Geoindex

GEOLOGICAL SURVEY PROFESSIONAL PAPER 1172

Geoindex

By PATRICIA FULTON *and* HAROLD JOHNSON,
assisted by WILLARD L. MCINTOSH, MARGARET EISTER, LAWRENCE BALCERAK,
DONALD HANSON, RICHARD THOENSEN, *and* PEARL PORTER

GEOLOGICAL SURVEY PROFESSIONAL PAPER 1172

Data base and data-base management system for the index to geologic maps

UNITED STATES GOVERNMENT PRINTING OFFICE, WASHINGTON:1982

UNITED STATES DEPARTMENT OF THE INTERIOR

JAMES G. WATT, *Secretary*

GEOLOGICAL SURVEY

Dallas L. Peck, *Director*

Library of Congress catalog-card No. 82-600504

For sale by the Superintendent of Documents, U.S. Government Printing Office
Washington, D.C. 20402

CONTENTS

ILLUSTRATIONS

CONVERSION FACTORS

Metric unit		Inch-Pound equivalent
	Length	
millimeter (mm)	= 0.03937	inch (in)
meter (m)	= 3.28	feet (ft)
kilometer (km)	= .62	mile (mi)
	Area	
square meter (m^2)	= 10.76	square feet (ft^2)
square kilometer (km^2)	= .386	square mile (mi^2)
hectare (ha)	= 2.47	acres
	Volume	
cubic centimeter (cm^3)	= 0.061	cubic inch (in^3)
liter (L)	= 61.03	cubic inches
cubic meter (m^3)	= 35.31	cubic feet (ft^3)
cubic meter	= .00081	acre-foot (acre-ft)
cubic hectometer (hm^3)	=810.7	acre-feet
liter	= 2.113	pints (pt)
liter	= 1.06	quarts (qt)
liter	= .26	gallon (gal)
cubic meter	= .00026	million gallons (Mgal or 10^6 gal)
cubic meter	= 6.290	barrels (bbl) (1 bbl=42 gal)
	Weight	
gram (g)	= 0.035	ounce, avoirdupois (oz avdp)
gram	= .0022	pound, avoirdupois (lb avdp)
metric tons (t)	= 1.102	tons, short (2,000 lb)
metric tons	= 0.9842	ton, long (2,240 lb)
	Specific combinations	
kilogram per square centimeter (kg/cm^2)	= 0.96	atmosphere (atm)
kilogram per square centimeter	= .98	bar (0.9869 atm)
cubic meter per second (m^3/s)	= 35.3	cubic feet per second (ft^3/s)

Metric unit		Inch-Pound equivalent
	Specific combinations—Continued	
liter per second (L/s)	= .0353	cubic foot per second
cubic meter per second per square kilometer [$(m^3/s)/km^2$]	= 91.47	cubic feet per second per square mile [$(ft^3/s)/mi^2$]
meter per day (m/d)	= 3.28	feet per day (hydraulic conductivity) (ft/d)
meter per kilometer (m/km)	= 5.28	feet per mile (ft/mi)
kilometer per hour (km/h)	= .9113	foot per second (ft/s)
meter per second (m/s)	= 3.28	feet per second
meter squared per day (m^2/d)	= 10.764	feet squared per day (ft^2/d) (transmissivity)
cubic meter per second (m^3/s)	= 22.826	million gallons per day (Mgal/d)
cubic meter per minute (m^3/min)	=264.2	gallons per minute (gal/min)
liter per second (L/s)	= 15.85	gallons per minute
liter per second per meter [(L/s)/m]	= 4.83	gallons per minute per foot [(gal/min)/ft]
kilometer per hour (km/h)	= .62	mile per hour (mi/h)
meter per second (m/s)	= 2.237	miles per hour
gram per cubic centimeter (g/cm^3)	= 62.43	pounds per cubic foot (lb/ft^3)
gram per square centimeter (g/cm^2)	= 2.048	pounds per square foot (lb/ft^2)
gram per square centimeter	= .0142	pound per square inch (lb/in^2)
	Temperature	
degree Celsius (°C)	= 1.8	degrees Fahrenheit (°F)
degrees Celsius (temperature)	=[(1.8×°C)+32]	degrees Fahrenheit

GEOINDEX

By PATRICIA FULTON and HAROLD JOHNSON,
assisted by WILLARD L. MCINTOSH,
MARGARET EISTER, LAWRENCE BALCERAK, DONALD HANSON,
RICHARD THOENSEN, and PEARL PORTER

ABSTRACT

The acquisition and the dissemination of information are ever-increasing problems for Federal agencies engaged in research. To facilitate the publication of its geologic index maps, the U.S. Geological Survey has moved toward computer-based operations. The index to geologic maps (Geoindex) has been established and developed as a data base and data-base management system that provides three main capabilities. The primary capability is to provide the means to generate rapidly geologic index maps for publication. A second capability is to provide users an immediate access to all items in the data base. The third capability is to provide nationwide summary information to policy makers.

INTRODUCTION

The first U.S. Geological Survey indexes to geologic maps were published in the 1940's. They consisted of State base maps at scales of 1:750,000 or 1:1,000,000 on which the outlines of published geologic maps were shown. By the mid-1960's, most of the indexes were out of date. Revision was delayed because of the rising cost of color printing (six press runs) and the mechanical difficulty of showing legibly the additional (doubled) coverage produced in the 1950's and 1960's.

Some of the problems were solved in the publication of the Montana index in 1969. Heretofore, geologic index maps had shown all the geologic coverage. In the Montana index, only maps equal or better in quality and comprehensiveness than the State geologic map were included. This limitation imposed a reasonable and standard criterion for determining what should be included in the index. All very small scale maps, as well as many sketchy or generalized maps, were omitted. Elimination of such material produced a more legible index without serious loss of geologic-map coverage. The revised index, like previous indexes, included both published and open-file maps of the U.S. Geological Survey, published maps of the State surveys, and maps published by other organizations.

When computer-assisted techniques were introduced, the project grew, and ideas continued to change and evolve. Henceforth, maps published at the following scale ranges will be indexed on three separate sheets:
1. Scales of 1:24,000 and larger
2. Scales smaller than 1:24,000, through and including 1:63,360
3. Scales smaller than 1:63,360, through and including 1:250,000

ACKNOWLEDGMENTS

The authors thank Joseph Moses Botbol and Roger Bowen for their assistance in the use of the GRASP system. Diane Lewis and Karen Shallcross completed compilation and entered the data into the Geoindex data base; Jane Timmins typed the manuscript and arranged it in the proper sequence; James Fisher, Kevin Laurent, and Maryjon McAvery helped in writing the computer programs and in constructing the data base. All these people are from the U.S. Geological Survey.

We also thank William Strauss, Denise Maurer, and Ray Wisecarver of the Johns Hopkins Applied Physics Laboratory for their invaluable assistance in the digitization of the data.

THE DATA-BASE MANAGEMENT SYSTEM

PURPOSE

The primary purpose of the Geoindex Data-Base Management System is to generate geologic index maps as quickly as possible. These published reports are widely distributed to a large, diverse group. A published geologic index consists of a series of map sheets and accompanying text material. The Geoindex is constructed State by State. Each map sheet consists of a State base map on which the outlines of published geologic maps and identifying numbers are superimposed (Fulton and McIntosh, 1977). Figure 1 shows a map sheet from the published index for the State of Kentucky.

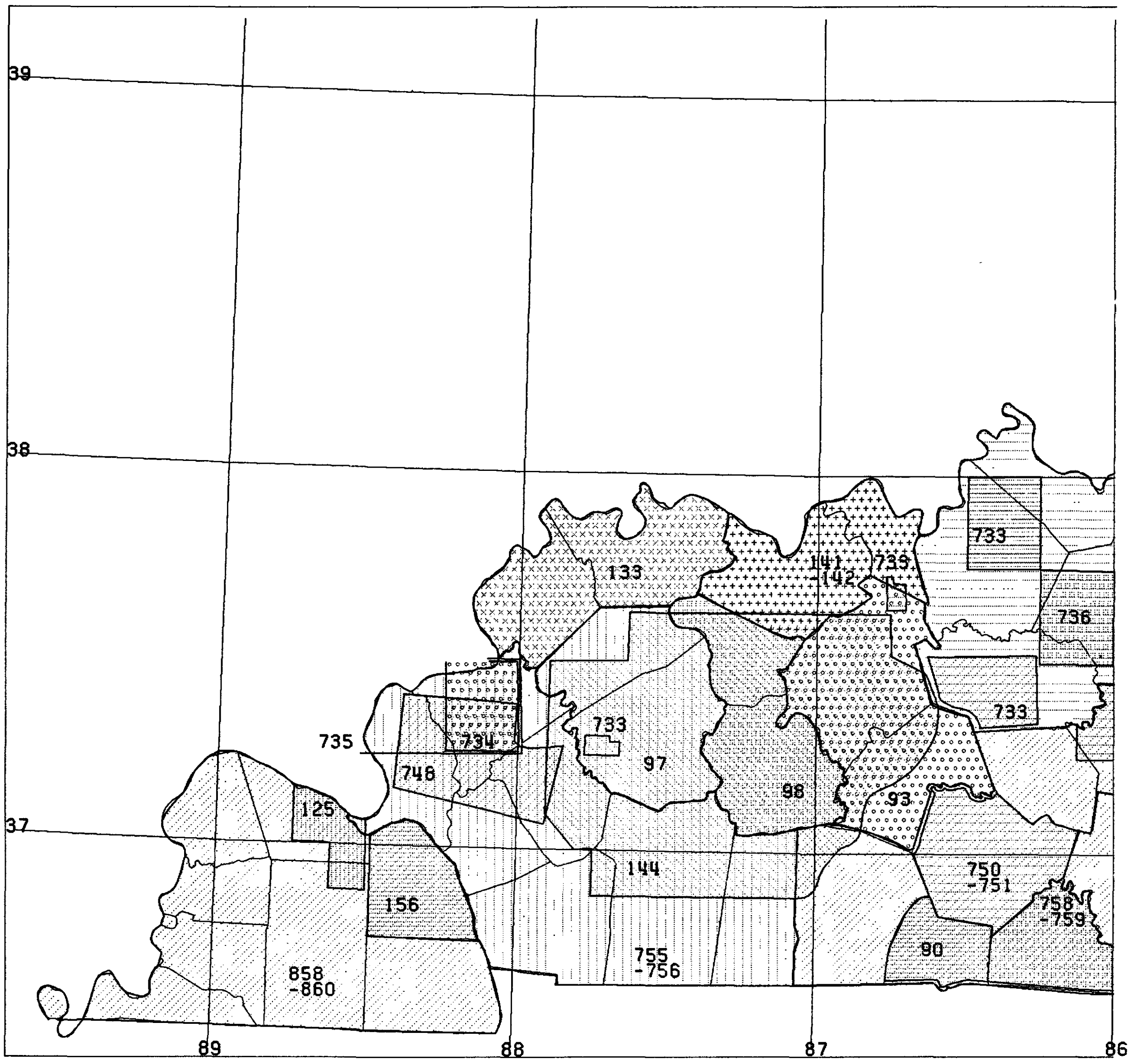
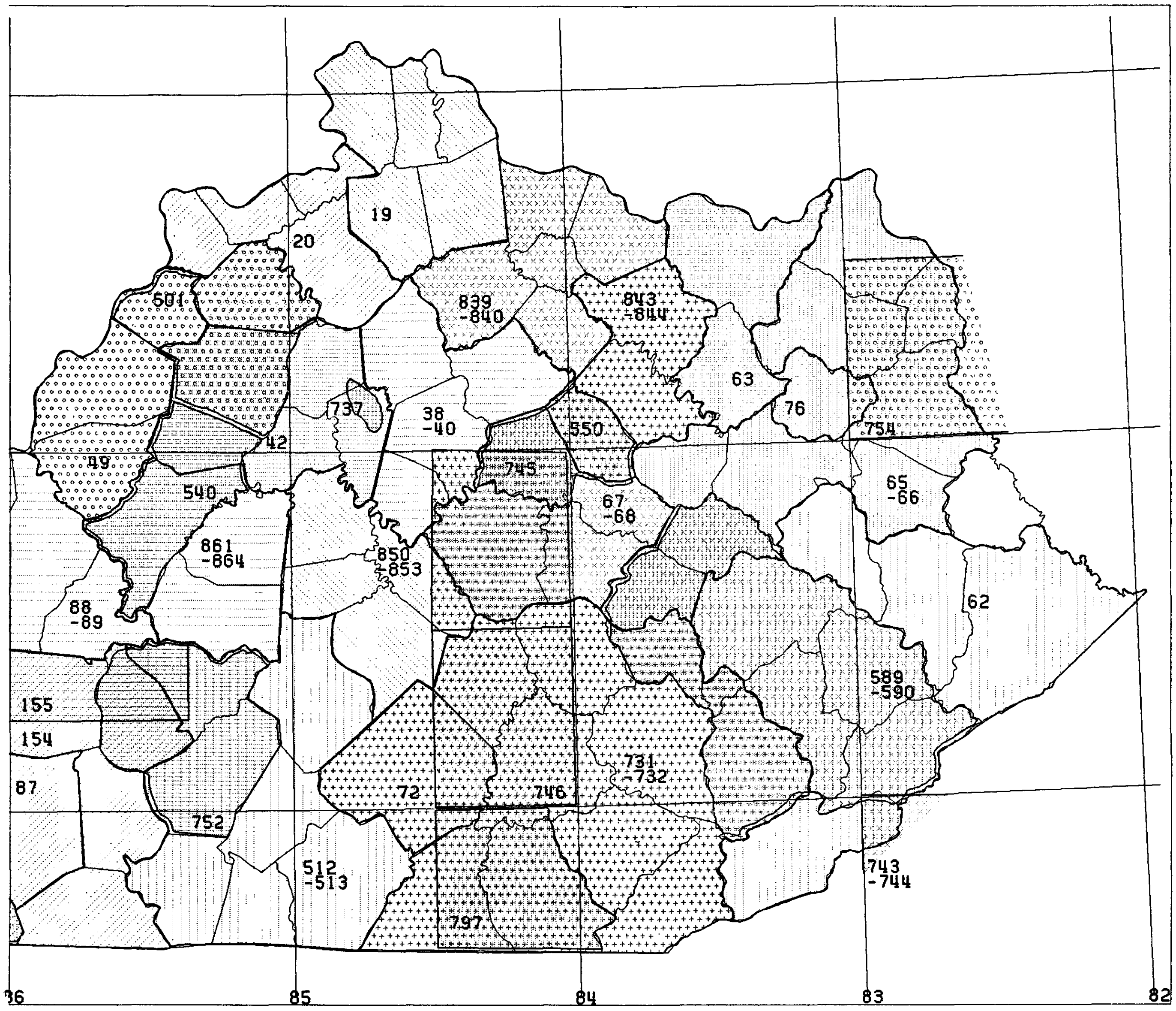


FIGURE 1.—Sheet from the published Kentucky index, showing geologic maps

The text material is composed of bibliographic references, each of which is numbered to correspond to the number on the matching map outline. Figure 2 shows part of a sheet of bibliographic text from the published index for the State of Kentucky.

Each index, then, contains the map outlines and bibliographic references for published geologic maps for one State. The reports are published in black and white and folded to a size that fits the standard file cabinet. Because the computer-based operations have introduced economies, geologic map indexes are distributed free to the public.

A second purpose of a data base in machine readable form is to furnish outside users a rapid access to all items contained in the data base. This second function is required by a more specialized group of outside users. For example, some specialists in the Earth sciences have active projects and need immediate answers to their questions. Typically, the procedures involve the retrieval of selected items from the data base, followed by a series of manipulations, and finally a display of various combinations of text and graphics. Alternatively, some outside users whose projects are still in the planning stage need information that is contained in the data base but does not appear on the published geologic index map. To satisfy the two types of users, both text and map data are available online from the computer system. Aspects of this type of usage of the data-base



whose scales range from smaller than 1:63,360 through and including 1:250,000.

management system are discussed later, under Storage and Retrieval System.

A third purpose of the data-base management system is to provide nationwide information to policy makers. Under this concept, one can access the data base as a single entity covering the entire United States instead of accessing merely one State at a time. The system can present a comprehensive overview of the whole country, but all the details shown on State maps are still available. For example, the total area for which geologic maps are available in the United States can be computed. Such statistics are valuable for national planning.

To fulfill these requirements, the geologic map index exists in two very different forms: as computer files in the Geoindex data base, and as published reports. An additional reason for having two different forms is that as soon as data have been stored in a computer, new applications become feasible. As a result, computer resident data files serve a much larger community of users than previously imagined. Because the data files are in digital form, they become multifunctional in that retrievals from the files can assume totally different appearances. These additional capabilities more than justify the initial cost in creating machine readable map files.

GLOSSARY AND SYSTEM STRUCTURE

A technical dicussion of the data base structure first requires the definition of some of the terms (Honeywell Information Systems, Inc., 1978).

1. **Data base.** An integrated collection of data upon which operations (such as read, write, and revise) can be performed.
2. **Data-base management system.** A software system that accesses an integrated collection of data.
3. **User.** A person who retrieves, updates, or deletes data within the data base. Such a person actively maintains the data base.
4. **Outside user.** A person who retrieves data from the data base. This is a person who uses the system but does not maintain it.
5. **Data model or schema.** The description of the data base that defines the characteristics and organization of the data within the data base.

The Geoindex data base is a relational data base and is derived from the mathematical theory of relations. The Geoindex Data Base Management System is a relational data-base management system. This structure is a natural consequence of having the Geologic Retrieval and Synopsis Program (GRASP) as the primary storage and retrieval program of the system because GRASP organizes data in relational form. GRASP is discussed more fully in the section entitled Storage and Retrieval System. The relational form is essentially a matrix composed of the familiar rows and columns.

The mathematical terminology specific to a relational data base must also be defined:

6. **File.** A collection of organized data, a relation.
7. **Record.** A representative "row" of data, a **tuple.**
8. **Attribute.** Name of a data field within a record, a column of information.
9. **Attribute value.** Value of a data field within a record.
10. **Domain.** The set of all values a data field may assume.
11. **Data submodel.** User's definition of the data base.
12 **Data model.** Total definition of the data base.

A relational data base is in matrix form where the tuples (records) constitute the rows and the attributes (data items) constitute the columns. All tuples within a given relation have the same format (all records within a file have the same format). This last statement is a definitive characteristic of a relational data base.

The Geoindex is both large and complex. Size, of course, contributes to complexity, but the major source of complexity is the nature of the data that the system must process. The data comprise two distinct types: text and graphics. A record ("row," definition 7) exists for each map. The attribute values (columns, definition 9) are composed of text data derived from the bibliographic reference and of graphic data derived from the map outline. See figures 1 and 2. The complete list of attributes handled by the data-base management system is shown in figure 3.

The data-base management system is functionally divided into four parts. The first part is composed of computer programs and procedures designed to perform two vital tasks. The first task is to capture and verify the data. The second task is to create the map sheets and bibliographic text sheets as camera copy ready for reproduction. The first part of the system will be discussed in the section Data Acquisition and the section Publication Copy. The second part of the system is composed of computer programs and procedures that have one task to accomplish. This part of the system loads the data as relations into files that are accessible to GRASP, the storage and retrieval program. The third part of the system consists of GRASP and several plot programs. This third part is described in the section Storage and Retrieval System. The fourth part of the system consists of computer programs and procedures that insure the safety and integrity of the data by providing backup files and permanent archival data storage.

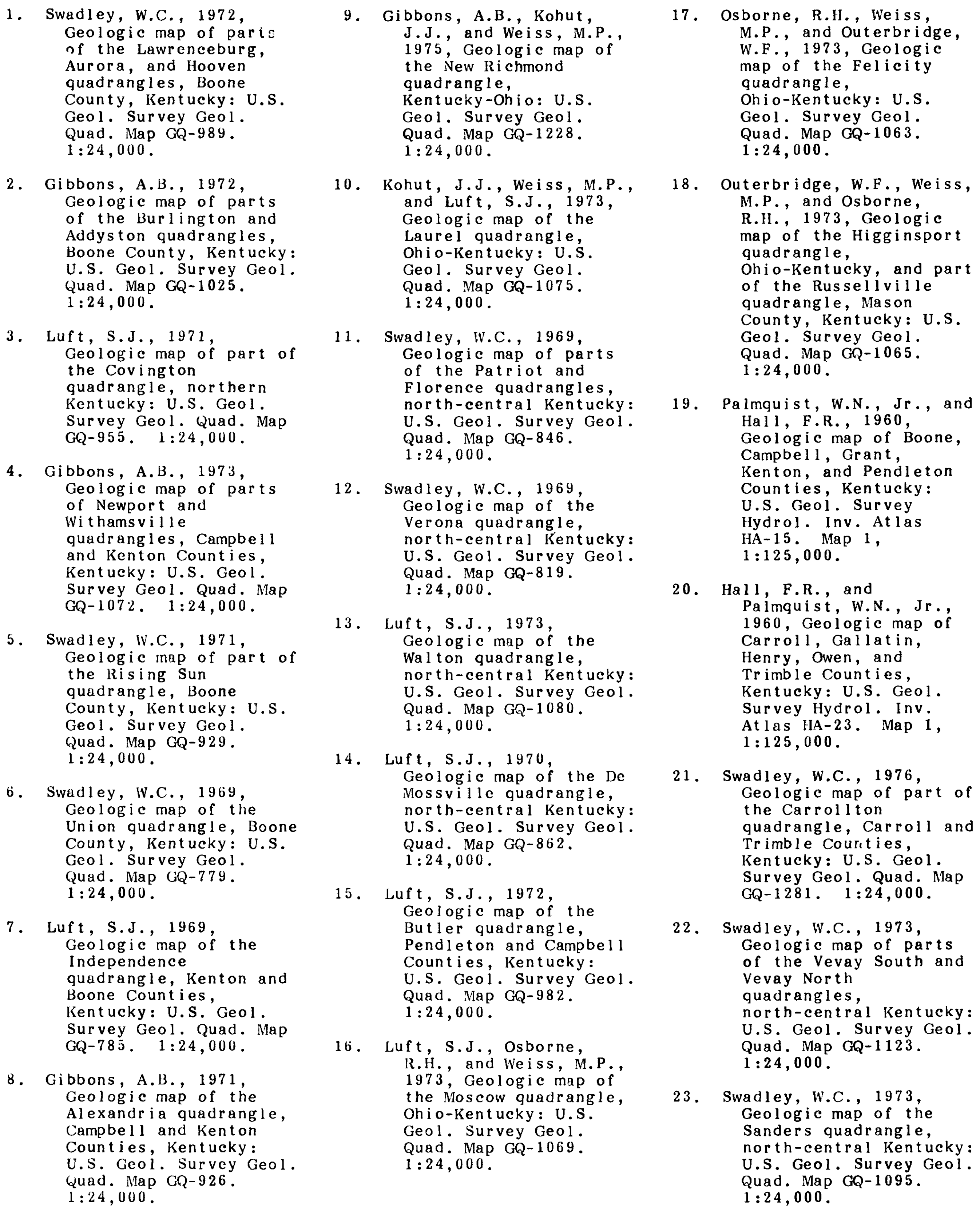
The system flow chart, illustrated in Appendix A, shows the chronological work flow that is virtually identical with parts one through four mentioned.

DATA ACQUISITION

TEXT DATA

The text data for an individual State are received in draft form, which is somewhat similar to that shown in figure 2. Each draft is examined, and a list of questions is prepared to cover any errors, omissions, or ambiguities that would slow the actual data-entry process. This list is returned to the geologist who compiled the index map and who then clarifies the uncertainties. After questions are answered and this list is returned, the physical keying of the text begins. The attributes, or record items, entered at this time are listed as follows: Identification number, author, year, title, publisher, county or region, emphasis, scale, and series.

The text material is prepared offline in card-image form on key-to-disk devices. These are word-processor computer terminals, which function both as stand-alone, offline, data-entry stations and as communications terminals. Several types are available; however, each hardware unit includes a keyboard for data entry, a cathode ray tube (CRT) screen that displays the characters entered from the keyboard and messages sent from a computer, and dual flexible disks that store data. Line printers, some switch-selectable among the units, supply the necessary hard copy.



1. Swadley, W.C., 1972, Geologic map of parts of the Lawrenceburg, Aurora, and Hooven quadrangles, Boone County, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-989. 1:24,000.

2. Gibbons, A.B., 1972, Geologic map of parts of the Burlington and Addyston quadrangles, Boone County, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1025. 1:24,000.

3. Luft, S.J., 1971, Geologic map of part of the Covington quadrangle, northern Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-955. 1:24,000.

4. Gibbons, A.B., 1973, Geologic map of parts of Newport and Withamsville quadrangles, Campbell and Kenton Counties, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1072. 1:24,000.

5. Swadley, W.C., 1971, Geologic map of part of the Rising Sun quadrangle, Boone County, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-929. 1:24,000.

6. Swadley, W.C., 1969, Geologic map of the Union quadrangle, Boone County, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-779. 1:24,000.

7. Luft, S.J., 1969, Geologic map of the Independence quadrangle, Kenton and Boone Counties, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-785. 1:24,000.

8. Gibbons, A.B., 1971, Geologic map of the Alexandria quadrangle, Campbell and Kenton Counties, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-926. 1:24,000.

9. Gibbons, A.B., Kohut, J.J., and Weiss, M.P., 1975, Geologic map of the New Richmond quadrangle, Kentucky-Ohio: U.S. Geol. Survey Geol. Quad. Map GQ-1228. 1:24,000.

10. Kohut, J.J., Weiss, M.P., and Luft, S.J., 1973, Geologic map of the Laurel quadrangle, Ohio-Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1075. 1:24,000.

11. Swadley, W.C., 1969, Geologic map of parts of the Patriot and Florence quadrangles, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-846. 1:24,000.

12. Swadley, W.C., 1969, Geologic map of the Verona quadrangle, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-819. 1:24,000.

13. Luft, S.J., 1973, Geologic map of the Walton quadrangle, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1080. 1:24,000.

14. Luft, S.J., 1970, Geologic map of the De Mossville quadrangle, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-862. 1:24,000.

15. Luft, S.J., 1972, Geologic map of the Butler quadrangle, Pendleton and Campbell Counties, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-982. 1:24,000.

16. Luft, S.J., Osborne, R.H., and Weiss, M.P., 1973, Geologic map of the Moscow quadrangle, Ohio-Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1069. 1:24,000.

17. Osborne, R.H., Weiss, M.P., and Outerbridge, W.F., 1973, Geologic map of the Felicity quadrangle, Ohio-Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1063. 1:24,000.

18. Outerbridge, W.F., Weiss, M.P., and Osborne, R.H., 1973, Geologic map of the Higginsport quadrangle, Ohio-Kentucky, and part of the Russellville quadrangle, Mason County, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1065. 1:24,000.

19. Palmquist, W.N., Jr., and Hall, F.R., 1960, Geologic map of Boone, Campbell, Grant, Kenton, and Pendleton Counties, Kentucky: U.S. Geol. Survey Hydrol. Inv. Atlas HA-15. Map 1, 1:125,000.

20. Hall, F.R., and Palmquist, W.N., Jr., 1960, Geologic map of Carroll, Gallatin, Henry, Owen, and Trimble Counties, Kentucky: U.S. Geol. Survey Hydrol. Inv. Atlas HA-23. Map 1, 1:125,000.

21. Swadley, W.C., 1976, Geologic map of part of the Carrollton quadrangle, Carroll and Trimble Counties, Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1281. 1:24,000.

22. Swadley, W.C., 1973, Geologic map of parts of the Vevay South and Vevay North quadrangles, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1123. 1:24,000.

23. Swadley, W.C., 1973, Geologic map of the Sanders quadrangle, north-central Kentucky: U.S. Geol. Survey Geol. Quad. Map GQ-1095. 1:24,000.

FIGURE 2.—Part of a sheet of bibliographic references taken from the published Kentucky Index.

Each map is uniquely identified and has the following list of attributes:

Mnemonic	Attribute
id	identifying number for the bibliographic reference
state	name of the state
author	authors
year	year of publication
title	title
county	county or region
publish	publisher
series	title of publication series
emphasi	type of geology - surficial, economic, stratigraphic, oil, gas, coal, metal
area	area covered by map
aunit	dimension for area, generally square kilometers
nlat	extreme north latitude
slat	extreme south latitude
wlong	extreme west longitude
elong	extreme east longitude
clat	center point latitude
clong	center point longitude
omaps	other maps not included as outlines, i.e., title
avail	depositories where maps can be obtained
base	USGS topo, DMA-TC topo, photomosaic, shaded relief
geology	only geology shown on the map
plate	plate or map or sheet identification
idstate	FIPS state code
scale	map scale - 1:24,000, 1:250,000, etc.
idsub	sub id number, i.e., more than one map
ibound	id number on the boundary outline ties together text and graphic or x,y files
ispan	secondary number on the boundary outline further ties text to graphic file
othermap	phrase used in Bibliography

FIGURE 3. – List of attributes (names of data fields) that constitute a record.

After a line of text data is entered on the keyboard, it is stored on the diskette. At the option of the operator, the data are simultaneously listed a line at a time on the printer or listed all together at the end of a session. This printer list is then checked for errors. The operator corrects data by using a key-to-disk technique. When text data are as error free as possible, the data-entry devices are operated as computer terminals, and the data are transmitted directly from the diskette to permanent storage on a large host computer.

MAP DATA

The map data, as previously mentioned, are the outlines of published geologic maps. These map data are received from the geologist as ink or colored-pencil outlines overlaid on base maps. The base maps consist of green-line images printed on dimensionally stable plastic.

The computer group codes a matching base on a stable material in preparation for the digitization of county outlines. These codes are the digits assigned to each county and State by the Federal Information Processing Standards Publications (FIPS PUBS). This digitization or conversion from graphics to machine-readable code is done manually on a type of drafting-table digitizer with a resolution of 0.025 mm (0.001 in.) (Fulton, 1975).

The lower left-hand corner of the neat line is designated as the origin, so that the entire map lies within the positive quadrant in the Cartesian coordinate system.

Only the end points of straight-line segments are recorded. Where the outlines are extremely convoluted, stream digitizing is performed and the spacing between points is generally about 1 mm (0.04 in.). The data are plotted for verification. Because of the complexity of the index maps, these plots must be drawn in different colors so that each outline and its identifying number is distinct.

PUBLICATION COPY

After the computer-generated plots for the Geoindex are judged acceptable, they need further processing to create the final version of the maps. The first few published indexes were drawn in black ink by means of a drum plotter. Several different pens created the various line weights so that overlapping areas could be distinguished.

Index maps convey information concerning areas. Shading portrays areal information very effectively. Thus, those who prepared the later maps took advantage of the newer technology inherent in a matrix plotter. Such a plotter now generates the final maps. This plotter has a resolution of 160 dots per inch and an effective plotting width of 18 in. To date, 10 different patterns supply sufficient contrast so that various mapped areas may be distinguished from one another. See figure 1, which shows a Kentucky map sheet listing the maps that range in scale from 1:63,360 through 1:250,000.

At the same time that the map plots are generated, the text data are processed. The file is in a machine-readable form almost identical with card-image form, but the final output must be in the traditional bibliographic form. This change is accomplished by processing the text data file through programs resident on a main-frame computer that strip out the extraneous data and codes and rearrange the order of the attributes (items). Then the modified file is transmitted over telephone lines back to the word-processor terminal. The final manuscript is printed on coated paper automatically. The map plots and text material are then sent for photographic reduction, a process that creates the photographic plates for mass production. See figure 2, which shows a copy of part of a sheet of text data from the published Kentucky index.

STORAGE AND RETRIEVAL SYSTEM

This storage and retrieval system is the Geologic Retrieval and Synopsis Program (GRASP), written and developed within the Geological Survey by Roger W. Bowen and Joseph Moses Botbol (1975). GRASP is used extensively within the U.S. Geological Survey. It has also been installed on the computers of national agencies of several countries in South America and Europe.

The GRASP system implements searches of the text files by individual items or by any combination of two or more items. All the references for the published geologic maps that meet the search criteria are retrieved. The entire contents of all the retrieved text records can be listed, or only one or two items can be selected. However, the boundary identification number, ***ibound***, the bibliographic identification number, ***id***, and the subidentification number, ***idsub***, are the only items required to generate a graphic image. Thus, the numeric value of these three items are listed on a formatted disk file for subsequent plotting. The GRASP system is exited and the plot program **pn16** is invoked. This program, too, executes in an interactive mode. Its options provide for plots of the State outline, the graticule, the county boundaries in solid or dotted lines, and, of course, the file of geologic-map outlines. These graphics can be plotted interactively on a CRT terminal and then printed by a hard-copy unit attached to the terminal.

After a map has been drawn on the CRT, information from other files may be added by direct overlay to the original graphic on the screen. In addition to the features already described, this program offers the opportunity to enlarge any part of the plot repetitively until a cluttered area becomes readable.

At present, the newly published geologic index maps are categorized according to scale. The retrieval can be executed in GRASP by designating the proper scale as the search criteria and querying the file. This first step in map generation from digital data is accomplished interactively on a graphics CRT terminal. This same file, which has been plotted interactively, can then be directed to a drum plotter for reproduction at the original scale. Figure 4 shows the maps published at scales smaller than 1:63,360 for the State of North Dakota as drawn on the CRT screen.

Several other programs enable a user to plot the entire United States. The programs take geographic coordinates from either a GRASP retrieval or directly from the map-data files. These programs convert the geographic coordinates to Cartesian coordinates. Then another plot program entitled **pin90** operates in an interactive mode similar to that of **pn16** to plot these files. It, too, has options for specifying various combinations of files and enlarging designated areas. Figure 5 shows a CRT plot of some areas covered by published maps.

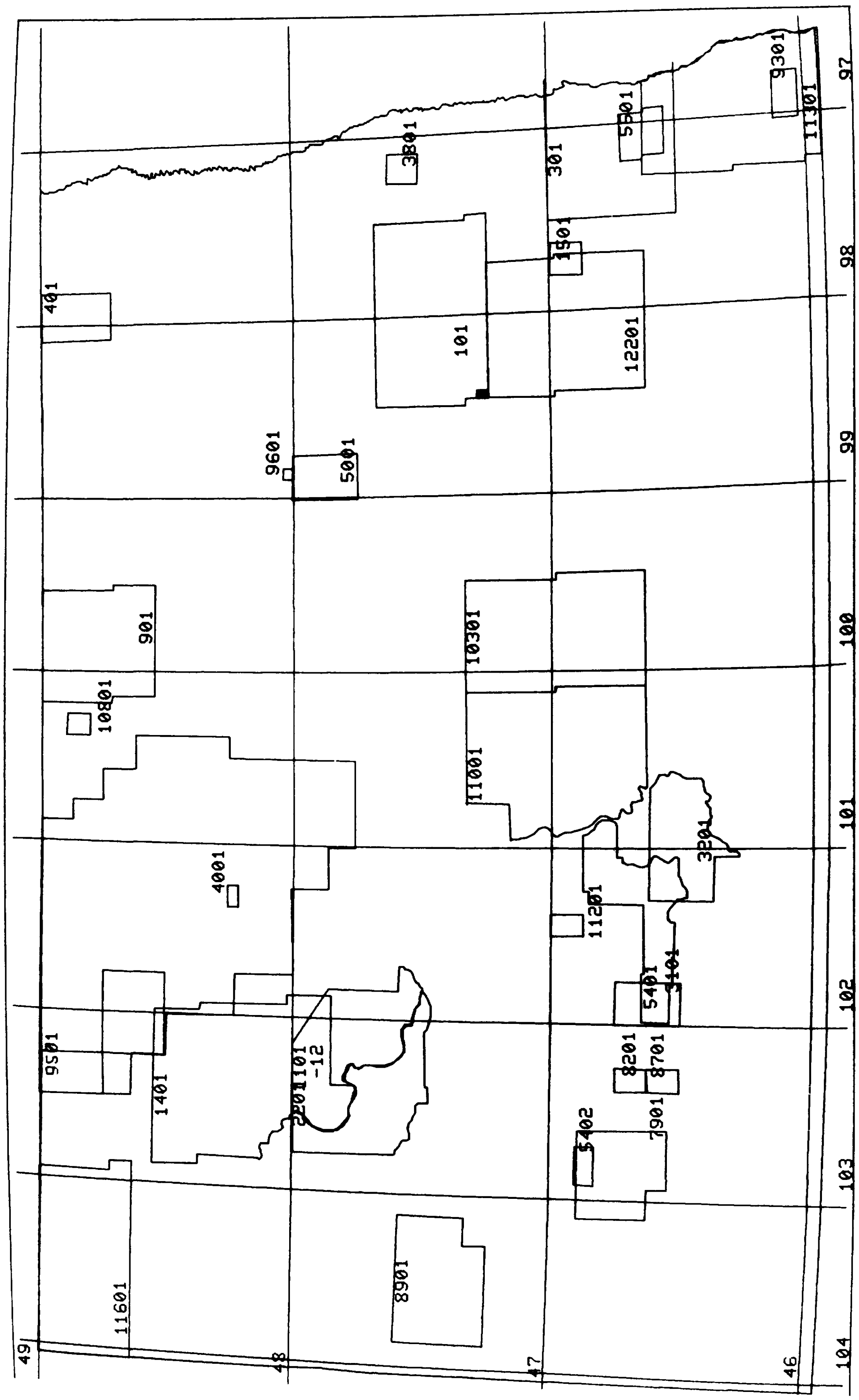


FIGURE 4.—CRT plot for the State of North Dakota, showing geologic maps published at scales smaller than 1:63,360. The figure is a photograph of the CRT screen.

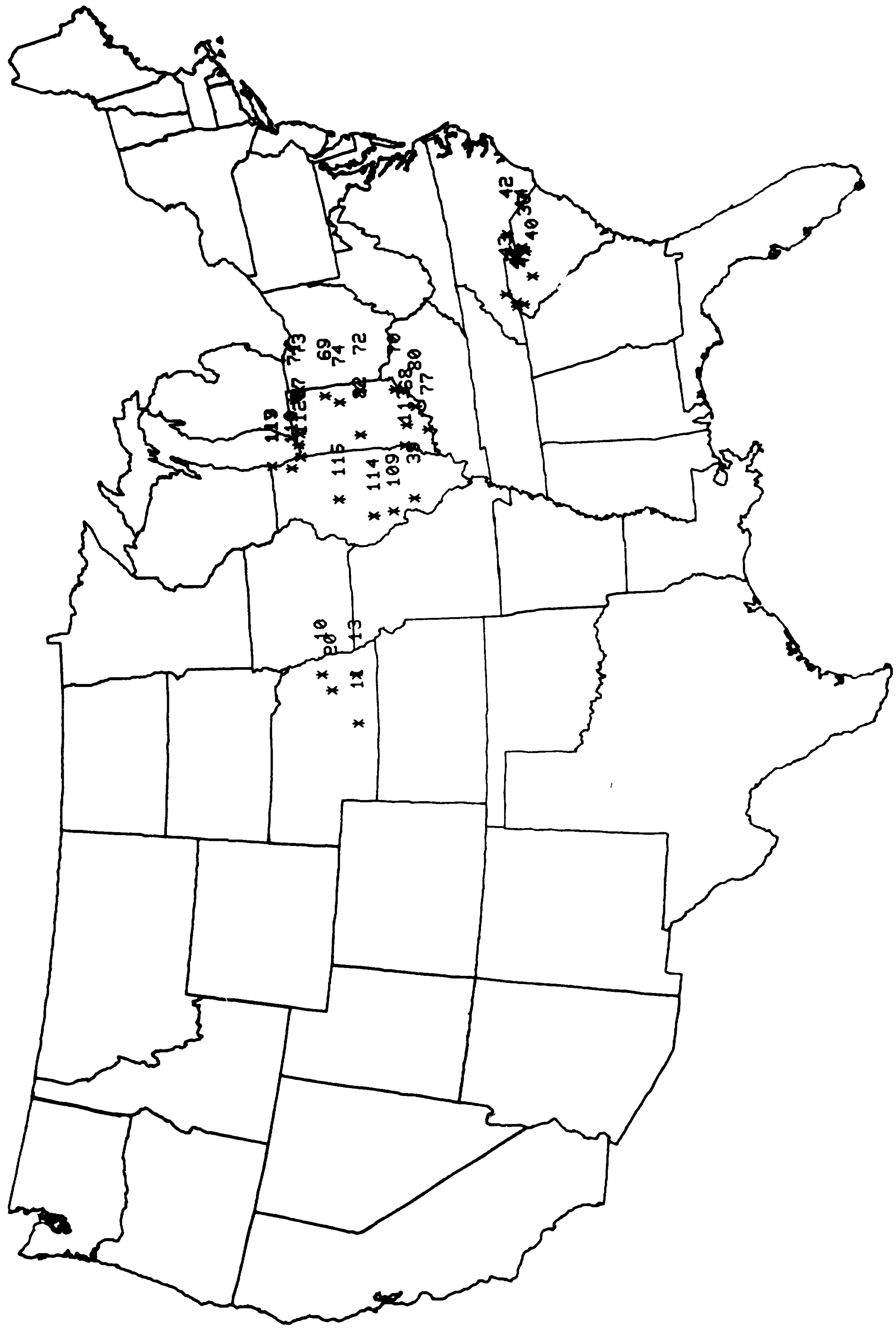


FIGURE 5.–CRT plot of the United States that shows some areas covered by published geologic maps. Figure is a photograph of the CRT screen.

The system also utilizes some machine-independent plotting packages that were obtained specifically to provide diversity of output for the Geoindex as well as for the entire U.S. Geological Survey. Figure 6, which illustrates geologic mapping on a regional basis, is a map showing all the areas covered by geologic maps in the western States of Idaho, Nevada, and Arizona.

The storage and retrieval part of the Geoindex system has the capability of providing nationwide summary information to policy makers. Examples of maps that present information of special interest to national planners are shown in figures 7 and 8. Both of these maps were generated from queries to the Geoindex and show geologic mapping on a national basis. These maps reflect the data resident in the data base at the time of query. Figure 7 shows the areas in the United States covered by geologic maps that were published at a scale of 1:250,000 after 1960. It includes both U.S. Geological Survey maps and non-Survey maps. Figure 8 shows geologic maps from all sources that were published at scales larger than 1:250,000.

DATA BASE

STRUCTURE OF THE DATA

TEXT DATA

As previously mentioned, the Geoindex comprises two distinct types of data: text and graphic. These two data types are handled separately throughout most of the system because of their dissimilar nature. After the bibliographic data are captured as keystrokes of coded data in digital form, they are usually called text data.

Initially, the text data in machine-readable form look very much like a listing of ordinary punched cards. The most obvious difference is that a printed line contains both uppercase and lowercase characters. Actual data fields vary from 4 to 60 characters. The Geoindex data base is generated on a State-by-State basis. Each State carries the two-digit code assigned by FIPS PUBS. Every file name for a particular State contains this same numeric code as a suffix. The leading three or four characters of the file name are descriptive of the type of data in the file. Thus, the initial reference data file for Colorado is named *ref08*.

The format for the text data is as follows:

State identification	2 digits.
Reference number	4 digits.
Item number	2 digits.
Informational data	4 to 60 characters.

Figure 9 shows the text data for reference number 90 (a record in file *ref08*) for the State of Colorado as it looks in its initial form. This is a reference that contains four separate maps, and it was chosen to illustrate the complexities of the data structure.

MAP DATA

As stated earlier, the map data are digitized as Cartesian coordinates in the positive quadrant. These coordinate files are structured so that several different types of map data are compatible and are handled efficiently within the one system. The two major types of graphic data are the index-map coordinates and the base-map coordinates. The base-map coordinates consist of political boundaries, such as State and county, and also the geographical positions of the graticule. One bibliographic reference may contain several maps, one map, or no map at all. Conversely, one map outline, typically a county, may be identified by a great number of bibliographic references. A unique identifier for each map is mandatory and is a combination of three attributes. A primary identification number (*id*), a secondary identification number (*idsub*), and a third number (second *idsub*) insure uniqueness. The data for each map outline are composed of two different parts. The first or header section under a format of (8I5) consists of number codes for the various map features. The features and the feature codes are listed as follows:

1. Identification or feature number (*id*): neatline = 900, State = 9*NM* (*NM* refers to FIPS code for the particular State)
2. Number of outlines that have this *id*
3. First subfeature: adjacent county *id* number, adjacent State = 9*NM*, national boundary = 993, lake boundary = 995
4. Number of points
5. Second subfeature number
6. State *id* number
7. Graticule = 991, county = 992, island = 994 (values recorded only for grid, county, and islands; blank for others)
8. Span—that is, one map outline for several references

The second part of the record has a format of (12F6.3) and contains the string of Cartesian (x, y) coordinates that define the boundary of the published geologic map. The first Cartesian pair indicates the position for the identifying number. A listing of both parts of the Cartesian coordinate data record for reference number 90 for the State of Colorado is shown in figure 10.

TEXT AND MAP DATA

After the data are in digital form, only the map file (fig. 10) contains the information that can be used to complete the record, the list of attributes named in figure 3. The area covered by each geologic map is computed from the Cartesian coordinates, and that information is then stored along with the unit of measurement (currently square kilometers). All Cartesian coordinates are also transformed into latitude and longitude and stored as radians. The header cards for the radian files are identical with the header cards for the Cartesian coordinate files. The data are in card-image form with a format of (6F12.9,I8). The decimal point is implied in the data files so that there are three latitude-longitude pairs per card image with space for sequence numbers. Figure 11 shows the radian data for reference number 90 for Colorado. The names of the files are similar.

Using the radian values, a program determines the maximum latitude and longitude for the four directions and then stores each map outline. The center of each map outline is computed in radians and is stored. The data-base management system performs these and other computations. The items listed above, derived from the map data, are merged into the record, or tuple, so that a complete record for each map contains all the information shown in figure 3. These two files, text and graphic, are compared to ensure that each reference is identified by the correct outline. Figure 12 shows a text-data record in its complete form. It represents the final form for reference number 90 for the State of Colorado. These text data correlate with the map-coordinate data shown in figure 10. The data shown in figure 12 are in the format required for the storage and retrieval system.

Twelve files are stored permanently on two sets of magnetic tapes. Each magnetic tape contains the data for five States. Figure 13 describes and identifies these files.

SYSTEM SPECIFICATIONS

The detailed system specifications are given in the appendixes. Appendix A contains the system flow chart, which also shows the input and output files. Appendix B contains the operational instructions, which detail the minimum set of information needed to execute the programs.

Appendix C contains the computer-program reference guide. This guide gives complete descriptions of the computer programs and listings of the source code for each program. Appendix D contains the formats and notes needed for data entry.

STATUS OF THE SYSTEM

The data-base management system has passed the operational phase and is a fully functional system. The primary objective, the generation of geologic index maps, is in a production mode, and the data base is growing daily. Figure 14 is another computer graphic that summarizes the present status of the Geoindex. The system automatically generates a new status map at the beginning of each month. The files can be accessed in an interactive mode, and personalized index maps plotted immediately, as shown in figures 4 and 5. The system becomes increasingly useful as a tool for policy makers as more States are added to the data base. Figures 6, 7, and 8 show summary maps that can be of value in making policy and plans.

REFERENCES CITED

Bowen, R. W., and Botbol, J. M., 1975, The Geologic Retrieval and Synopsis Program (GRASP): U.S. Geological Survey Professional Paper 966, 87 p.

Fulton, P. A., 1975, Mapping and Computers, *in* Rubinoff, Morris, and Yovits, M. C., eds., Advances in Computers, v.13: New York, Academic Press, p. 73–108.

Fulton, P. A., and McIntosh, W. L., 1977, Computerized Data Base for the Geomap Index: The American Cartographer, v. 4, no. 1, pp. 29–37.

Honeywell Information Systems, Inc., 1978, Level 68 Software Multics Relational Data Store (MRDS) Reference Manual, p. 1-1, 2-3.

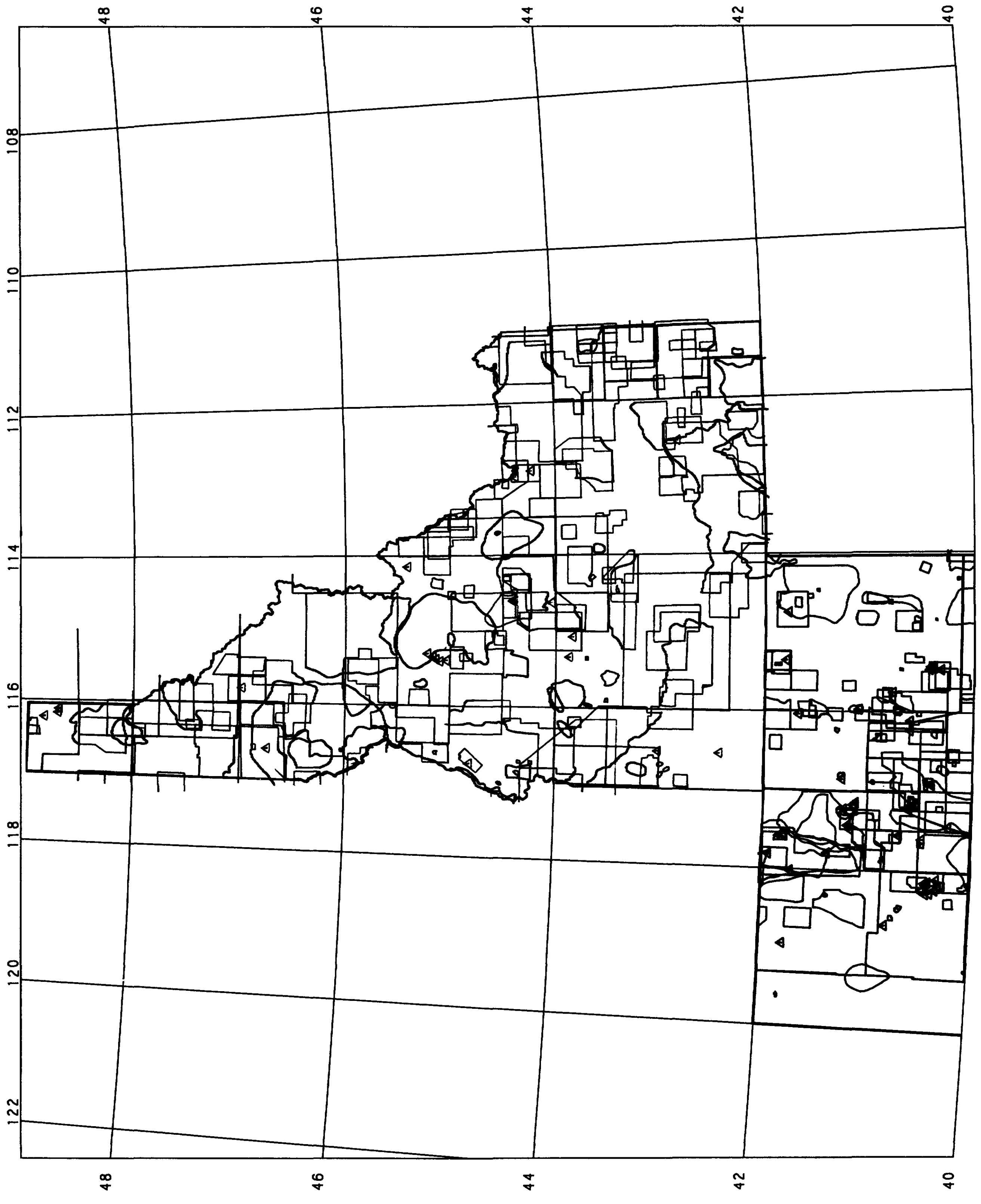


FIGURE 6.—Plot showing all the areas for which geologic maps have been published in the States of Idaho, Nevada, and Arizona.

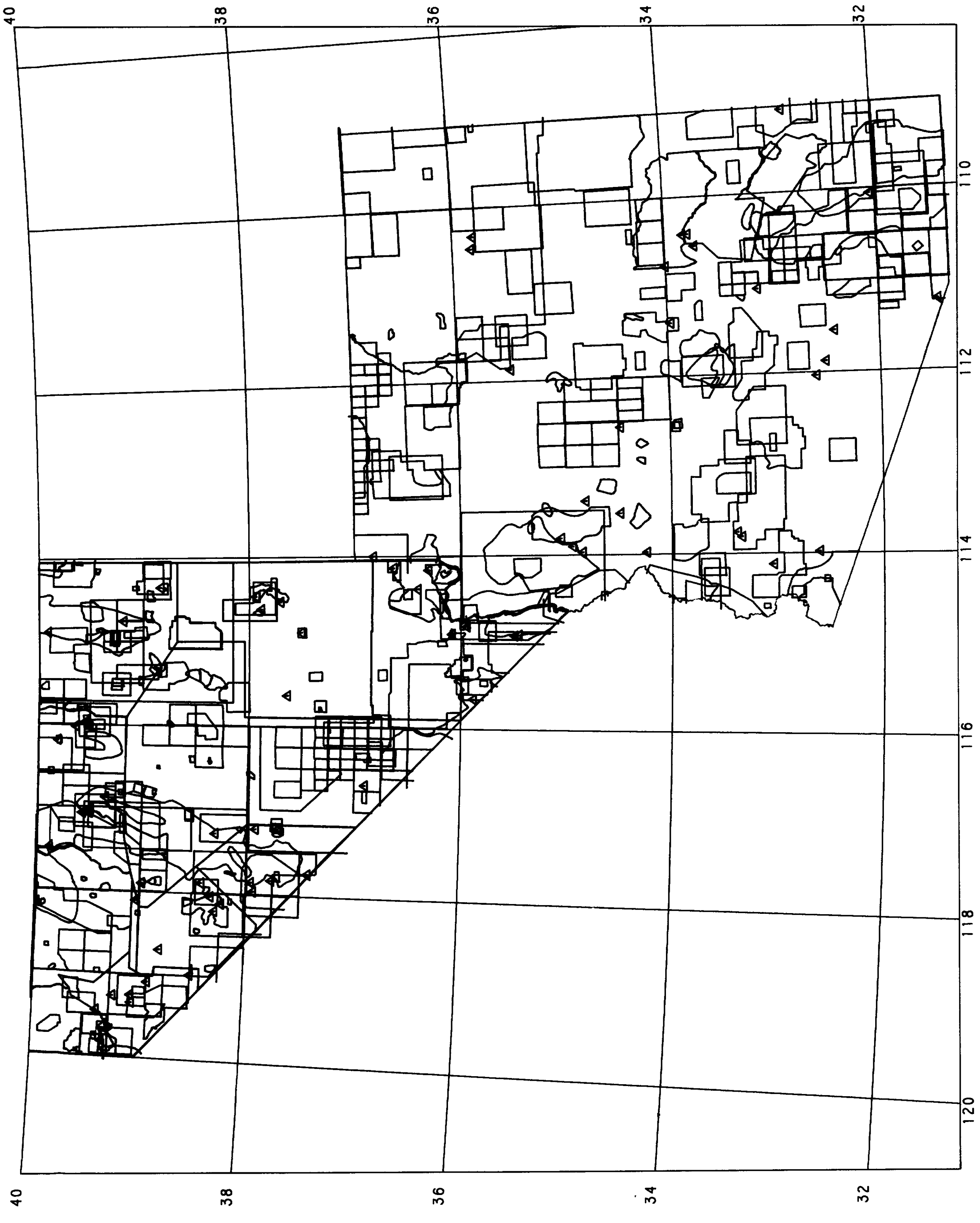


FIGURE 6.—Continued.

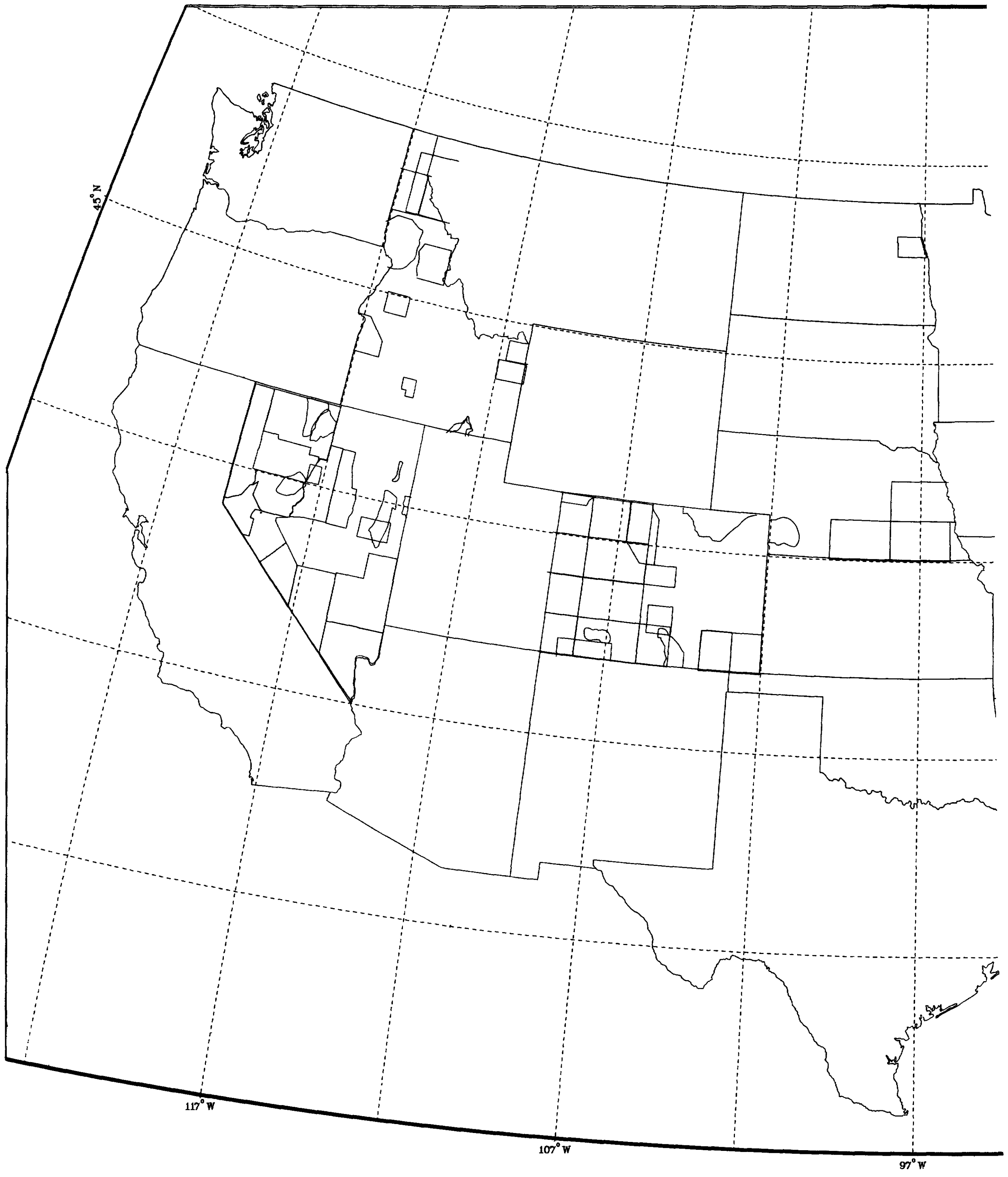
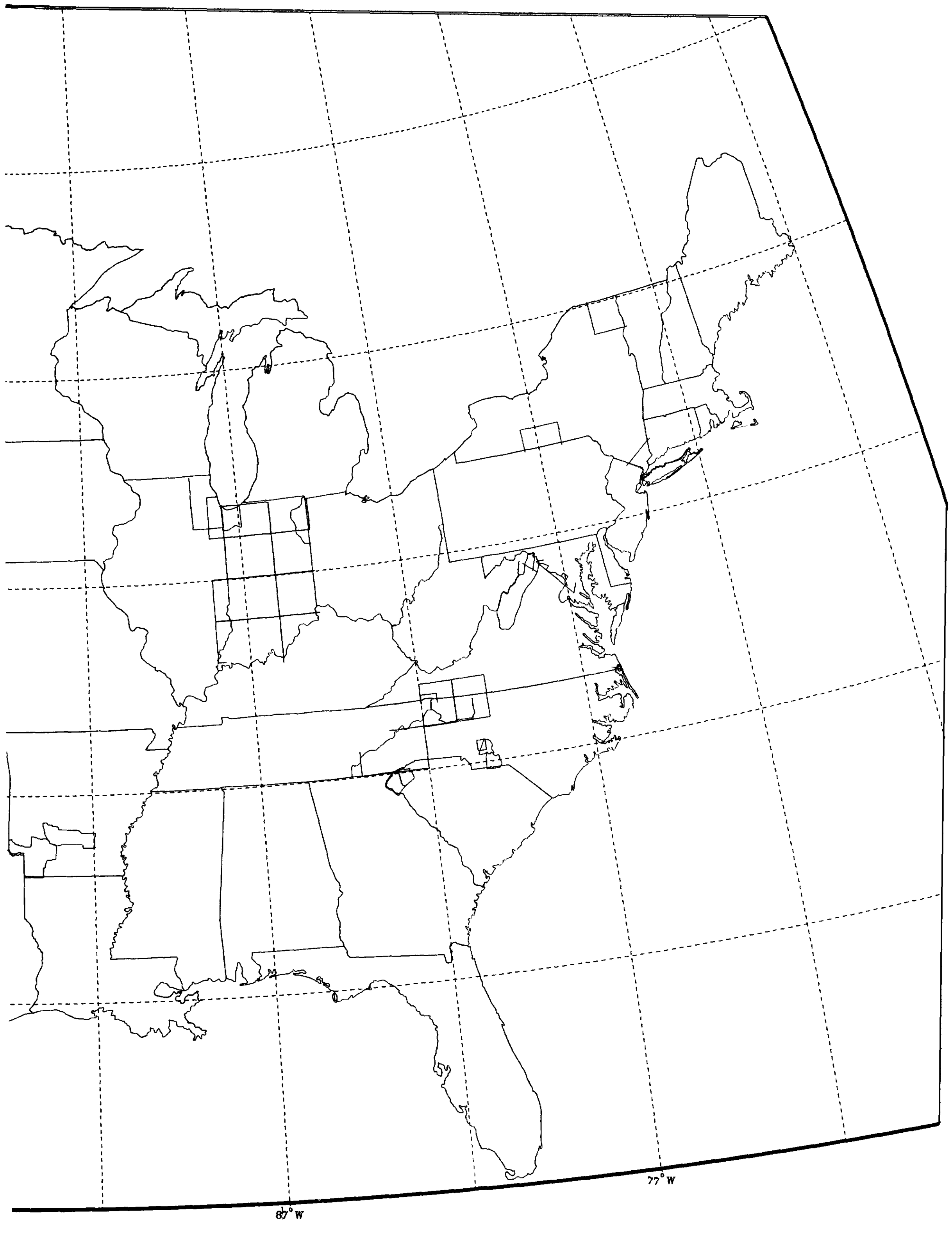


FIGURE 7. – Plots from the on-line data base showing maps published after 1960 in the United States



at a scale of 1:250,000. This plot includes both U.S. Geological Survey maps and non-Survey maps.

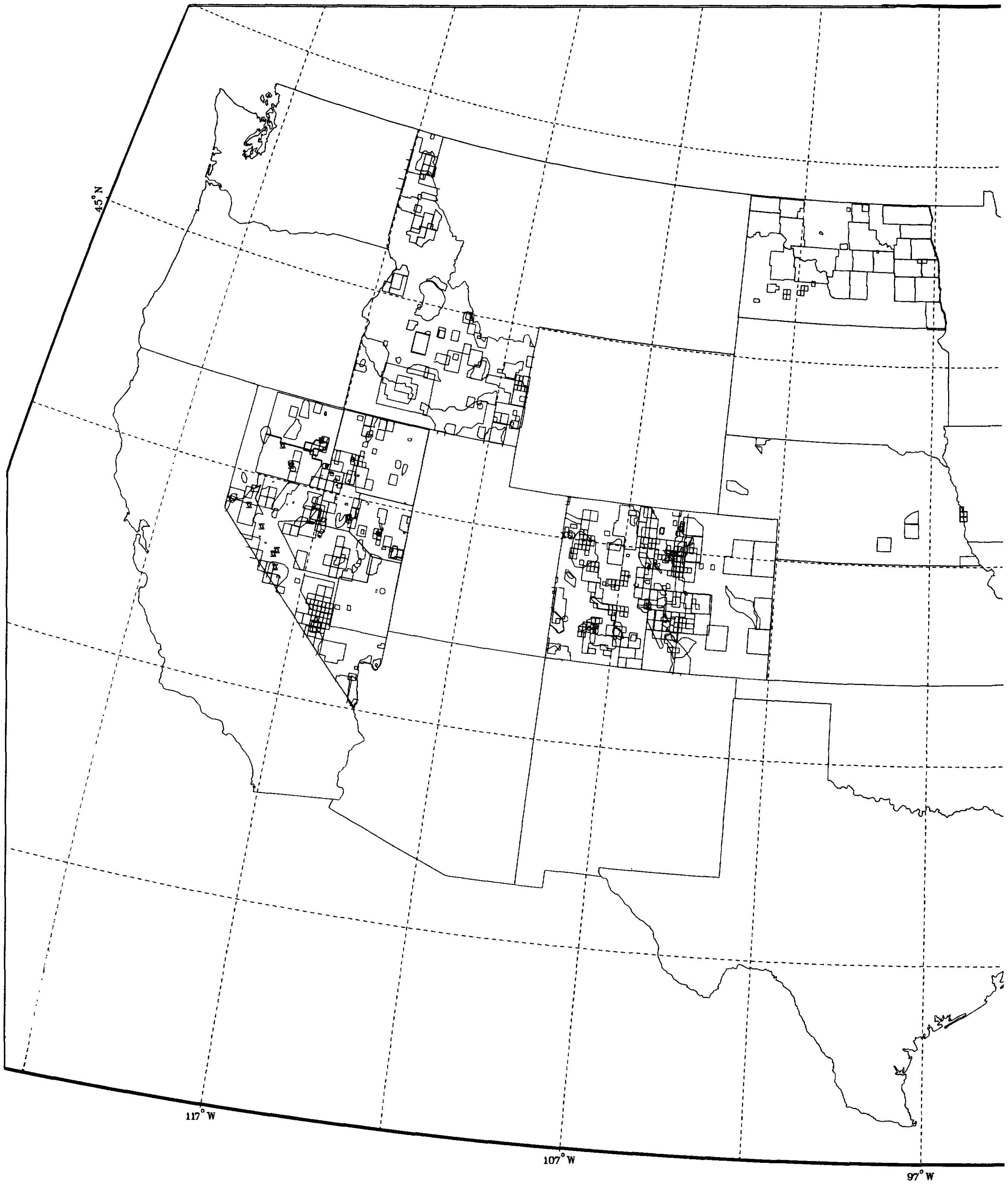
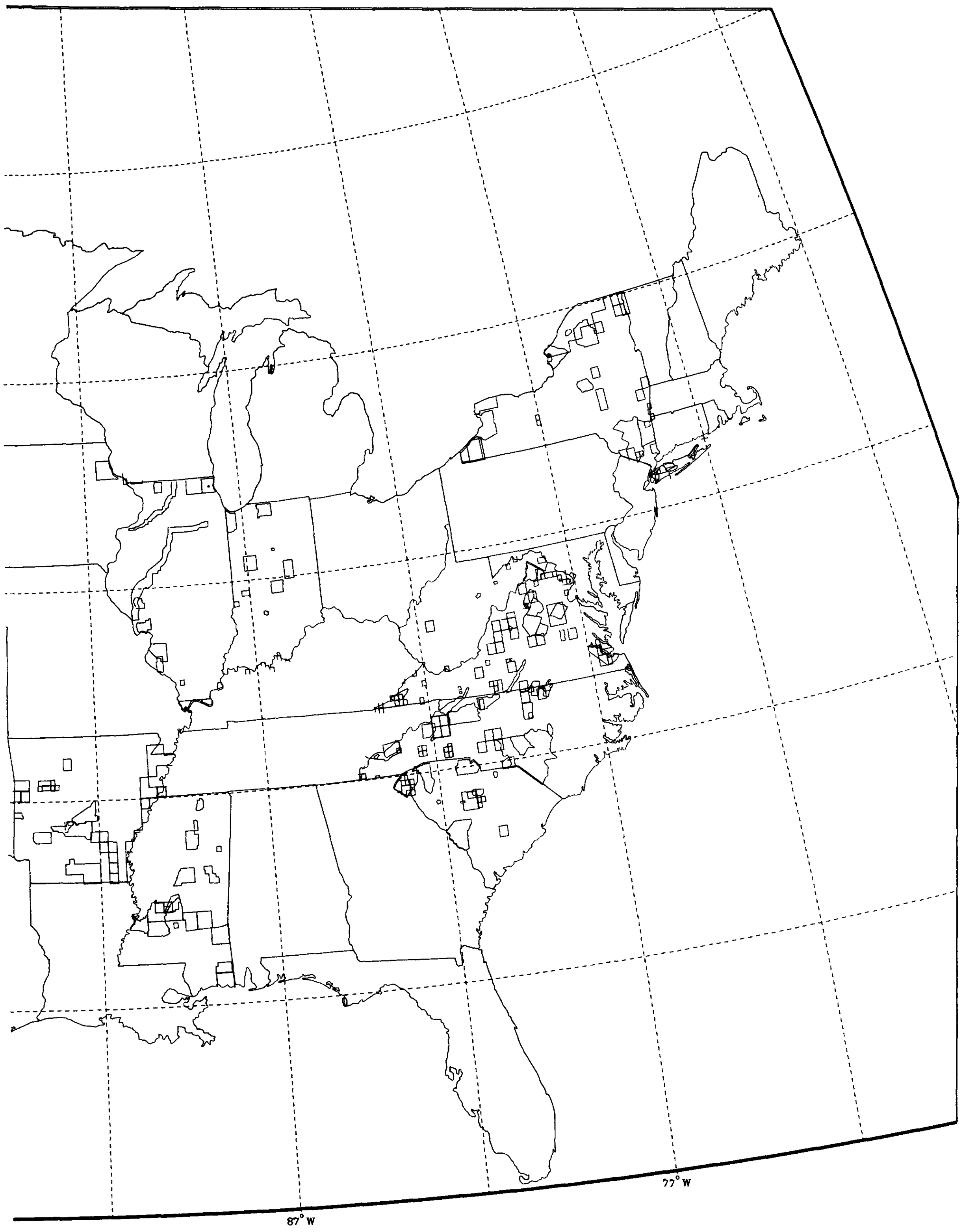


FIGURE 8. – Plots from the on-line data base showing geologic maps published from



all sources, U.S. Geological Survey and non-Survey, at scales larger than 1:250,000.

```
8 90 2Colorado
8 90 3Atwood, W.W.
8 90 81918
8 90 9Relation of landslides and glacial deposits to reservoir
8 9010sites in the San Juan Mountains, Colorado:
8 9012mineral, hinsdale, la plata
8 9017U.S. Geol. Survey
8 901848000
8 901925000
8 902093750
8 902184480
8 9023Bull. 685.
8 9024engineering
8 9038geology
8 9039Fig. 3,
8 9040fig. 4,
8 9041fig. 6,
8 9042fig. 7,
8 9044 8
8 90451
8 90462
8 90473
8 90484
8 90509001
8 90519002
8 90529003
8 90539004
8 9086Also detailed maps.
```

FIGURE 9. – Partial list of text data showing attribute values, in initial (card image) form, for reference number 90, Colorado.

```
  90     4     1     6     0     8     0     0
6232  3662  6162  3979  6689  3947  6677  3597  6146  3615  6162  3979
  90     4     2     2     0     8     0     0
6700  3683  6627  3765     0     0     0     0     0     0     0     0
  90     4     3     6     0     8     0     0
6650  4088  6570  4421  7015  4403  7005  4025  6558  4022  6570  4421
  90     4     4     6     0     8     0     0
7133  4248  7104  4142  7374  4136  7379  3804  7098  3803  7104  4142
```

FIGURE 10. – Map data in Cartesian coordinates for reference number 90, Colorado.

```
    90     4     1     6     0     8     0     0
-1873730163  0658147465 -1874116297  0659414456 -1871448603  0659327192
-1871476238  0657921102 -1874159720  0657951797 -1874116297  0659414456
    90     4     2     2     0     8     0     0
-1871368160  0658268095 -1871744737  0658591828
    90     4     3     6     0     8     0     0
-1871659176  0659890362 -1872095935  0661221254 -1869840993  0661181741
-1869857850  0659663364 -1872118161  0659618394 -1872095935  0661221254
    90     4     4     6     0     8     0     0
-1869229955  0660567668 -1869367424  0660140064 -1868000881  0660134288
-1867947965  0658801627 -1869368043  0658778571 -1869367424  0660140064
```

FIGURE 11. - Map data in radians for reference number 90, Colorado.

```
  90Colorado            Atwood, W.W.,

                                            1918Relation of landslides and glaci
al deposits to reservoir    sites in the San Juan Mountains, Colorado:

                                                mineral, hinsdale, la plata

                                                                    U.S. Geol. S
urvey                                           Bull. 685.

        engineering                                                 123.5   sq.
km. 3746053     3741045    10722053    10713033     3744017    10718016

                                                                 geology     Fig
. 3,                        848000   1 9001       Also detailed maps.

  90Colorado            Atwood, W.W.,

                                            1918Relation of landslides and glaci
al deposits to reservoir    sites in the San Juan Mountains, Colorado:

                                                mineral, hinsdale, la plata

                                                                    U.S. Geol. S
urvey                                           Bull. 685.

        engineering
    3744004     3744004    10714035    10714035     3744004    10714035

                                                                 geology     fig
. 4,                        825000   2 9002       Also detailed maps.

  90Colorado            Atwood, W.W.,

                                            1918Relation of landslides and glaci
al deposits to reservoir    sites in the San Juan Mountains, Colorado:

                                                mineral, hinsdale, la plata

                                                                    U.S. Geol. S
urvey                                           Bull. 685.

        engineering                                                 113.2   sq.
km. 3753006     3747036    10715052    10708002     3750022    10712000

                                                                 geology     fig
. 6,                        893750   3 9003       Also detailed maps.

  90Colorado            Atwood, W.W.,

                                            1918Relation of landslides and glaci
al deposits to reservoir    sites in the San Juan Mountains, Colorado:

                                                mineral, hinsdale, la plata

                                                                    U.S. Geol. S
urvey                                           Bull. 685.

        engineering                                                  60.5   sq.
km. 3749023     3744042    10706024    10701031     3747003    10704002

                                                                 geology     fig
. 7,                        884480   4 9004       Also detailed maps.
```

FIGURE 12.—Complete text-data record for reference number 90, Colorado. This shows the data in the format required by the storage and retrieval system.

Files in Permanent Storage

Description of file	Name of file	Name of file	Name of file
	File is composed of alpha-numeric data	File is composed of Cartesian coordinates	File is composed of latitude and longitude in radians
Identification file written on tape for each State. It names all the files that follow belonging to that State	bginNM		
Outlines of maps shown on the index		coorNM	cordNM
State outline		statNM	strdNM
County outlines		counNM	curdNM
Graticule		gridNM	
Neat line		bordNM	
Centers of map outlines		centrNM	
Parameters used to transform Cartesian coordinates to geographic coordinates for each State	paraNM		
Final form of the text files	redyNM		

NM is the FIPS code for each State

FIGURE 13. Table showing names of files in permanent storage.

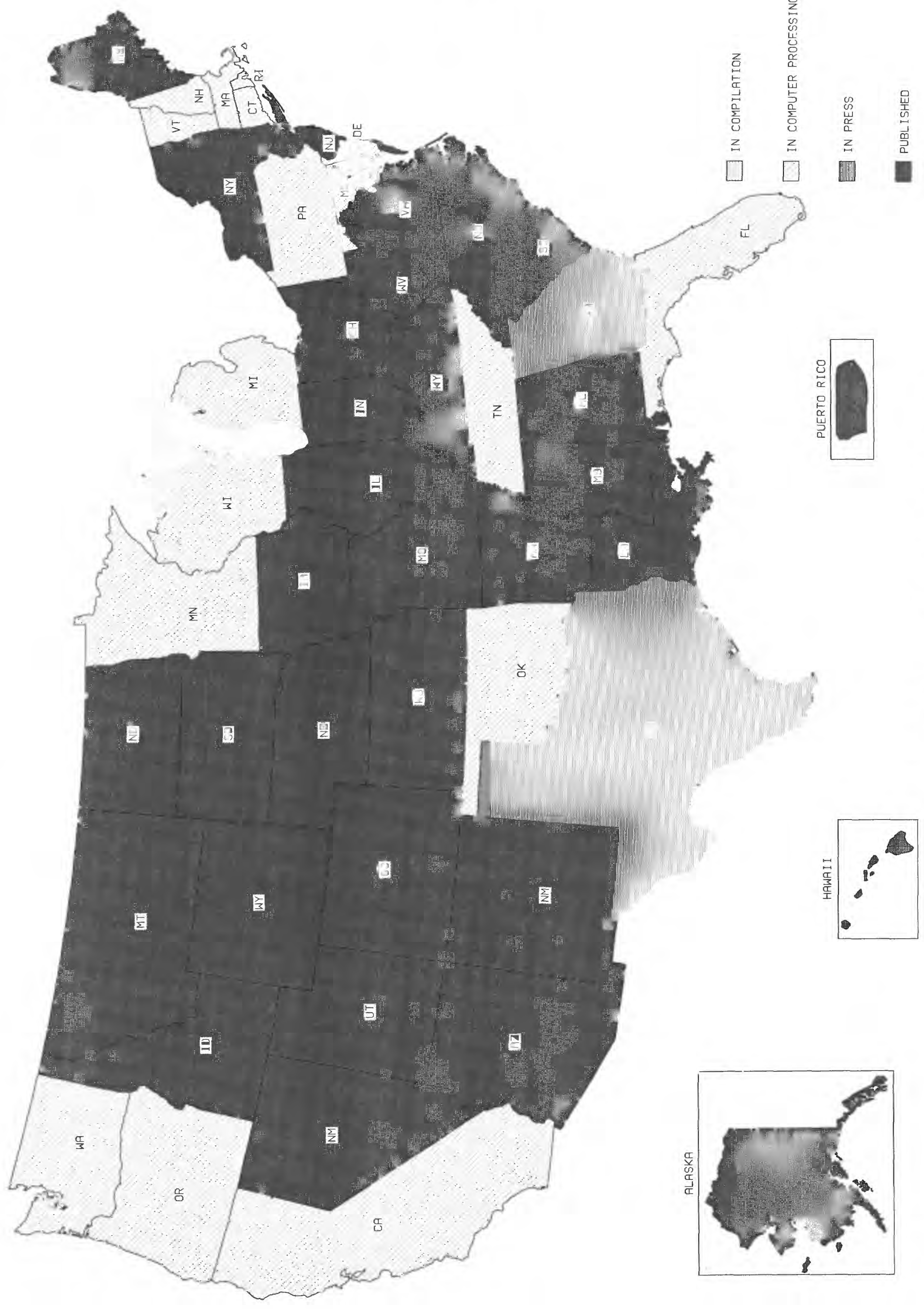
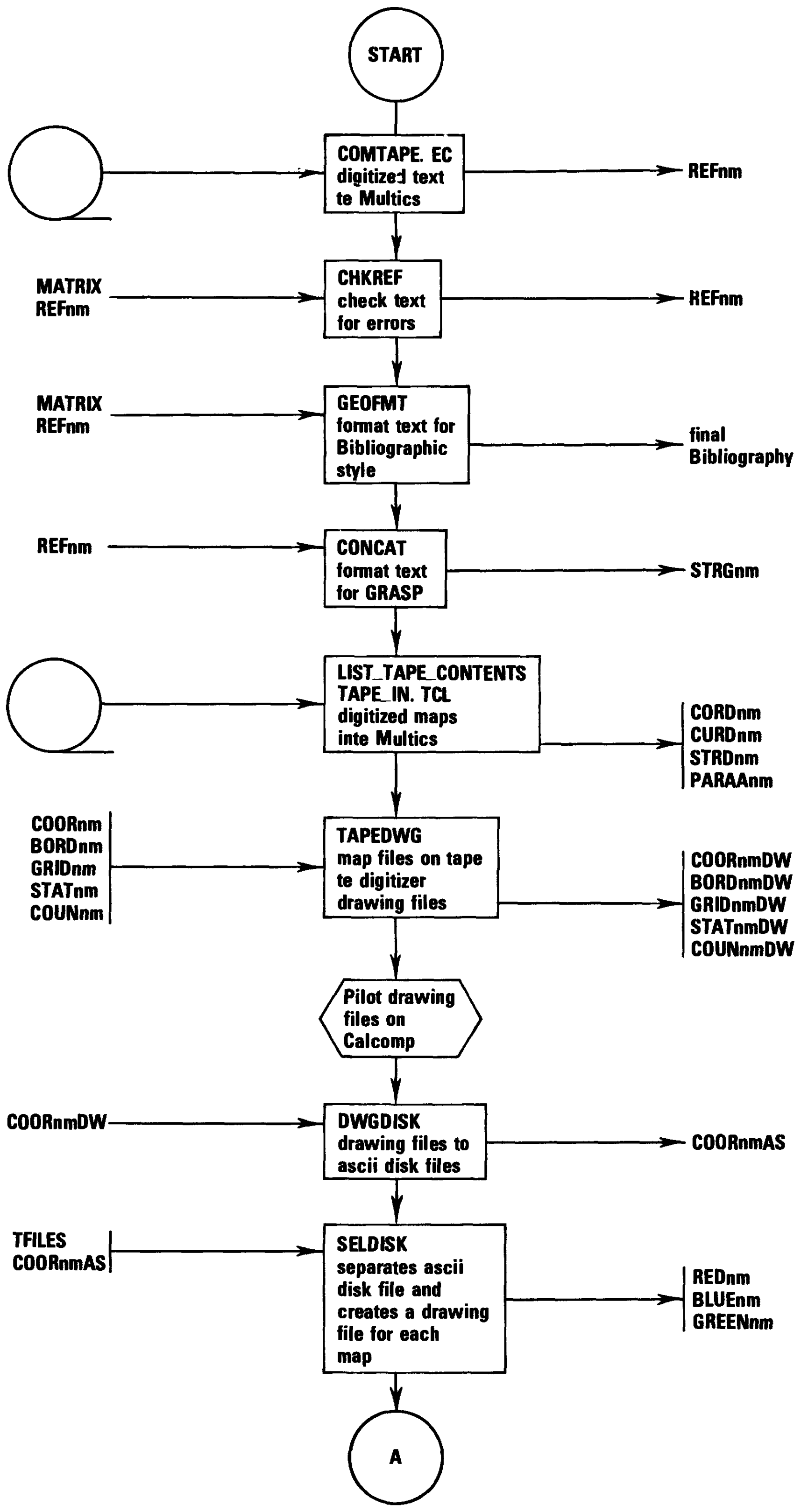


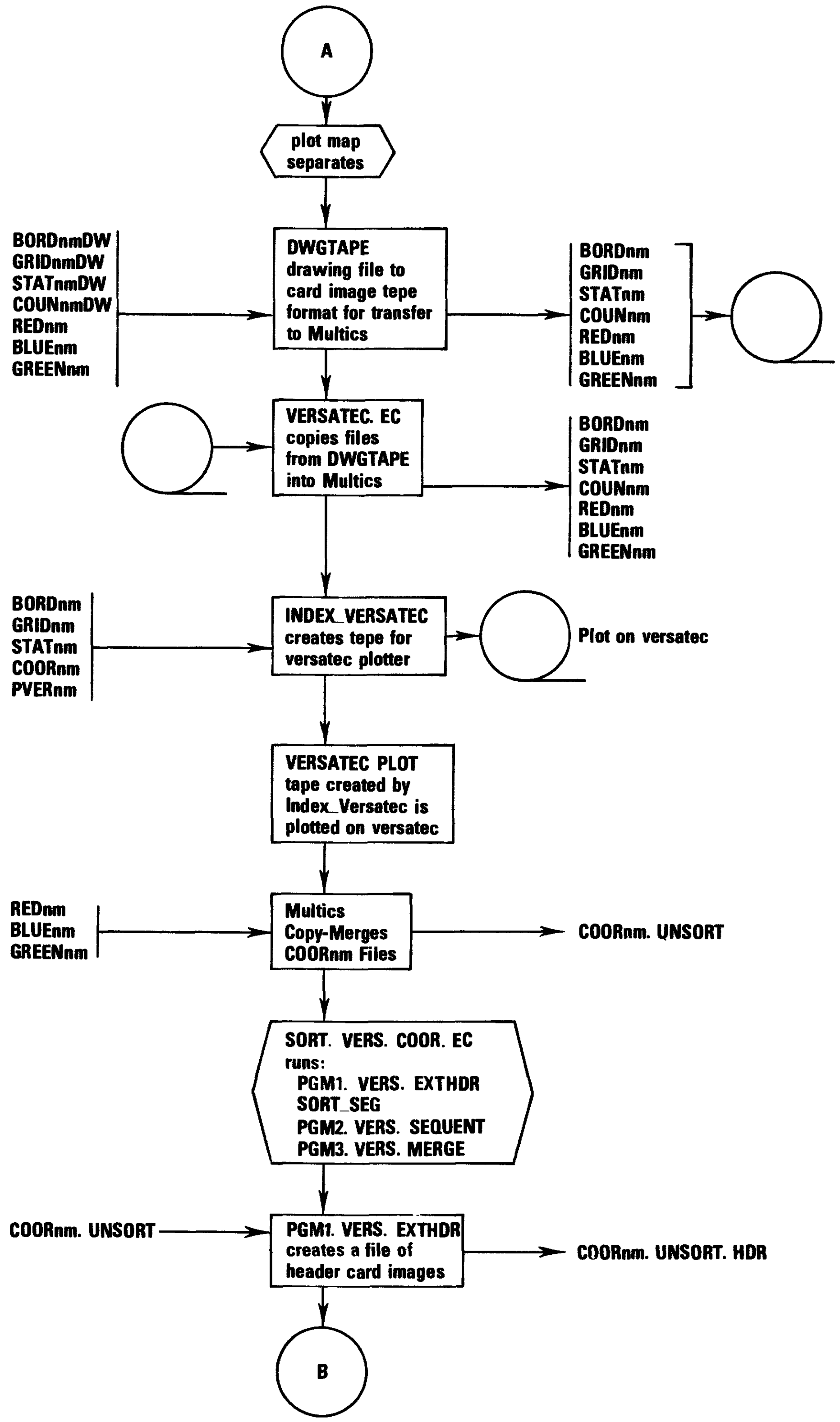
FIGURE 14.—Status map for the Geoindex, automatically generated monthly.

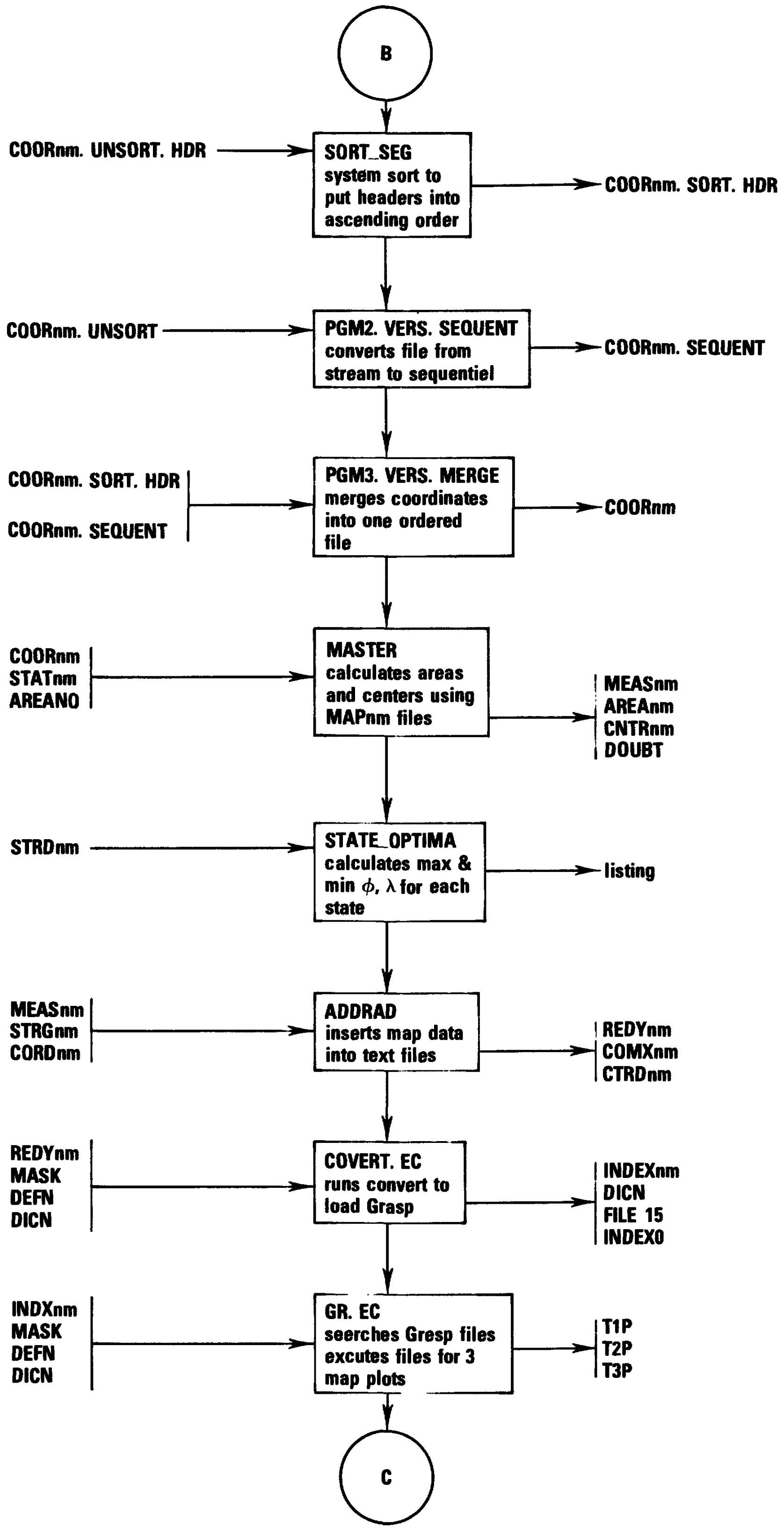
APPENDIXES: SYSTEM SPECIFICATIONS

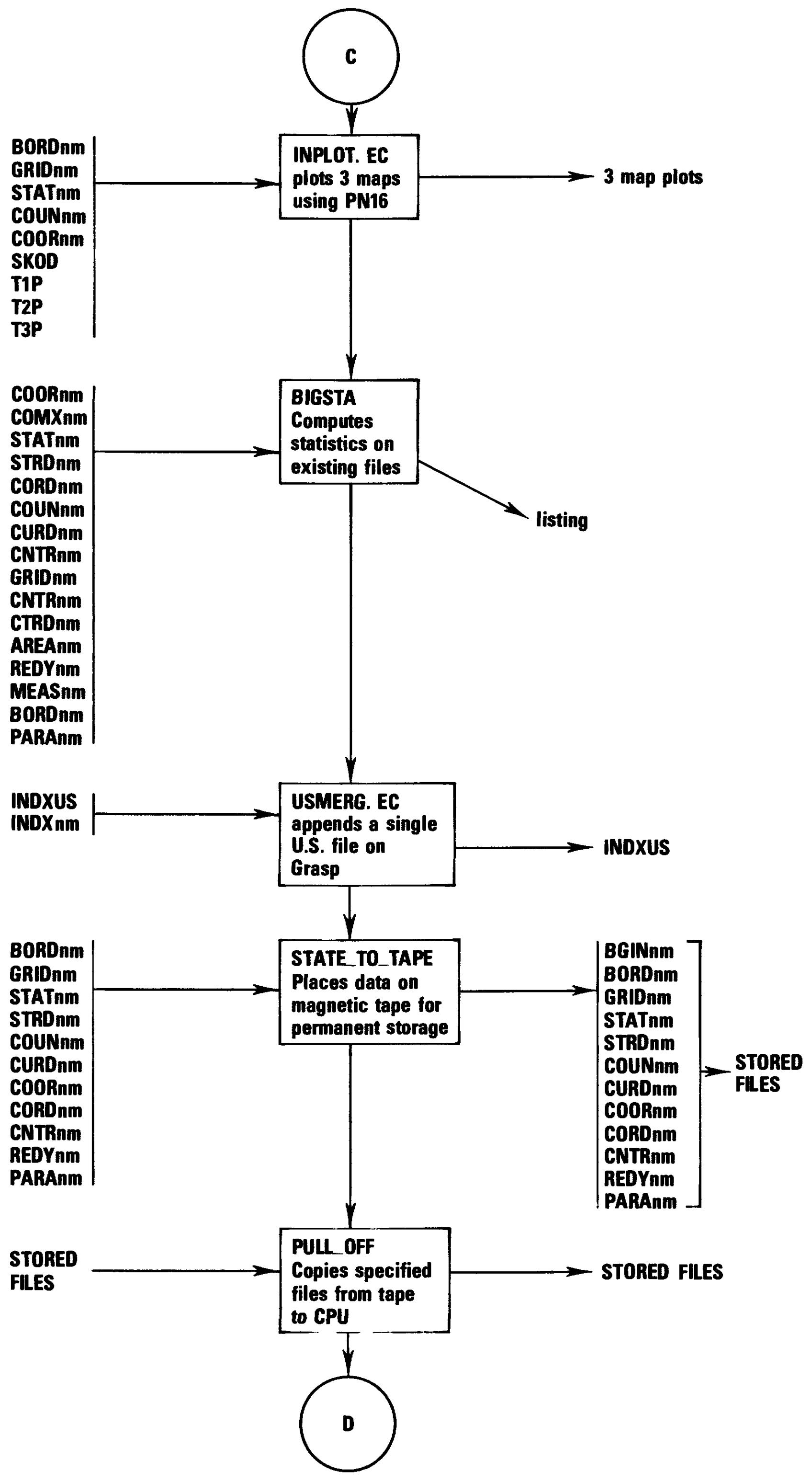
Note:
Program and subroutine names are printed in bold sans-serif type: **chkref**.
Variable names are printed in italic sans-serif type: *itype*.
Permanent-file names are printed in sans-serif type: matrix.
Ordinary variables are printed in italics: x, y.

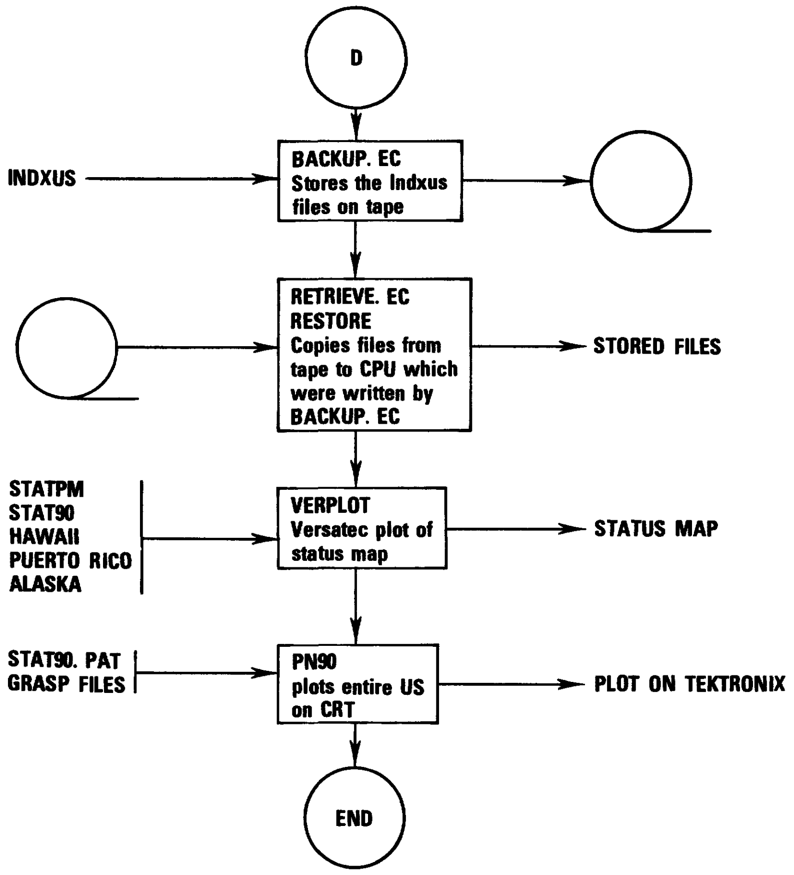
APPENDIX A. SYSTEM FLOWCHART


START
COMTAPE. EC digitized text te Multics
REFnm
MATRIX REFnm
CHKREF check text for errors
REFnm
MATRIX REFnm
GEOFMT format text for Bibliographic style
final Bibliography
REFnm
CONCAT format text for GRASP
STRGnm
LIST_TAPE_CONTENTS TAPE_IN. TCL digitized maps inte Multics
CORDnm CURDnm STRDnm PARAAnm
COORnm BORDnm GRIDnm STATnm COUNnm
TAPEDWG map files on tape te digitizer drawing files
COORnmDW BORDnmDW GRIDnmDW STATnmDW COUNnmDW
Pilot drawing files on Calcomp
COORnmDW
DWGDISK drawing files to ascii disk files
COORnmAS
TFILES COORnmAS
SELDISK separates ascii disk file and creates a drawing file for each map
REDnm BLUEnm GREENnm
A


A
plot map separates
BORDnmDW
GRIDnmDW
STATnmDW
COUNnmDW
REDnm
BLUEnm
GREENnm
DWGTAPE drawing file to card image tepe format for transfer to Multics
BORDnm
GRIDnm
STATnm
COUNnm
REDnm
BLUEnm
GREENnm
VERSATEC. EC copies files from DWGTAPE into Multics
BORDnm
GRIDnm
STATnm
COUNnm
REDnm
BLUEnm
GREENnm
BORDnm
GRIDnm
STATnm
COORnm
PVERnm
INDEX_VERSATEC creates tepe for versatec plotter
Plot on versatec
VERSATEC PLOT tape created by Index_Versatec is plotted on versatec
REDnm
BLUEnm
GREENnm
Multics Copy-Merges COORnm Files
COORnm. UNSORT
SORT. VERS. COOR. EC runs: PGM1. VERS. EXTHDR SORT_SEG PGM2. VERS. SEQUENT PGM3. VERS. MERGE
COORnm. UNSORT
PGM1. VERS. EXTHDR creates a file of header card images
COORnm. UNSORT. HDR
B


B
COORnm. UNSORT. HDR
SORT_SEG system sort to put headers into ascending order
COORnm. SORT. HDR
COORnm. UNSORT
PGM2. VERS. SEQUENT converts file from stream to sequentiel
COORnm. SEQUENT
COORnm. SORT. HDR
COORnm. SEQUENT
PGM3. VERS. MERGE merges coordinates into one ordered file
COORnm
COORnm
STATnm
AREANO
MASTER calculates areas and centers using MAPnm files
MEASnm
AREAnm
CNTRnm
DOUBT
STRDnm
STATE_OPTIMA calculates max & min ϕ, λ for each state
listing
MEASnm
STRGnm
CORDnm
ADDRAD inserts map data into text files
REDYnm
COMXnm
CTRDnm
REDYnm
MASK
DEFN
DICN
COVERT. EC runs convert to load Grasp
INDEXnm
DICN
FILE 15
INDEXO
INDXnm
MASK
DEFN
DICN
GR. EC seerches Gresp files excutes files for 3 map plots
T1P
T2P
T3P
C


C
BORDnm
GRIDnm
STATnm
COUNnm
COORnm
SKOD
T1P
T2P
T3P
INPLOT. EC plots 3 maps using PN16
3 map plots
COORnm
COMXnm
STATnm
STRDnm
CORDnm
COUNnm
CURDnm
CNTRnm
GRIDnm
CNTRnm
CTRDnm
AREAnm
REDYnm
MEASnm
BORDnm
PARAnm
BIGSTA Computes statistics on existing files
listing
INDXUS
INDXnm
USMERG. EC appends a single U.S. file on Grasp
INDXUS
BORDnm
GRIDnm
STATnm
STRDnm
COUNnm
CURDnm
COORnm
CORDnm
CNTRnm
REDYnm
PARAnm
STATE_TO_TAPE Places data on magnetic tape for permanent storage
BGINnm
BORDnm
GRIDnm
STATnm
STRDnm
COUNnm
CURDnm
COORnm
CORDnm
CNTRnm
REDYnm
PARAnm
STORED FILES
STORED FILES
PULL_OFF Copies specified files from tape to CPU
STORED FILES
D



APPENDIX B. OPERATIONAL INSTRUCTIONS

HOW TO RUN COMTAPE.EC ON MULTICS

Purpose of the program: **comtape.ec** allows the user to read the tapes containing digital text in ASCII code from outside sources into the Multics system.

Input files: Any outside tape

Output file: Segment named by the user

To run the program:

A. Label tape "for Multics use," and forward tape to be processed to production control.

B. Send a message to sys op asking him to locate tape *nnnnnn*.

Example: sm sys op Please locate tape *nnnnnn*

C. After you have been notified that the tape has been found, type:

ec comtape *nnnnnn SEGNAME*

where *nnnnnn* is the 6-position volume name, and *SEGNAME* is a name the user wishes to call the file, for example, ref21.

D. You will be informed when the tape is mounted and the tape drive on which it is mounted. You will also receive a count number of the records copied into *SEGNAME*. This file will automatically be dprinted.

Example: copy_file -ids "tape_ibm_&1 -nlb -nb 2 -fmt fb -den 800 -rec 80 -bk 800" -ods "record_stream_-target vfile_&2" dp &2

E. The tape number will be substituted for the *&1*. The *SEGNAME* given by the user will be inserted where *&2* appears.

HOW TO RUN CHKREF ON MULTICS

Purpose of the program: **chkref** reads through a reference file ref*NM* and checks for various errors that might occur.

To run the program:

A. Before running **chkref**, you must link to it. Type:

lk >udd>Geoindx>HJohnson>chkref

lk >udd>Geoindx>HJohnson>matrix

After the first link, you can run it without linking again.

B. Type: chkref

C. When asked for the file name, type in the name, such as, ref98

D. Study any error messages that **chkref** gives, make corrections to the reference file, and run **chkref** again until no error messages occur.

HOW TO RUN GEOFMT.EC ON MULTICS

Purpose of the program: **geofmt.ec** executes a series of commands and programs to read the reference file, to extract selected data, to arrange it in a predetermined order, and to create a columnarized output segment ready for printing.

Linking: Before running for the first time, you must link to the following segments:

```
lk >udd>Geoindx>PPorter>geofmt.ec
lk >udd>Geoindx>PPorter>geofmt
lk >udd>Geoindx>PPorter>geofmt.qedx
lk >udd>Geoindx>PPorter>geofmta.qedx
lk >udd>Geoindx>PPorter>geofmt2.qedx
lk >udd>Geoindx>PPorter>geofmt3.qedx
lk >udd>Geoindx>PPorter>embed_tabs
lk >udd>Geoindx>PPorter>to-nbiPP
```

Special instructions:

A. The program, **geofmt.ec,** can be run on any terminal. The only time that you must be on the NBI is to run the **to-nbiPP** program where you must use the proportional space printer.

B. To get a rough draft, type anything other than *nbipp* as the third argument of the exec_com.
   Example: ec geofmt 84 10 nbino
   Upon termination, type:
   dp -dl -nep geofmt.columns
   If a file has more than 550 references, do only half the file, and after you get the dprint, run the program again using the last half of the file starting with a four-column page through the end of the file.

C. When three columns are desired on the first page with the map, follow the instructions in paragraph B with the exception of 8 and 9. For number of columns, enter 3 and return. In answer to the column width, enter 48 and return.

D. When using the seven-column option, you can print a single three-column page giving the reference numbers for that page, but if more than one page is to be printed, you must start with a four-column page.

To run the program:

A. Type: ec geofmt *pagelength lines nbipp*
   Example: ec geofmt 84 10 nbipp
   Note: *Pagelength* specifies the number of lines on the page. It can be any number, but presently only 84 or 140 are used. *Lines* is an argument specifying the number of lines that should be available at the bottom of the page in order to print a complete reference. There must be a third argument. If you want proportional space printing, type: *nbipp*. Otherwise, type some letters or some word for the third argument.

B. Then you must respond to the following questions or statements:
   1. ENTER FILE NAME: Enter the names (*refNM*) and cr (carriage return).
   2. TYPE IN STATE NUMBER: Enter 2-digit FIPS code and cr.
   3. WHAT IS YOUR STARTING REFERENCE NUMBER? (use 3 digits): Enter 3 digits and cr. Note: We strongly recommend that you do no more than 550 references at one time.
   4. WHAT IS YOUR ENDING REFERENCE NUMBER? (use 3 digits): Enter 3 digits and hit cr.
   5. fortran_io_: CLOSE FILES? Type yes and cr.
   6. DO YOU NEED TO EDIT? Type yes or type no, and cr. If you typed no, go to step 7. If you typed yes, the following will appear on the screen:
      EDIT.
      Enter q to exit editor.
      You are now in the edit mode. Line length is set to 80 to make the entire line visible on the screen. One line of ***geofmt.data*** may print out as three lines on the NBI screen. Make the necessary changes and be sure to write the segment before you exit the editor. Under no circumstances should you break or interrupt while in **qx**. After all changes, type:
      w
      q
   7. DO YOU WANT 7 COLUMNS? If you want a combination of 4,3,4,3, and so forth, type: yes. If you want four columns on every page, type: no.
   8. EMBED_TABS ENTER NUMBER OF COLUMNS: Enter 4 and cr.
   9. EMBED_TABS ENTER COLUMN WIDTH: Enter 42 and cr.
   10. The ready message will appear on the screen, and the job is completed.

C. Before recording ***geofmt.columns*** on the diskette, enter **qedx** and check the beginning of each page to make sure that a new reference begins in each column. Also check the last page (not the final page) to make sure that you specified enough references to fill the page.

If you are using the seven-column option and doing only part of the reference file, you should end with a three-column page (unless you are doing a single page). Delete the lines of the four-column page, write *geofmt.columns* and quit the editor.

D. The file, *geofmt.columns*, is now ready to be recorded on the diskette for printing. Before running **to-nbiPP**, issue the following commands:
    stty -modes lfecho
  Type ct, and then cr. Four options will appear on the screen. Type 4, representing computer 4, but not cr. Now type:
    to-nbiPP cr
  1. The following message will appear:
      Multics file name to be sent (or q to quit):
    Enter *geofmt.columns* followed by cr.
  2. Multics will respond with ?.
  3. Hold down SHIFT key and press the XMIT keys. The NBI now receives the document line by line. When the last line has been received, a single Greek character will remain on the screen.
  4. Hit BREAK key. CONVERSATIONAL will appear on the screen.
  5. Type: q cr
    Multics will respond with STOP and ready message.
  6. Hit HOME key. READY will appear on the screen and you are now back in NBI word processing.
  7. Name the document by typing:
      co le,1,*document name*
    followed by cr.
  8. To print the document, insert and aline paper and type the following command:
      pr li,s:*document name*

HOW TO RUN TO-NBID ON MULTICS

Purpose of the program: **to-nbiD** allows the user to record segments from Multics on the NBI diskette (communications disk—four options only) while using the NBI System II as a terminal.

To run the program:

A. Turn on machine and insert disk. After READY appears on the screen, type ct and cr. Type 3, representing computer 3, but not cr. If you accidently hit cr, COMMAND ERROR will appear on the screen. If this happens, again type ct, and when the four options appear on the screen, type: 3.

B. Type cm l,c and cr. Note: The l is an alphabetic character and not the number one. CONVERSATIONAL will appear on the screen. Insert telephone in modem and dial Multics number. When carrier light comes on, hit cr. Wait for the two-line Multics greeting. If you lose the carrier light or the greeting does not appear, hang up and redial. You are still in CONVERSATIONAL mode.

C. Login to Multics as you usually do. If you cannot login, hit cr and go back to step B.

D. You must have the following link in your working directory:
    lk >udd>Geoindx>PPorter>to-nbiD
  Note: To print the greater-than sign, press the key and CTRL key at the same time.

E. To execute the program, type: to-nbiD cr

F. The following message will appear:
    Multics file name to be sent (or q to quit):

G. Type in name of file and then cr.

H. Multics will respond with ?.

I. Hold down SHIFT key and press the XMIT key. The NBI now receives the document line by line. When the last line has been received, a single Greek character will remain on the screen.

J. Hit BREAK key. CONVERSATIONAL will appear on the screen.

K. Type: q cr
    Multics will respond: STOP
  fortran_io_: CLOSE FILES?
  Type: yes
  Multics will then respond with a ready message.

L. You may now logout of Multics as you usually do, or you may wish to edit the document to insure that it was received correctly. You are still in CONVERSATIONAL mode.

M. Hit HOME key. READY will appear on the screen and you are now back in NBI word processing.

N. Name the document by typing:
    co le,1,*document name* cr
  Note: This command expanded means copy letter, drive 1, and name you wish to call document.

O. To print the document, insert and aline paper and type the following print lines command:
    pr li,s:*document name* cr
  NBI will buzz, giving you a chance to make sure that the paper is inserted correctly. Hit cr. Document will start printing.

P. If you are no longer on Multics, go to step Q. If you did not logout in step L, you will now have to get back in CONVERSATIONAL mode. Type:
    cm l,c (as in step B)
  CONVERSATIONAL will appear on the screen. Logout the way you usually do. After the logout

message appears on the screen, press the HOME key. READY will appear on the screen.

Q. Type: off cr
   Remove disk during countdown, and then turn off machine.

HOW TO RUN CONCAT ON MULTICS

Purpose of the program: **concat** takes a reference file and builds from it a file suitable for input into Bowen's (Bowen and Botbol, 1975) program **convert**. Be sure *refNM* is the file you want.
   Input files: *refNM*, matrix
   Output file: *strgNM*

To run the program:

A. Before running **concat** the first time, you must link it to your working directory. Type:
   lk >udd>Geoindx>HJohnson>concat
B. Type: concat
C. When asked for the State code, type in the FIPS code for this State.
   Example: concat

```
  ENTER THE 2-DIGIT CODE FOR THE
  STATE BEING PROCESSED
Type: 15
  YOU GOT TO MAIN
  YOU WROTE THE 25th VECTOR TO THE
  STRG FILE
  STOP
  FORTRAN IO : CLOSE FILES?
Type: yes
```

HOW TO RUN LIST_TAPE_CONTENTS ON MULTICS

Purpose of the program: **list_tape_contents** abstracts files from outside tapes containing digital map data in ASCII code.

To run the program: After the normal procedure of taking the tape to production control, sending a message to the operator asking her to locate the tape, and being informed that tape is there, you then list the contents of the tape. This can be done by **list_tape_contents**, which prints information about files recorded on 9-track magnetic tape. This command will list only ANSI (American National Standards Institute) standard labeled and IBM OS/370 standard labeled tapes.

Example: list_tape_contents *nnnnnn* -long -iom tape_ibm_ where *nnnnnn* is the volume number, *-long* is an argument that will cause an extensive amount of information to be printed about the files, and **-iom tape_ibm_** invokes the I/O module to attach and read the specified tape volume. The **tape_ibm_** subroutine is specified in order to list OS standard labeled tapes. See listing 1.

Now that you have a list of the tape contents, determine which files you want to transfer to the disk. Transfer can be accomplished by the **tape_in** command, which uses a control file written by the user in the tape control language. See listing 2 for an example. The volume statement is the volume number of the tape. For most outside tapes, *Tape, Storage, Density, Format, Record,* and *Block* will be the same as those shown in listing 2. There will be a file number and path statement for each file to be transferred. The argument of the file statement will be an asterisk. The number statement will specify which file number it is on the tape. The argument for the path statement will be the name that you wish the file to be called after it has been transferred to disk.

The control file must have a suffix of .tcl. After you have created the control segment with a text editor, you can accomplish semantic checking with the following command:
   tape_in rdtape.tcl -ck

The -ck argument does not cause a tape to be mounted. If any errors occur, check the **tape_in** command in the Honeywell Information Systems' "MPM Peripheral Input/Output Manual." After making corrections, type:
   tape_in rdtape.tcl

To simplify the process, procedures are listed below in steps:

1. sm sys op Please locate tape *nnnnnn.*
2. list_tape_contents *nnnnnn* -long -iom tape_ibm_
3. Create the .tcl segment. Use uppercase and lowercase as shown in listing 2.
4. tape_in rdtape.tcl -ck
5. new_proc
6. tape_in rdtape.tcl

HOW TO RUN TAPEDWG ON DATA GENERAL

Purpose of the program: **tapedwg** reads a hexadecimal ASCII (American Standard Code for Information Interchange) tape written by a 32 bit/word computer and places it in the format of a System 101 drawing file. The data consists of *x, y* coordinates in the format already specified for map data.

To run the program:

A. Bring the empty drawing file onto the table. Be sure to display it as a check that it is empty.
B. Run the overlay program **tapedwg**.

C. The program will print:

    PAUSE MOUNT TAPE ON UNIT 0

  and will pause until you enter cr. After the tape is mounted, enter cr.

D. The program will print:

    UNIT 1 OR 0??

  and will wait until you enter 1 or 0 followed by cr.

E. The program will print:

    CHARACTER HEIGHT =

  and wait. The usual response is 0.14 and cr.

F. The program will print:

    SYMBOL # =

  and wait. The usual response is 1 and cr.

G. The program will print:

    # OF PENS = 1, 2, OR 3

  and wait. The usual response is 3 and cr.

H. The program will print:

    TEXT WANTED?? 1=YES, 0=NO.

  and wait. The usual response is 1 and cr.

I. The program will print:

    SKIP FILES??

  and wait. The usual response is no and cr.

J. If the drawing file is filled, the program will print:

    DRAWING FILE FULL!!
    DO NOT REWIND TAPE!!
    DO NOT REWIND TAPE!!
    SAVE DRAWING FILE, GET NEW DRAWING FILE AND RECALL TAPEDWG OVERLAY

  The program will then wait. Do exactly as the program instructs.

K. If the drawing file is not full, but an EOF is encountered, the program will print:

    END OF FILE REACHED?
    REWIND TAPE?

  and wait. The response is y for yes, and cr if there are no more files to be read. The response is n for no if there are more files to be read.

L. The program will print:

    PROGRAM FINISHED

  This means a successful completion of the program.

LISTING 1.–*Example of* **list_tape_contents**

```
Mounting volume AAR793 with no write ring.
AAR793 mounted on tape_01.

File listing of OS Labeled Volume AAR793 Recorded at 1600 bpi.

ID: BID.DAW.OHIOBOR          Format:   FB       Blksize:     6400      Lrecl:           80
Number:      1               Mode:     ****     Created: 10/27/78      Expires:    unknown
                             Section:     1     Version:        0      Generation:       0

ID: BID.DAW.OHIOLL
Number:      2

ID: BID.DAW.OHIOSB
Number:      3

ID: BID.DAW.OHIOCB
Number:      4

ID: BID.DAW.OHIO
Number:      5

ID: BID.DAW.OHIOPARM
Number:      6

ID: BID.DAW.OHIOSBRD
Number:      7

ID: BID.DAW.OHIOCBRD
Number:      8

ID: BID.DAW.OHIORAD
Number:      9

Displayed characteristics for the last 9 files are identical.
Finished listing volume AAR793 as specified.
r 1256 2.427 52.966 633
```

M. System error messages are printed whenever a problem occurs. Consult the manuals and take appropriate measures.

HOW TO RUN DWGDISK ON DATA GENERAL

Purpose of the program: **dwgdisk** reads System 101 drawing files and creates an ASCII disk file containing the coordinate outline data in Geoindex standard format. The sequence is eight integer values comprising the header card information, followed by ***isfno*** (number of coordinate pairs) pairs of real numbers. The second outline immediately follows the first and so on.

To run the program:

A. Bring desired drawing file onto table. Make sure that the drawing file does not contain extraneous information or the program will not execute correctly. For example, if you have deleted something from a drawing file, the deletion will change only certain parts of that particular record to a zero. The record still exists and will cause problems in the program. To delete an unwanted record, save the file and then bring it back.

B. Run the overlay program **dwgdisk**.

C. The program will print:

```
PAUSE FOR OPERATORS
```

and the program will wait for you to enter cr.

D. The program will print:

```
NAME OF DISK OUTPUT FILE = ??
```

and will wait until you type a name in, followed by cr. Depending upon your answer, the program will type:

```
OLD-FILE OK??
NEW-FILE OK??
```

or print an error message, or end the program (if

LISTING 2.—*Example of a tcl*

```
rdtape.tcl      11/21/78  1455.7  est Tue

Volume: AAR793
Tape: ibmsl;
Storage: unstructured;
Density: 1600;
Format: fb;
Record: 80;
Block: 6400;
File: *;
path: ohiobor;
number: 1;
File: *;
path: ohioll;
number: 2;
File: *;
path: ohiosb;
number: 3;
File: *;
path: ohiocb;
number: 4;
File: *;
path: ohio;
number: 5;
File: *;
path: ohioparm;
number: 6;
File: *;
path: ohiosbrd;
number: 7;
File: *;
path: ohiocbrd;
number: 8;
File: *;
path: ohiorad;
number: 9;
End;

r 1455 0.077 0.828 27
```

cr is the first character or the escape sequence, control d, is entered). A negative answer to the first two will cause it to ask the question again. After printing the error message, it will ask the question again.

E. The program will print:

   DO YOU WISH TO WRITE AN EOF FLAG ON THIS FILE??

   and wait for a y (yes) or n (no). The last part of the outline file must have an EOF flag.

F. The program will print:

   TYPE IN 2 DIGIT STATE NUMBER

   Use the FIPS code for this State. This information goes on the header card.

G. The program will then ask:

   IS THIS THE GRID BEING PUNCHED

   Answer y (yes) or n (no). No other answer will be accepted. This is needed to fill in the header card. If the answer is yes, the next question will be skipped.

H. The program will ask:

   IS THIS THE COUNTIES BEING PUNCHED

   Answer y(yes) or n(no).

I. The program will start to process the data. It assumes that there is text in the drawing file. If not, it will print the message:

   NO TEXT IN FILE!

   and exit from the program.

J. When execution is complete, the program prints the message:

   DONE

   rings a bell, and returns control to the table.

HOW TO RUN SELDISK ON DATA GENERAL

Purpose of the program: **seldisk** reads an outline disk file in Geoindex standard format and creates a System 101 drawing file. The format used is a header card with (8I5) format followed by the outline with x, y coordinate pairs in (12F6.3) format on each card. The drawing file will contain only those outlines identified by cards that have a feature number and a subfeature number. Note: The program does not replace the file on the table; instead it appends this data at the end of the existing file.

The ASCII disk file will usually be created by using the program dwgdisk.

The drawing file will consist of various outlines all having the following characteristics: Each outline resides in the subfile with a number equal to the feature number or 1,000 less than the feature number. There are from one to four lines of text, followed by one pen up and then a series of pen downs. The text consists of the feature number, the subfeature number (if ***ifno*** greater than 1), the ***span*** (if different from 0) and the second subfeature number (if different from 0).

The outlines will alternate through the three pens so that the colors will change for better visibility.

To run the program:

A. Perform the steps necessary to make files available from the digitizing table. If you wish to place the incoming data in a drawing file by itself, CLEAR the drawing file.

B. ACTIVATE AND DISPLAY the drawing file. This tells the program where to write the information.

C. Load the cards in the hopper. First will come the ***T-file***. This consists of cards with the feature number and subfeature number of those outlines wanted in the drawing file. These are in (I8,I2) format. Follow these cards with a card that has a −1 as a feature number (columns 7 and 8). This will be used as a flag for the end of the ***T-file***.

D. Run the overlay program **seldisk**.

E. The program will print:

   !SELDISK OVERLAY

   !PAUSE TURN ON CARD READER

   and wait for you to enter cr. This is a reminder to make sure it is turned on. If it is not correctly turned on, the program will print:

   !FOPFL ERROR!

   and exit.

F. The program will print:

   !CHARACTER HEIGHT =

   and wait for you to type an answer. This will be the height in inches of all text read in (usually 0.14).

G. The program will print:

   !SYMBOL # =

   and wait for you to type an answer. The answer is the number of the symbol that is drawn wherever there is a single point for an outline.

H. The program will print:

   NAME OF COORDINATE OUTLINE FILE = ??

   and wait for you to type in the name of a disk file. If this is a file that does not exist, the program will print:

   NEW FILE TRY AGAIN!

   and return to ask the question again. Any other error will cause an error message to be printed, and then the question will be asked again. A control d or a cr on the first character will cause the program to terminate.

I. The program will execute and when finished it returns control to the table with the new drawing file. Any error in execution will automatically put you into the command mode. Some kind of error message will be given, and the error should be corrected; then the whole procedure must be started again. As the program executes, it will print the feature number of all outlines selected for the drawing file.

HOW TO RUN DWGTAPE ON DATA GENERAL

Purpose of the program: **dwgtape** reads a System 101 drawing file and writes to tape the header card and data cards for all outlines. Each separate outline has a header card with (8I5) format followed by data cards in (12F6.3) format. The text position is in the first position on the first data card. The program makes no attempt to sort the outlines; it just starts at the beginning of the drawing file and processes the file in sequential order.

Several options are available. You can punch cards for the whole file or you can pick one subfile number (feature number). It will process all outlines that have the subfile number you have chosen. The second option gives you the choice of punching all the data cards of each outline or of punching only the header card and first data card for each outline. This is useful when many text position changes have been made that would affect only the first data card. The third option lets you skip files on the tape so that your file can be placed on a tape with other files.

To run the program:

A. Bring desired drawing file onto table. Make sure that the drawing file does not contain extraneous information or the program will not execute correctly. For example, if you have deleted something from a drawing file, the deletion will change only certain parts of that particular record to a zero. The record still exists and will cause problems in the program. To delete an unwanted record, save the file and then bring it back.

B. Run the overlay program **dwgtape**.

C. The program will print:

```
PAUSE MOUNT TAPE ON UNIT 0
```

and will wait for you to enter cr. This gives you a chance to mount the tape if you have not already done so. The tape must have a write ring.

D. The program will print:

```
!SUBFILE# = , TYPE 9999 FOR ALL!
```

This gives you the option of punching only one feature number (subfile) or everything in the file.

E. The program will print:

```
!DO YOU WISH THE FIRST DATA CARD
ONLY???
```

Answer y (yes) or answer n (no). No other answer will be accepted.

F. The program will print:

```
TYPE IN 2 DIGIT STATE NUMBER.
```

Use the FIPS code for this State. This information goes on the header card.

G. The program will then ask:

```
IS THIS THE GRID BEING PUNCHED??
```

Answer y (yes) or answer n (no). No other answer will be accepted. This is needed to fill in the header card. If the answer is yes, the next question will be skipped.

H. The program will ask:

```
IS THIS THE COUNTIES BEING
PUNCHED??
```

Answer y or answer n.

I. The program will ask:

```
SKIP FILES??
```

Answer y or answer n.

J. The program will ask:

```
HOW MANY FILES??
```

Type in the number of files you wish to be skipped.

K. The program will start to process the data. It assumes that there is text in the drawing file. If not, it will print the message:

```
!NO TEXT IN FILE!
```

and exit.

L. When execution is complete, the program prints the message:

```
!DONE!
```

and returns control to the table.

HOW TO RUN VERSATEC.EC ON MULTICS

Purpose of the program: **versatec.ec** creates the files that are to be used for input to the Multics Versatec programs. The input files are those files that were created on the Data General minicomputer by the program **dwgtape**.

To run the program:

A. Before you run this program for the first time, the program must be linked. This is done by typing:

```
lk >udd>Geoindx>PPorter>versatec.ec
```

B. After this is done, the next step is to send the systems operator a message to locate the tape:

sm sys op Please locate tape *tape number*

After the operator has located the tape, then type in:

ec versatec *tape number numbers of files file1 file2 . . . filen. file1, file2 . . . filen* must not contain more than 32 characters including blanks.

C. If there is more than one coordinate file, then these coordinate files must be merged into one large coordinate file. This can be done in the editor qedx.

D. The output file of the merge must be of the form: **coor*NM*.unsort**, where *NM* is the State code.

HOW TO RUN INDEX_VERSATEC ON MULTICS

Purpose of the program: **index_versatec** plots the various *x*, *y* data files that constitute an index map using the 18-in. Versatec plotter.

To run the program:

A. First link to the following:
   lk >udd>Geoindx>PPorter>index_versatec
   lk >udd>Geoindx>PPorter>init_vals

B. For each State *NM* you wish to plot, place the following files in your working directory (or link). The program will access the data via these file numbers:

bordNM	10
gridNM	11
statNM	12
counNM	13
coorNM	14
pverNM	15

   The format of ***pverNM*** is explained in listing 3. Note: Before executing any program on Multics, you should do a **new_proc**.

C. Execute the plot program by typing: **index_versatec**
   The program will respond:
   TYPE IN TWO DIGIT STATE NUMBER
   Type in the State number. The program will execute and will periodically print out information.

D. The versaplot software will do as many as 100 different plots and automatically store the output in segments named ***vplt00***, ***vplt01***, ***vplt02***, . . ., ***vplt98***, ***vplt99***. These will write over any existing segments with that name. Therefore, the first thing you should do after completing the program is to rename these segments.

E. To execute for another State:
   1. Make sure you have renamed the ***vpltNM*** segments.
   2. Do a **new_proc**.
   3. Go to step B.

F. To put plot segments onto tape: There is an exec_com that will place as many segments as you wish onto a tape. The best way to do this is to copy the exec_com into your segments along with all its six names. Type:
   copy >udd>Geoindx>PPorter>gpt.ec- all

G. To use:
   1. Take a tape to production control. A label "For Multics Use" should be on the tape.
   2. Send a message to system operator to get the tape:
      sm sys op Please locate tape *nnnnnn*

LISTING 3.--*Parameter cards for a Versatec plot*

I. Read input parameter from cards for each data file.

Files	Columns
A. Neat line	1-22 card 1
B. Grid	23-44 card 1
C. State	45-66 card 1
D. Counties	1-22 card 2
E. File of selected outlines	23-44 card 2
F. Outline coordinate	45-66 card 2
G. 0 = end of all plotting	67 card 1
1 = more cards follow describing another plot	

II. Procedure: plot each of the first four files.

III. Sort the file of selected outlines into ascending order (eliminate duplicates).
   Plot these selected outlines from the master file, or plot all of the master file.
   Each 22-column section contains the following information:

Column 1	0 = no plot
	1 = plot
Column 2	0 = plot character and lines
	1 = plot lines (points) only
	2 = plot characters only

   There will be a separate subroutine for each of the column 2 choices.

Column 3	Number of different line widths to be used in plotting this file.

   If there are more than one, the program will rotate through those specified, one for each outline.
   Columns 4 through 12 provide values for each line width. Start at left. Lines widths are 1 through 5, which yield lines from 1 to 5 dots wide.
   Columns 13 through 22 are not used for versatec plotting on Multics.

Example: sm sys op Please locate tape aar711.

3. Wait for message from operator saying that she has the tape(s).
4. Execute the exec_com. Usage is:

   ec gpt &1 &2 &3 ... &n

   where *&1* is the tape number, and *&2* to *&n* are segments to be put on this tape. Example:

   ec gpt aar711 plot1 plot2 plot3

   which places three plots on the tape. Note: You can put approximately 1,100 pages of segments on a 1,000-inch tape and about 2,200 pages onto a 2,000-inch tape. The exec_com will split a segment between two tapes, but you do not want to do this. The plotter cannot handle a multiple-tape file produced in this fashion.

HOW TO RUN SORT.VERS.COOR.EC ON MULTICS

Purpose of the program: **sort.vers.coor.ec** creates the files that are to be used for input to the Multics Versatec programs. The input files are those files on the tape that was created on the digitizer by program **dwgtape**.

To run the program:

A. Before you run these programs for the first time, the programs must be linked. This is done by typing:

```
lk > udd> Geoindx> PPorter> pgm1.vers.exthdr
lk > udd >Geoindx >PPorter >pgm2.vers.sequent
lk >udd>Geoindx> PPorter> pgm3.vers.merge
lk > udd>Geoindx> PPorter> sort.vers.coor.ec
```

B. If there is more than one coordinate file, then these coordinate files must be merged into one large coordinate file. This can be done in the editor **qedx**.

C. The output file of the merge must be of the form:

   coorNM.unsort

   where *NM* is the State code.

D. You must have ***coorNM.unsort*** in your directory or be linked to it. To run, type:

   ec sort.vers.coor*NM*

   where *NM* is the State code.

E. This exec com is made up of three programs and one sort. **pgm1.vers.exthdr** creates an unsorted header record file from the unsorted coordinate file that was created in the **qedx** editor. This unsorted header record file is input to the system sort where a sorted header record file is created.

F. **pgm2.vers.sequent** converts the unsorted coordinate file from a stream to a sequential file.

G. **pgm3.vers.merge** merges the sorted header file and the sequential coordinate file into the sorted coordinate file.

H. As **sort.vers.coor.ec** is executing, messages are displayed on the terminal indicating the progress of the job.

I. The files created by **versatec.ec** and **sort.vers.-coor.ec** can be input to the **index_versatec** programs.

HOW TO RUN MASTER ON MULTICS

Purpose of the program: **master** reads coordinate files for map outlines and calculates areas for each outline. It begins with the area for the entire State. After computing this area, the program compares the area with the true area from a file named **areano**. The true area divided by the computed area gives a factor that is used to adjust each area computed for each outline.

At the same time, a center point is computed for each area. Then these centers are tested to make sure that they lie inside each outline and that they are not too close to the boundary. If they pass the test, they are written to a file ***cntrNM***. Otherwise, they are put in a file named ***doubt***. This must be checked by hand and, if necessary, adjusted by hand:

Input files: **areano**, ***statNM***, ***coorNM***.
Output files: ***areaNM***, ***cntrNM***, ***measNM***, ***doubt***

To run the program:

A. Before running **master** for the first time, you must link it to your directory by typing:

```
lk >udd>Geoindx>HJohnson>master
lk >udd>Geoindx>HJohnson>areano
```

B. To run, type: master

C. When asked for it, type the FIPS code for the State.

D. When asked for it, type the denominator of the map scale for the map used for this State, format, (F8.0). For example, where the scale is given as 1/1,000,000, type 1000000.. Where the scale is 1/750,000, type 750000.. Be sure to include the decimal point.

E. After the State outline is used to compute State area, the machine will tell you the factor. This should be close to 1.0. If it is very different from 1.0, you may have the scale wrong or something may be wrong with the State file.

F. After the run is complete, list ***doubt*** and make corrections.

HOW TO RUN STATE_OPTIMA ON MULTICS

Purpose of the program: **state_optima** prints the maximum latitudes and longitudes that border a State outline.

To run the program:

A. You must have the following links:
   lk >udd>Geoindx>HJohnson>state_optima
   You must also be linked to the State radian file *strdNM.*

B. Type: state_optima
   and follow directions. You will be asked for FIPS code for the State.

HOW TO RUN ADDRAD ON MULTICS

Purpose of the program: **addrad** inserts correct areas, latitudes, longitudes, centers, and other data into the *strgNM* files for final input to the GRASP **convert** program.
   Input files: *strgNM, measNM, conxNM, ctrdNM*
   Output file: *redyNM*

To run the program:

A. Before running **addrad** for the first time, you must link it to your directory by typing:
   lk >udd>Geoindx>HJohnson>addrad

B. To run, type: addrad

C. When asked, type the FIPS code for the State.

To check out error messages in addrad:

A. While running **addrad**, the program may write error messages in the form:
   THERE IS NO AREA WITH IF = 28 AND ISF = 1

B. These messages must all be checked out.

C. The messages are caused by two conditions:
   1. The outline for this *IF* and *ISF* is a single point. This condition is evident from an inspection of *coorNM* file, where the *ISFNO* number in the header card is 2. When this condition occurs, no record appears in *measNM.*
   2. An error has occurred. When there is no outline for this *IF* and *ISF*, then *IBOUND* and *ISPAN* should not be present, or a record should be found in *measNM.*
      Inspect the reference file to see whether an outline is present, which is indicated by values in items 50-59, 76-85 (*IBOUND* and *ISPAN*). No record occurs in *measNM* for this *IF* and *ISF.*

D. Try running **master** again to see if the record in *measNM* was somehow dropped.

HOW TO RUN COVERT.EC ON MULTICS

Purpose of the program: **covert.ec** reads the *redyNM* file and creates a GRASP file for the State.

To run the program:

A. Before running **covert.ec** for the first time, you must create the following links:
   lk >udd>Geoindx>PFulton>covert.ec
   lk >udd>Geoindx>PFulton>index0
   lk >udd>Geoindx>PFulton>dicn
   lk >udd>Geoindx>PFulton>defn
   lk >udd>Geoindx>PFulton>mask
   lk >udd>Geoindx>PFulton>setmas
   lk >udd>Geoindx>PFulton>grasp
   lk >udd>Grasp>grasp
   lk >udd>Grasp>convert

B. Before running, print the **index0**, **dicn**, and **mask** file.

C. Type:
   ec covert *NM state*
   where *NM* is the FIPS code number for the State and *state* is the State name. Example:
   ec covert 45 SouthCarolina

D. After running **covert.ec**, print **index0** again to be sure it has been updated properly. Also compare the run printout with the sample to be sure it was successfully completed.

E. After running, be sure you give access on the new *indxNM* file to *.Gmap-Indx.*

F. If the State has to be run through **covert.ec** again, be sure to delete the *indxNM* file before running **covert.ec**, and delete the State line from **index0**.

HOW TO RUN GR.EC ON MULTICS

Purpose of the program: **gr.ec** sorts the State index file by scale and creates three files:
   t1p for scales less than or equal to 1:24000
   t2p for scales greater than 1:63360 and
   t3p for scales between 1:24001 and 1:63360

To run the program:

A. Before running **gr.ec** for the first time, create the following link:
   lk >udd>Geoindx>PFulton>gr.ec

B. Before running **gr.ec**, you must first have run **covert.ec** for the selected State.

C. Type: ec gr *NM*
   where *NM* is the FIPS code for the selected State.
   Example for Illinois: ec gr 17

D. The program will print the files *t1p*, *t2p*, and *t3p*. These files should be inspected for accuracy. If any discrepancy is found, the *redyNM* file must be corrected and **covert.ec** rerun for the selected State. Then **gr.ec** must be rerun to insure that the corrections were entered properly.

E. The three files *t1p*, *t2p*, and *t3p* must be kept and used in two succeeding programs:

1. They must be used in **inplot.ec**.
2. They must be used in **index_versatec**. After the final Versatec plots have been sent out for reproduction, the files should be deleted. If several States are processed through these programs at the same time, then the files should be renamed ***t1pNM***, where *NM* is the FIPS code for the selected State. In this way, the data can be stored safely until the programs are actually executed, at which time the files must resume their original names.

F. Caution: If the execution fails, be sure to delete the files that may have been created: ***t1, t2, t3, t1p, t2p, t3p, and output_file***. If not, the files will cause other retrievals to fail.

HOW TO RUN INPLOT.EC ON MULTICS

Purpose of the program: **inplot.ec** plots the three files created by **gr.ec**, which are ***t1p***, ***t2p***, and ***t3p***. It provides a visual check of the integrity of the plot files.

To run the program:

A. Before running for the first time, you must establish the links:

```
lk >udd>Grasp>assoc
lk >udd>Grasp>closer
lk >udd>Geoindx>PFulton>inplot.ec
lk >udd>Geoindx>PFulton>pn16
lk >udd>Geoindx>PFulton>pos
lk >udd>Geoindx>PFulton>plo
lk >udd>Geoindx>PFulton>plod
lk >udd>Geoindx>PFulton>ploc6
```

This program is run soon after **gr.ec** for the selected State and to access Tektronix routines type:

setup_tektronix_tcs.

B. Before any execution of the program, you must also have all the *x*, *y* coordinate files available, such as ***bordNM***, ***coorNM***, ***statNM***, and ***counNM***. This program must be run on a Tektronix terminal because it plots directly on the screen.

C. To run the program, type: ec inplot *NM state* where *NM* is the FIPS code for the State, and *state* is the name of the selected State. Example, for Illinois: ec inplot 17 Illinois

HOW TO RUN PN16 ON MULTICS

Purpose of the program: **pn16** plots a State index map interactively on a Tektronix CRT screen. This is a two-step process. First, a GRASP retrieval is executed wherein a disk file is created that contains the links to the coordinate Geoindex files. This GRASP file is identified as unit 13 and is described below. However, the program is also constructed so that the user has the option of plotting any combination of the input files.

To run the program:

A. Before running **pn16** you must link to it:

```
lk >udd>Geoindx>PFulton>pn16
lk >udd>Geoindx>PFulton>skod
```

B. Input: You must also have the State base-sheet files or link to them.

These files are:

coorNM, ***bordNM***, ***statNM***, ***counNM***, and ***gridNM***

Optionally, you need files created by a GRASP retrieval or a file in the same format as the ***coorNM*** file.

C. Type: pn16

D. The program will supply the following prompts. The user will supply the replies:

Prompt 1:	NEED STATE CODES (Enter y for yes.)
Reply:	y (The **skod** file is then printed; n or cr presents prompt 3.)
Prompt 2:	TYPE 1 AND HIT RETURN KEY WHEN READY.
Reply:	1 (MANDATORY REPLY)
Prompt 3:	ENTER STATE ID NUMBER.
Reply:	18 (example showing FIPS code for Indiana)
Prompt 4:	IF A COORDINATE FILE IS TO BE PLOTTED, ENTER Y.
Reply:	y (presents prompt 5) n or cr (presents prompt 7)
Prompt 5:	ENTER NAME OF FILE TO BE PLOTTED.
Reply:	t1p (example showing name of a file created by GRASP)
Prompt 6:	IF INPUT SHOULD BE SORTED, REPLY WITH A Y FOR YES. (This file should be sorted the first time it is used because the program expects the numeric identifiers in ascending order; n or cr presents prompt 7.)
Reply:	y
Prompt 7:	ENTER TITLE FOR MAP.
Reply:	Indiana for years greater than 1970 (example)
Prompt 8:	TO PLOT STATE ENTER 1.
Reply:	1 (causes State boundary to be plotted) 0 (No State boundary will be plotted.)

cr (No State boundary will be plotted.)

Prompt 9: COUNTY PLOT (ENTER 1 FOR SOLID LINE, 2 FOR DOTTED, ELSE 0.)

Reply: 1 (County boundaries will be plotted in solid lines.)
2 (County boundaries will be plotted in dotted lines.)
0 (No county boundaries will be plotted—presents next prompt.)
cr (No county boundaries will be plotted—presents next prompt.)

Prompt 10: TO PLOT GRID ENTER 1.

Reply: 1 (Latitude and longitude will be plotted.)
0 (No latitude and longitude will be plotted—presents next prompt.)
cr (No latitude and longitude will be plotted—presents next prompt.)

Prompt 11: TO SUPERIMPOSE ANOTHER FILE, ENTER 0 FOR NO, 1 FOR LINES ONLY, 2 FOR LINES AND CHARACTERS. (This prompt causes another plot file to be superimposed on the map. The file must have the same format as the coordinate files—for example, the locations of silver deposits could be plotted on the base or index map of Nevada.)

Reply: 1 (plots outlines and (or) points only; presents prompt 12)
2 (plots outlines and (or) points with an identifying number; presents prompt 12)
0 or cr (No superimposed file will be plotted; next erases screen and starts plotting.)

Prompt 12: ENTER FILE NAME. (This question is asked only if 1 or 2 were the replies to prompt 9.)

Reply: cu3 (example of the name of a plot file).

The screen is then erased, and the files are plotted. A bell rings when the plots are completed. For one hard copy, type the c and cr keys. For multiple hard copies, use the copy switch. When finished, cr.

Prompt 13: FOR AN ENLARGEMENT OF A PART OF THIS PLOT, TYPE Y.

Reply: n or cr (No enlargement will be made—presents prompt 16.)
y (An enlargement will be made according to the replies from prompts 14 and 15.)

Prompt 14: POSITION CURSOR AT LOWER LEFT OF DESIRED AREA; TYPE C.

Reply: Physically move the crosshair cursor to the required position and type c and cr.

Prompt 15: POSITION CURSOR AT UPPER RIGHT OF DESIRED AREA; TYPE C.

Reply: Physically move the crosshair cursor to the required position and type c and cr. (For best results, the window defined by the crosshair cursor should approximate the shape of the original plot; otherwise, geometric distortions will be introduced into the plot.)

The program from prompt 7 through 13 is then repeated.

Prompt 16: TO PLOT ANOTHER FILE ENTER Y FOR YES.

Reply: y (causes a return to prompt 4)
n or cr (ends the program and causes the message "good" to be printed to show a successful execution)

The system prints STOP fortran_io: Close files?
Reply: yes

HOW TO RUN BIGSTA ON MULTICS

Purpose of the program: **bigsta** compiles statistics on most of the files that we use in processing one State. It counts cards, computes lengths, and so forth, whenever appropriate.

Input files: ***coorNM***, ***comxNM***, ***statNM***, ***strdNM***, ***cordNM***, ***counNM***, ***curdNM***, ***cntrNM***, ***ctrdNM***, ***gridNM***, ***areaNM***, ***redyNM***, ***measNM***, ***bordNM***, and any others you wish to count

Output files: None

To run the program:

A. Before running **bigsta** for the first time, you must link it to your directory by typing:

```
lk >udd>Geoindx>HJohnson>bigsta
lk >udd>Geoindx>HJohnson>bigcal_bigsta
lk >udd>Geoindx>HJohnson>out2_bigsta
lk >udd>Geoindx>HJohnson>prim_bigsta
lk >udd>Geoindx>HJohnson>rads_bigsta
lk >udd>Geoindx>HJohnson>cards_bigsta
```

lk >udd >Geoindx >HJohnson >out1_bigsta
lk >udd >Geoindx >HJohnson >ftnumber

B. Type: bigsta
C. When asked, enter the two-digit FIPS State code.
D. After running through the standard files, the program asks for any other files you wish processed. You may wish to type the following: *paraNM*
E. After the run is complete, the total number of cards is also given. The printout of the total should be filed.

HOW TO RUN USMERG.EC ON MULTICS

Purpose of the program: **usmerg.ec** takes as input a newly created *indxNM* file and appends it to the existing indxus. indxus is the GRASP file that contains all the States. The output file is named usall. At the end of the run, it is dprinted for checking.

To run the program:

A. Before running **usmerg.ec** for the first time, you must establish the following links:
lk >udd>Geoindx>PFulton>usmerg.ec
lk >udd>Geoindx>PFulton>indxus
After establishing the link to indxus, copy it into your directory.
B. To run, type: ec usmerg *NM*
where *NM* is the FIPS code for the State that is to be added to the indxus file. Example: ec usmerg 17 will take indx17 for the State of Illinois and append it to indxus.
C. Study the program run listing for any errors. usall will be dprinted. Study the listing of usall for any errors.
D. If there are no errors, save the existing indxus by copying into a file called sindxus. Then delete indxus, and rename usall, indxus.
E. Run GRASP and check new indxus.
F. If there is an error:
 1. Delete usall and indxus.
 2. Copy sindxus to indxus.
 3. Rerun, starting at C.

HOW TO RUN STATE_TO_TAPE ON MULTICS

Purpose of the program: **state_to_tape** must be run on a console that gives a printout.

To run the program:

A. Be sure you have created the following links to the program:
lk >udd >Geoindx >HJohnson >state_to_tape
lk >udd >Geoindx >HJohnson>
heading_state_to_tape
lk >udd >Geoindx >HJohnson >sts_begin
lk >udd >Geoindx >HJohnson>
list_state_tape.ec
lk >udd >Geoindx >HJohnson>
disk_to_tape_fb_retain.ec
lk >udd >Geoindx >HJohnson>
disk_to_tape_vbs_retain.ec
B. Be sure you have the following files in your directory or that you are linked to an actual segment containing them:
coorNM, cordNM, statNM, strdNM, counNM, curdNM, cntrNM, gridNM, bordNM, redyNM, paraNM
C. Look up the number of this tape.
D. Look up the last file number that was written to this tape. If you have never run this program on this tape, then the last file number is 1 (the file that initialized the tape).
E. Type:
sm sys op Please find *tape number*
(*Tape number* being your tape number)
F. Wait until the operator sends a message to your console that he has found the tape.
G. Type: state_to_tape
(The program will prompt you for the information it needs.)
H. As directed by the machine, make two copies of the printout. Store these printouts for future reference. Then put the original printout in your log for this tape. You will need to refer to it whenever you add more records to the tape or whenever you want to print a listing of these files.
I. Remember to use a second backup tape with these files.
J. To drop these files from your directory, link to:
lk >udd >Geoindx >HJohnson >drop.ec
and type: ec drop *NM*
where *NM* is the State number.

HOW TO RUN PULL_OFF ON MULTICS

Purpose of the program: **pull_off** enables the user to select files from the Geoindex files and to write the selected files to disk.

To run the program:

A. Make the following links:
lk >udd >Geoindx >HJohnson >pull_off
lk >udd >Geoindx >HJohnson >
separate_pull_off
lk >udd >Geoindx >HJohnson >state_pull_off
lk >udd >Geoindx >HJohnson >up_file_number
lk >udd >Geoindx >HJohnson >
tape_to_disk_vbs_retain.ec

lk >udd >Geoindx >HJohnson>
tape_to_disk_fb_retain.ec
lk >udd >Geoindx >HJohnson>
list_state_tape.ec

B. You need to know the tape number.

C. You need to decide whether to take off certain separate files, or to use all the files for one State.

D. To take off separate files, you must know their exact names, and their file numbers (positions) on the tape. This information can be obtained from a tape map or by using **list_state_tape.ec**.

E. Send a message to the operator to get your tape:
sm sys op Please find tape *nnnnnn*

F. Type: pull_off
Follow directions.

G. This program prints out a tape map at the end. If you do not want that, just hit break key after it starts printing.

H. Remember that many of the tape files of ***paraNM***, ***cntrNM***, and ***redyNM*** have an extra record that does not end in a newline character. You may have to edit these files before using them.

HOW TO RUN BACKUP ON MULTICS (COMPLETE DUMP)

Purpose of the program: **backup** enables you to dump one or more segments to your tape. You can even dump whole directories. The program creates a file named control.dump. You must type in all the absolute path names of the segments or directories you want to dump to tape. Note: These must be entered in alphabetical order.

How to run the program:

A. First you must make the followings links:
lk >udd >Geoindx > HJohnson > backup
lk >udd >Geoindx > HJohnson > backup.ec
lk >udd >Geoindx > HJohnson > backup1
lk >udd >Geoindx > HJohnson > backup2
lk >udd >Geoindx > HJohnson > dump.ec

B. You must know the absolute path name of the segments you want to dump for backup. Example:
>udd >Geoindx > HJohnson >indxus
>udd >Gmap_Indx >HJohnson
This would dump the one segment **indxus** and the whole directory >udd > Gmap_Indx >HJohnson

C. You must know your tape number.

D. You must send a message to the operator to find your tape:
sm sys op Please find tape *nnnnnn*

E. From operator: Go Ahead

F. Type: backup
Computer will respond: backup1

G. Prompt: DID YOU SEND A MESSAGE TO THE OPERATOR TO FIND YOUR TAPE? IF YOU DID, TYPE A 1
Response: 1
Computer responds with a prompt. Last part of prompt is:
NOW TYPE IN THE ABSOLUTE PATH NAME OF THE NEXT SEGMENT OR DIRECTORY YOU WANT TO BACKUP. TYPE ITS ABSOLUTE PATH NAME
Example: >udd >Geoindx > HJohnson > indxus

H. Prompt: IF YOU WANT TO DUMP MORE PATHS, TYPE 1; OTHERWISE, 0
Stop
Prompt: FORTRAN IO : CLOSE FILES?
Response: Yes
Message: io close file10
io detach file10
Computer prints: backup2
Prompt: TYPE YOUR TAPE NUMBER, FORMAT A6
Response: *nnnnnn* (example, 111849)
Computer prints:
Complete_dump_control.dump HHJ -debug
> udd > Geoindx >HJohnson >indxus
Prompt: TYPE PRIMARY_DUMP_TAPE LABEL
Response: 111849
Computer prints:
TAPE_: MOUNTING TAPE 111849 FOR WRITING TAPE_: TAPE 111849 MOUNTED ON DRIVE 1 DUMP FINISHED.

I. The computer prints:
THIS ROUTINE ADDS 1 OR 2 MESSAGE FILES TO YOUR DIRECTORY WHICH ARE AUTOMATICALLY DPRINTED. THEY ARE VERY IMPORTANT AND SHOULD BE PICKED UP AND SAVED IN A SAFE PLACE.
THEY ARE THE DUMP.MAP AND POSSIBLE ERROR MESSAGE. SAVE THEM IN A SAFE PLACE. THROW AWAY ANY OLD DUMP.MAPS FOR THIS TAPE, SINCE THEY ARE COMPLETELY OBSOLETE.
Type: Rename indxus indxus_true

HOW TO RUN RESTORE ON MULTICS

Purpose of the program: **restore** enables you to put a file from a backup tape, (created by the program backup) onto the Multics system

How to run the program:

A. First make the following links:
```
lk >udd >Geoindx >HJohnson >restore
lk >udd >Geoindx >HJohnson >retrieve.ec
```
Decide which segments you want to restore from your backup tape. Check the dump.map for that tape to make sure that the segments are present. You must know their exact pathnames.

B. You must have these segments listed in the same order that they appear in the dump.map.

C. Be sure to rename your file before bringing it back from tape, such as change indxus to *indxus_true*.

Then you can use the command:
```
compare indxus indxus_true
```
to see if there are any differences.

D. Type: new_proc
and wait for system to respond; then type: restore

E. Prompt: DID YOU SEND A MESSAGE TO THE OPERATOR TO FIND YOUR BACKUP TAPE? IF YOU DID, TYPE: 1

F. Prompt: NOW TYPE THE ABSOLUTE PATH NAME OF THE NEXT SEGMENT OR DIRECTORY THAT YOU WANT TO RESTORE; THIS NAME IS ON YOUR BACKUP TAPE.
Use its absolute path name. Example:
```
lk >udd >Geoindx > HJohnson > indxus
```

G. Prompt: IF YOU WANT TO RESTORE MORE PATHS, TYPE 1; OTHERWISE, TYPE 0

H. Prompt: TYPE THE NUMBER OF YOUR BACKUP TAPE, FORMAT (A6). Example: 111849

Computer prints:
```
RETRIEVE CONTROL.RETRIEVE -DEBUG
INPUT TAPE LABEL: 111849
TAPE_: MOUNTING TAPE 111849 FOR READING
TAPE_: TAPE 111849 MOUNTED ON DRIVE 1
BEGIN AT 01/25/78 2053.3 EST WED.
END OF READABLE DATA.
BK_INPUT: ARE THERE ANY MORE TAPES TO BE RELOADED?
```
User responds:
```
No
```
Computer prints:
```
NORMAL TERMINATION 01/25/78 2053.4 EST WED.
DPRINT -DL CONTROL.RETRIEVE.
                         RETRIEVE.MAP
1 REQUEST SIGNALLED, 0 ALREADY IN PRINTER QUEUE 3
```
This routine automatically dprints a retrieve map. Check to make sure that the requested files are in your directory.

User should then issue **list** command to find out whether the file has been restored. Example:
ls indxus

User should then issue **compare** command to insure the segment restored is the same as the segment that was written to tape using **backup**. Example: Compare indxus indxus_true

Computer responds:
```
NO DISCREPANCIES FOUND.
```

HOW TO RUN VERPLOT ON MULTICS

Purpose of the program: **verplot** generates the status map for the Geoindex. This program reads a file of commands and creates a Versatec plot file using the instructions from that file.

To run the program:

A. Link to the following Multics files:
```
lk >udd >Geoindx >PPorter >verplot
lk >udd >Geoindx >PPorter >init_vals
```
Copy or link to the following coordinate files:
```
lk >udd >Geoindx >PPorter >stat90
lk >udd >Geoindx >PPorter >hawaii
lk >udd >Geoindx >PPorter >alaska
lk >udd >Geoindx >PPorter >puerto_rico
```

B. Create or link to some command file. See listing 4 for instructions on the contents of this file. An example is shown in listing 5.

C. Run the program on Multics by typing: verplot

D. The program will ask the following question:
```
WHAT IS THE NAME OF YOUR COMMAND FILE??
USE NO MORE THAN SIX CHARACTERS!
```
and will read your answer, then attach and open this file for reading. A nonexistent file will give an error message on the terminal.

E. The program reads the data and calls the subroutine corresponding to that command. Any error in a command causes this error message to be written along with the entire data record:

THIS LINE CANNOT BE IDENTIFIED AS A COMMAND.

Note: All error messages are written to a file called ***temp10***, which is created by the program and at the end of the run dprinted to provide a hard copy, which is always provided unless the program does not run to completion (example: hitting break key); if it is not dprinted, you have no means to get the information in ***temp10***. An abnormal termination leaves ***temp10*** as a zero length file.

F. As each subroutine is called, it will process the information given in the command line and write messages to ***temp10*** for both valid operations and for errors. We have tried to take into account every type of possible error for which it gives an appropriate error message and have the program continue. This program should give a plot and a progress and error report to cross-check and to identify any errors and omissions. Any error for which a report is not given is not a common typing or omission error and must be resolved in a different manner.

G. After the program has finished the plots, it prints this message:

```
PLOT FINISHED
N VECTORS LOST
N ACTIVE LINES USED
1 request signalled, N already in printer queue 3.
Stop.
```

H. To put plot segments onto tape: An exec_com will place as many segments as you wish onto a tape. The best way to do this is to copy the exec_com into your segments along with all six of its names. Type: copy >udd >Geoindex >PPorter >gpt.ec-all

I. To Use: Label a tape "For Multics Use" and take it to production control. Send message to system operator:

sm sys op Please locate tape number *nnnnnn*

Wait for message from operator saying he has the tape(s). Execute the exec_com. Usage is:

ec gpt &1 &2 &3 . . . &n

where *&1* is the tape number, and *&2* to *&n* are segments to be put on this tape. Example: ec gpt aar730 plot1 plot2

LISTING 4.–*Formation of the command file for* **verplot** *and an example command file*

The file is composed of one or more instructions taken from a list of eight commands along with a variety of keywords that give almost limitless scope in creating a Versatec plot file. Restrictions for a command will be explained in that particular section.

I. All commands consist of records that have a maximum of 80 characters. This record length facilitates the use of cards if wanted.
II. All commands and keywords can be either uppercase or lowercase, but must all be of one type in a particular word. The types can be mixed within a record.
III. Each command must start in column 1 (card-image terminology used) and must be immediately followed by a semicolon.
IV. Keywords can be in any order but must be separated by commas. For each command, certain keywords are required and others are optional; default values are present if an optional keyword is missing. Many keywords include some data values.
V. Except for the PLOT command, all commands and keywords must be on the same record.
VI. Blanks are ignored except in the following cases:
   A. All commands and keywords must be in a continuous string.
   B. Commands must start in column 1 and be immediately followed by a semicolon.
   C. Keywords that require a data value must be immediately followed by the character =.
   D. When used in a data value as a place holder. Example: x = !!1!!, will be interpreted as x = 100. Example: y = 2!.!!, will be interpreted as y = 20.
   Note: "!" is the symbol used for a blank space.
VII. If at any time the same keyword occurs twice in a record, the second occurrence will take precedence.
VIII. In all commands, if an error in the data value for a keyword occurs, that value will be either ignored or set to the default value if one exists.
IX. In all commands, if any required keyword is missing, the command is ignored.
X. All numbers can be in either integer or real number format.
XI. An example of a command file, **statpm**, follows:

```
          statpm

outline; npoint=5,shade=13
 19.4,0.67,19.4,0.92,19.73,0.92,19.73,0.67,19.4,0.67
outline; npoint=5,shade=12
 19.4,1.59,19.4,1.84,19.73,1.84,19.73,1.59,19.4,1.59
outline; npoint=5, shade=1
 19.4,2.51,19.4,2.76,19.73,2.76,19.73,2.51,19.4,2.51
```

LISTING 4.—*Formation of the command file for* **verplot** *and an example command file*—Continued

```
outline; npoint=5, shade=4
 19.4,3.43,19.4,3.68,19.73,3.68,19.73,3.43,19.4,3.43
legend; x=19.88,y=0.72,height=0.1,nchar=9
PUBLISHED
legend; x=19.88,y=1.64, height=0.1,nchar=8
IN PRESS
legend; x=19.88,y=2.56,height=0.1,nchar=22
IN COMPUTER PROCESSING
legend; x=19.88, y=3.48,height=0.1,nchar=14
IN COMPILATION
reorg; x=1.0, y=1.0
outline; npoint=5
 0.0,0.0,0.0,3.2,3.4,3.2,3.4,0.0,0.0,0.0
legend; x=1.35, y=3.3, height=0.1, nchar=6
ALASKA
plot; name=alaska, shadeall, pattern=1,13
end plot;
reorg; x=5.6, y=-0.4
outline; npoint=5
 0.0,0.0,0.0,1.3,2.0,1.3,2.0,0.0,0.0,0.0
legend; x=0.68, y=1.42, height=0.1, nchar=6
HAWAII
plot; name=hawaii, shadeall, pattern=1,13
end plot;
reorg; x=8.0, y=0.7
outline; npoint=5
 0.0,0.0,0.0,0.7,2.0,0.7,2.0,0.0,0.0,0.0
legend; x=0.37,y=0.8, height=0.1, nchar=11
PUERTO RICO
plot; name=puerto_rico, shadeall, pattern=1,13
end plot;
reorg; x=-14.6, y=-1.3
linwid; 4
outline; npoint=5
 0.0,0.0,0.0,17.75,22.65,17.75,22.65,0.0,0.0,0.0
legend; x=6.775, y=16.8, height=0.3, lwidth=4, nchar=30
STATUS OF GEOLOGIC MAP INDICES
legend; x=7.725, y=16.0, height=0.3, lwidth=4, nchar=23
SATURDAY, MARCH 1, 1980
linwid; 1
plot; name=stat90, textfield=1,6, refclear=, height=0.1,
   select, selshade
    1     1   13
    4     1   13
    5     1   13
    6     1    1
    8     1   13
    9     1    1
   12     1    1
```

LISTING 4.–*Formation of the command file for* **verplot** *and an example command file* –Continued

```
    13     1    12
    16     1    13
    17     1    13
    18     1    13
    19     1    13
    20     1    13
    21     1    13
    22     1    13
    23     1    13
    24     1     1
    25     1     1
    26     1     1
    27     1     1
    28     1    13
    29     1    13
    30     1    13
    31     1    13
    32     1    13
    33     1     1
    34     1    13
    35     1    13
    36     1    13
    37     1    13
    38     1    13
    39     1    13
    40     1     1
    41     1     1
    42     1     1
    45     1    13
    46     1    13
    47     1     1
    48     1    12
    49     1    13
    50     1     1
    51     1    13
    53     1     1
    54     1    13
    55     1     1
    56     1    13
END PLOT;
plot; name=stat90, textfield=1,6, height=0.1, select,selshade
    10     1     1
    44     1     1
end plot;
plot; name=stat90, select, selshade
    11     1     1
    25     2     1
    25     3     1
    51     2    13
end plot;
```

LISTING 5.–*Commands for* **verplot**

END PLOT

Purpose:

The command END PLOT informs the program that the information describing the plotting of a file is at an end. The only use is in conjunction with the PLOT command.

Command usage:

END PLOT; (or end plot;)–The program will continue reading the command file until the end plot command is reached.

LEGEND

Purpose:

The command LEGEND plots the character string given in a manner described by the keywords.

Command usage:

LEGEND; (or legend;) (keywords) followed by the text string on the next record.

Required keywords:

x= (or *X*=)–*x* coordinate of start of text string
y= (or *Y*=)–*y* coordinate of start of text string
height= (or *HEIGHT*=)–Height of each character
nchar= (or *NCHAR*=)–Number of characters in text string on next record (Always start in column 1 and use no more than 80 characters, which can be either uppercase or lowercase (or mixed).)

Optional keywords:

angle= (or *ANGLE*=)–The angle at which the text string is plotted (Default = 0 degrees.)
lwidth= (or *LWIDTH*=)–The width of the line in dots (Default = 1 dot wide.)

LINWID

Purpose:

The command LINWID changes the line width of all subsequent plotting to the value given.

Command usage:

linwid; (or LINWID;) (number)–The number must be 1, 2, 3, 4, or 5 because the Versatec software will accept no others. If no number is present or an error is in the data, the default value of 1 will be used.

OUTLINE

Purpose:

The command OUTLINE plots an outline whose coordinates are on the following record(s).

Command usage:

outline; (or OUTLINE;) (keywords)–Followed by the record(s) containing the data points.

Required keywords:

npoint= (or *NPOINT*=)–The number of data pairs of *x*, *y* coordinate points that follow (All coordinates must be separated by commas. No more than 20 pairs can be used, with no more than 20 values per record.)

Optional Keywords:

shade= (or *SHADE*=)–Tells the program to shade this outline and gives the reference number of the pattern to use in shading (The valid reference numbers are 1 through 20 with any other number defaulting to 1. However, numbers 14 through 20 are blank patterns that are reserved for use with the PATTERN command.)
noline (or *NOLINE*)–Causes the outline not to be plotted (This should not be used unless the shade option is also used).

PATTERN

Purpose:

The command PATTERN reads data values and stores these in an array that will be used as a pattern for shading at some later point in the program.

Command usage:

pattern; (or PATTERN;) (keywords)–Followed by one or more records containing the data values.

LISTING 5.–*Commands for* **verplot**–Continued

Required keywords:

refnum= (or *REFNUM=)*–The reference number to be used in identifying this pattern (It must be from 14 through 20.)
numword= (or *NUMWORD=*)–The number of data words on the following record(s) (This number can only be 1, 2, 4, 8, or 16. Any other number will give an error. The maximum number of words accepted by Versatec is 16, and all the others divide evenly into 16.)
type= (or *TYPE=*)–The only choices are *INTEGER* or *OCTAL*. The data values start on the next record. The program will read as many records as necessary to satisfy the *numword* variable. All data values are separated by commas.

PLOT

Purpose:

The command PLOT reads the name of a file, attachs and opens it for input and plots the data in that file according to the other keywords or to the default values.

Command usage:

plot; (or PLOT;) (keywords)–Followed by records containing a selective records list. The END PLOT command must always be used in conjunction with this.

Required keywords:

name= (or *NAME=*)–Contains the name of the file to be plotted (Not more than 20 characters may be used.)
Example: PLOT; NAME=*filename*
END PLOT;
This constitutes the simpliest use of the plot command.
Note: This command will plot only those files whose outlines have a header card where the number of pairs of points in the outlines are listed in integer format in columns 16–20. The data points will follow in (12F6.3) format with the first data point being a text position.

Optional keywords:

height= (or *HEIGHT=*)–The height of each character in the header card text; default value of 0.14 in.
noline= (or *NOLINE=*)–Does not plot the outline but allows all other options, such as character plotting and shading (Default is to plot the outlines.)
pattern= (or *PATTERN=*)–Followed by a series of numbers (The first is the count of how many more numbers follow. The remaining numbers are a sequence of pattern reference numbers, through which the program will rotate when shading outlines. Default pattern sequence is 1 through 10.)
refclear= (or *REFCLEAR=*)–Will clear the area around text when shading (This has one major problem. The Versatec software does not have a clearing function. Instead, if two or more areas are given in one shading command, the software will alternate shading and clear areas as the areas overlap. For our purpose, this is acceptable if the area to be cleared lies entirely within the outline. However, if areas overlap or the text lies outside, effects will be confusing. Default is not to clear the area.)
select= (or *SELECT=*)–Will plot only those outlines listed in the selective file following the keywords.
selshade= (or *SELSHADE=*)–Will shade only those outlines that have a valid pattern reference number listed in the selective file following the keywords.
shadeall= (or *SHADEALL=*)–Will shade all outlines and will rotate through the pattern sequence given (or 1–10 by default) (An outline pattern can be changed by listing a valid pattern reference number in the selective file following the keywords.)
textfield= (or *TEXTFIELD=*)–Followed by a series of numbers, the first number gives the count of how many numbers follow. The remaining numbers are a sequence representing some of the eight fields on the header cards. These are numbered from 1 to 8 from left to right. This sequence of numbers tells what fields will be plotted and in what order. Any blank or zero valued field will be ignored.
Example: text field=3,8,1,6
There are three fields to plot. First field 8, then field 1, and then field 6. Each field will be lined up underneath the previous one.

The keywords may occupy more than one record. All records containing keywords, except the last, must have a comma as the last entry on the record. This is the only indication that there are more keywords given.

A keyword with associated data values must be contained on one record. They cannot span records.

The selective file following the keywords comprises records containing the reference number (field one of the header card), the subfeature number (field three), and an optional pattern reference number. These are all five character fields contained in column 1 through 15 of the record. The two outline identifiers must be in this field exactly as they are in the five character header card field. The pattern reference number is in format (I5).

REORG

Purpose:

The command REORG changes the software origin of the plot file. This has the effect of moving the subsequent plotting commands in relation to those done previously.

LISTING 5.—*Commands for* **verplot**—Continued

Command usage:

reorg; (or REORG;) (keywords)
One, but not both, of the following keywords must be present. If one is missing, the default of 0 for that value will be used. A movement to the left or down is negative; right or up is positive.

Keywords:

x= (or *X*=)—The amount of movement (inches) in a left or right direction
y= (or *Y*=)—The amount of movement (inches) up or down

SCALE

Purpose:

The command SCALE changes the scale of all subsequent plotting.

Command usage:

scale (or SCALE) (number)
A blank data value or an error will default to a scale of 1.

SYMBOL

Purpose:

The command SYMBOL changes the character plotted when a single point is encountered.

Command usage:

symbol (or SYMBOL) (number)
The number represents some character. Any error in the number will default to a small triangle (number 2).

HOW TO RUN PIN90 ON MULTICS

Purpose of the program: **pin90** will plot the U.S. map and then, if the user wishes, will plot numbers, symbols, and outlines using GRASP files and will also plot the grid file.

Input files: *file14*–**stat90**; *file15*–GRASP files; *file16*–grid file

Output files: None (The only output is the plot on the screen.)

To run the program:

A. Before running **pin90** for the first time you must link it to your directory by typing:

```
lk >udd>Geoindx>PPorter>pin90
lk >udd>Geoindx>PPorter>enlrg
lk >udd>Geoindx>PPorter>indiv
lk >udd>Geoindx>PPorter>min-max
lk >udd>Geoindx>PPorter>plocv
lk >udd>Geoindx>PPorter>grid
```

B. To run, type: pin90

C. The user will then respond to the following questions:

1. NEED SYMBOL CODES? (ENTER Y FOR YES.) If you want to see the symbol and corresponding number, type y and cr. Otherwise, just enter cr and proceed to C3.
2. TAP 1 AND RETURN KEY WHEN READY
3. Screen is erased.
4. ENTER SYMBOL NUMBER AND FILE TO BE PLOTTED. Example: 43silver You may enter as many as five files (maximum number of eight characters for file name). After each entry, enter cr and the message will appear again. When you have entered the last file or if you have no entry, just enter cr.
5. FOR SYMBOL AND NUMBERS (WITH PLOTTING), TYPE 1; FOR SYMBOL AND (OR) OUTLINE (NO NUMBERS), TYPE 2; FOR NUMBERS ONLY (NO SYMBOLS OR PLOTTING), TYPE 3. If you had no entry for C4 just enter cr. Otherwise, type in the number and cr.
6. Screen is erased.
7. ENTER TITLE FOR MAP. Example: U.S. MAP
8. TO PLOT INDIVIDUAL STATES, ENTER 1–FOR ENTIRE U.S., ENTER 2. You must enter 1 or 2.
9. TO PLOT GRID, ENTER 1. If you want the grid, type 1 and cr; otherwise just type cr.
10. IF YOU WANT A HARD COPY UPON COMPLETION, TYPE C.
11. Screen is erased. If you entered 1 in response to C8, the following will appear on the screen:

    GIVE NUMBER OF STATES TO BE PLOTTED
    LIMIT OF 10 IN ASCENDING ORDER
    MUST BE A 2 DIGIT NUMBER, 01-51

    For example, to plot the States of Illinois (12), Indiana (13), Kentucky (16), Ohio (34), and West Virginia (49), respond with the code number for each State, as follows: 1213163449.

    This means a group of as many as 10 States. The States and their corresponding numbers appear on listing 6.

    Screen is erased. The States along with GRASP files or grid file are plotting; if the user typed c in response to C10, a hard copy will be made automatically at this time.

12. FOR AN ENLARGEMENT OF PART OF THIS PLOT, TYPE Y. Type y and cr if you want an enlargement; otherwise enter cr and proceed to C17.
13. FOR A HARD COPY AFTER ENLARGEMENT, TYPE C. If the user wants an automatic hard copy, type c and cr; otherwise just enter cr.
14. POSITION CURSOR AT LOWER LEFT OF DESIRED AREA: TYPE C. Postion the vertical and horizontal cursors at the desired location, type c and cr.
15. POSITION CURSOR AT UPPER RIGHT OF DESIRED AREA, TYPE C. Position the vertical and horizontal cursors at the upper right location, type c and cr.
16. Screen is erased. An enlargement of the desired area is plotted, and an automatic hard copy is made upon completion if the user typed c in response to C13.

LISTING 6. – *The 48 conterminous States and District of Columbia and their corresponding two-digit numbers that are used for plotting individual States*

[These are codes used on file **stat 90** (not FIPS codes)]

Alabama	AL	01	Nebraska	NE	26
Arizona	AZ	02	Nevada	NV	27
Arkansas	AR	03	New Hampshire	NH	28
Calfornia	CA	04	New Jersey	NJ	29
Colorado	CO	05	New Mexico	NM	30
Connecticut	CT	06	New York	NY	31
Delaware	DE	07	North Carolina	NC	32
District of Columbia	DC	08	North Dakota	ND	33
Florida	FL	09	Ohio	OH	34
Georgia	GA	10	Oklahoma	OK	35
Idaho	ID	11	Oregon	OR	36
Illinois	IL	12	Pennsylvania	PA	37
Indiana	IN	13	Rhode Island	RI	38
Iowa	IA	14	South Carolina	SC	39
Kansas	KS	15	South Dakota	SD	40
Kentucky	KY	16	Tennessee	TN	41
Louisiana	LA	17	Texas	TX	42
Maine	ME	18	Utah	UT	43
Maryland	MD	19	Vermont	VT	44
Massachusetts	MA	01-04	Virginia	VA	45,46,47
Michigan	MI	21	Washington	WA	48
Minnesota	MN	22	West Virginia	WV	49
Mississippi	MS	23	Wisconsin	WI	50
Missouri	MO	24	Wyoming	WY	51
Montana	MT	25			

APPENDIX C. COMPUTER-PROGRAM REFERENCE

EXEC_COM NAME: COMTAPE.EC

Author: Pearl Porter
Purpose of the program: **comtape.ec** reads an outside ASCII tape into the Multics system and writes the tape into a segment given by the user. **Comtape.ec** is written in Multics command language. All the programs in the Geoindex are written in Fortran IV unless otherwise specified.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec comtape *nnnnnn segname*
Arguments:
-ids– Input description of the tape
-ods– Output description
Subroutines called: None
Common data referenced: None
Input file: **refNM** on magnetic-tape
Output file: **refNM**
Arrays used: None
Called by: None
Error checking and reporting: The **comtape.ec** segment will work for tapes that are unlabeled and second in the file sequence and have fixed length format, density of 800, record length of 80, and block size of 800. After the **comtape.ec** has been executed, do not try to type in other commands until the process has been completed because this can cause errors.
Constants: None
Program logic:
1. An outside ASCII tape is read into the Multics system and written to a segment that the user specified.
2. The user will receive a count number of the records copied onto *SEGNAME*, and the file will automatically be dprinted.

```
copy_file -ids "tape_ibm_ &1 -nlb -nb 2 -fmt fb -den 1600 -rec 80 -bk
800" -ods "record_stream_ -target vfile_ &2" dp &2
&quit
```

PROGRAM NAME: CHKREF

Author: Harold Johnson

Purpose of the program: **chkref** is used to check the accuracy of the reference files. It checks whether certain records are integer or real numbers, whether the records exceed their prescribed lengths, whether the State number is consistent, whether the records are in the correct order within each individual reference, whether the separate references are in correct order, and whether the first record is correct.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: None

Arguments: None

Subroutines called: **assoc**, **setup_chkref**, **clean_chkref**, **checkitem_chkref**, **reup_chkref**, **rfskip_chkref**

Common data referenced: None

Input files: *refNM*, matrix

Output files: *refNM*

Arrays used: None

Called by: None

Error checking and reporting: Messages are written to the interactive user when the State number is wrong, when the *id* number is out of order, when the item number is out of order, when the initial item number for a reference is not 2, when the State name is wrong, and whenever a read error occurs. The user is informed of the total number of records that were read in the reference files.

Constants: None

Program logic:

1. The file to be checked and the control file matrix are attached to Fortran numbers 30 and 22 by calling **assoc**.
2. Vectors *itype*, *ichar*, and *item* are set up by the subroutine **setup**, which describes the type and number of character postions allocated to each reference item.
3. Subroutine **clean** puts blanks into the words of *ifile*, *jfile*, and *ifile1*.
4. A reference record is read, noting its State number *istate*, reference number *jf*, and item number *jtm*. The reference data on this record is read into *jfile*.
5. Subroutine **checkitem** uses *jtm* to check if the type of data in *jfile* is integer or floating point as prescribed by *itype* for this *jtm* and whether or not *jfile* lies in limits prescribed by *ichar*.
6. *jfile* is read into *ifile1* and **reup_chkref** transfers information from *jf*, *jtm*, and *jfile* into *if*, *itm*, and *ifile*, respectively.
7. A new record is read into *jstate*, *jf*, *ftm*, and *jfile*, and they are checked by **checkitem_chkref**. *istate* and *jstate* are compared to see if they are identical.
8. Where *if* is greater than *jf*, the *if* numbers are out of order, and an error message notes this; where *if* is less than *jf*, a new reference file has been reached, and it is checked to see if item number *jtm* is a 2.
9. Where *if* is equal to *jf*, item numbers are compared to see if *jtm* is greater than *itm*. Unless *itm* is 87 (indicating that there are repetitions in this reference), an error message appears.
10. After an error or after satisfactorily passing each test, **reup_chkref** is called to move *jf*, *jtm*, and *jfile* to *if*, *itm*, and *ifile*. Then control passes to step 7.
11. After all records have been checked, *ncard*, the number of cards read, is written in a message to the user.

```
c ******** CHKREF PROGRAM *****
c     SEPT. 16, 1976         H. JOHNSON
c
external io(descriptors)
c
c     THE FOLLOWING FILES ARE REQUIRED FOR THIS PROGRAM:
c     INPUT FILES:
c     22 = "MATRIX" FILE WHICH FORMATS IRIS RECORDS.
c     30 = REFERENCE FILE TO BE CHECKED.
c     OUTPUT FILE:
c     06 = MESSAGE FILE.
c
c Converted to Multics  February 17, 1977 H. Johnson
c     THIS PROGRAM IS WRITTEN TO RUN ON THE NEWEST REFERENCE FILES
c     WHICH ARE MADE UP TO USE WITH THE NEWEST GRASP PROGRAM, IRIS AN
\cD C
c     EATE
```

```
c     THIS PROGRAM IS DESIGNED TO RUN THROUGH THE REFERENCE FILES AND
c     CHECK FOR THEIR ACCURACY.
c     IT CHECKS WHETHER THE FILE IS INTEGER OR REAL IF IT IS SUPPOSED
\c TO
c     BE
c     IT CHECKS IF THE FILE IS WITHIN THE PRESCRIBED LIMITS
c     IT CHECKS ON THE STATE NUMBER
c     IT CHECKS ON THE ORDER OF THE SEPARATE FILE SUBJECTS.
c     IT CHECKS ON THE OREDER OF THE ITEMS WITHIN A SINGLE SUBJECT.
c
c
       dimension itype(46),ichar(46),item(46,10),ifile(73),jfile(73)
      dimension ialpha(5),ifilel(50)
       character*32 filename
      data iblank/"    "/
    call io ("attach","file22","vfile_ ","matrix","-append","-ssf")
      call io ("open","file22","si")
      write(6,890)
890    format(" enter the file name to be checked :")
      read 895, filename
895    format (a32)
      call io ("attach","file30","vfile_",filename,"-append","-ssf")
      call io ("open","file30","si")
c
c
c
c
           idim=46
c     IDIM IS THE NUMBER OF RECORDS IN THE MATRIX FILE
c
           call setup_chkref(itype,ichar,item,idim)
c     THIS SET UP THE MATRICES ITYPE(IDIM),ICHAR(IDIM),ITEM(IDIM,IWII
\cDE)
c     WHICH DESCRIBE THE TYPE AND NUMBER OF CHARACTER POSITIONS
c     ALLOCATED TO EACH ITEM.
c
           nfile=73
           call clean_chkref(ifile,nfile)
c     THIS ROUTINE PUTS BLANKS INTO THE WORDS OF IFILE.
c
           call clean_chkref(jfile,nfile)
           nfile=40
           call clean_chkref(ifilel,nfile)
c
c
c
           call rfskip_chkref(ncard)
c     THIS ROUTINE READS DOWN FILE 30 LOOKING FOR THE FIRST TRUE RECOR
\cD,
c     SKIPPING THE CARDS WHICH MERELY DESCRIBE THE REFERENCE FILE.
c     NCARD IS THE KEY NUMBER FOR THE FIRST RECORD.
c     IT POSITIONS 30 READY TO READ THE FIRST RECORD.
c
           ncard=1
```

```
c
          read(30,930,end=1000,err=500)istate,jf,jtm,(jfile(k),k=1,73)
930       format(i2,i3,i2,73a1)
c
          call checkitem_chkref(idim,ncard,item,jtm,jfile ,itype,ichar
\c)
c     THIS ROUTINE CHECKS IF THE FILE IS INTEGER OR FLOATING POINT
c     WHEN ITS ITM NUMBER INDICATES THAT.
c     IT ALSO CHECKS IF THE FILE IS CONTAINED WITHIN THE BOUNDARY SET
\cBY
c     ITS ITM NUMBER.
c     THESE TYPES AND LIMITS ARE READ FROM MATRIX AND FOUND HERE IN TH
\cE
c     FILES ITYPE AND ICHAR.
          do 5 k=1,40
          ifile1(k)=jfile(k)
5         continue
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
c     THIS ROUTINE PUTS INFORMATION IN THE J-FILES,JF,JT,JFILE,
c     INTO THE IFILES IF,ITM,IFILE.
c
10        ncard=ncard+1
          read(30,930,end=1000,err=500)jstate,jf,jtm,(jfile(k),k=1,73)
          call checkitem_chkref(idim,ncard,item,jtm,jfile,itype,ichar)
          if(istate .eq. jstate) go to 20
          write(6,940)ncard
940       format(" THE STATE NUMBER IS WRONG ON RECORD NUMBER ",i6)
          write(6,931)jstate,jf,jtm,jfile
931       format(i2,i3,i2,73a1)
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
          go to 10
c
20        if(if .le. jf) go to 30
          write(6,950)ncard
950       format(" THE IF NUMBER IS OUT OF ORDER IN RECORD NUMBER ",i6
\c)
          write(6,931)jstate,jf,jtm,jfile
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
          go to 10
c
30        if(if .lt. jf) go to 40
          if((itm .lt. jtm) .or. (itm .eq. 87)) go to 35
          write(6,960)ncard
960 format(" THE ITEM NUMBER IS OUT OF ORDER AROUND RECORD NUMBER",i6)
          write(6,931)jstate,jf,jtm,jfile
35  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
          go to 10
c
40        if(jtm .eq. 2) go to 50
          write(6,970)ncard
970       format(" THE ITEM SHOULD BE 2 IN RECORD NUMBER ",i6)
          write(6,931)jstate,jf,jtm,jfile
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
          go to 10
```

```
c
50          do 60 k=1,40
            if(ifilel(k) .ne. jfile(k)) go to 70
60          continue
c
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
            go to 10
c
70          write(6,980)ncard
980         format(" THE STATE NAME IS WRONG IN RECORD NUMBER ",i6)
            write(6,931) jstate,jf,jtm,jfile
  call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
            go to 10
c
c
500         write(6,990)ncard
990         format(" THERE WAS A READ ERROR ON RECORD NUMBER ",i6)
   call reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,jfile)
            go to 10
c
1000        write(6,1900)ncard
1900        format(" YOU REACHED THE EOF AFTER READING ",i6," RECORDS.")
c      *******************************************************
    call io ("close","file22")
    call io ("close","file30")
    call io ("detach","file22")
    call io ("detach","file30")
c      *******************************************************
       stop
end
c ********  END **********
```

SUBROUTINE NAME: RFSKIP_CHKREF

Author: Harold Johnson

Purpose of the program: **rfskip_chkref** checks to locate the record having reference number 1 and item number 2, which should be the first record in most reference files.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call rfskip_chkref (j)

Arguments: *j*—The record number in the reference file, which has reference number 1 and item number 2

Subroutines called: None

Common data referenced: None

Input files: ***refNM***

Output files: None

Arrays used: None

Called by: **chkref**

Error checking and reporting: The record number of the beginning record is always reported. If no correct first record is found, a message to that effect is sent to the user.

Constants: None

Program logic:

1. Each record is read to determine its reference number, ***ib***, and its item number, ***ic***.
2. These are compared with 1 and 2 until a match is found. This matching record is reported to the user.
3. If no match is found, a warning message is written to the user, the file is closed, and control returns to **chkref**.

```
c ******* SUBROUTINE RFSKIP_CHKREF *******
c      *
c      ************************************************************
c
         subroutine rfskip_chkref(j)
         data ione/"  1 "/,itwo/"2   "/
         j=1
1        read(30,900,end=100)ia,ib,ic
900      format(a2,a4,a1)
         if(ib .eq. ione .and. ic .eq. ic) go to 10
         j=j+1
         go to 1
10       write(6,910)j
910      format(" THE REFERENCE DATA BEGINS AT THE",i4,"TH RECORD")
         backspace 30
         return
c      ******************************************************
100 write(6,920)
920 format(" THERE WAS NO FIRST RECORD FOUND! what's wrong?")
rewind 30
return
end
```

SUBROUTINE NAME: CLEAN_CHKREF

Author: Harold Johnson
Purpose of the program: **clean_chkref** inserts blank characters in each word of the vector *ifile*.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call clean_chkref (ifile,nfile)
Arguments:
ifile – A vector
nfile – The number of elements in *ifile*

Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: *ifile*
Called by: **chkref**, **reup_chkref**
Error checking and reporting: None
Constants: None
Program logic:
1. The blank character is inserted into each word of *ifile* by a do loop.

```
c ******* SUBROUTINE CLEAN_CHKREF *******
         subroutine clean_chkref(ifile,nfile)
c      THIS ROUTINE PUTS BLANK WORDS INTO THE FILE IFILE.
c
c HJohnson February 16, 1977
c
         dimension ifile(nfile)
         data iblank/"    "/
         do 10 k=1,nfile
         ifile(k)=iblank
10       continue
         return
         end
c********* END CLEAN_CHKREF *********
```

SUBROUTINE NAME: REUP_CHKREF

Author: Harold Johnson
Purpose of the program: **reup_chkref** transfers the characters in *jfile* to *ifile*.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call reup_chkref (itype,ichar,item,-idim,if,itm,ifile,jf,jtm,jfile)
Arguments:
ifile–A vector containing the reference data in a refer-record
jfile–The same kind of vector as *ifile*
Subroutines called: **clean_chkref**
Common data referenced: None
Input files: None
Output files: None
Arrays used: *ifile(73), jfile(73)*
Called by: **chkref**
Error checking and reporting: None
Constants: None
Program logic:
1. **clean_chkref** is called to put blanks into *ifile*.
2. Characters in *jfile* are written into *ifile*.

```
c ******* SUBROUTINE REUP_CHKREF *******
  subroutine reup_chkref(itype,ichar,item,idim,if,itm,ifile,jf,jtm,j
\cfile)
  dimension itype(idim),ichar(idim),item(idim,10),ifile(73),jfile(73
&)
c H Johnson February 17, 1977
c
          if=jf
          itm=jtm
          nfile=73
          call clean_chkref(ifile,nfile)
c
          do 10 k=1,73
          ifile(k)=jfile(k)
10        continue
          return
c     ********************************************************
c
          end
c ******* END REUP_CHKREF *******
```

SUBROUTINE NAME: BLANKCHECK_CHKREF

Author: Harold Johnson
Purpose of the program: **blankcheck_chkref** checks to see if the information in the current record being checked is within the limits prescribed by matrix.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call blankcheck_chkref (ichar,jfile,-ncard)
Arguments:
ichar–The admissible length of the current reference item
jfile–The vector of characters obtained from the current reference record
ncard–The number of the record in the reference file currently being examined
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: *jfile(73)*
Called by: **checkitem_chkref**
Error checking and reporting: When *jfile* is too long, a message explaining the problem, giving the record number along with a printing of the file *jfile* and the number of the erroneous nonblank characters is sent to the user.
Constants: None
Program logic:
1. All characters after the *ichar*th are compared with the blank character. Discrepancies are reported.

```
c ******* SUBROUTINE BLANKCHECK_CHKREF.FORTRAN *******
          subroutine blankcheck_chkref(ichar,jfile,ncard)
c this routine checks to see if the file jfile(73) is contained
c within ichar spaces by seeing if the other spaces are blank
c
c H Johnson February 16, 1977
c
          dimension jfile(73)
          data iblank/"    "/
1         do 10 k=1,4
          i=ichar+k
          if(jfile(i) .ne. iblank) go to 20
10        continue
          return
20        write(6,900)ncard,i
900       format(" THE FILE IS TOO LONG ON RECORD NUMBER ",i6," BECAUSE
  CHARACTER ",i2," IS NOT BLANK")
          write(6,901)(jfile(k),k=1,73)
901       format(" THIS DATA IS ",73a1)
          return
c     ******************************************
c
          end
c *******END BLANKCHECK_CHKREF *******
```

SUBROUTINE NAME: TYPECHECK_CHKREF

Author: Harold Johnson
Purpose of the program: **typecheck_chkref** checks to see whether the information in the current reference record represents an integer or floating-point number when that type is indicated by its item number.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call typecheck_chref (itype,ifile,ncard)
Arguments:
itype– One of the type of 1 or 2
ifile– A vector of characters obtained from a record of the reference file being checked
ncard– The number of the current record being checked
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **checkitem_chkref**
Error checking and reporting: When *ifile* is wrong, this is reported to the user, along with the record number and copy of *ifile*.
Constants: None
Program logic:
1. If the item number is 1, each character in *ifile* is checked to determine if it is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or blank.
2. If the item number is 2, each character is compared with these same integers and then with a period. The program checks that exactly one period occurs.
3. Any discrepancies are reported to the user.

```
c ******* SUBROUTINE TYPECHECK_CHKREF.FORTRAN *******
          subroutine typecheck_chkref(itype,ifile,ncard)
          dimension ifile(73),number(12)
c
c this subroutine checks whether the type indicated by itype
c corresponds to what is found in ifile.
c H Johnson February 16, 1977
```

```
c
          data number/"0    ","1    ","2    ","3    ","4    ","5    ","6
\c","7
&","8    ","9    ","    ",".    "/
1         if(itype .eq. 2) go to 20
c
c     WHEN ITYPE IS 1 WE TEST TO SEE IF IFILE CONTAINS ONLY INTEGERS.
          do 10 j=1,20
          do 15 k=1,11
          if(ifile(j) .eq. number(k)) go to 10
15        continue
          go to 500
10        continue
          return
c
c     WHEN ITYPE IS 2 WE TEST TO SEE IF IFILE IS A REAL NUMBER.
20        continue
          iflag=0
          do 30 j=1,20
          do 35 k=1,11
          if(ifile(j) .eq. number(k)) go to 30
35        continue
          if(ifile(j) .ne. number(12)) go to 510
          iflag = iflag+1
          if(iflag .ne. 1) go to 510
30        continue
          if(iflag .ne. 1) go to 510
          return
c
500       write(6,920)ncard
920 format(" THERE IS SUPPOSED TO BE AN INTEGER IN RECORD NUMBER ",i6
\c)
          write(6,921)ifile
921       format(" THIS DATA IS ",73a1)
          return
c
510       write(6,930)ncard
930       format(" THERE IS SUPPOSED TO BE A REAL NUMBER IN RECORD ",
\ci6)
          write(6,921)ifile
          return
c     ****************************************************
c
          end
c ******* END TYPECHECK_CHKREF *******
```

SUBROUTINE NAME: LOCAT1_CHKREF

Author: Harold Johnson

Purpose of the program: **locat1_chkref** determines the line and column of matrix in which a given *item* number occurs by using the matrix, *item*, which was constructed from matrix.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call locat1_chkref (jtm,idim,item,-line,kolumn)

Arguments:

jtm – An item number of the current reference record

idim – The number of rows in *item*

line–The line in *item* where *jtm* is located
kolumn–The column in *item* where *jtm* is located
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: *item(idim,10)*
Called by: **checkitem_chkref**
Error checking and reporting: Done by **checkitem_chkref** when *line* = 0
Constants: None
Program logic:
1. *jtm* is compared with each element of *item* using a do loop, checking by columns first, since most items occur in the first column.

```
c ******* SUBROUTINE LOCAT1_CHKREF *******
         subroutine locat1_chkref(jtm,idim,item,line,kolumn)
         dimension item(idim,10)
         line=0
         kolumn=0
         do 10 k=1,10
         do 10 j=1,idim
         if(jtm .eq. item(j,k)) go to 20
10       continue
         return
20       line=j
         kolumn=k
         return
c    ****************************************************
c
         end
c ******* END LOCAT1_CHKREF *******
```

SUBROUTINE NAME: CHECKITEM_CHKREF

Author: Harold Johnson
Purpose of the program: **checkitem_chkref** checks whether *jfile* is integer if *jtm* is 1 or floating point when *jtm* is 2. It checks whether the number of nonblank characters in *jfile* is at most *ichar(lin)*.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call checkitem_chkref (idim,ncard,-item,jfile,itype,ichar)
Arguments:
idim–The number of lines in matrix
ncard–The number of records that have been read in the reference file
jtm–The item number on the current record being checked
jfile–The reference data in the current record, a vector
itype–The vector of types from matrix
ichar–The vector of maximum allowable lengths from matrix
Subroutines called: **locat1_chkref**, **typecheck_chkref**, **blankcheck_chkref**
Common data referenced: None
Input files: None
Output files: None
Arrays used: *jfile(73)*, *item(idim,10)*, *itype(idim)*, *ichar(idim)*
Called by: **chkref**
Error checking and reporting: When **locat1_chkref** cannot match *jtm* with any item number in *item*, a message is written along with the file *jfile* to the user.
Constants: None
Program logic:
1. **locat1_chkref** is called to determine which line of matrix contains *jtm*.
2. If *itype(line)* is 1 or 2, **typecheck_chkref** is called to check whether *jfile* is a character representation of integer or floating-point data.
3. **blankcheck_chkref** is called to check whether *jfile* contains at most *ichar(line)* nonblank characters.

```
c ******* SUBROUTINE CHECKITEM_CHKREF *******
  subroutine checkitem_chkref(idim,ncard,item,jtm,jfile,itype,ichar)
        dimension jfile(73),item(idim,10),itype(idim),ichar(idim)
c     THIS CHECKS TWO THINGS ABOUT JFILE.
c     DOES THE FILE CONTAIN INTEGERS OR FLOATING-
c     POINT NUMBERS WHEN JTM IS 1 OR 2?
c
c     IS THE FILE CONTAINED IN THE LIMITS SET BY MATRIX FOR THIS ITEM?
c
c H Johnson February 16, 1977
1       call locatl_chkref(jtm,idim,item,line,kolumn)
c
c     THIS SUBROUTINE LOCATE FINDS THE LINE AND COLUMN OF MATRIX
c     WHICH CONTAINS THE ITEM = JTM.
c
        if(line .gt. 0) go to 2
        write(6,900)ncard
900   format(" THE MATRIX FILE DOES NOT CONTAIN THE ITEM ON RECORD ",i6
\c)
        write(6,901)jfile
901     format(" THIS DATA IS ",73al)
        return
c
c     WHEN THE TYPE OF THE ITEM IS .GT. 2 JFILE CAN BE ANY CHARACTER.
2       if(itype(line) .gt. 2 .or. itype(line) .eq. 0)go to 50
        ity=itype(line)
5       call typecheck_chkref(ity,jfile,ncard)
c     THIS SUBROUTINE CHECKS WHETHER THE TYPE INDICATED BY ITYPE(LINE)
c     CORRESPONDS TO WHAT IS FOUND IN JFILE.
c
50      ich=ichar(line)
        call blankcheck_chkref(ich,jfile,ncard)
c     THIS CHECKS TO SEE IF JFILE IS CONTAINED IN
c     ICHAR(LINE) SPACES.
c
        return
c     *****************************************************
c
        end
c *******END CHECKITEM_CHKREF *******
```

SUBROUTINE NAME: SETUP_CHKREF

Author: Harold Johnson

Purpose of the program: **setup_chkref** reads the file **matrix** to construct vectors *itype* and *ichar* and matrix *item* that indicates for each item in the reference file its type, its allocated space, and its item number.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call setup_chkref (itype, ichar,item,idim)

Arguments:

itype–The vector of length **idim** whose *k*th entry is the type of the *k*th kind of record in **matrix** (integer, floating point, or alphanumeric)

ichar–The vector of length *idim* whose *k*th entry is the maximum allowable length of the *k*th kind of record described by **matrix** *item*–The *idim* by 10 matrix giving the item numbers in **matrix** allocated to the various kinds of records

idim–The number of records in **matrix**

Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **chkref**
Error checking and reporting: When **matrix** does not contain exactly the number of records indicated by the value of *idim*, an error message is sent to the user.
Constants: None
Program logic:

1. Records in **matrix** are read into *a1, a2, itype(j), b1, b2, ichar(j), c1, c2, (item(j,k),K=1,10)*.
2. When the EOF of **matrix** is sensed, the number of read records is compared with *idim* to see whether they are equal.

```
c ***** SUBROUTINE SETUP_CHKREF *******
          subroutine setup_chkref(itype,ichar,item,idim)
          dimension itype(idim),ichar(idim),item(idim,10)
          j=0
1         j=j+1
read(22,900,end=100)a1,a2,itype(j),b1,b2,ichar(j),c1,c2,c3,(item
\c(j,k),
&k=1,10)
900       format(2a4,i2,2a4,i6,2a4,a1,10i3)
          go to 1
100       if(j-1.eq. idim)go to 200
          write(6,910)j,idim
910       format(" J = ",i3," BUT IDIM = ",i3)
200       return
c     ***************************************************
c
c
      end
c ****** END SETUP_CHKREF *****
```

EXEC_COM NAME: GEOFMT.EC

Authors: Kevin W. Laurent, Larry C. Harms, and Pearl Porter
Purpose of the program: **geofmt.ec** executes a series of command lines and routines without user intervention.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec geofmt *page lines nbipp*
Arguments:

page–Number of lines on the page

lines–Number of lines needed at bottom of page to write a complete reference

nbipp–Used to designate whether or not the proportional-space printer is to be used

Subroutines called: **geofmt**, **geofmt.qedx**, **geofmt2.qedx**, **geofmt3.qedx**, **geofmt4.qedx** are all executed.
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: None
Error checking and reporting: The user must enter the arguments when executing **geofmt.ec** or she will receive a message stating the current buffer and level and the commands that have not been executed. This same message will appear if the user asks for too many references to be processed at one time. We recommend that 550 references be the maximum number processed at one time.
Constants: None
Program logic:

1. A command is given to turn off the COMMAND_LINE to prevent the commands from being written out.
2. The file name given by the user is attached to *file10.*
3. The output from **geofmt**, *geofmt.data*, is attached to *file11.*
4. The fortran program **geofmt** is executed.

5. *file10* and *file11* are detached.
6. The user is asked if she needs to edit.
7. If the third argument is *nbipp*, subroutine **geofmta.qedx** is executed. Otherwise, subroutine **geofmt.qedx** is executed.
8. DL GEOFMT.RUNOFF. This will delete the old copy (if one exists).
9. FO GEOFMT. RUNOFF; RF GEOFMT; FO is a command to direct **geofmt.runoff** to a segment, and RF will run off **geofmt**; CO directs output back to the terminal.
10. The user is queried whether she wants 7 columns or not. This is a combination of 4 on the first page and 3 on the next page and so forth. If she responds yes, **geofmt2.qedx** is executed. If she answers no, **geofmt3.qedx** is executed, and every page will have four columns.
11. DL GEOFMT.COLUMNS. This will delete the old copy (if one exists).
12. If the third argument is *nbipp*, then **embed_tabs** is executed. Otherwise, the next statement is executed.
13. Four segments called *overlay1*, *overlay2*, *overlay3*, and *overlay4* are combined, using the overlay command, into one segment called *geofmt.columns*.
14. Eight segments created during this process are deleted.
15. Quit.

```
&
&          /* The geofmt exec_com is used to perform steps necessary to
&                 create a columnar print of input reference data. */
&
&command_line off
&
&          /* run reformat program */
&
io attach file10 vfile_ [response "ENTER FILE NAME:"]
io attach file11 vfile_ geofmt.data
geofmt
&
&
io detach file10
io detach file11
&
&          /* Edit geofmt.data */
&
&if [query "Do you need to edit?"]
&then
&else &goto nextstep
&input_line off
&attach
qx
r geofmt.data
estty -modes ll80
eioa_ "Edit.^/Enter ""q"" to exit editor."
&detach
&
&label nextstep
&
stty -modes ll132
&
&          /* create runoff segment */
&
&if [equal &3 "nbipp"]
&then qx geofmta &1 &2
```

```
&else qx geofmt &1 &2
&
&          /* put runoff output into segment */
&
dl geofmt.runout -bf
fo geofmt.runout;rf geofmt;co
&
&          /* break output into 4 files (columns) */
&
&if [query "DO YOU WANT 7 COLUMNS?"]
&then qx geofmt2 &1
&else qx geofmt3 &1
&
&          /* create columnized output */
&
dl geofmt.columns -bf
&if [equal &3 "nbipp"]
&then embed_tabs &1
&else do "fo geofmt.columns;overlay overlay1 overlay2 -in 34 overlay3 -in
&67 overlay4 -in 100 -pl &1;co"
&
&      for linolex -- qx geofmt4 &1
&
&          /* delete intermediate segments */
&
dl geofmt.(runoff runout data)  -bf
dl overlay(1 2 3 4) -bf
&quit
```

SUBROUTINE NAME: GEOFMT.QEDX

Authors: Kevin W. Laurent, Larry C. Harms, and Pearl Porter

Purpose of the program: **geofmt.qedx**, an edit routine, creates a RUNOFF segment using *geofmt.data*, which was created during the execution of **geofmt**.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: None

Arguments:

&1 – Passed from the exec_com and contains the page length (usually 84 or 140)

&2 – Contains the number of lines needed at the bottom of the page to type a complete reference

Subroutines called: None

Common data referenced: None

Input files: *geofmt.data*

Output files: *geofmt.runoff*

Arrays used: None

Called by: **geofmt.ec**

Error checking and reporting: Located in **geofmt.ec**

Constants: None

Program logic:

1. The two arguments used when executing **geofmt.ec** are read into a buffer called ***args***, and the first argument (which is the page length) is moved to a buffer called *lines*. The second argument (number of lines needed at bottom of page for printing a complete reference) is moved to a buffer called *need*. These two argements are used with .PL and .NE, respectively, as runoff commands. The initialization routine puts RUNOFF commands into buffer ***0***.
2. Segment *geofmt.data* is read into a buffer called *file*.
3. A special character {, a brace that is made by depressing the shift key and left bracket key simultaneously, is appended to the end of *geofmt.data* as an end of file indicator.
4. The RUNOFF commands and one line of data at a time is moved from buffer ***file to buffer 0.***
5. Step 4 is repeated until the special end of file indicator is detected, at which time it is deleted.
6. Write *geofmt.runoff.*
7. Quit to exit from text editor.

```
b(main)
$a
b(file)
1m(input)
b0
$a
.un 7
.ne \c\b(need)
\c\b(input)
\c\f
s/{/{/
d
w geofmt.runoff
q
\f
b(loop)
$a
\c\b(main)
\c\b(loop)
\f
b(args)
1m(lines)
1m(need)
b(lines)
1s/\c
//
b(need)
1s/\c
//
b0
$a
.pl \b(lines)
.ll 30
.ma 0
.na
.in 7
\f
b(file)
r geofmt.data
$a{\f
b0
\b(loop)
```

SUBROUTINE NAME: GEOFMT

Authors: Kevin W. Laurent, Larry C. Harms, and Pearl Porter

Purpose of the program: **geofmt** reads the reference file, extracts selected data, arranges it in a predetermined order, and writes it out as a string of data, *geofmt.data*.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: geofmt

Arguments: None

Subroutines called: None

Common data referenced: None

Input files: refNM (*NM*, two-digit FIPS State code) used on unit 10

Output files: geofmt.data format(i3,". ",396a1) used on unit 11

Arrays used: iordr(34), istor(34), icoma(34), nuse(34), icode(34,60), *iout(396)*

Called by: **geofmt.ec**

Error checking and reporting: Located in **geofmt.ec**

Constants: None

Program logic:

1. Files are attached and opened in **geofmt.ec**.
2. The user is prompted for the State number, starting reference number, and ending reference number.
3. The arrays are loaded with blanks except ***nuse***, which is loaded with zeros.
4. The reference file is read until the beginning reference number given by the user, ***ibgin***, matches the record just read, ***iref***. Load ***iref*** into ***jref***.
5. The item number is matched against the array ***iordr(i)***.
   If no match is found, go to step 4.
   If item number is 87, load ***flag87*** with an 87.
   If equal, ***icard*** is loaded into ***icode(i)***, ***item*** is loaded into ***istor(i)***, and a 1 is stored in ***nuse(i)***.
6. Continue steps 4 and 5 until ***jref*** no longer equals ***iref***.
7. If ***nuse(i)***, where ***i*** equals 13, 15, . . ., 31, equals 1, load ***icoma(i)*** with a 1. Add 1 to ***ipnct***. Load ***kk*** with ***i***.
   This routine checks for scales and commas can be inserted later. ***ipnct*** has the number of commas that will have to be inserted, and ***kk*** contains the number as specified by ***ordr*** of the last scale. This field, ***kk***, will be checked later to determine whether or not a period and an extra space are required for output.
8. The number of characters and spaces in ***icode(i,j)*** is loaded into ***isave***.
9. If the current record is a scale, check ***ipnct***.
   If ***ipnct*** = 0, go to step 12.
   If ***flag87*** = 87, go to step 10. If the reference has an ITEM 87, a comma will be placed after the scale rather than a period, as more data will follow the scale.
   If ***ipnct*** = 1, to to step 11. This indicates that there is only one scale field and that no punctuation will be needed.
10. If ***icoma(i)*** = 0, go to step 12. Add 1 to ***isave***. Load ***icode(i,isave)*** with a comma.
11. Subtract 1 from ***ipnct***.
12. If the current record is not the year, ITEM 8, go to step 13. Load ***icode(i,isave)*** with a comma.
    Go to step 16.
13. If the current record is not the publisher, ITEM 17, go to step 14.
    If the last character of this record is a period, insert an extra blank after the period, add 1 to ***isave***, and then go to step 16.
14. If the current record is not the series, ITEM 23, go to the step 14A.
    If the current record is ITEM 23, check ***istor(11)*** = 60, which means that the series is continued on another record.
    Otherwise, put an extra space after ITEM 23 data.
    Go to step 16.
    If ***istor(i)*** is not = 60, go to 15. Put extra space after ITEM 60 data.
    Go to step 16.
15. If the current record is not a scale (item 18–22, 61–65), go to step 16. If ***i*** = ***kk***, which means this is the last scale in the reference, load a period at the end of the field.
    Load ***iout*** with 1:
    Load ***iout*** with the scale and necessary commas separating the field.
    Go to step 17.
16. ***iout(istart)*** = ***icode(i,jj)*** where ***jj*** = 1, ***save***. This routine will load ***iout*** with the current record using ***isave*** as the counter to move the exact number of characters and spaces for that record.
17. If ***i*** = ***kk***, meaning this record is the last scale, ***istart*** = ***istart*** + 1.
    This will insert another space after the scale, giving a total of 2 spaces between the scale and next record.
18. If the value of ***istor(i)*** is not equal to 86, go to step 19.
    If ***istor(33)*** equals blank, and ***istor(34)*** equals blank, go to 20.
    istart = ***istart*** + 1 will give a total of 2 spaces between the ITEM 86 record and the next record.
19. If ***istor(i)*** is not equal to ITEM 35, go to step 20. If ***istor(34)*** equals blank, go to step 20.
    istart = ***istart*** + 1. This will insert an extra space after the ITEM 35 record has been written to ***iout***.
20. ***istart*** = ***istart*** + 1, ***iout(istart)*** = blank space. This will put a space between each record written to ***iout***. Repeat steps 8–20 until ***i*** = 34, then go to next step.
21. If ***flag87*** = 87, continue reading the records, steps 8–20, until a new reference number is found. Load ***flag87*** = 0. Write ***geofmt.data*** on unit 11 using ***jref***, (***iout(i)***, ***i*** = 1, ***istart***), and format (i3,-". ",396a1).
    If ***jref*** = ***iendref***, go to step 24. (This means the reference number of the record just written is the last record to be processed. ***iendref*** is the ending reference number given by the user.)
    If ***iend*** = 1, go to step 24. This is the last record on the file.
22. Load ***jref*** with ***iref***. ***iref*** is the reference number of the last record read before processing the current reference data.
23. Initialize the arrays by loading them with blanks. Go to step 5.
24. Stop.

```
c    GEOFMT.FORTRAN
c          This program reads the reference file, extracts selected records
c       as determined by iordr, stores them in icode according to iordr
c       and writes it out as a string of data.
c
c    General outline of program written by Larry Harms of CCD.
c    Written in detail by Pearl B. Porter, April, 1978
c
        dimension iordr(34),istor(34),nuse(34),icoma(34)
        character*1 icard(65),icode(34,65),iout(396)
        data iordr/3,4,5,8,9,10,11,37,17,23,60,39,18,40,19,
     &         41,20,42,21,43,22,66,61,67,62,68,63,69,
     &         64,70,65,86,35,34/

        write (6,10)
 10     format(" TYPE IN STATE NUMBER")
        read (5,20)jsta
 20     format(i2)
        write (6,22)
c    ATTENTION For Calif., ref number will be 4 digits.
 22     format (" WHAT IS YOUR STARTING REFERENCE NUMBER? (use 3 digits)")
        read (5,24) ibgin
c    ATTENTION  Change (i3) to (i4) for California.
 24     format (i3)
        write (6,26)
c    ATTENTION  For Calif., ref number will be 4 digits.
 26     format (" WHAT IS YOUR ENDING REFERENCE NUMBER? (use 3 digits)")
        read (5,24) iendref
c
c          Initialize arrays to blanks.
c
        do 30 i=1,65
 30     icard(i)=" "
        do 40 j=1,34
        do 40 i=1,65
 40     icode(j,i)=" "
        do 50 k=1,34
 50     icoma(k)=" "
        do 60 i=1,34
        nuse(i)=0
 60     istor(i)=" "
        do 70 i=1,396
 70     iout(i) =" "
        ipnct=0
        flag87=0
c
c          Read reference file until the current record equals the
c          beginning reference number given by the user.
c
 80     read (10,110,end=900) ista,iref,item,icard
        if (iref .ne. ibgin) go to 80
        jref = iref
        go to 120
c
```

```
c        jref contains the reference number of the data being processed.
c
 100   read(10,110,end=155) ista,iref,item,icard
c   ATTENTION  For Calif., change  i3  to  i4.
 110   format(i2,i3,i2,65a1)
       if (jref .ne. iref) go to 160
       if (flag87 .eq. 87) go to 160
c
c        Does record contain an item code = to an item code in iordr
c
 120   do 130 i=1,34
       if (item .eq. iordr(i)) go to 140
       if (item .eq. 87) flag87 = 87
 130   continue
       go to 100
c
c        Load icode with icard as determined by iordr, store item in istor
c        and turn on nuse(i) which indicates there's data for this
c        particular item.
c
 140   do 150 j=1,65
 150   icode(i,j)=icard(j)
       istor(i)=item
       nuse(i)=1
       go to 100
c
 155   iend = 1
c
c        Check scales to determine how many commas will be needed
c        when written out to file11 (geofmt.data).
c
 160   do 170 i=13,31,2
       if (nuse(i) .eq. 0) go to 170
c        If nuse(i) = 0, there's no data for this record.
       icoma(i)=1
       kk=i
       ipnct = ipnct+1
 170   continue
c
c        Store in isave the total number of characters and
c        significant spaces contained in the record.
c
       if (flag87 .eq. 87) go to 175
       istart = 0
 175   do 500 i=1,34
       isave = 0
       if (nuse(i) .eq. 0) go to 500
       do 180 j=1,65
       if (icode(i,j) .eq. " ") go to 180
       isave=j
 180   continue
c
c        Check for scales and if there's more than one scale,
c        insert commas after the scales.
```

```
c
      if ((istor(i) .ge. 18 .and. istor(i) .le. 22) .or.
     &    (istor(i) .ge. 61 .and. istor(i) .le. 65)) go to 190
      go to 220
 190  if (ipnct .eq. 0) go to 220
      if (flag87 .eq. 87) go to 200
      if (ipnct .eq. 1) go to 210
c
 200  if (icoma(i) .eq. 0) go to 220
      isave=isave+1
      icode(i,isave)=","
 210  ipnct=ipnct-1
c
c       If item 8 (year), put comma after the year.
c
 220  if (istor(i) .ne. 8) go to 230
      isave=isave+1
      icode(i,isave)=","
      go to 400
c
c       If the publisher has a period at end of field,
c       two spaces must follow the period.
c
 230  if (istor(i) .ne. 17) go to 240
      if (icode(i,isave) .eq. ".") isave=isave+1
      go to 400
c
c       There must be 2 spaces after the series.
c
 240  if (istor(i) .ne. 23) go to 245
      if (istor(11) .eq. 60) go to 245
      isave=isave+1
      go to 400
c
 245  if (istor(i) .ne. 60) go to 250
      isave = isave+1
      go to 400
c
c       When working with scales, precede the scales with "1:"
c
 250  if ((istor(i) .ge. 18 .and. istor(i) .le. 22) .or.
     &   (istor(i) .ge. 61 .and. istor(i) .le. 65)) go to 260
      go to 400
 260  if (i .ne. kk) go to 270
      ipnct = 0
      if (flag87 .eq. 87) go to 270
      isave = isave+1
      icode(i,isave)="."
 270  iout(istart+1)= "1"
      iout(istart+2)= ":"
      istart = istart+2
      ikorp = 0
      if (isave .le. 3)go to 400
      if (icode(i,isave) .eq. "," .or. icode(i,isave) .eq. ".")
     &     ikorp = 1
```

```
      ithi = 3
      if (ikorp .eq. 1) ithi = 4
      if (isave .ne. 9) go to 280
      ifir = 2
      isec = 3
      go to 330
c
c       Depending on the size of the scale field, there will be
c       from 1 to 3 moves to load the data in the output.
c       ithi (3rd load) will contain a 3 unless the last
c       position of the scale is a period or comma, then it
c       will contain 4.  isec (2nd) and ifir (1st) will
c       be loaded according to the number of digits in the
c       scales.  A comma is loaded after ifir and isec.
c
 280  if (isave .ne. 8) go to 290
      isec = 3
      ifir = 2
      if (ikorp .eq. 1) ifir = 1
      go to 330
c
 290  if (isave .ne. 7) go to 300
      isec = 3
      ifir = 1
      if (ikorp .eq. 1) ifir = 0
      go to 330
c
 300  if (isave .ne. 6) go to 310
      ifir = 0
      isec = 3
      if (ikorp .eq. 1) isec = 2
      go to 330
c
 310  if (isave .ne. 5) go to 320
      ifir = 0
      isec = 2
      if (ikorp .eq. 1) isec = 1
      go to 330
c
 320  ifir = 0
      isec = 1
      if (ikorp .eq. 1) isec = 0
c
c       Load the first (ifir) set of digits followed by comma.
c
 330  ipos = 1
      isecmv = isec
      if (ifir .eq. 0) go to 350
      do 340 jj=ipos,ifir
      istart = istart+1
 340  iout(istart)=icode(i,jj)
      iout(istart+1)=","
      istart = istart+1
      ipos = ifir+1
      isec = isec+ifir
```

```
c
c          Load the second (isec) set of digits followed by comma.
c
 350    if (isec .eq. 0) go to 370
        do 360 jj=ipos,isec
        istart = istart+1
 360    iout(istart)=icode(i,jj)
        iout(istart+1)=","
        istart = istart+1
        ipos = ipos+isecmv
        ithi = ithi+isec
c
c          Load third (ithi) set of digits.
c
 370    do 380 jj=ipos,ithi
        istart = istart+1
 380    iout(istart)=icode(i,jj)
        go to 425
c
c          Load iout with icode depending on isave.
c
 400    do 420 jj=1,isave
        istart = istart+1
 420    iout(istart) = icode(i,jj)
 425    if (i .eq. kk) istart = istart+1
c
c          If any data follows "Also other maps",
c          put 2 spaces after "Also other maps".
        if (istor(i) .ne. 86) go to 430
        if ((istor(33) .eq. " ") .and. (istor(34) .eq. " ")) go to 450
        istart = istart+1
        go to 450
c
 430    if (istor(i) .ne. 35) go to 450
        if (istor(34) .eq. " ") go to 450
        istart = istart+1
c
c          Increase istart by 1 so a space appears between records of data.
c
 450    istart = istart+1
        iout(istart) = " "
 500    continue
c
        if (flag87 .ne. 87) go to 525
        if (jref .ne. iref) go to 510
        if (iend .ne. 1) go to 565
 510    istart = istart-1
        if (iout(istart) .eq. ",") iout(istart) = "."
        if (iout(istart) .ne. " ") go to 525
        go to 510
c
c
c          Write the string of data to file11 (geofmt.data)
```

```
c
 525    write (11,550) jref,(iout(i),i=1,istart)
c   ATTENTION  For Calif., change  i3  to  i4.
 550    format(i3,". "396a1)
        istart = 0
c
c          iendref is the ending reference number given by the user.
        if (jref .eq. iendref) go to 1000
        if (iend .eq. 1) go to 1000
c
        jref = iref
c
c          Initialize arrays before processing next reference.
c
        do 560 i=1,396
 560    iout(i) = " "
 565    do 570 j=1,34
        do 570 i=1,65
 570    icode(j,i) = " "
        do 590 i=1,34
        icoma(i) = " "
        nuse(i) = 0
        istor(i) = " "
 590    continue
        flag87 = 0
        go to 120
c
 900    write (6,905)
 905    format (" THE BEGINNING RECORD WAS NOT FOUND")
c
 1000       stop
        end
```

SUBROUTINE NAME: GEOFMT2.QEDX

Authors: Kevin W. Laurent, Larry C. Harms, and Pearl Porter
Purpose of the program: **geofmt2.qedx** formats the file ***geofmt.runout*** into alternating 4 and 3 columnar output acceptable for use by the overlay command to create a columnized output segment.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: qx geofmt2 *&1*
Arguments: &1–It contains the page length (84 or 140) and was passed from the exec_com, **geofmt.ec**.
Subroutines called: None
Common data referenced: None
Input files: ***geofmt.runout***
Output files: ***overlay1***, ***overlay2***, ***overlay3***, ***overlay4***
Arrays used: None
Called by: Executed by **geofmt.ec**
Error checking and reporting: **geofmt.ec**
Program logic:
1. The initialization routine creates four segments: ***overlay1*** to ***overlay4***. A segment, ***geofmt.runout***, is read into buffer, ***file***. A special character is appended to the segment as an end of file indicator.
2. The loop macro will move a specified number of lines (determined by the page length argument when **geofmt.ec** was executed) into each of the four overlay segments. Then the specified number of lines are moved into ***overlay1***, ***overlay2***, and ***overlay3***. This loop alternates in moving data to all four segments, and then three segments, until the end of file indicator is read.
3. All four segments are written.
4. Quit the text editor.

```
b(4col)
$a
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay1
1,$w overlay1
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay2
1,$w overlay2
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay3
1,$w overlay3
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay4
1,$w overlay4
\c\b(3col)
\f
b(3col)
$a
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay1
1,$w overlay1
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay2
1,$w overlay2
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay3
1,$w overlay3
\c\b(args)+1,$d
1,$s/^.*$//
0r overlay4
1,$w overlay4
\c\b(4col)
\f
b(test)
$a
b(file)
1s/^{/{/
q
\f
b(args)
1s/\c
```

```
//
b0
ecr overlay(1 2 3 4)
b(file)
r geofmt.runout
$a{\f
b0
\b(4col)
```

SUBROUTINE NAME: GEOFMT3.QEDX

Authors: Kevin W. Laurent, Larry C. Harms, and Pearl Porter

Purpose of the program: **geofmt3.qedx** formats ***geofmt.runout*** into four files acceptable for use by the OVERLAY command to create a columnized output segment.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: qx geofmt3 *&1*

Arguments: &1–Passed from the exec_com, **geofmt.-ec**, and contains the page length (84 or 140)

Subroutines called: None

Common data referenced: None

Input files: ***geofmt.runout***

Output files: ***overlay1***, ***overlay2***, ***overlay3***, ***overlay4***

Arrays used: None

Called by: Executed by **geofmt.ec**

Error checking and reporting: Located in **geofmt.ec**

Constants: None

Program logic:

1. The initialization routine creates four segments: ***overlay1*** to ***overlay4***. A segment, ***geofmt.runout***, is read into buffer, ***file***, and a special character {, a brace, which is made by depressing the shift key and left bracket key simultaneously, is appended as an EOF indicator.
2. The loop macro moves a predetermined number of lines (page length argument used in **geofmt.ec**) first to ***overlay1***, ***overlay2***, and so forth, until the end of file indicator is read.
3. Write the four output segments.
4. Quit the text editor.

```
b(loop)
$a
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay1
1,$w overlay1
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay2
1,$w overlay2
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay3
1,$w overlay3
\c\b(test)
1,\c\b(args)m(input)
b(input)
0r overlay4
1,$w overlay4
\c\b(loop)
\f
```

```
b(test)
$a
b(file)
1s/^{/{/
q
\f
b(args)
1s/\c
//
b0
ecr overlay(1 2 3 4)
b(file)
r geofmt.runout
$a{\f
b0
\b(loop)
```

SUBROUTINE NAME: GEOFMTA.QEDX

Authors: Kevin Laurent and Pearl Porter
Purpose of the program: **geofmta.qedx**, an edit routine, creates a RUNOFF segment using the file *geofmt.-data*, which was created during the execution of **geofmt**.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: qx geofmta *&1 &2*
Arguments:
&1 – It contains the page length (usually 84 or 140) and was passed from **geofmt.ec.**
&2 – It contains the number of lines needed at the bottom of the page to type a complete reference and was passed from **geofmt.ec.**
Subroutines called: None
Common data referenced: None
Input files: geofmt.data
Output files: geofmt.runoff
Arrays used: None
Called by: **geofmt.ec**
Error checking and reporting: Located in **geofmt.ec**
Constants: None
Program logic:

1. This **qedx** routine is executed if the third argument of **geofmt.ec** is *nbipp*. It is very similar to **geofmt.qedx**, except the .UN and .IN commands for RUNOFF were increased to allow for proportional-space printing. The line length was increased from 30 to 42.
2. The two arguments used when executing **geofmt.ec** are read into a buffer called *args*, and the first argument (which is the page length) is moved to a buffer called *lines*. The second argument (number of lines needed at the bottom of the page for printing a complete reference) is moved to a buffer called *need*. These two arguments are used with .PL and .NE respectively as RUNOFF commands.
3. The initialization routine puts RUNOFF commands into buffer *0*.
4. Segment *geofmt.data* is read into a buffer called *file*.
5. A special character {, a brace, which is made by depressing the shift key and left bracket key simultaneously, is appended to the end of *geofmt.data* as an end-of-file indicator.
6. The RUNOFF commands, and one line of data at a time is moved a from buffer *file* to buffer *0*.
7. Step 6 is repeated until the special end-of-file indicator is detected, at which time it is deleted.
8. Write *geofmt.runoff.*
9. Exit from text editor.

```
b(main)
$a
b(file)
1m(input)
b0
$a
.un 11
.ne \c\b(need)
```

```
\c\b(input)
\c\f
s/{/{/
d
w geofmt.runoff
q
\f
b(loop)
$a
\c\b(main)
\c\b(loop)
\f
b(args)
1m(lines)
1m(need)
b(lines)
1s/\c
//
b(need)
1s/\c
//
b0
$a
.pl \b(lines)
.ll 42
.ma 0
.na
.in 11
\f
b(file)
r geofmt.data
$a{\f
b0
\b(loop)
```

SUBROUTINE NAME: EMBED_TABS

Author: Kevin Laurent

Purpose of the program: **embed_tabs**, a PL/1 program, embeds tab commands between the columns of the file ***geofmt.columns*** to produce a columnar proportional spaced printout.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: embed_tabs *&1*

Arguments: &1 – It contains the page length (usually 84 or 140) and is passed from **geofmt.ec**.

Subroutines called: None

Common data referenced: None

Input files: ***overlay1***, ***overlay2***, ***overlay3***, ***overlay4***

Output files: ***geofmt.columns***

Arrays used: None

Called by: None

Error checking and reporting: If an error occurs the error code is passed to the command processor, and a Multics system message is printed.

Constants: None

Program logic:

1. The program asks for the number of columns and the column width.
2. Tab positions are calculated depending on the column width given by the user.
3. The program writes a SET command to ***geofmt.columns*** to be used by the NBI. Left margin is set to 1; top margin is set to 0; a code, ***J-1***, is passed that specifies proportional spaced printing; line length is set to 132; page length and text length are set according to the first argument given when executing **geofmt.ec**.
4. The four overlay segments are concatenated and written to ***geofmt.columns***.
5. If an entire line is blank, only the first tab will be written, in order to act as a line feed.
6. Files are closed and detached.

```
embed_tabs: et: proc (page_len);                                 /* embed
\ctab commands in nbi proportional spaced printer stream */

/* Written by KLaurent,USGS,CCD,BSAP 6/27/78 */

/* The embed_tabs program accepts input from the overlay files and ad
\cds
   the embedded commands between line segments to tab the columns. */

dcl  page_len char (*);                                          /* page l
\cength parameter */
dcl  ioa_$ioa_switch entry options (variable);                   /* for wr
\citing our concatenated lines to geofmt.columns */
dcl  iox_$attach_name entry (char (*), ptr, char (*), ptr, fixed bin
\c(35)); /* for all attachments */
dcl  iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35)
\c);
dcl  iox_$get_line entry (ptr, ptr, fixed bin (21), fixed bin (21), f
\cixed bin (35));
dcl  iox_$close entry (ptr, fixed bin (35));
dcl  iox_$detach_iocb entry (ptr, fixed bin (35));
dcl  com_err_ entry options (variable);                          /* com_er
\cr_ will interpret all the errors and print a standard message */
dcl  error_table_$end_of_info external fixed bin (35);           /* use fo
\cr end of file check */
dcl  command_query_ entry options (variable);                    /* use to
\c ask for ncols and col_width */
dcl  continue_to_signal_ entry (fixed bin (35));                 /* use to
\c pass along the error code to the command processor */

dcl 1 query_info,                                                /* for pa
\cssing info to command_query_ */
    2 vers fixed bin init (2),                                   /* versio
\cn of structure */
    2 yes_or_no_sw bit (1) unal init ("0"b),
    2 suppress_name bit (1) unal init ("0"b),
    2 status_code fixed bin (35) init (0),                       /* not us
\ced here */
    2 query_code fixed bin (35) init (0);                        /* not us
\ced here */

dcl  answer char (256) var;                                      /* answer
\c returned from command_query_ */
dcl  query_info_ptr pointer;                                     /* used f
\cor command_query_ */
dcl (ncols, col_width) fixed bin;                                /* number
\c of columns; column width */

/* Retrieve number of columns and column width */

          query_info_ptr = addr (query_info);
          call command_query_ (query_info_ptr, answer, "embed_tabs",
\c"Enter number of columns:");
          ncols = answer;                                        /* conver
\ct to internal format */
```

```
          call command_query_ (query_info_ptr, answer, "embed_tabs",
\c"Enter column width:");
          col_width = answer;

          begin;


dcl  line (ncols) char (256) var  ;                    /* array
\cfor line segments to be concatenated */
dcl  line_str_var char (256) var init ("");            /* varyin
\cg length string */
dcl  line_str char (79) init ("");                     /* concat
\cenated string */
dcl  line_buff char (50) init (" ");                   /* buffer
\c for input */
dcl  chars fixed bin (21) init (0);
dcl (sub, temp) pic"999";
dcl  stmt_no pic"9999";                                /* statem
\cent number */
dcl  i fixed bin;
dcl  code fixed bin (35);                              /* standa
\crd error code */
dcl (ov_ptr (4), tabout_ptr) pointer;                  /* pointe
\crs to io control blocks */
dcl  buff_ptr pointer;                                 /* pointe
\cr to input buffer */
dcl 1 tab_format_array,
    2 tab (ncols+1) char (9) var;

/* Initialization */

               buff_ptr = addr (line_buff);            /* store
\caddress of input buffer in buffer pointer */
               line (*) = "";                          /* initia
\clize lines to blanks */
               do sub = 1 to ncols+1;                  /* calcul
\cate tab positions */
                    temp = (sub-1)* (col_width-9)+1;
                    tab (sub) = "#(ta," || temp || ")";
               end;
               call iox_$attach_name ("ov1", ov_ptr (1), "vfile_ over
\clay1", null (), code);
               call iox_$attach_name ("ov2", ov_ptr (2), "vfile_ over
\clay2", null (), code);
               call iox_$attach_name ("ov3", ov_ptr (3), "vfile_ over
\clay3", null (), code);
               call iox_$attach_name ("ov4", ov_ptr (4), "vfile_ over
\clay4", null (), code);
               call iox_$attach_name ("tabout", tabout_ptr, "vfile_ g
\ceofmt.columns", null (), code);

               do i = 1 to ncols;
                    call iox_$open (ov_ptr (i), 1, "0"b, code); /* op
\cen overlay segs */
               end;
               call iox_$open (tabout_ptr, 2, "0"b, code);
               call ioa_$ioa_switch (tabout_ptr, "F 0001 #(se ml,t0,j
\c-1,1132,p^a,x^a)", page_len, page_len, code); /* set margins & prop
\c print */
               do stmt_no = 2 to 9999;                        /* do unt
\cil endfile */
                    do i = 1 to ncols;
                         call iox_$get_line (ov_ptr (i), buff_ptr, 50
\c, chars, code);
                         if ((code = error_table_$end_of_info) & (i =
\c 1)) then goto endup;
                         line (i) = substr (line_buff, 1, max (chars-
\c1, 0)); /* move input line to hold area */
                    end;
                    if ((line (1) = "") & (line (2) = "") & (line (3)
\c = "") & (line (4) = ""))
                    then do;
                         line_str_var = "F " || stmt_no || " " || tab
\c (1);
                         line_str = line_str_var;             /* move v
\carying length string to non-varying string */
                         call ioa_$ioa_switch (tabout_ptr, line_str,
\ccode);
                    end;
                    else do;
                         do i = 1 to ncols;

                              if i = 1
                              then line_str_var = "F " || stmt_no ||
\c" " || tab (i) || line (i) || "#(EX)";
                              else if i = ncols
                              then line_str_var = "F " || stmt_no ||
\c"+" || tab (i) || line (i);
                              else line_str_var = "F " || stmt_no ||
\c"+" || tab (i) || line (i) || "#(EX)";
                              line_str = line_str_var;      /* move v
\carying length string to non_varying string */
                              call ioa_$ioa_switch (tabout_ptr, line_
\cstr, code);
                         end;
                    end;
               end;

endup:
               do i = 1 to ncols;
                    call iox_$close (ov_ptr (i), code);
                    call iox_$detach_iocb (ov_ptr (i), code);
               end;
               call iox_$close (tabout_ptr, code);
               call iox_$detach_iocb (tabout_ptr, code);
          end;                                                /* end be
\cgin block */

     end embed_tabs;
```

SUBROUTINE NAME: TO-NBIPP

Author: NBI personnel. Modified for use with **geofmt** programs by Pearl Porter.

Purpose of the program: **to-nbiPP** allows the user to record segments from Multics on the NBI diskette when using the NBI System II as a terminal.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: to-nbiPP

Arguments: None

Subroutines called: None

Common data referenced: None

Input files: ***geofmt.columns***

Output files: None

Arrays used: None

Called by: None

Error checking and reporting: Provisions exist for checking and reporting erroneous names and aborting program.

Constants: None

Program logic:

1. The user is prompted with the message:
   MULTICS FILE NAME TO BE SENT (OR Q TO QUIT):
   The user types the file name, which can be a maximum of 32 characters.
   At the end of transmission, the user types: q.
2. If ***file_name*** = ***q***, go to 13.
3. Attach and open ***file10***.
4. If ***istat*** (which is the error code) is not equal to 0, go to 1.
5. Read a line from the Multics segment into a format (79a1).
   At end, go to step 10.
6. Write the line using a format (1h,79a1)
7. Call **ioa_$nal("?")**. This prints a ? on the screen.
8. Read this character into ***iack***.
9. If ***iack*** = 11, go to 5. Otherwise, go to 6. The above loop, steps 5 through 9, sends each line and inputs each ACK.
10. The next loop holds computer in program so it does not receive ACKs while it is in ready, and NBI still receives.
11. Read a value into ***end***, using format (a1).
12. If value of ***end*** is not equal to ***q***, go to 11.
13. Close and detach ***file10*** and stop the program.

to-nbiPP.fortran

```
c     NBI-MULTICS HANDSHAKE PROGRAM FOR RECEIVING TO NBI'S DISK.
c       (10/4/77)
c     name = to-nibPP:SPECIAL TO SEND IN DISKETTE FORMAT - AUTOMATIC
c        LINES
      dimension line(79)
      character file_name*32
      double precision ec
      equivalence(istat,ec)
c     REQUEST AND ACCESS DESIRED "FILE" TO BE SENT.   (PROVISIONS EXIST
c           FOR ERRONEOUS NAMES AND ABORTING PROGRAM).
13    print ,"Multics file name to be sent (or q to quit): "
      read17, file_name
17    format(v)
      if(file_name.eq."q") go to 5
      call io ("attach","file10","vfile_",file_name)
      call io ("open","file10","si")
      if(istat.ne.0) go to 13
      call ioa_$nnl ("?")
      read(5,20) iack
c     LOOP IN PROGRAM WHICH SENDS EACH LINE AND INPUTS EACH ACK
c       (OR NAK).
1     read (10,10,end=3) line
10    format(79a1)
```

```
2     write(6,11) line
11    format(1h,79a1)
      call ioa_$nnl ("?")
      read (5,20) iack
20    format(i2)
      if(iack.eq.11) go to 1
      go to 2
c     EXIT PROGRAM: INPUT LOOP HOLDS COMPUTER IN PROGRAM SO IT DOES NOT
c          RECEIVE "ACKS" WHILE IT IS IN READY AND NBI STILL RECEIVES.
c          THEN IT CLOSES FILES.
3     read(5,30) end
30    format(a1)
      if(end.ne."q") go to 3
5     call io ("close","file10")
      call io ("detach","file10")
      stop
      end
```

PROGRAM NAME: TO-NBID

Author: **to-nbiD** was written by NBI personnel in October 1977. It is written in Fortran and is compiled on the Honeywell Series 60 computer.

Purpose of the program: **to-nbiD** allows the user to record segments from Multics on the NBI diskette when using the NBI System II as a terminal.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: to-nbiD

Arguments: None

Subroutines called: None

Common data referenced: None

Input files: Name of file to be transferred

Output files: Name of file transferred

Arrays used: None

Called by: None

Error checking and reporting: Provisions exist for detecting erroneous name and halting program.

Constants: None

Program logic:

1. The user is prompted with the message:
   MULTICS FILE NAME TO BE SENT (OR Q TO QUIT):
   The user types the file name, which can be as many as 32 characters long.
   At the end of transmission, the user types: q.
2. If *file_name* = *q*, go to 13.
3. Attach and open *file10*.
4. If *istat* (which is the error code) is not equal to 0, go to 1.
5. Read a line from the Multics segment into a format (133a1). At end go to step 10.
6. Write the line using a format (1h,133a1)
7. Call **ioa_$nal("?")**. This prints a ? on the screen.
8. Read this character into *iack*.
9. If *iack* = 11 go to 5.
   Otherwise, go to 6. The above loop, steps 5 through 9, sends each line and inputs each ACK.
10. The next loop holds computer in program so it does not receive ACKs while it is in ready and NBI still receives.
11. Read *end* using format (a1).
12. If *end* is not equal to *q*, go to 11.
13. Close and detach *file10* and stop the program.

```
to-nbiD.fortran


c     NBI-MULTICS HANDSHAKE PROGRAM FOR RECEIVING TO NBI'S DISK.
c        (10/4/77)
c     name = to-nbiD:SPECIAL TO SEND IN DISKETTE FORMAT - AUTOMATIC
c       LINES
      dimension line(133)
      character file_name*32
      double precision ec
      equivalence(istat,ec)
c     REQUEST AND ACCESS DESIRED "FILE" TO BE SENT.   (PROVISIONS EXIST
c           FOR ERRONEOUS NAMES AND ABORTING PROGRAM).
13    print ,"Multics file name to be sent (or q to quit): "
      read17, file_name
17    format(v)
      if(file_name.eq."q") go to 5
      call io ("attach","file10","vfile_",file_name)
      call io ("open","file10","si")
      if(istat.ne.0) go to 13
      call ioa_$nnl ("?")
      read(5,20) iack
c     LOOP IN PROGRAM WHICH SENDS EACH LINE AND INPUTS EACH ACK
c       (OR NAK).
1     read (10,10,end=3) line
10    format(133a1)
2     write(6,11) line
11    format(1h,133a1)
      call ioa_$nnl ("?")
      read (5,20) iack
20    format(i2)
      if(iack.eq.11) go to 1
      go to 2
c     EXIT PROGRAM: INPUT LOOP HOLDS COMPUTER IN PROGRAM SO IT DOES NOT
c          RECEIVE "ACKS" WHILE IT IS IN READY AND NBI STILL RECEIVES.
c          THEN IT CLOSES FILES.
3     read(5,30) end
30    format(a1)
      if(end.ne."q") go to 3
5     call io ("close","file10")
      call io ("detach","file10")
      stop
      end
```

PROGRAM NAME: CONCAT

Author: Harold Johnson
Purpose of the program: **concat** prepares reference files for input to the GRASP system. Each set of records from one reference are concatenated in the format assigned by matrix into one long vector.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: concat
Arguments: None
Subroutines called: **ftnumber, main_concat**
Common data referenced: None
Input files:
matrix used onunit 22 (*file22*)
refNM used on unit 30 (*file30*)
Output files:
temp77 used on unit 77 (*file77*)
strgNM used on unit 40 (*file40*)
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. *file22* is attached to matrix, *file77* is attached to a temporary file named *temp77*, and the user is asked for the State *id* number for the reference file being processed. *file30* is attached to this State's reference file, and *file40* is attached to a new output file named *strgNM*, where *NM* is the State FIPS code.
2. **main_concat** is called where the main work is processed.
3. All files are closed.

```
c ******* CONCAT *******
 character*32 filename
 character*1 iblank
  character*4 file, mode
  character*2 state
  character*6 outfile
      dimension iout(1211),ifirst(46),ichar(46),item(46,10),iate(5)
data iblank/" "/
c  UPDATED AS OF DEC 27, 1976  H. JOHNSON
c  Converted to Multics February 21, 1977 H Johnson
c Modified to allow item 87 to indicate extra records, March 3, 1977
c  H Johnson.
c WARNING: ANY CHANGE IN nrec MUST BE MADE BY HAND IN THE modify
c SUBROUTINE **********************************!!!!!!!!!!!!!!!!!!!!!
c
c    -----------------------------------------------------------------
c    I                                                               I
c    I                                                               I
c    I      THE PURPOSE OF THIS PROGRAM IS TO PREPARE REFERENCE FILES I
c    I   FOR INPUT TO THE "CREAT" PROGRAM OF "IRIS".  A REFERENCE FILE I
c    I   IS READ FOR EACH IF, ISF, AND A LONG VECTOR RECORD IS CREATED I
c    I   WITH REFERENCE ENTRIES LOCATED IN PRE-ASSIGNED POSITIONS      I
c    I                                                               I
c    I      REQUIRED INPUT FILES:                                    I
c    I   30 = REFERENCE FILE  = "refNM"                              I
c    I   22 = MATRIX FILE DESCRIBING THE ASSIGNED LOCATIONS.         I
c    I   05 = INPUT TO TELL REFERENCE FILE NAME AND NUMBER OF LINES OF I
c    I   EXPLANATORY DATA TO BE SKIPPED .                            I
c    I                                                               I
c    I      REQUIRED OUTPUT FILES:                                   I
c    I   77 = WORK FILE                                              I
c    I   40 = "strgNM" IS THE MAIN OUTPUT FILE.                      I
c    I                                                               I
c    I                                                               I
c    I                                                               I
c    -----------------------------------------------------------------
c THIS PROGRAM CONCATENATES THE DIFFERENT "IF" FILES
c FROM THE REF AND REFU FILES INTO LONG FILES FOR INPUT
c TO THE CREATE PROGRAM OF GRASP.
c
c          MAIN PROGRAM
c
 call io ("attach","file22","vfile_","matrix","-append","-ssf")
 call io ("open","file22","si")
 call io ("attach","file77","vfile_ ","temp77")
 call io ("open","file77","sio")
          write(6,890)
890 format(" ENTER THE 2-DIGIT CODE FOR THE STATE BEING PROCESSED")
read(5,891) state
891 format(a2)
encode(outfile,893)state,iblank
893 format("ref",a2,a1)
mode = "si"
     call ftnumber(30,outfile,mode)
```

```
encode(outfile,895)state
895 format("strg",a2)
mode = "so"
     call ftnumber(40,outfile,mode)
c
c MATRIX IS THE INPUT MATRIX WHICH DESCRIBES WHERE THE INPUT
c RECORDS ARE TO BE LOCATED AMONG THE POSITIONS IN THE OUTPUT
c FILE TEMPO1 WHICH IS SET UP FOR GRASP "CREATE"INPUT.
c
c IT IS ALSO REQUIRED TO EQUATE 30 TO THE INPUT REFU OR REF
c FILE.
c EQUATE 77 TO A TEMPORARY FILE USED ONLY IN THIS PROGRAM.
c THE OUPUT FILE IS CALLED TEMPO1
c
            nrec=1211
            iwide=10
            idim=46
c NREC IS THE LENGTH OF THE OUTPUT FILES IN TEMPO1, OR 40.
c IWIDE IS THE NUMBER OF POSSIBLE PLATES ON THE SAME OUTLINE.
c IDIM IS THE NUMBER OF DIFFERENT KINDS OF FILES IN REF OR REFU.
c
nskip = 0
            call main_concat(nrec,idim,iwide,iout,ifirst,ichar,item,nskip)
c THIS READS THE REFU OR REF FILE AND SETS UP, FOR EACH "IF"
c A VECTOR IOUT CONTAINING DATA FROM THE REF FILE IN POSITIONS
c DESCRIBED BY THE MATRIX.  IT THEN WRITES THESE VECTORS OUT
c TO FILE 40 WHOSE RECORD LENGTH IS NREC.
endfile 40
c
c THIS ROUTINE ADDS A BLANK RECORD TO THE END OF STRGnm
c THIS IS NECESSARY BECAUSE OF A PECULIARITY IN MULTICS.
c
   call  io ("close","file22")
   call io ("close","file40")
   call io ("close","file30")
   call io("detach","file22")
   call io ("detach","file40")
    call io ("detach","file30")
c
   call io ("close","file77")
    call io ("detach","file77")
            stop
c
            end
```

SUBROUTINE NAME: MAIN_CONCAT

Author: Harold Johnson

Purpose of the program: **main_concat** calls **contrl_concat**, which sets up the control vectors and matrices that determine positioning of data in output vectors. It repeatedly calls **vector_concat** to write this information into long vectors in memory. It calls **wryte_concat** to output these vectors to the ***strgNM*** file. Each time it checks for repeated data using ***ndflg***, the flag for ITEM 87.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call main_concat (nrec,idim,iwide,-iout,ifirst,ichar,item)

Arguments:
nrec–The length of the output vectors
idim–The number of different kinds of items–that is, the number of rows in **matrix**
iwide–The maximum number of different items associated with a single row of **matrix**
iout–The output vector of length ***nrec***
ifirst–The control vector of length ***idim*** whose *k*th entry is the starting position in ***iout*** of data associated with the name in the *k*th row of **matrix**
ichar–The control vector of length ***idim*** whose *k*th entry is the number of allowable characters for data associated with the name of the *k*th row of **matrix**
item–The control matrix of dimension ***idim*** by 10 of item numbers occurring in **matrix**

Subroutines called: **contrl_concat**, **vector_concat**, **wryte_concat**, **modify_concat**
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: concat
Error checking and reporting: The user is informed:
YOU GOT TO MAIN
Number of records written to ***strgNM*** is counted, and the user is informed every 25th record because, during the long interactive running of this program, the user may become anxious about loops and long CPU time.
Constants: None
Program logic:
1. Subroutine **contrl_concat** is called to set up the control matrices ***idim***, ***ichar***, and ***item***.
2. Subroutine **vector_concat** is called to set up the output vector for one reference.
3. **wryte_concat** is called to output that vector.
4. When an ITEM 87 is found, indicating repeated data, **modify_concat** is called to modify the previous output vector according to the data that come after ITEM 87.
5. The count of output vectors is incremented and a message written each time the count equals a multiple of 25.

```
           subroutine main_concat(nrec,idim,iwide,iout,ifirst,ichar,
\citem,
& nskip)
c subroutine used in main program "concat"
c updated as of dec. 27, 1976  h. johnson
c converted to multics february 27, 1977 h johnson
           dimension iout(nrec),ifirst(idim),ichar(idim),ifile(60)
           dimension item(idim,10)
           write(6,9100)
9100        format(" you got to main")
c
           call contrl_concat(idim,ifirst,ichar,item)
c this sets up the control matrix to run this subroutine.
c item(line,kolumn) is the item number in refu.
c ifirst(line) is the starting position in the output file
c for the item.
c ichar(line) is the number of positions for item(line,kolumn)
c in the outfile.
c
           kount=1
c
1 call vector_concat(nrec,idim,iout,ndflg,iwide,ichar,ifirst,item,kf
\clg)
c this reads through one "if" file in refu. for those items
c in kolumn=1, it sets up a vector iout(nrec) which is to be the
c first output for this "if". it writes the other cards
c having item with kolumn .gt. 1 into a file 77 which will
c be read repeatedly to produce new vectors iout.  kflg =
c the number of cards written to 77.  ndflag = 1 when end of
c file 30 is reached.
c
```

```
           call wryte_concat(iout,nrec,item,idim,iwide,kflg,
& ichar,ifirst)
c this writes the iout received from vector to the output
c file, 40.  it then reads through file 77 using the kolumn
c =2 to change iout, writes this new iout vector, then goes
c to kolumn=2,3,...
c
 if(ndflg .ne. -1) go to 5
 call modify_concat(iout,nrec,item,idim,iwide,kflg,ichar,ifirst,ndfl
\cg)
c
c this routine reads throught records which follow item = 87 until
c the next item = 87 is encountered.  it modifies iout only in
c those records which it finds in file 30 between these two items 87
c and then writes the resulting vector and proceeds to modify until
c it senses a new "if".
c
c
5   continue
           kount=kount+1
           if(25*(kount/25) .lt. kount) go to 10
           write(6,9110)kount
9110       format(" you wrote the ",i5,"th vector to the strg file")
10         if(ndflg .eq. 1)return
           go to 1
c
           end
```

SUBROUTINE NAME: ALTER_CONCAT

Author: Harold Johnson

Purpose of the program: **alter_concat** modifies the output vector when more than one item occurs associated with the same name classification. Those data have been written to *file77*, and **alter_concat** processes them.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call alter_concat (iout,nrec,item,idim,-kolumn,mstop,ichar,ifirst)

Arguments:

iout, nrec, item, idim, ichar, ifirst–See **main_concat.**

kolumn–An assigned column of *item*, which is to determine what data will be used to modify *iout*

Subroutines called: **locate_concat**

Common data referenced: None

Input files: None

Output files: None

Arrays used: *iout(nrec), ichar(idim), ifirst(idim), ifile(60)*

Called by: **wryte_concat**

Error checking and reporting: None

Constants: None

Program logic:

1. *mstop* is set to 0.
2. *file77* is read, and *id* is compared with those in column number *kolumn* in *item*.
   If a match is found, *mstop* is set to 1, *iout* is modified according to the data in this record of *file77*, and the reading is repeated.
3. If no match is found, the next record in *file77* is read.

```
c ******* SUBROUTINE ALTER_CONCAT *******
          subroutine alter_concat(iout,nrec,item,idim,kolumn,mstop,
         ichar,ifirst)
c SUBROUTINE USED IN MAIN PROGRAM "CONCAT"
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c Converted to Multics FEBRUARY 18, 1977 H. Johnson
```

```
c
            dimension iout(nrec),item(idim,10),ifile(60),iff(3)
            dimension ichar(idim),ifirst(idim)
            mstop = 0
1           read(77,900,end=100)istate,(iff(j),j=1,3),itm,
           (ifile(k),k=1,60)
900         format(i2,3a1,i2,60a1)
c
            call locate_concat(itm,idim,item,line,kolumn)
            if(line .gt. 0) go to 10
            go to 1
c
10          mstop=1
            no=ichar(line)
            do 20 j=1,no
            iout(ifirst(line)+j-1)=ifile(j)
20          continue
            go to 1
c
100         rewind 77
             return
            end
c ******* END ALTER_CONCAT *******
```

SUBROUTINE NAME: LOCATE_CONCAT

Author: Harold Johnson
Purpose of the program: **locate_concat** searches the rows of ***item*** under an assigned column to match a given item number.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call locate_concat (itm,idim,item,-line,kolumn)
Arguments:
idim, ***item*** – See **main_concat.**
itm – A given item number that is to be found in ***item***
kolumn – A given column number whose column in ***item*** is to be searched
line – The line number in ***item*** where ***itm*** is found in the ***kolumn***th column
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: ***item*** (***idim***,***10***)
Called by: **vector_concat**, **alter_concat**, **modify_-concat**
Error checking and reporting: None
Constants: None
Program logic:
1. ***line*** is set at 0.
2. The column number ***kolumn*** in array ***item*** is searched for a match with ***itm***.
3. If found, that column is equated to ***line***.

```
c ******* SUBROUTINE LOCATE_CONCAT *******
            subroutine locate_concat(itm,idim,item,line,kolumn)
c SUBROUTINE USED IN MAIN PROGRAM "CONCAT"
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c converted to Multics February 21, 1977 H. Johnson
c
            dimension item(idim,10)
            line=0
            do 10 j=1,idim
            if(itm .eq. item(j,kolumn)) go to 20
```

```
10          continue
            return
20          line=j
            return
            end
c  ******* END LOCATE_CONCAT *******
```

SUBROUTINE NAME: MODIFY_CONCAT

Author: Harold Johnson
Purpose of the program: **modify_concat** reads through the records in a reference file that lie between two successive ITEM 87s or between 87 and the next reference number and then modifies the previous ***iout*** vector according to those intermediate records.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call modify_concat (iout,nrec,item,-idim,iwide,kflg,ichar,ifirst,ndflg)
Arguments:
iout, ***nrec***, ***item***, ***idim***, ***iwide***, ***ichar***, ***ifirst***–See **main_concat**.
kflg, ***ndflg***, ***iout***–See **vector_concat.**
Subroutines called: **wryte_concat**, **locate_concat**
Common data referenced: None
Input files: None
Output files: None
Arrays used: ***newout(1211)***, ***ifirst(idim)***, ***iout(nrec)***
Called by: **main_concat**
Error checking and reporting: None
Constants: None
Program logic:
1. A vector ***newout*** is made identical to the input vector ***iout.***
2. A record from the reference file is read.
   If this represents a new reference file as indicated by a new ***if*** number, the input reference file is backspaced, the ***newout*** vector is written, and program control returns.
   If the record is a new ITEM 87, ***newout*** is written and step 1 is repeated.
   If the record is neither of these, the record data is entered into ***newout***, and the next record is read and step 2 is repeated.

```
c  ******* SUBROUTINE MODIFY_CONCAT *******
       subroutine modify_concat(iout,nrec,item,idim,iwide,kflg,ichar,
                                ifirst,ndflg)
c
c Subroutine used in the new version of concat, updated to allow
c extra items flagged by item no. 87.
c H Johnson, March 3, 1977.
c
c   This subroutine read through records which follow item = 87 until
c the next item 87 is noted.  It modifies iout only in those records
c which it finds between items 87.  It then writes the resulting
c vector and continues.  If it senses a new "if" it returns.
c
       dimension iout(nrec),ifirst(idim),ichar(idim),ifile(60)
       dimension item(idim,10),newout(1211)
c
ndflg = 0
c first, make newout  the same as iout, the original vector..
5   do 7 j=1,nrec
    newout(j) = iout(j)
7 continue
c
10 read(30,910,end=1001)istate,(newout(j),j=2,4),itm,(ifile(k),k=1,60)
910 format(i2,3a1,i2,60a1)
```

```
c
c check to see if a new "if" has been encountered; if it has, backspace
c and return.
c
     do 20 j=2,4
    if(newout(j) .ne. iout(j)) go to 1000
20 continue
c
c Check to see if a new item 87 has been encountered.  If it has,
c write the newout vector and repeat the process.
c
       if(itm .ne. 87) go to 100
   kflg = 0
    call wryte_concat(newout,nrec,item,idim,iwide,kflg,ichar,ifirst)
  go to 5
c
100 continue
c Now locate the line in matrix which this last-read item occurs in.
c
     do 110 kolumn = 1,10
    call locate_concat(itm,idim,item,line,kolumn)
    if(line .ne. 0) go to 120
110 continue
c
120 continue
c Modify newout according to ifile in the positions indicated by
c ichar(line) and ifirst(line).
c
   no = ichar(line)
   do 130 j = 1,no
130 newout(ifirst(line) + j - 1) = ifile(j)
c
   go to 10
c
1000 backspace 30
do 1020 j = 2,4
1020 newout(j) = iout(j)
    call wryte_concat(newout,nrec,item,idim,iwide,kflg,ichar,ifirst)
    return
c
1001 ndflg=1
return
end
c ******* END MODIFY_CONCAT *******
```

SUBROUTINE NAME: VECTOR_CONCAT

Author: Harold Johnson

Purpose of the program: **vector_concat** sets up the output vector for one reference file. When ITEM 87 is found, it writes the remaining records for the reference to a temporary holding *file77.*

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call vector_concat (nrec,idim,iout,-ndflg,iwide,ichar,ifirst,item,kflg)

Arguments:

nrec, idim, iout, iwide, ichar, ifirst, item–See **main_concat.**

ndflag–Indicates by 1 that the end of the input reference has been reached

kflg—The number of records from the reference file that **vector_concat** temporarily stored in *file77*
Subroutines called: **locate_concat**
Common data referenced: None
Input files: *refNM*
Output files: None
Arrays used: *iout*(*nrec*), *ifile*(*60*), *ichar*(*idim*), *ifirst*(*idim*)
Called by: **main_concat**
Error checking and reporting: None
Constants: None
Program logic:
1. The output vector *iout* is first blanked out.
2. The first reference record for the current reference file is read to set up the reference *id* and the State FIPS code in *iout*.
3. The rest of the records are read and the row and column of the **matrix** file is determined where the corresponding item is located. This is done by calling **locate_concat.**
4. Data in each record is inserted into *iout* using the information determined in 3.
5. When ITEM 87 has been read, control is returned to **main_concat**.
6. Whenever items are found that occur in columns of *item* other than the first, those records are written to a temporary holding *file77*. **wryte_concat** processes *file77* to update *iout*.

```
c ******* SUBROUTINE VECTOR_CONCAT *******
          subroutine vector_concat(nrec,idim,iout,ndflag,iwide,ichar,
         ifirst,item,kflg)
c SUBROUTINE USED IN MAIN PROGRAM "CONCAT"
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c  Converted to Multics February 21, 1977 H Johnson.
c
          dimension iout(nrec),ifile(60),ifirst(idim),ichar(idim)
          dimension item(idim,10),iff(3)
c
          data iblank/"    "/
          ndflag=0
          kflg=0
c
c FIRST, MAKE IOUT ALL BLANK
          do 10 j=1,nrec
          iout(j)=iblank
10        continue
c
c READ THE FIRST RECORD TO SET UP THE "ID" NUMBER
c AND THE STATE IN IOUT
c
          read(30,900,end=1001)istate,(iout(j),j=2,4),itm,
         (ifile(k),k=1,60)
900       format(i2,3a1,i2,60a1)
          do 20 j=1,20
          iout(4+j)=ifile(j)
20        continue
c
30        read(30,900,end=1001)istate,(iff(j),j=1,3),itm,(ifile(k),
          k=1,60)
c CHECK TO SEE IF THIS IS A NEW IF; IN WHICH CASE, STOP.
c
          do 35 j=1,3
          if(iff(j) .ne. iout(1+j)) go to 1000
35           continue
c
  if(itm .ne. 87) go to 37
  ndflag=-1
```

```
 if(kflg .gt. 0)rewind 77
return
37    continue
c
c  When itm = 87 is encountered, it indicates that the last output
c vector is to be modified by the next records which follow
c until the next item = 87 is encountered.  Any further records
c with the same "if" will modify the last output vector obtained from
c the original file.
c NOW LOCATE THE LINE AND KOLUMN IN WHICH ITM OCCURS.
c IF IT OCCURS IN KOLUMN 1, WRITE TO IOUT; OTHERWISE WRITE
c TO FILE 77.
c
            kolumn=1
            call locate_concat(itm,idim,item,line,kolumn)
c
c WHEN LINE = 0, NO MATCH HAS BEEN FOUND IN KOLUMN 1.
c IT IS NECESSARY TO CHECK THE OTHER COLUMNS.
c
            if(line .eq. 0) go to 50
            no=ichar(line)
            do 40 j=1,no
            iout(ifirst(line)+j-1)=ifile(j)
40          continue
            go to 30
c
50          kflg=kflg+1
c
            write(77,900)istate,(iff(j),j=1,3),itm,(ifile(k),k=1,60)
            go to 30
c
1000        backspace 30
            if(kflg .gt. 0)rewind 77
            return
c
1001         ndflag=1
            if(kflg .gt. 0)rewind 77
            return
            end
c ******* END VECTOR_CONCAT *******
```

SUBROUTINE NAME: WRYTE_CONCAT

Author: Harold Johnson
Purpose of the program: **wryte_concat** is used to write the vector *iout* to the output file *strgNM*. It also modifies *iout* when multiple item numbers occur that have been written to *file77*.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call wryte_concat (iout,nrec,item,-idim,iwide,kflg,ichar,ifirst)
Arguments:
iout, nrec, item, idim, iwide, ichar, ifirst–See **main_concat**.
kflg–See **vector_concat.**
Subroutines called: **alter_concat**
Common data referenced: None
Input files: None
Output files: None
Arrays used: *iout(nrec), leftov(80)*
Called by: **main_concat, modify_concat**
Error checking and reporting: None
Constants: None

Program logic:

1. An 80-character vector named ***leftov*** is blanked out.
2. Each successive 80-character segment of ***iout*** is written to ***strgNM*** until a segment of less than 80 characters remains.
3. This last segment, with blanks from ***leftov*** added to its right-hand side to make up 80 characters, is written to ***strgNM***.
4. If ***kflg*** indicates that data exists in ***file77*** for this reference, **alter_concat** is called to modify ***iout***. Then step 2 is repeated. ***kolumn*** begins at 1 and is incremented by 1 until no match is found by **alter_concat**. In this way, repeated data are introduced into ***iout*** one column at a time according to matrix.

```
c ******* SUBROUTINE WRYTE_CONCAT *******
           subroutine wryte_concat(iout,nrec,item,idim,iwide,kflg,
          ichar,ifirst)
c SUBROUTINE USED IN MAIN PROGRAM "CONCAT"
c WARNING: THERE IS A SUBROUTINE CALLED WRYTE IN THE PROGRAM "PUT44"
c UPDATED DEC. 27, 1976  H. JOHNSON
c converted to Multics February 18, 1977 H. Johnson
c
           dimension iout(nrec),item(idim,10),v(3)
           dimension ichar(idim),ifirst(idim),leftov(80)
            data iblank/"    "/
           do 1000 j=1,80
           leftov(j)=iblank
1000       continue
           kolumn=1
           m=80
1           no=nrec/80
           do 10 j=1,no
           write(40,900)(iout(80*(j-1)+k),k=1,80)
900        format(80a1)
10         continue
           mo=(no+1)*80 - nrec
           no=80*no+1
           write(40,900)(iout(k),k=no,nrec),(leftov(j),j=1,mo)
           if(kflg .eq. 0)return
c
           kolumn=kolumn+1
           if(kolumn .gt. iwide)return
c
           call alter_concat(iout,nrec,item,idim,kolumn,mstop,
          ichar,ifirst)
c THIS RUNS THROUGH FILE 77 AND COMPARES ITM WITH ITEM
c (J,KOLUMN),J=1,IDIM.  WHEN A MATCH IS FOUND, IOUT IS CHANGED.
c MSTOP = 0 WHEN NO MATCH HAS BEEN FOUND.
           if(mstop .eq. 0) return
           if(kolumn .eq. iwide)return
           go to 1
c
           end
c ******* END WRYTE_CONCAT *******
```

SUBROUTINE NAME: CONTRL_CONCAT

Author: Harold Johnson
Purpose of the program: **contrl_concat** sets up the control vectors *ifirst* and *ichar* and the matrix *item* that are used to process the records.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call contrl_concat (idim,ifirst,ichar,-item)
Arguments: See **main_concat**.
Subroutines called: None
Common data referenced: None
Input files: **matrix** used on unit 22 (*file22*)
Output files: None
Arrays used: None
Called by: **main_concat**
Error checking and reporting: None
Constants: None
Program logic:

1. By means of a do loop, the file **matrix** is read to determine for each row the locations of the first character, the number of admissible characters, and the associated item numbers for each kind of data.

```
c ******* SUBROUTINE CONTRL_CONCAT *******
          subroutine contrl_concat(idim,ifirst,ichar,item)
c SUBROUTINE USED IN MAIN PROGRAM "CONCAT"
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c Converted to Multics February 21, 1977 H Johnson
c
          dimension ifirst(idim),ichar(idim),item(idim,10)
          do 10 j=1,idim
         read(22,900)a,b,c,ifirst(j),ichar(j),not,
         (item(j,k),k=1,10)
900       format(2a4,a2,i5,i9,i9,10i3)
10        continue
          return
          end
c ******* END CONTRL_CONCAT ******
```

FILE NAME: MATRIX

Purpose of the file: **matrix** assigns to the item numbers that occur in the reference files the following data: acronyms, type numbers, intitial character positions, maximum character lengths, and terminal character positions in the *strgNM* and *redyNM* files that are used as input to the GRASP routines.
Format: Each record contains *iacron*, *itype*, *ifirst*, *ichar*, *ilast*, and from 1 to 10 items, located as follows: a9 (left-justified), i1, i5, i9, i6, 3X, and from 1 to 10 as i3.
Arguments:
iacron–An acronym associated with the items
itype–A code for the GRASP program to indicate what type of data occurs in the record (1 means integer; 2 means floating point number; 3 means dictionary character; 6 means character string)
ifirst–The initial position in the records of *strgNM* and *redyNM* files where this information is to be stored
ichar–Maximum allowable length of this information
ilast–Last position in the records of *strgNM* and *redyNM* where this information is allowed
item–From 1 to 10 item numbers that are associated with this acronym
Referenced by: **chkref**, **concat**, GRASP

```
          matrix

ID        1     1        4      4     1
STATE     3     5       20     24     2
AUTHOR1   6    25       60     84     3
AUTHOR2   6    85       60    144     4
AUTHOR3   6   145       60    204     5
YEAR      1   205        4    208     8
```

TITLE1	6	209	60	268	9									
TITLE2	6	269	60	328	10									
TITLE3	6	329	60	388	11									
TITLE4	6	389	60	448	37									
COUNTY1	6	449	60	508	12									
COUNTY2	6	509	60	568	13									
COUNTY3	6	569	60	628	14									
PUBLISH	6	629	60	688	17									
SERIES	6	689	60	748	23									
SERIES2	6	749	60	808	60									
EMPHASI	6	809	60	868	24									
AREA	2	869	8	876	25									
AUNIT	6	877	7	883	26									
NLAT	1	884	12	895	27									
SLAT	1	896	12	907	28									
WLONG	1	908	12	919	29									
ELONG	1	920	12	931	30									
CLAT	1	932	12	943	31									
CLONG	1	944	12	955	32									
OMAPS	6	956	60	1015	34									
AVAIL	6	1016	60	1075	35									
BASE	3	1076	30	1105	36									
GEOLOGY	3	1106	12	1117	38									
PLATE	6	1118	30	1147	39	40	41	42	43	66	67	68	69	70
IDSTAT	1	1148	2	1149	44									
SCALE	1	1150	8	1157	18	19	20	21	22	61	62	63	64	65
IDSUB	1	1158	2	1159	45	46	47	48	49	71	72	73	74	75
IBOUND	1	1160	6	1165	50	51	52	53	54	76	77	78	79	80
ISPAN	1	1166	6	1171	55	56	57	58	59	81	82	83	84	85
ALSOMAP	6	1172	30	1201	86									
DUM0	1	1202	1	1202	87									
DUM1	1	1203	1	1203	88									
DUM2	1	1204	1	1204	89									
DUM3	1	1205	1	1205	90									
DUM4	1	1206	1	1206	91									
DUM5	1	1207	1	1207	92									
DUM6	1	1208	1	1208	93									
DUM7	1	1209	1	1209	94									
DUM8	1	1210	1	1210	95									
DUM9	1	1211	1	1211	96									

PROGRAM NAME: TAPEDWG

Author: Richard Thoensen

Purpose of the program: **tapedwg** reads a group of card images from a tape created on a 32 bit machine and creates a System 101 drawing file. The header card is followed by data cards that contain 6 points per card in (12F6.3) format.

Data base: Geoindex

Computer: Data General Nova 1220

Operating system: System 101

Calling sequence: tapedwg

Arguments: None

Subroutines called: **save**, **msgot**, **numin**, **yesno**, **modlc**, **fckbk**, **rdtape**, **ckcrd**, **hex8**, **rdhdr**, **asflc**, **erase**, **rdcrd**, **rwcon**, **rewin**, **bell**, **ovrly**, **exit2**, **skip**

Common data referenced: */blk/*, */pntr/*, */menu1/*, */exec/*, */dskbf/*, */ident/*, */tape/*, */tpdw1/*, */tpdw2/*, */font/*

Input files: Tape that contains *mapNM*

Output files: *coorNMdw*, *bordNMdw*, *gridNMdw*, *statNMdw*, *counNMdw*

Arrays used: None

Called by: None

Error checking and reporting: A message will be printed out if a data error or tape error occurs.

Constants: None
Program logic:

1. Initialize data fields.
2. Send message to terminal:
   TAPEDWG OVERLAY
   PAUSE MOUNT TAPE ON UNIT 0
3. Send message to terminal:
   CHARACTER HEIGHT =
   and store the response in *texth*.
4. Send message to terminal:
   SYMBOL # =
5. Send message to terminal:
   # OF PENS=,1,2, or 3
   and store response in *ipen*.
6. Send message to terminal:
   TEXT WANTED?? 1=yes, 0=no
   and store in *iftext*
7. Multiply the character height *texth* by 1.5 and store in *sfact*.
8. Send message to terminal:
   SKIP FILES?
   Call **yesno**.
   If yes, call **skip**.
   Otherwise, go to next step.
9. Call **rdtape** to read the tape.
   If *jstat* = 4, go to step 24.
   If *jstat* not = 0, go to step 25.
10. Call **hex8**.
11. Call **rhhdr**.
12. Call **asflc** to find the number *temp* in *ibuff(1)*.
    isubf = *temp* + 0.5.
    Write the value of *isubf* using (I6) format.
    If *isubf* = 9999, then go to step 29.
    If *isubf* is greater than 1000, then subtract 1000 from *isubf*.
    If *isubf* equals 1000, load *isubf* with 998.
13. Call **erase** to clear the screen.
14. Call **asflc** to find the number *temp* in *ibuff(6)*, *isf* = *temp* + 0.5.
    If the error code *istat* = 1, set *isf* = 1.
15. Call **asflc** to find number *temp* in *ibuff(11)*.
    If the error code is 1, then set *not* = 0; otherwise, set *not* = *temp* + 0.5.
16. Call **asflc** to find the number *temp* in *ibuff(16)*.
    If the error code is 1, then set *ispan* = 0; otherwise, set *ispan* = *temp* + 0.5.
17. If *isfno* is greater than 6, then set *ie* = 6; otherwise, set *ie* = *isfno*.
18. Call **rdtape** to read the tape.
19. Call **ckcrd.**
    If *ibad* = 1 (error code) go to 18.
20. Call **hex8**.
21. Call **rdcrd** to read the data card.
    If *kstat* = 1 go to 26.
22. If *iftext* is not equal to 1, call **rwcon** to format symbols only. *iftext* will be = 1 if text was wanted, otherwise it will be a 0.
23. Call **rwcon** to format the drawing file.
    Continue steps 18 through 23 until *jstat* = 4.
24. Call **msgot** to print on the terminal:
    END OF FILE REACHED?
    Go to step 27.
25. Message:
    TAPE ERROR # (*NM* and the error code.)
    Go to step 27.
26. Message:
    DATA ERROR
27. Call **msgot** to send message to the terminal:
    REWIND TAPE?
28. Call **yesno**.
    If *ians* = 1 call **rewin** to rewind the tape.
29. Call **msgot** to send message to the terminal:
    PROGRAM FINISHED!!

```
C      TAPEDWG
C
C    10THOENSEN76
C
C
C      SOURCE=<TAPEDWG:F>
C      OBJECT=<TAPEDWG:R>
C
C      PURPOSE:
C              TO READ A GROUP OF CARD IMAGES FROM A
C              TAPE CREATED ON A 32 BIT MACHINE
C              AND CREATE A SYST 101 DWG FILE.
C              HEADER CARD FOLLOWED BY DATA CARDS
C              W/ 6 PTS PER CARD 12F6.3 FORMAT
C
C
```

```
C.... REMARKS:
C.... THIS PROGRAM HAS KNOWLEDGE OF FILE STRUCTURE.
C
C     WHEN RWCON READS A RECORD IT TRANSFERS
C     THE DATA TO COMMON /LINBF/ LTYPE,LWIDE
C     AND TO COMMON /SYMBF/ MIRSY,SKLSY
C
C     WHEN RWCON WRITES A RECORD IT TRANSFERS THE DATA
C     FROM COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
C
C     THE CURRENT SYST 100 VALUES FOR LINE WIDTH
C     AND TYPE ARE STORED IN COMMON /MENU1/
C
C
      COMMON /BLK/X(30),Y(30),A(10),K(30),KP,ID(80)
      COMMON /PNTR/KPT(3,2)
      COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
      COMMON /EXEC/ IEXEC(64),REXEC(64)
      COMMON /DSKBF/ IDUM(3),LENG
      COMMON /IDENT/ IDA(3)
      COMMON /TAPE/ IBUF(40),ICRD(80)
      COMMON /TPDW1/ XX(6),YY(6),IBLANK,ISF,ISF2,IBUFF(20)
      COMMON /TPDW2/ LS(5),LSF(5),ISFNO,LSF2(5),LSPAN(5)
      COMMON /FONT/ IFONT
C
      EQUIVALENCE (K(1),K1)
      EQUIVALENCE (K(2),K2)
      EQUIVALENCE (K(4),KF)
      EQUIVALENCE (K(11),KEY)
      EQUIVALENCE (K(12),ISUBF)
      EQUIVALENCE (K(13),NSYMB)
      EQUIVALENCE (K(15),NCHAR)
      EQUIVALENCE (IEXEC(31),JUSTH)
      EQUIVALENCE (IEXEC(32),JUSTV)
      EQUIVALENCE (X(1),X1)
      EQUIVALENCE (Y(1),Y1)
      EQUIVALENCE (A(1),ANGLE)
      EQUIVALENCE (A(2),TEXTH)
      EQUIVALENCE (IEXEC(19),LNMSV),(IEXEC(20),LNWSV)
      EQUIVALENCE (IDA(1),IFNO)
      EQUIVALENCE (IDA(2),NOR)
      EQUIVALENCE (IDA(3),NIF)
C
      DATA IBLANK /2H  /
C
C
C
C
C
      CALL SAVE(1)
      KNT=0
      IBAD=0
      ISTAT=0
      JSTAT=0
      KSTAT=0
      LSTAT=0
      MSTAT=0
      MRFLG=0
      SFACT=1
      LNMOD=1
      LNWID=0
      IFONT=0
      ANGLE=0
      JUSTH=1
      JUSTV=1
      IDA(1)=0
      IDA(2)=0
      IDA(3)=0
C
      CALL MSGOT("|TAPEDWG OVERLAY")
      CALL NUMIN("||TAPE DRIVE NO. (0 OR 1) ",TEMP)
      MTUNIT=TEMP
      PAUSE MOUNT TAPE PLEASE
C
  200 CALL NUMIN("|CHARACTER HEIGHT=",TEXTH)
      IF (TEXTH) 200,210,210
  210 CALL NUMIN("|SYMBOL # =",TEMP)
      NSYMB=TEMP
      IF (NSYMB) 210,210,220
  220 IF (NSYMB-200) 230,230,210
 230  CALL NUMIN("|# OF PENS=,1,2,OR 3",TEMP)
      IPEN=TEMP
      IF (IPEN .LT. 1)  GO TO 230
      IF (IPEN .GT. 3)  GO TO 230
C
      CALL NUMIN("|TEXT WANTED??  1=YES,0=NO",TEMP)
      IFTEXT=TEMP
C
      SFACT=1.5*TEXTH
C
      CALL MSGOT("|SKIP FILES?")
      CALL YESNO(IANS)
      IF(IANS .EQ. 1)CALL SKIP(MTUNIT,JSTAT)
      IF(JSTAT .NE. 0)GO TO 6000
C
 300  LNWID=LNWID+1
      IF (LNWID.GT.IPEN) LNWID=1
C
C     READ HEADER CARD
C
      IFNO=0
      ISFNO=0
      NOR=0
      NIF=0
C
C    CHECK FOR FULL DRAWING FILE
C
      CALL MODLC(KPT(1,2),-3000,MSTAT)
      GO TO (800,7300,7300),MSTAT
C
```

```
800    CONTINUE
       CALL FCKBK(LSTAT)
       IF(LSTAT .NE. 0)GO TO 7000
       DO 1000 I=1,40
       IBUF(I)=IBLANK
1000   CONTINUE
       CALL RDTAPE(MTUNIT,IBUF,40,0,80,JSTAT)
       IF(JSTAT .EQ. 4)GO TO 5000
       IF(JSTAT .NE. 0)GO TO 6000
C
       CALL CKCRD(IBAD)
       IF(IBAD .EQ. 1)GO TO 800
C
       CALL HEX8
C
       CALL RDHDR
C
C
       NCHR1=5
       NCHR2=5
       NCHR3=5
       NCHR4=5
C
C
       DO 320 I=1,5
       IF (LS(I) .EQ. IBLANK) NCHR1=NCHR1-1
       IF (LSF(I) .EQ. IBLANK)  NCHR2=NCHR2-1
       IF (LSF2(I).EQ. IBLANK)  NCHR3=NCHR3-1
       IF(LSPAN(I) .EQ. IBLANK)NCHR4=NCHR4-1
 320   CONTINUE
C
C
       J=5-NCHR1
       DO 330 I=1,NCHR1
       LS(I)=LS(I+J)/400K
       IBUFF(I)=LS(I)
 330   CONTINUE
C
       WRITE(10,335) (LS(I),I=1,NCHR1)
 335   FORMAT(50I6)
C
       J=5-NCHR2
       DO 340 I=1,NCHR2
       LSF(I)=LSF(I+J)/400K
       IBUFF(I+5)=LSF(I)
 340   CONTINUE
C
C
       IF (NCHR3.EQ.0) GO TO 352
       J=5-NCHR3
       DO 350 I=1,NCHR3
       LSF2(I)=LSF2(I+J)/400K
       IBUFF(I+10)=LSF2(I)
 350   CONTINUE
C
```

```
C
 352  IF (NCHR4 .EQ. 0)  GO TO 354
      J=5-NCHR4
      DO 353 I=1,NCHR4
      LSPAN(I)=LSPAN(I+J)/400K
      IBUFF(I+15)=LSPAN(I)
 353  CONTINUE
C
C
  354 CALL ASFLC (IBUFF(1),NCHR1,TEMP,ISTAT)
      ISUBF=TEMP+.5
C
      WRITE(10,20) ISUBF
   20 FORMAT(I6)
      IF (ISUBF .EQ. 9999)  GO TO 500
C
C    CLEAR SCREEN
C
      KNT=KNT+1
      IF(KNT .NE. 26)GO TO 1100
      KNT=0
      CALL ERASE
1100  CONTINUE
C
      CALL ASFLC(IBUFF(6),NCHR2,TEMP,ISTAT)
      ISF=TEMP+.5
      IF(ISTAT .EQ. 1)ISF=1
      CALL ASFLC (IBUFF(11),NCHR3,TEMP,ISTAT)
      NOT=TEMP+.5
      IF (ISTAT.EQ.1) NOT=0
      CALL ASFLC(IBUFF(16),NCHR4,TEMP,ISTAT)
      ISPAN=TEMP+.5
      IF(ISTAT .EQ. 1)ISPAN=0
      IE=ISFNO
      IF(ISFNO .GE. 6)IE=6
C
C     READ FIRST DATA CARD
C
1200  CONTINUE
      CALL FCKBK(LSTAT)
      IF(LSTAT .NE. 0)GO TO 7000
      DO 1500 I=1,40
      IBUF(I)=IBLANK
1500  CONTINUE
      CALL RDTAPE(MTUNIT,IBUF,40,0,80,JSTAT)
      IF(JSTAT .EQ. 4)GO TO 5000
      IF(JSTAT .NE. 0)GO TO 6000
C
      CALL CKCRD(IBAD)
      IF(IBAD .EQ. 1)GO TO 1200
C
      CALL HEX8
C
      CALL RDCRD(KSTAT)
      IF(KSTAT .EQ. 1)GO TO 6500
```

```
C
C
C
      IF (IFTEXT .NE. 1)  GO TO 400
C
      X1=XX(1)
      Y1=YY(1)
      NCHAR=NCHR1
      DO 360 I=1,NCHR1
      ID(I)=LS(I)
  360 CONTINUE
      KEY=8
      KP=1
      CALL RWCON(KF,2)
      KEY=16
      CALL RWCON(KF,2)
C
C
      IF (ISF.EQ.1.AND.IFNO.EQ.1) GO TO 375
      NCHAR=NCHR2
      DO 370 I=1,NCHR2
      ID(I)=LSF(I)
  370 CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
 375  IF(ISPAN .EQ. 0)GO TO 380
      NCHAR=NCHR4
      DO 377 I=1,NCHR4
      ID(I)=LSPAN(I)
 377  CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
  380 IF (NOT.EQ.0) GO TO 400
      NCHAR=NCHR3
      DO 390 I=1,NCHR3
      ID(I)=LSF2(I)
  390 CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
  400 X1=XX(2)
      Y1=YY(2)
C
C     SYMBOL NEEDED IF THIS IS SINGLE PT
C
      IF (ISFNO.GT.2) GO TO 410
      KEY=7
      KP=1
      CALL RWCON(KF,2)
      GO TO 300
```

```
C
C      WRITE PEN UP
C
  410 KEY=1
      KP=1
      CALL RWCON(KF,2)
      JJ=3
      GO TO 430
C
420   CONTINUE
      CALL FCKBK(LSTAT)
      IF(LSTAT .NE. 0)GO TO 7000
      DO 2000 I=1,40
      IBUF(I)=IBLANK
2000  CONTINUE
      CALL RDTAPE(MTUNIT,IBUF,40,0,80,JSTAT)
      IF(JSTAT .EQ. 4)GO TO 5000
      IF(JSTAT .NE. 0)GO TO 6000
C
      CALL CKCRD(IBAD)
      IF(IBAD .EQ. 1)GO TO 420
C
      CALL HEX8
C
      CALL RDCRD(KSTAT)
      IF(KSTAT .EQ. 1)GO TO 6500
C
C
C
C      WRITE PEN DOWN
C
  430 KEY=6
      KP=1
      DO 440 I=JJ,IE
      X1=XX(I)
      Y1=YY(I)
      CALL RWCON(KF,2)
  440 CONTINUE
      JJ=1
      IF (ISFNO-6) 300,300,450
  450 ISFNO=ISFNO-6
      IE=ISFNO
      IF (ISFNO.GE.6) IE=6
      GO TO 420
C
C    EOF FOUND
C
5000  CONTINUE
      CALL MSGOT("|END OF FILE REACHED?")
      GO TO 7000
C
C    TAPE ERROR
C
6000  CONTINUE
      WRITE(10,1001)JSTAT
```

```
1001  FORMAT(1X,'TAPE ERROR #',I1)
      GO TO 7000
C
C   DATA ERROR
C
6500  CONTINUE
      WRITE(10,1002)
1002  FORMAT(1X,'DATA ERROR')
C
C   REWIND UNIT 0
C
7000  CONTINUE
      CALL MSGOT('|REWIND TAPE?')
      CALL YESNO(IANS)
      IF(IANS .NE. 1)GO TO 7500
      CALL REWIN(MTUNIT)
      GO TO 7500
7300  CONTINUE
      CALL BELL
      WRITE(10,1003)
1003  FORMAT(1X,'DRAWING FILE FULL|',2(/1X,'DO NOT REWIND TAPE||'),/1X
\c,'
     *SAVE DRAWING FILE,GET NEW DRAWING FILE AND RECALL TAPEDWG OVERLA
\cY'
     *)
7500  CONTINUE
C
C     DONE
C
  500 ISUBF=999
      CALL MSGOT ("|PROGRAM FINISHED||")
      KEY=31
      CALL RWCON(KF,2)
      KP=0
      CALL OVRLY(1,IER)
      CALL EXIT2
      END
```

PROGRAM NAME: DWGDISK

Author: Lawrence Balcerak

Purpose of the program: **dwgdisk** reads a System 101 drawing file and writes an ASCII disk file containing the header card and data cards for each feature outline.

Data base: Geoindex

Computer: Data General Nova 1220

Operating system: System 101

Calling sequence: dwgdisk

Arguments: None

Subroutines called: **fclfl, fopfl, save, numin, msgot, yesno, rwcon, asflc, fcnot, ovrly, exit2, flnam, fdffl, xdmsg**

Common data referenced: */PUNCH/* Most Bendix subroutines read from or write to common blocks. Read "System 100 Programmers manual" (S100PM) for further information.

Input files: ***bordNMdw, gridNMdw, statNMdw, counNMdw, redNM, blueNM, greenNM***

Output files: ***bordNM, gridNM, statNM, counNM, redNM, blueNM, greenNM***

Arrays used: None

Error checking and reporting: None

Constants: None

Program logic:

1. Pause. Stops execution of program until a return is sent. Prints message:
   PAUSE FOR OPERATOR
2. Call **fclfl** (clears a file and releases slot 0).
3. Send message to terminal:
   NAME OF DISK OUTPUT FILE = ??

4. Call **flnam** to receive a file name from the keyboard and store in *name*.
   igood is the returned status code.
5. If *igood* = 1, go to step 6 (acceptable file name).
   If *igood* = 2, go to step 3 (file name too long).
   If *igood* = 3, go to step 78. Control d or cr on first character was entered.
6. Call **fopfl**. This opens the file name for writing and assigns it to file slot 0. *istat* is the monitor error code.
7. If *istat* is not equal to 0, go to step 9. At this point a nonzero value is an error.
8. Send message to terminal:
   OLD FILE-OK???.
   Call **yesno**. If yes, go to step 15. If no, go to step 2.
9. If *istat* is not equal to octal 204, go to step 14. This is the status code for a new file.
10. Send message to terminal:
    NEW FILE-OK??
    Call **yesno**. If yes, go to step 11. If no, go to step 2.
11. Call **fdffl**, which defines a file name *name*. *istat* is the monitor error code.
12. If *istat* is not equal to 0, go to step 14.
13. Call **fopfl**. This opens the file name for writing and assigns it to file slot 0. If there are no errors then *istat* = 0; go to step 15.
14. Call **xdmsg**(*istat*). This prints a disk operating system error message based on *istat*. Go to step 2.
15. Send message to terminal:
    !DO YOU WISH TO WRITE AN EOF FLAG ON THIS FILE??.
    If you must run this program several times, you will later concatenate the several files and will need an EOF flag only in the last file.
16. Call **save(1)**. This saves critical constants from the systems common blocks that are parameters describing the drawing file. These constants will be needed at the end of the program to restore the operation to the table with the same parameters.
17. Set *nif* = 0. This variable on the header card indicates grid, county, and so forth.
18. Send message to terminal:
    TYPE 2 DIGIT STATE NUMBER
    Call **numin** to receive number.
    Set *nor* = to returned number.
19. Send message to terminal:
    IS THIS THE GRID BEING WRITTEN??
20. Call **yesno**. If yes, go to step 21. If no, go to step 22.
21. Set *nif* = 991 (indicates the grid).
    Go to step 25.
22. Send message to terminal:
    IS THIS THE COUNTIES BEING WRITTEN??
23. Call **yesno**. If yes, got to step 24. If no, go to step 25.
24. Set *nif* = 992 (indicates the counties).
25. Set *iifno*(*i*) = 0 for *i* = 1,2000.
26. Set *kpt*(*kf*,*1*) = 1. Set the read pointer for the drawing file to the first record. Steps 27–30 will read the drawing file until the first text position that occurs in the drawing file is read.
27. Set *kp* = 1. When reading a drawing file, **rwcon** uses *x*(*kp*) and *y*(*kp*).
28. Call **rwcon** (*kf*,1). *kf* (equivalent to *k*(4)) is the active file. The 1 indicates a read.
29. If *key* = 31 (EOF), go to step 77.

Note: Several assumptions are made concerning the drawing file. Each outline begins with a text string identifying the feature number, subfeature number, span and second subfeature number, with a default of 0 for any absent text. There can be any number of line segments that make up an outline.

30. If *key* is not equal to 8 (text position), go to step 27. Steps 31–39 read through the drawing file counting the number of points for each outline and counting the number of outlines that have the same feature number. This must be done before punching starts because the information is on the header card.
31. Set *kount* = 1. This is a count of the number of points in an outline. The text position is the first point.
    Set *inum* = 1. This is a count of the number of outlines.
    Set *numtext* = 0. This is the count of how many lines of text are in the outline being read.
32. Set *kp* = 1.
    Call **rwcon** to read a record.
33. If *key* = 31 (EOF), go to step 37.
    If *key* = 8 (text position), go to step 37.
    If *key* = 16 (text string), go to step 34.
    If *key* = 1 (pen up), or if *key* = 6 (pen down), or if *key* = 7 (symbol position), add 1 to *kount*.
    Go to step 32.
34. Add 1 to *numtext* (one more text string found).
    If *numtext* is greater than 1, go to step 37. We are interested in only the first text string at this time.
    Set *ibuf*(*i*) = *id*(*i*). This contains the character string just read.
35. Call **asflc** to find the number, *temp*, represented by the text in *ibuf*(*i*).
    If *istat* = 1, go to step 32. An error code of 1 is returned for any abnormality.
36. Add 1 to the count of number of outlines that have same feature number as the new outline just started. Go to step 32.

37. Set *iburp*(*inum*) = *kount*. This is a count of the number of points for each outline.
    Set *kount* = 1. Start count over.
    Add 1 to *inum* (sequence number of next outline).
38. If *key* = 31 (EOF), go to step 40. There are two ways to reach this step: *key* = 31 or *key* = 8.
39. The only way to reach this step was if *key* = 8 (text position), which starts a new outline. Go to step 32. Steps 40–44 will read the drawing file until the first text position that occurs in the drawing file is read. These statements start reading the drawing file from the first record.
40. Set *kpt*(*kf*,*1*) = 1. This sets the read pointer for the drawing file to the first record.
41. Set *inum* = 1 (outline count).
42. Set *kp* = 1. Call **rwcon** to read a record.
43. If *key* = 31 (EOF), go to step 79. Then the program is almost finished.
44. If *key* is not equal to 8 (text position), go to step 42.
45. Set *kount* = 0. This is a counter for the number of text strings found for an outline. None is found yet.
    Set *knum* = 2. This is a counter for the number of the point to be processed. One is already processed.
    Set *isfno* = *iburp*(*inum*) (the number of points).
    Add 1 to *inum* (sequence number of next outline).
    Set *iup* = 1. This is the counter for the number of pen ups or symbol positions found in one outline. The first is treated differently from the rest.
    Set *isf* = 1. This will be 1 unless changed in a text string.
    Set *not* = 0.
    Set *ispan* = 0 (default values).
    Set *xp*(*1*) = *x1*.
    Set *yp*(*1*) = *y1*. This is the text position.
46. For *i* = *knum*, 6:
    Set *xp*(*i*) = 0.
    Set *yp*(*i*) = 0.
47. If *knum* = 1, set *knum* = 0. This will be equal to 1 when a card has just been punched and more points are needed to complete the outline. It then branches to the previous step, where it must be 1, but logic further along demands that it be 0.
48. Set *kp* = 1.
    Call **rwcon** to read a record.
49. If *key* = 31 (EOF), go to step 72.
50. If *key* = 8 (text position), go to step 72. This is true for all outlines except the first outline.
51. If *key* = 16 (text string), go to step 55.
52. If *key* = 1 (pen up) or 7 (symbol position), go to step 67.
53. If *key* = 6 (pen down), go to step 70.
54. Go to step 48.
55. Add 1 to *kount*. Another text string found for this outline.
56. If *kount* is greater than 4, go to step 48. *kount* should never be greater than 4, because there are only four possible pieces of information.
57. For *i* = 1, 5, set *ibuf*(*i*) = *id*(*i*). *id*(*i*) contains the text string from the record read in statement 48.
58. Call **asflc** to find the number, *temp*, represented by the text in *ibuf*(*i*).
59. If *istat* = 1, go to step 48. An error code of 1 is returned for any abnormality.
60. If *kount* = 1, go to step 66 (the first text string).
    If *nif* = 992, go to step 64. If this is the county file, *isf* represents a bordering county or other boundary.
61. If there is only one outline and *temp* is greater than 0, set *kount* = 4.
    This must be a second subfeature number, but there should not be four text strings.
    If there is only one outline and *temp* is less than 0, set *kount* = 3. This must be a span, but there should not be three text strings.
62. Set *i* = *kount* −1.
    If *i* = 1, go to step 64.
    If *i* = 2, go to step 65.
    If *i* = 3, go to step 66.
63. Set *isf* = *temp*. To get to this step, one of three conditions existed:
    this must have been the second text string with *ifno* greater than 1, or this is the counties, or *temp* = 0. Go to step 48.
64. Set *ispan* = *temp*. This was the third text string, or second text string with *temp* less than 0. Go to step 48.
65. *not* = *temp*. This was the fourth text string, or *temp* greater than 0 and only one outline. Go to step 48.
66. *iff* = *temp*. This is the feature number. *ifno* = *iifno*(*iif*). This is the count of outlines with same feature number. Go to step 48.
67. If *key* = 1 (pen up) and *iup* is greater than 1, go to step 70. This is another line segment that must be concatenated to previous segments.
68. Add 1 to *iup*, which is a flag to show what position the next pen up has in the outline (used in previous step).
69. Set *xp(2)* = *x1*
    Set *yp(2)* = *y1*.
    Program writes the header card to the disk file. Go to step 48.
70. Add 1 to *knum*, which is counter for next position.
    Set *xp*(*knum*) = *x1*.
    Set *yp* (*knum*) = *y1*.

71. If *knum* is less than 6, go to step 48.
    Otherwise, go to next step. The card should have six points to be written.
72. If *knum* = 0, go to step 74. The last card written had six points on it and finished an outline.
73. Program writes *xp*(*i*) and *yp*(*i*) to the disk file.
74. If *key* = 31 (EOF), go to step 78. Then, the program is almost finished.
75. If *key* = 8 (text position), go to step 45. A new outline is to be processed; default values must be reset.
76. Set *knum* = 1.
    Go to step 46. There are more points in this outline.
77. Program writes message to terminal:
    NO TEXT IN FILE!
    Go to step 79.
78. Program writes message to terminal:
    !DONE!
79. If *ieof* = 1, go to step 80.
    If *ieof* = 2, go to step 81. This is an indicator for whether or not an EOF flag is to be written. This was done in step 15.
80. Set *iif* = 9999 (the EOF flag). Program writes a header card to the disk file.
81. Close file slot 0.
    Call **fcnot** (" **7** ") several times to ring the bell. This produces an audible signal to the operator. Program writes message to terminal:
    PROGRAM FINISHED
    Set *kp* = 0.
    Call **save(2)** (restores critical constants).
    Call **overly**, a routine that overlays user memory with selected main program (returns control to the table).
    Call **exit2** (overlays signoff for the system).

```
         C      DWGDISK
C
C       WRITTEN   2MAR78   BALCERAK
C
C        SOURCE=<DWGDISK:F>
C        OBJECT=<DWGDISK:R>
C
C        PURPOSE:
C               TO READ A SYSTEM 101 DRAWING FILE LOADED ON
C               THE DRAWING TABLE-GET THE X,Y COORDINATES
C               OF THE TEXT REFERENCES AND OF THE LINES
C               AND WRITE TO DISK IN 12F6.3 FORMAT.
C
C               THE HEADER CARD FOR EACH OUTLINE WILL ALSO
C               BE WRITTEN WITH ALL RELAVENT INFORMATION.
C
C
C
        COMMON /BLK/ X(30),Y(30),A(10),K(30),KP,ID(80)
        COMMON /PNTR/ KPT(3,2)
        COMMON /LINBF/ LTYPE,LWIDE
        COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
        COMMON /EXEC/ IEXEC(64),REXEC(64)
        COMMON /DSKBF/ IDUM(3),LENG
        COMMON /IDENT/ IDA(3)
        COMMON /PUNCH/ XP(6),YP(6),IBUF(5),IBURP(2000),NAME(10),
       $ IIFNO(2000)
C
        EQUIVALENCE (K(1),K1)
        EQUIVALENCE (K(2),K2)
        EQUIVALENCE (K(4),KF)
        EQUIVALENCE (K(11),KEY)
        EQUIVALENCE (K(15),NCHAR)
        EQUIVALENCE (X(1),X1)
        EQUIVALENCE (Y(1),Y1)
```

```
C
C
C
C
      PAUSE FOR OPERATOR
1000  CALL FCLFL (0,IER)
1010  CALL MSGOT ("|NAME OF DISK OUTPUT FILE=??  ")
      CALL FLNAM (NAME,IGOOD)
      GO TO (1020,1010,500),IGOOD
1020  CALL FOPFL (NAME,0,1,ISTAT)
      IF (ISTAT .NE. 0)  GO TO 1030
      CALL MSGOT ("|OLD-FILE.  OK??  ")
      CALL YESNO (IANS)
      GO TO (1,1000),IANS
1030  IF (ISTAT .NE. 204K)  GO TO 1050
      CALL MSGOT ("|NEW-FILE.  OK??  ")
      CALL YESNO (IANS)
      GO TO (1040,1000),IANS
C
1040  CALL FDFFL(NAME,ISTAT)
      IF (ISTAT .NE. 0)  GO TO 1050
      CALL FOPFL (NAME,0,1,ISTAT)
      IF (ISTAT .EQ. 0)  GO TO 1
1050  CALL XDMSG (ISTAT)
      GO TO 1000
C
  1   CALL MSGOT ("|DO YOU WISH TO WRITE AN EOF FLAG ON THIS FILE??  "
\c)
      CALL YESNO (IEOF)
C
      CALL SAVE(1)
      NIF=0
      CALL NUMIN ("|TYPE 2 DIGIT STATE NUMBER  ",TEMP)
      NOR=TEMP
      CALL MSGOT ("|IS THIS THE GRID BEING WRITTEN??  ")
      CALL YESNO (ISTAT)
      GO TO (2,3),ISTAT
   2  NIF=991
      GO TO 5
   3  CALL MSGOT ("|IS THIS THE COUNTIES BEING WRITTEN??  ")
      CALL YESNO (ISTAT)
      GO TO (4,5),ISTAT
   4  NIF=992
   5  DO 6  I=1,2000
      IIFNO(I)=0
   6  CONTINUE
      KPT(KF,1)=1
C
C      START READING RECORDS AND WRITTING TO DISK
C
  10  KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31) GO TO 350
      IF (KEY .NE. 8)  GO TO 10
      KOUNT=1
      INUM=1
      NUMTEXT=0
   20 KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31)  GO TO 30
      IF (KEY .EQ. 16)  GO TO 25
      IF (KEY .EQ.  8)  GO TO 30
      IF ( (KEY .EQ. 1) .OR. (KEY .EQ. 6) .OR. (KEY .EQ. 7)  )
     *  KOUNT=KOUNT+1
      GO TO 20
C
   25 NUMTEXT=NUMTEXT+1
      IF (NUMTEXT .GT. 1)  GO TO 20
      DO 27  I=1,5
   27 IBUF(I)=ID(I)
      CALL ASFLC (IBUF,NCHAR,TEMP,ISTAT)
      IF (ISTAT .EQ. 1)  GO TO 10
      IIF=TEMP+0.5
C
   30 IBURP(INUM)=KOUNT
      KOUNT=1
      INUM=INUM+1
      IF (KEY .EQ. 31)  GO TO 40
      IIFNO(IIF)=IIFNO(IIF)+1
      NUMTEXT=0
      GO TO 20
C
C
   40 KPT(KF,1)=1
      INUM=1
C
C
   50 KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31)  GO TO 500
      IF (KEY .NE. 8)  GO TO 50
   55 KOUNT=0
      KNUM=2
      ISFNO=IBURP(INUM)
      INUM=INUM+1
      ISF=1
      IUP=1
      NOT=0
      ISPAN=0
      XP(1)=X1
      YP(1)=Y1
   60 DO 65 I=KNUM,6
      XP(I)=0
      YP(I)=0
   65 CONTINUE
      IF (KNUM .EQ. 1) KNUM=0
C
C
   70 KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31)  GO TO 140
      IF (KEY .EQ.  8)  GO TO 140
      IF (KEY .EQ. 16)  GO TO 80
      IF ( (KEY .EQ. 1) .OR. (KEY .EQ. 7)  )  GO TO 110
      IF (KEY .EQ. 6)   GO TO 130
      GO TO 70
C
C
   80 KOUNT=KOUNT+1
      IF (KOUNT .GT. 4)  GO TO 70
      DO 85 I=1,5
   85 IBUF(I)=ID(I)
      CALL ASFLC (IBUF,NCHAR,TEMP,ISTAT)
      IF (ISTAT .EQ. 1)  GO TO 70
      IF (KOUNT .EQ. 1)  GO TO 105
      IF (NIF .EQ. 992)  GO TO 90
      IF( (IFNO .EQ. 1) .AND. (TEMP .GT. 0.0) )  KOUNT=4
      IF ( (IFNO .EQ. 1) .AND. (TEMP .LT. 0.0) )  KOUNT=3
      I=KOUNT-1
      GO TO (90,95,100),I
   90 ISF=TEMP
      GO TO 70
   95 ISPAN=TEMP
      GO TO 70
  100 NOT=TEMP
      GO TO 70
  105 IIF=TEMP+0.5
      IFNO=IIFNO(IIF)
      GO TO 70
C
C
  110 IF ( (KEY .EQ. 1) .AND. (IUP .GT. 1) )  GO TO 130
      IUP=IUP+1
      XP(2)=X1
      YP(2)=Y1
      WRITE (0,120) IIF,IFNO,ISF,ISFNO,NOT,NOR,NIF,ISPAN
  120 FORMAT (1X,8I5)
      GO TO 70
C
C
  130 KNUM=KNUM+1
      XP(KNUM)=X1
      YP(KNUM)=Y1
      IF (KNUM .LT. 6)  GO TO 70
  140 IF (KNUM .EQ. 0) GO TO 160
      WRITE (0,150) (XP(I),YP(I),I=1,6)
  150 FORMAT (1X,12F6.3)
  160 IF (KEY .EQ. 31) GO TO 500
      IF (KEY .EQ.  8)  GO TO 55
      KNUM=1
      GO TO 60
C
C     DONE
C
  350 CALL FCNOT ("|NO TEXT IN FILE|")
```

```
C
 500   CALL FCNOT ("|DONE|")
       GO TO (510,520),IEOF
 510   IIF=9999
       WRITE (0,120) IIF,IFNO,ISF,ISFNO,NOT,NOR,NIF,ISPAN
 520   CALL FCLFL (0,IER)
       CALL FCNOT ("<7>")
       CALL FCNOT ("<7><7><7>")
       CALL FCNOT ("<7>")
       KP=0
       CALL SAVE(2)
       CALL OVRLY (1,IER)
       CALL EXIT2
       END
```

PROGRAM NAME: SELDISK

Author: Lawrence Balcerak
Purpose of the program: **seldisk** reads through an ASCII disk file containing coordinate outlines and places selected outlines into a drawing file.
Data base: Geoindex
Computer: Data General Nova 1220
Operating system: System 101
Calling sequence: seldisk
Arguments: None
Subroutines called: **fclfl**, **fcnot**, **flnam**, **fopfl**, **msgot**, **numin**, **ovrly**, **rwcon**, **save**, **xdmsg**, **asflc**
Common data referenced: */Punch/* Most Bendix subroutines read from or write to common blocks. Read "System 100 Programmers Manual" (S100PM) for further information.
Input files: None
Output files: None
Arrays used: None
Error checking and reporting: None
Constants: None
Program logic:

1. Call **save(1)** (saves critical constants from the systems common blocks that are parameters describing the drawing file). These will be needed at the end of the program to restore operation to the table with the same parameters.
2. Set *mrflg* = 0. *mrflg* is the mirror flag. 0 indicates no mirroring.
   Set *sfact* = 1. *sfact* is scale factor.
   Set *lnmod* = 1. *lnmod* is line type. 1 indicates a solid line.
   Set *lnwin* = 0. *lnwin* is pen number (initial value, which will be changed at a later time).
   Set *ifont* = 0 (system requirement).
   Set *angle* = 0 (angle of rotation for text).
   Set *justh* = 1. *justh* is horizontal text justification. 1 indicates that the text will be left justified starting at the text location.
   Set *justv* = 1. *justv* is vertical text justification. 1 indicates that the text will be above the text location.
3. Send message to terminal:
   SELDISK OVERLAY
   PAUSE TURN ON CARD READER
   and wait for the return to be pushed. This gives a chance to make sure that the card reader has been prepared correctly.

Steps 4–6 open the card reader for use.

4. Call **fclfl** (clears a file and releases slot 0).
5. Call **fopfl** (opens the card reader for reading and assigns it to slot 0).
   If *ier* is equal to 0, go to step 7. Otherwise, go to step 6. *ier* is the monitor error code (See S100PM, p. 236).
6. Write message to terminal:
   FOPFL ERROR
   and then go to step 72.
7. Call **numin**. This sends the message:
   !CHARACTER HEIGHT =
   to the terminal and waits for an answer, which it places in the real variable *texth*. This is usually 0.14 inches.
8. If *texth* is less than 0, go back to step 7. The system will accept a zero height but not a negative height.
9. Call **numin**. Sends message:
   !SYMBOL # =
   and receives answer into *temp*.
   Set *nsymb* = *temp*. Change to an integer.
10. If *nsymb* is greater than 0, go to step 11.
    Otherwise, go back to step 9. This must be a positive integer.
11. If *nsymb* is greater than 200, go back to step 9. The range of possible symbol numbers is 1–200.

12. Set ***sfact*** = 1.5 times ***texth***. This is the symbol scale factor. Our standard symbol for single points (a small triangle) is constructed 1 inch high. We usually plot it at 0.21 inches.
13. Set ***icount*** = 1. Index counter for the array ***ifile***, which will contain the feature and subfeature numbers read from the ***T-file***.
14. Read a feature and subfeature number with a format of (I8,I2) and place into ***ifile***(***icount***,1) and ***ifile***(***icount***,2).
15. If the feature number = -1, go to step 17. This is a flag for the end of the ***T-file***.
16. Add 1 to ***icount*** (to read another card).
    Go to step 14.
17. Subtract 1 from ***icount***. We don't want to count the flag.

Steps 18–26 open the coordinate outline file for reading.

18. Call **fclfl** (clears a file and releases slot 0).
19. Send message to terminal:
    NAME OF COORDINATE OUTLINE FILE = ??
    and wait for an answer.
20. Call **flnam** to receive the file name, and place it in the array ***name***.
    igood is the returned status code.
21. If ***igood*** = 1, go to step 22. An acceptable file name has been read in.
    If ***igood*** = 2, go back to step 19 (file name too long).
    If ***igood*** = 3, go to step 72. Control d or cr on first character was entered.
22. Call **fopfl**. This opens the file ***name*** for reading and assigns it to slot 0. ***istat*** is the monitor error code.
23. If ***istat*** is not equal to octal 204, go to step 25. This is the code for a new file. This will be the most common error.
24. Send message to terminal:
    !NEW FILE TRY AGAIN!!
    Go to step 18.
25. If ***istat*** is equal to 0, go to step 27. This indicates an old file that has no problems in opening.
26. Call **xdmsg**(***istat***). This prints a disk operating system error message based on ***istat***. Go to step 18 to try again.
27. Add 1 to ***lnwid***. This is the pen number, which changes whenever a new outline starts.
    If ***lnwid*** is greater than 3, then set ***lnwid*** equal to 1. Only three pens are on this plotter.
28. Read a header card. The feature, subfeature, second subfeature, and span are read in as characters. The rest are read as integers.
29. Set ***nchr1*** = 5.
    Set ***nchr2*** = 5.
    Set ***nchr3*** = 5.
    Set ***nchr4*** = 5. These are the character counts for each of the four strings read from the header cards.
30. Check each of the four character strings for blanks. Subtract 1 from the character count for each blank found.
31. Set ***J*** = 5 −***nchr1***. This is the number of blank characters.
32. In the character string, ***ls*** (feature number) divides each nonblank character by octal 400. This moves the bit pattern from the left half of the word to the right half.
    Set the array ***ibuf*** starting at element 1 equal to the right justified nonblank characters.
33. Repeat steps 31 and 32 for the subfeature number, ***lsf***, and store in ***ibuf*** starting at element 6.
34. Repeat steps 31 and 32 for the second subfeature number, ***lsf2***, and store in ***ibuf*** starting at element 11.
35. Repeat steps 31 and 32 for the span, ***lspan***, and store in ***ibuf*** starting at element 16. In the last three steps, there is a possibility that the number of characters is 0 (a blank field on the card). This will be accounted for when the subroutine **asflc** is called for each number.
36. Call **asflc** to change the ***nchr1*** characters starting at ***ibuf***(***1***) to the real number ***temp***.
    Set ***isubf*** = ***temp*** + 0.5. Change to an integer, but add 0.5 first to make sure the number is truncated correctly. This is the feature number as well as the subfile number.
37. If ***isubf*** is equal to 9999, go to step 72. This is the end-of-file flag, EOF, so the job is finished. The system does not recognize an end = option in a read statement, hence the need for the end-of-file flag.
    If ***isubf*** is greater than 1,000, subtract 1,000 from ***isubf***.
    If ***isubf*** is equal to 1,000, set ***isubf*** = 998. More than 1,000 outlines are possible, but only 999 subfiles are.
38. Call **asflc** using the ***nchr2*** characters starting at ***ibuf***(6) to find ***isf*** the subfeature number. ***istat*** is the returned error code that is 0 for no errors and equal to 1 for an error. The possible errors are as follows: no characters, a nonnumeric character, more than one plus or minus sign or decimal point, a plus or a minus sign somewhere other than position number one.
39. If ***istat*** is equal to 1, set ***isf*** = 1. This will be the default value.
40. Call **asflc** using ***nchr3*** characters starting at ***ibuf***(***11***) to find ***not***, the second subfeature number.

41. If *istat* is equal to 1, set *not* = 0 (the default value).
42. Call **asflc** using *nchr4* characters starting at *ibuf(16)* to find *ispan*, the span (should be negative or zero).
43. If *istat* is equal to 1, set *ispan* = 0 (the default value).
44. Go through the *ifile* array to see if an entry matches the feature number, *isubf*, and the subfeature number, *isf*.
    If a match is found, go to step 47.
45. Read a data point record. Skip this outline.
46. Subtract 6 from *isfno*. Six data points are on each record.
    If there are more data point records for this outline, *isfno* is greater than 0, go to step 44. Otherwise, go to step 28 to read the next header card.
47. Set *ie* equal to the minimum of (*6,isfno*). There are a maximum of six points per record.
48. Read *ie* data points from the next record into *xx* and *yy*.
49. Set *x1* = *xx(1)*.
    Set *y1* = *yy(1)*. This is the data point written to the drawing file when **rwcon** is called.
50. Set *nchar* = *nchr1*. This is the number of characters to be written as a text string when **rwcon** is called.
    For *i* = 1, *nchr1*, set *id(i)* = *ls(i)* (feature number). *id* is the array from which **rwcon** gets the text string.
51. Set *key* = 8 (the indicator that is a text position).
52. Set *kp* = 1. When writing a drawing file record, **rwcon** uses *x(kp)* and *y(kp)*.
    Call **rwcon**(*kf,2*). *kf* (equivalent to *k(4)*) is the active file. The 2 indicates a write.
53. Set *key* = 16 (text string indicator).
    Call **rwcon** to write a record. This writes the text string record containing the feature number.
54. Write the feature number to the terminal. This leaves a record of what has been done to date, which may be needed if there is some sort of system failure.
55. If the subfeature number is not to be placed as text, go to step 57.
56. Set *nchar* = *nchr2*.
    Set *id(i)* = *lsf(i)*, for *i* = 1, *nchr2*.
    Set *key* = 16.
    Call **rwcon** to write a text string record.
57. If the span is not to be placed as text, go to step 59.
58. Set *nchar* = *nchr4*.
    Set *id(i)* = *lspan(i)*, for *i* = 1, *nchr4*.
    Set *key* = 16.
    Call **rwcon** to write a text string record.
59. If the second subfeature number is not to be placed as text, go to 61.
60. Set *nchar* = *nchr3*.
    Set *id(i)* = *lsf2(i)*, for *i* = 1, *nchr3*.
    Set *key* = 16.
    Call **rwcon** to write a text string record.
61. Set *x1* = *xx(2)*.
    Set *y1* = *yy(2)*. Process the second point.
62. If *isfno* is greater than 2, go to step 64. Greater than 2 indicates a line segment. If equal to 2, it would indicate a single point.
63. Set *key* = 7 (the indicator for a symbol).
    Set *kp* = 1.
    Call **rwcon** to write a symbol record.
    Go to step 27 to start on the next outline.
64. Set *key* = 1 (the indicator for a pen up).
    Set *kp* = 1.
    Call **rwcon** to write a pen-up record.
    Set *jj* = 3. We have already processed the first 2 points.
    Go to step 66.
65. Read *ie* data points from the next record into *xx* and *yy*.
66. Set *key* = 6 (the indicator for pen down).
    Set *kp* = 1.
67. Do for *i* = *jj*, *ie*.
    Set *x1* = *yy(i)*.
    Set *y1* = *yy(i)*.
    Call **rwcon** to write a pen-down record.
68. Set *jj* = 1 (will start with first data point next time).
69. If there are no more data points for this outline, go to step 27 to start on the next.
70. Subtract 6 from *isfno*. This computes how many more points are left to complete the outline.
71. Set *ie* = minumum of (*6,isfno*).
    Go to step 64.
72. Set *isubf* = 999. Subfile 999 indicates that the whole drawing file is being referred to.
73. Set *key* = 31 (the indicator for an EOF).
    Call **rwcon** to write an EOF record.
    Call **fclfl** to clear the disk file and release slot 0.
74. Call **fcnot** ("*7*") several times to ring the bell. This produces an audible signal to the operator.
    Write to the terminal:
    !PROGRAM FINISHED!!
    Set *kp* = 1.
    Call **ovrly**. This overlays user memory with selected main program; it returns control to the table.
    Call **exit2** (overlays signoff for the system).

```
C      SELDISK
C
C      WRITTEN 2MAR78  BALCERAK
C
C      SOURCE=<SELDISK:F>
C      OBJECT=<SELDISK:R>
C
C      PURPOSE:
C              TO READ A GROUP OF PTS FROM A DISK FILE
C              AND CREATE A SYST 101 DWG FILE.  THERE WILL
C              BE A HEADER CARD FOLLOWED BY DATA CARDS
C              WITH 6 PTS PER CARD IN 12F6.3 FORMAT.
C              THE PROGRAM SELECTS ONLY CERTAIN FILES .
C
C
C.... REMARKS:
C.... THIS PROGRAM HAS KNOWLEDGE OF FILE STRUCTURE.
C
C      WHEN RWCON READS A RECORD IT TRANSFERS
C      THE DATA TO COMMON /LINBF/ LTYPE,LWIDE
C      AND TO COMMON /SYMBF/ MIRSY,SKLSY
C
C      WHEN RWCON WRITES A RECORD IT TRANSFERS THE DATA
C      FROM COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
C
C      THE CURRENT SYST 100 VALUES FOR LINE WIDTH
C      AND TYPE ARE STORED IN COMMON /MENU1/
C
C
       COMMON /BLK/X(30),Y(30),A(10),K(30),KP,ID(80)
       COMMON /PNTR/KPT(3,2)
       COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
       COMMON /EXEC/ IEXEC(64),REXEC(64)
       COMMON /DSKBF/ IDUM(3),LENG
       COMMON /PUNCH/ XX(6),YY(6)
       COMMON /CRDWG/ LS(5), LSF(5),ISFNO,LSF2(5),LSPAN(5),NAME(10)
       COMMON /CRDWG/  ISF,ISF2,IBLANK,IBUF(20),IFILE(1000,2)
       COMMON /FONT/ IFONT
C
C      EQUIVALENCE (K(1),K1)
C      EQUIVALENCE (K(2),K2)
       EQUIVALENCE (K(4),KF)
       EQUIVALENCE (K(11),KEY)
       EQUIVALENCE (K(12),ISUBF)
       EQUIVALENCE (K(13),NSYMB)
       EQUIVALENCE (K(15),NCHAR)
       EQUIVALENCE (IEXEC(31),JUSTH)
       EQUIVALENCE (IEXEC(32),JUSTV)
       EQUIVALENCE (X(1),X1)
       EQUIVALENCE (Y(1),Y1)
       EQUIVALENCE (A(1),ANGLE)
       EQUIVALENCE (A(2),TEXTH)
```

```
C
      DATA IBLANK /2H  /
C
C
C
C
C
      CALL SAVE(1)
      MRFLG=0
      SFACT=1
      LNMOD=1
      LNWID=0
      IFONT=0
      ANGLE=0.
      JUSTH=1
      JUSTV=1
C
      CALL MSGOT("|SELDISK OVERLAY")
      PAUSE TURN ON CARD READER
      CALL FCLFL(0,IER)
      CALL FOPFL("/CDR",0,0,IER)
      IF (IER) 150,200,150
  150 CALL MSGOT("|FOPFL ERROR|")
      GO TO 500
C
  200 CALL NUMIN("|CHARACTER HEIGHT=",TEXTH)
      IF (TEXTH) 200,210,210
  210 CALL NUMIN("|SYMBOL # =",TEMP)
      NSYMB=TEMP
      IF (NSYMB) 210,210,220
  220 IF (NSYMB-200) 230,230,210
  230 SFACT=1.5*TEXTH
C
      ICOUNT=1
 240  READ (0,250) IFILE(ICOUNT,1),IFILE(ICOUNT,2)
 250  FORMAT (I8,I2,)
      IF (IFILE(ICOUNT,1) .EQ. -1)  GO TO 260
      ICOUNT=ICOUNT+1
      GO TO 240
 260  ICOUNT=ICOUNT-1
C
 270  CALL FCLFL (0,IER)
 280  CALL MSGOT ("|NAME OF COORDINATE OUTLINE FILE=??  ")
      CALL FLNAM (NAME,IGOOD)
      GO TO (290,280,500),IGOOD
 290  CALL FOPFL (NAME,0,0,ISTAT)
      IF (ISTAT .NE. 204K)  GO TO 295
      CALL MSGOT ("|NEW FILE   TRY AGAIN||  ")
      GO TO 270
 295  IF (ISTAT .EQ. 0)  GO TO 300
      CALL XDMSG (ISTAT)
      GO TO 270
C
  300 LNWID=LNWID+1
      IF (LNWID.GT.3) LNWID=1
```

```
C
C     READ HEADER CARD
C
 305  READ (0,310) (LS(I),I=1,5),IFNO,(LSF(J),J=1,5),
     * ISFNO,(LSF2(M),M=1,5),NOR,NIF,(LSPAN(L),L=1,5)
 310  FORMAT (5A1,I5,5A1,I5,5A1,2I5,5A1)
      NCHR1=5
      NCHR2=5
      NCHR3=5
      NCHR4=5
C
C
      DO 320 I=1,5
      IF (LS(I) .EQ. IBLANK) NCHR1=NCHR1-1
      IF (LSF(I) .EQ. IBLANK)  NCHR2=NCHR2-1
      IF (LSF2(I).EQ. IBLANK)  NCHR3=NCHR3-1
      IF (LSPAN(I) .EQ. IBLANK) NCHR4=NCHR4-1
 320  CONTINUE
C
C
      J=5-NCHR1
      DO 330 I=1,NCHR1
      LS(I)=LS(I+J)/400K
      IBUF(I)=LS(I)
 330  CONTINUE
C
      J=5-NCHR2
      DO 340 I=1,NCHR2
      LSF(I)=LSF(I+J)/400K
      IBUF(I+5)=LSF(I)
 340  CONTINUE
      IF (NCHR3.EQ.0) GO TO 352
C
C
      J=5-NCHR3
      DO 350 I=1,NCHR3
      LSF2(I)=LSF2(I+J)/400K
      IBUF(I+10)=LSF2(I)
 350  CONTINUE
C
C
 352  J=5-NCHR4
      DO 353 I=1,NCHR4
      LSPAN(I)=LSPAN(I+J)/400K
      IBUF(I+15)=LSPAN(I)
 353  CONTINUE
C
C
      CALL ASFLC (IBUF(1),NCHR1,TEMP,ISTAT)
      ISUBF=TEMP+.5
C
      IF (ISUBF .EQ. 9999)  GO TO 500
      IF (ISUBF .GT. 1000)  ISUBF=ISUBF-1000
      IF (ISUBF .EQ. 1000) ISUBF=998
```

```
C
      CALL ASFLC(IBUF(6),NCHR2,TEMP,ISTAT)
      ISF=TEMP+.5
      IF (ISTAT .EQ. 1) ISF=1
      CALL ASFLC (IBUF(11),NCHR3,TEMP,ISTAT)
      NOT=TEMP+.5
      IF (ISTAT .EQ. 1)  NOT=0
      CALL ASFLC (IBUF(16),NCHR4,TEMP,ISTAT)
      ISPAN=TEMP+.5
      IF (ISTAT .EQ. 1) ISPAN=0
C
      DO 1000 I=1,ICOUNT
      IF (IFILE(I,1) .NE. ISUBF) GO TO 1000
      IF (IFILE(I,2) .EQ. ISF)  GO TO 1050
 1000 CONTINUE
 1010 READ (0,355) XX(1)
      ISFNO=ISFNO-6
      IF (ISFNO) 305,305,1010
C
 1050 IE=ISFNO
      IF(ISFNO .GE. 6)IE=6
C
C     READ FIRST DATA CARD
C
      READ(0,355)(XX(I),YY(I),I=1,IE)
  355 FORMAT(12F6.3)
      X1=XX(1)
      Y1=YY(1)
      NCHAR=NCHR1
      DO 360 I=1,NCHR1
      ID(I)=LS(I)
  360 CONTINUE
      KEY=8
      KP=1
      CALL RWCON(KF,2)
      KEY=16
      CALL RWCON(KF,2)
C
      WRITE (10,365)  ISUBF
 365  FORMAT (I6)
C
      IF ( (ISF .EQ. 1) .AND. (IFNO .EQ. 1) )  GO TO 375
      NCHAR=NCHR2
      DO 370 I=1,NCHR2
      ID(I)=LSF(I)
  370 CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
 375  IF (ISPAN .EQ. 0)  GO TO 380
      NCHAR=NCHR4
      DO 377 I=1,NCHR4
      ID(I)=LSPAN(I)
 377  CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
  380 IF (NOT  .EQ.0) GO TO 400
      NCHAR=NCHR3
      DO 390 I=1,NCHR3
      ID(I)=LSF2(I)
  390 CONTINUE
      KEY=16
      CALL RWCON(KF,2)
C
C
  400 X1=XX(2)
      Y1=YY(2)
C
C     SYMBOL NEEDED IF THIS IS SINGLE PT
C
      IF (ISFNO.GT.2) GO TO 410
      KEY=7
      KP=1
      CALL RWCON(KF,2)
      GO TO 300
C
C     WRITE PEN UP
C
  410 KEY=1
      KP=1
      CALL RWCON(KF,2)
      JJ=3
      GO TO 430
C
  420 READ(0,355) (XX(I),YY(I),I=1,IE)
C
C     WRITE PEN DOWN
C
  430 KEY=6
      KP=1
      DO 440 I=JJ,IE
      X1=XX(I)
      Y1=YY(I)
      CALL RWCON(KF,2)
  440 CONTINUE
      JJ=1
      IF (ISFNO-6) 300,300,450
  450 ISFNO=ISFNO-6
      IE=ISFNO
      IF (ISFNO.GE.6) IE=6
      GO TO 420
C
C     DONE
C
```

```
500 ISUBF=999
    KEY=31
    CALL RWCON(KF,2)
    CALL FCLFL(0,IER)
    CALL FCNOT ("<7>")
    CALL FCNOT ("<7>")
    CALL FCNOT ("<7>")
    CALL MSGOT ("!PROGRAM FINISHED!!")
    KP=0
    CALL SAVE (2)
    CALL OVRLY(1,IER)
    CALL EXIT2
    END
```

PROGRAM NAME: DWGTAPE

Author: Lawrence Balcerak

Purpose of the program: **dwgtape** reads a System 101 drawing file, writes to tape the binary representation of the header card and data cards for each feature outline. Options are to write all or only one of the feature numbers. Also either all or only the first data card for each outline can be written.

Data base: Geoindex

Computer: Data General Nova 1220

Operating system: System 101

Calling sequence: dwgtape

Arguments: None

Subroutines called: **save**, **numin**, **msgot**, **yesno**, **rwcon**, **asflc**, **fcnot**, **ovrly**, **exit2**, **rdtape**, **wrtape**, **wreof**

Common data referenced: */Punch/* Most Bendix subroutines read from or write to common blocks. Read "SYSTEM 100/101 Programmers Manual" (SC100PM) for further information.

Input files: None

Output files: None

Arrays used: None

Called by: None

Error checking and reporting: None

Constants: None

Program logic: This program is designed to write the exact bit pattern of integers and real numbers used on an IBM/370. The tape will be read using a 20A4 format that preserves the bit pattern. The Bendix minicomputer has a four-byte real number, which is exactly the same as the IBM real number. However, the integer is only two bytes versus four for the IBM.

Using a 12-element integer array, which is made equivalent to a 6-element real array, the program writes the integers in binary to the tape. When writing the header card, expand each integer to four bytes by writing alternate zeros.

1. Pause. Stops execution of program until a return is sent. Prints message:
   TAPE UNIT NO. (0 OR 1)
2. Call **save(1)** (saves critical constant).
3. Call **numin** (sends message to terminal asking what feature number you want punched). This real number is then placed in ***temp***. Use 9999 if you want all features.
4. Set ***ifnum*** = ***temp***. Change to an integer.
5. Send message to terminal:
   DO YOU WISH THE FIRST DATA CARD ONLY??
6. Call **yesno** to receive a yes or no, which then sets the variable ***ianswer*** = 1(yes) or 2(no). It will not accept any other answer.
7. Set ***nif*** = 0. This variable on the header card indicates grid, county, and so forth.
8. Send message to terminal:
   TYPE 2 DIGIT STATE NUMBER
   Call **numin** to receive number.
9. Set ***nor*** = State number, which user types in.
10. Send message to terminal:
    IS THIS THE GRID BEING PUNCHED??
11. Call **yesno**.
    If yes, go to step 12. If no, go to step 13.
12. Set ***nif*** = 991 (indicates the grid).
    Go to step 16.
13. Send message to terminal:
    IS THIS THE COUNTIES BEING PUNCHED??
14. Call **yesno**.
    If yes, go to step 15. If no, go to step 16.
15. Set ***nif*** = 992 (indicates counties).
16. Set ***iifno***(***I***) = 0 for ***I*** = 1,1500.
17. Send message to terminal:
    SKIP FILES??
    Call **yesno** to receive answer.
    If yes, go to step 18. If no, go to step 23.

18. Call **numin**. Send message:
    HOW MANY FILES ??
    Receive answer and store in ***temp***.
19. Set ***iskip*** = ***temp***.
    If ***iskip*** is less than 0, go to step 18.
    If ***iskip*** = 0, go to step 23.
    If ***iskip*** is greater than 0, go to step 20.
20. Do steps 21–22 for ***i*** = 1, ***iskip***
21. Call **rdtape** to read the tape. ***jstat*** is the status return code.
22. If ***jstat*** = 4 (EOF), go to step 20 to read next file.
    If ***jstat*** does not = 0, go to step 84.
    Otherwise, go to step 21.
23. Set ***kpt(kf,1)*** = 1. Sets the read pointer for the drawing file to the first record. Steps 24–28 will read the drawing file until the first text position is read for the appropriate feature number.
24. Set ***kp*** = 1. When reading a drawing file, **rwcon** uses ***x(kp)*** and ***y(kp)***.
25. Call **rwcon(*kf,1*)**. ***kf***, equivalent to ***k(4)***, is the active file. The 1 indicates a read.
26. If ***key*** = 31 (EOF), go to step 83.

Note: Several assumptions are made about the drawing file. The subfile number is the same as the feature number. Each outline begins with a text string identifying the feature number, subfeature number, span and second subfeature number, with a default of 0 for any absent text. Any number of line segments can make up an outline.

27. If ***isubf***, subfile number, is not equal to ***ifnum***, and if ***ifnum*** is not equal to 9999, go to step 24. This searches for the appropriate feature number.
28. Set ***kount*** = 1. This is a count of the number of points in an outline. The text position is the first point.
    Set ***inum*** = 1 (a count of the number of outlines).
29. Set ***iifno(isubf)*** = 1. This is the count of how many outlines that have the same feature number.
30. Set ***kp*** =1.
    Call **rwcon** to read a record.
31. If ***key*** = 31 (EOF), go to step 36.
32. If this is not an appropriate outline (check ***isubf***), go to step 30.
33. If ***key*** = 8 (text position), go to step 36.
34. If ***key*** = 1 (pen up), or
    If ***key*** = 6 (pen down), or
    If ***key*** = 7 (symbol position), add 1 to ***kount***.
35. Go step 30.
36. Set ***iburp(inum)*** = ***kount***. This is a count of the number of points for each outline.
    Set ***kount*** = 1. Start count over.
    Add 1 to ***inum*** (sequence number of next outline).
37. If ***key*** = 31 (EOF), go to step 39. There are two ways to reach this step: ***key*** = 31 or ***key*** = 8.
38. Add 1 to the count of number of outlines that have same feature number as the new outline just started. This step can be reached only if ***key*** = 8 (text position), which starts a new outline. Go to step 30.

Steps 39–43 will read the drawing file until the first text position is read for the appropriate feature number.

39. Set ***kpt(kf***,1) = 1. Sets the read pointer for the drawing file to the first record.
40. Set ***inum*** = 1 (outline count).
41. Set ***kp*** = 1.
    Call **rwcon** to read a record.
42. If ***key*** = 31 (EOF), go to step 85. (Program is finished.)
43. If this is not an appropriate outline (check ***isubf***), go to step 41.
44. Set ***kount*** = 0. This is a counter for the number of text string found for an outline. None is found yet.
    Set ***knum*** = 2. This is a counter for the number of the points to be processed. One is already processed.
    Set ***iif*** = ***isubf*** (feature number).
    Set ***ifno*** = ***iifno(isubf)***. This is the number of outlines with same feature number.
    Set ***isfno*** = ***iburp(inum)*** (the number of points).
    Add 1 to ***inum***. This is a sequence number of the next outline.
    Set ***isf*** = 1. This will be 1 less changed in a text string.
    Set ***iup*** = 1. This is a counter for the number of pen ups or symbol positions in one outline. The first is treated differently from the rest.
    Set ***not*** = 0; set ***ispan*** = 0 (default values).
    Set ***xp(1)*** = ***x1***; set ***yp(1)*** = ***y1***. This is the text position.
45. For ***i*** = ***knum***, 6, set ***xp(i)*** = 0; set ***yp(i)*** =0.
46. If ***knum*** = 1, set ***knum*** = 0. This will be equal to 1 when a card image has just been written and more points are needed to complete the outline. It then branches to the previous step where it must be a 1, but logic further along demands that it be 0.
47. Set ***kp*** = 1; call **rwcon** to read a record.
48. If ***key*** = 31 (EOF), go to step 76.
49. If this is not an appropriate outline (check ***isubf***), go to step 47. More than one outline may have the same feature number.
50. If ***key*** = 8 (text position), go to step 76. This would be true when an outline other than the first comes up.
51. If ***key*** = 16 (text string), go to step 55.

52. If *key* = 1 (pen up) or 7 (symbol position), go to step 68.
53. If *key* = 6 (pen down), go to step 74.
54. Go to step 47.
55. Add 1 to *kount*. Another text is string found for this outline.
56. If *kount* = 1, or *kount* is greater than 4, go to step 47. If equal to 1, it is the feature number, which is the same as the subfile number. It should never be greater than 4.
57. For *i* = 1, 5, set *ibuf(i)* = *id(i)* (the text string).
58. Call **asflc** to find the number, *temp*, represented by the text in *ibuf(i)*.
59. If *istat* = 1, go to step 47. An error code of 1 is returned for any abnormality.
60. If *nif* = 992, go to step 64. If this is the counties, *isf* represents a bordering county or other boundary.
61. If there is only one outline and *temp* is greater than 0, set *kount* = 4. This must be a second subfeature number, but there may not be four text strings.
62. If there is only one outline and *temp* is less than 0, set *kount* = 3. This must be a span, but there may not be three text strings.
63. Set *i* = *kount* −1. If *i* = 1, go to step 64. If *i* = 2, go to step 65. If *i* = 3, go to step 66.
64. Set *isf* = *temp*. This must have been the second text string with *ifno* greater than 1, or this is the counties, or *temp* = 0. Go to step 47.
65. Set *ispan* = *temp*. This was the third text string, or second text string with *temp* less than 0. Go to step 47.
66. Set *not* = *temp*. This was the fourth text string, or *temp* greater than 0 and only one outline. Go to step 47.
67. If *key* = 1 (pen up) and *iup* is greater than 1, go to step 74. If these two conditions are met, then this is another line segment that must be concatenated to previous segments.
68. Add 1 to *iup*. This is a flag to show what position the next pen up has in the outline; used in previous step.
69. Set *xp(2)* = *x1*; set *yp(2)* = *y1*.
70. Set *iout(i)* = *izero*, for *i* = 1, 24.
71. Set *iout(2)* = *iif* (feature number).
    Set *iout(4)* = *ifno* (the number of outlines).
    Set *iout(6)* = *isf* (subfeature number).
    Set *iout(8)* = *isfno* (number of points).
    Set *iout(10)* = *not* (second subfeature number).
    Set *iout(12)* = *nor* (State number).
    Set *iout(14)* = *nif* (graticule identifier).
    Set *iout(16)* = *ispan* (span).
72. If *ispan* is less than 0, set *iout(15)* = *ineg* (makes the whole word negative.
73. Call **wrtape** to write the header card to the tape.
    If *istat* (error code) not = 0, go to step 84.
    Otherwise, go to step 47.
74. Add 1 to *knum* (counter for next position).
    Set *xp(knum)* = *x1*; set *yp(knum)* = *y1*.
75. If *knum* is less than 6, go to step 47.
    Otherwise, go to the next step. The card should have six points to be punched.
76. If *knum* = 0, go to step 80. Then, the last card punched had six points on it and finished an outline.
77. Set *m* = −1; set *l* = −3 (to start counters used later at the proper place in the arrays).
    Do steps 78–79 for *i* = 1, 12, 2.
78. Add 4 to *l*.
    Set *iout(l)* = *itemp(i)*.
    Set *iout(l + 1)* = *itemp(i + 1)*.
    Add 4 to *m*.
    Set *iout(m)* = *jtemp(i)*.
    Set *iout(m + 1)* = *jtemp(i + 1)*.
79. Call **wrtape** to write a data card to the tape.
    If *jstat* (error code) not = 0, go to step 84.
80. If *key* = 31 (EOF), go to step 85. Program is almost finished.
81. If *key* = 8 (text position), go to step 44. A new outline is to be processed; default values must be reset.
82. Set *knum* = 1.
    Go to step 45. More points are in this outline.
83. Write message to terminal:
    NO TEXT IN FILE!!
    Go to step 85.
84. Write *jstat* (error code) to the terminal.
85. Write message to terminal:
    !DONE!
    Call **wreof** to write EOF on tape.
    Set *kp* = 0.
    Call **save(2)** (restores critical constants).
    Call **ovrly** (overlays user memory with selected main program). Here it returns control to table.
    Call **exit2** (overlays signoff for the system).

```
C       DWGTAPE
C
C       WRITTEN   23NOV76    BALCERAK
C
C        SOURCE=<DWGTAPE:F>
C        OBJECT=<DWGTAPE:R>
```

```
C
C      PURPOSE:
C             TO READ A SYSTEM 101 DRAWING FILE LOADED ON
C             THE DRAWING TABLE-GET THE X,Y COORDINATES
C             OF THE TEXT REFERENCES AND OF THE LINES
C             AND WRITE TO TAPE IN 12F6.3 FORMAT.
C             OPTIONS INCLUDE WRITING ALL OR ONLY ONE OF
C             THE SUBFILES.  ALSO, ONLY THE FIRST DATA
C             CARD CAN BE WRITTEN OUT INSTEAD OF ALL.
C
C             THE HEADER CARD FOR EACH OUTLINE WILL ALSO
C             BE WRITTEN WITH ALL RELAVENT INFORMATION.
C
C
C
      COMMON /BLK/ X(30),Y(30),A(10),K(30),KP,ID(80)
      COMMON /PNTR/ KPT(3,2)
      COMMON /LINBF/ LTYPE,LWIDE
      COMMON /MENU1/ KODE,MRFLG,SFACT,LNMOD,LNWID
      COMMON /EXEC/ IEXEC(64),REXEC(64)
      COMMON /DSKBF/ IDUM(3),LENG
      COMMON /IDENT/ IDA(3)
      COMMON /PUNCH/ XP(6),YP(6),IBUF(5),IBURP(1000),IFNUM,IANSWER,
     $ IIFNO(1500),INEG,IZERO
      COMMON /UNIT/ MTUNIT
C
      DIMENSION ITEMP(12),JTEMP(12)
C
      EXTERNAL MT80
      EQUIVALENCE (K(1),K1)
      EQUIVALENCE (K(2),K2)
      EQUIVALENCE (K(4),KF)
      EQUIVALENCE (K(11),KEY)
      EQUIVALENCE (K(12),ISUBF)
      EQUIVALENCE (K(15),NCHAR)
      EQUIVALENCE (X(1),X1)
      EQUIVALENCE (Y(1),Y1)
      EQUIVALENCE (IEXEC(19),LNMSV)
      EQUIVALENCE (IEXEC(20),LNWSV)
      EQUIVALENCE (XP(1),ITEMP(1))
      EQUIVALENCE (YP(1),JTEMP(1))
C
C
      DATA INEG /177777K/
      DATA IZERO /000000K/

C
C
C
      CALL XIOIT(MT80)
      CALL SAVE(1)
      CALL NUMIN("|TAPE UNIT NO. (0 OR 1)",TEMP)
      MTUNIT=TEMP
      PAUSE MOUNT TAPE PLEASE
      CALL NUMIN ("|SUBFILE # =, TYPE 9999 FOR ALL",TEMP)
      IFNUM=IFIX(TEMP)
      CALL MSGOT ("|DO YOU WISH THE FIRST DATA CARD ONLY??")
      CALL YESNO (IANSWER)
      NIF=0
      CALL NUMIN ("|TY E 2 DIGET STATE NUMBER  ",TEMP)
      NOR=TEMP
      CALL MSGOT ("|IS THIS THE GRID BEING PUNCHED??  ")
      CALL YESNO (ISTAT)
      GO TO (2,3),ISTAT
    2 NIF=991
      GO TO 5
    3 CALL MSGOT ("|IS THIS THE COUNTIES BEING PUNCHED??  ")
      CALL YESNO (ISTAT)
      GO TO (4,5),ISTAT
    4 NIF=992
    5 DO 6  I=1,1500
      IIFNO(I)=0
    6 CONTINUE
C
C
      CALL MSGOT (" SKIP FILES??  ")
      CALL YESNO (IT)
      GO TO (7,12),IT
    7 CALL NUMIN(" HOW MANY FILES??  ",TEMP)
      ISKIP=TEMP
      IF (ISKIP) 7,12,8
    8 DO 11 I=1,ISKIP
    9 CALL RDTAPE (MTUNIT,IBURP,1000,0,JACNT,JSTAT)
      IF (JSTAT .EQ. 4)  GO TO 11
      IF (JSTAT .NE. 0)  GO TO 12
      GO TO 9
   11 CONTINUE
C
   12 CONTINUE
C
C
      KPT(KF,1)=1
C`
C      START READING RECORDS AND PUNCHING OUT CARDS
C
   10 KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31) GO TO 350
      IF ( (ISUBF .NE. IFNUM) .AND. (IFNUM .NE. 9999) )  GO TO 10
      IF (KEY .NE. 8)  GO TO 10
      KOUNT=1
      INUM=1
      IIFNO(ISUBF)=1
   20 KP=1
      CALL RWCON (KF,1)
      IF (KEY .EQ. 31)  GO TO 30
      IF ( (ISUBF .NE. IFNUM) .AND. (IFNUM .NE. 9999) )  GO TO 20
      IF (KEY .EQ.  8)  GO TO 30
      IF ( (KEY .EQ. 1) .OR. (KEY .EQ. 6) .OR. (KEY .EQ. 7)  )
```

```
     *  KOUNT=KOUNT+1
        GO TO 20
C
C
   30   IBURP(INUM)=KOUNT
        KOUNT=1
        INUM=INUM+1
        IF (KEY .EQ. 31)  GO TO 40
        IIFNO(ISUBF)=IIFNO(ISUBF)+1
        GO TO 20
C
C
   40   KPT(KF,1)=1
        INUM=1
C
C
   50   KP=1
        CALL RWCON (KF,1)
        IF (KEY .EQ. 31)  GO TO 500
        IF ( (ISUBF .NE. IFNUM) .AND. (IFNUM .NE. 9999) )  GO TO 50
        IF (KEY .NE. 8)  GO TO 50
   55   KOUNT=0
        KNUM=2
        IIF=ISUBF
        IFNO=IIFNO(ISUBF)
        ISFNO=IBURP(INUM)
        INUM=INUM+1
        ISF=1
        IUP=1
        NOT=0
        ISPAN=0
        XP(1)=X1
        YP(1)=Y1
   60   DO 65 I=KNUM,6
        XP(I)=0.
        YP(I)=0.
   65   CONTINUE
        IF (KNUM .EQ. 1) KNUM=0
C
C
   70   KP=1
        CALL RWCON (KF,1)
        IF (KEY .EQ. 31)  GO TO 140
        IF ( (ISUBF .NE. IFNUM) .AND. (IFNUM .NE. 9999) )  GO TO 70
        IF (KEY .EQ.  8)  GO TO 140
        IF (KEY .EQ. 16)  GO TO 80
        IF ( (KEY .EQ. 1) .OR. (KEY .EQ. 7)  )  GO TO 110
        IF (KEY .EQ. 6)   GO TO 130
        GO TO 70
C
C
   80   KOUNT=KOUNT+1
        IF ( (KOUNT .EQ. 1) .OR. (KOUNT .GT. 4) )  GO TO 70
        DO 85 I=1,5
```

```
  85  IBUF(I)=ID(I)
      CALL ASFLC (IBUF,NCHAR,TEMP,ISTAT)
      IF (ISTAT .EQ. 1)  GO TO 70
      IF (NIF .EQ. 992)  GO TO 90
      IF( (IFNO .EQ. 1) .AND. (TEMP .GT. 0.0) )  KOUNT=4
      IF ( (IFNO .EQ. 1) .AND. (TEMP .LT. 0.0) )  KOUNT=3
      I=KOUNT-1
      GO TO (90,95,100),I
  90  ISF=TEMP
      GO TO 70
  95  ISPAN=TEMP
      GO TO 70
 100  NOT=TEMP
      GO TO 70
C
C
 110  IF ( (KEY .EQ. 1) .AND. (IUP .GT. 1) )  GO TO 130
      IUP=IUP+1
      XP(2)=X1
      YP(2)=Y1
      WRITE(22,120)IIF,IFNO,ISF,ISFNO,NOT,NOR,NIF,ISPAN
 120  FORMAT(1X,8I5)
      GO TO 70
C
C
 130  KNUM=KNUM+1
      XP(KNUM)=X1
      YP(KNUM)=Y1
      IF (KNUM .LT. 6)  GO TO 70
 140  IF (KNUM .EQ. 0) GO TO 160
      WRITE(22,150)(XP(I),YP(I),I=1,6)
 150  FORMAT(1X,12F6.3)
 160  IF (KEY .EQ. 31) GO TO 500
      IF (KEY .EQ.  8)  GO TO 55
      IF (IANSWER .EQ. 1)  GO TO 50
      KNUM=1
      GO TO 60
C
C     DONE
C
 350  CALL FCNOT ("|NO TEXT IN FILE|")
C
 500  CALL FCNOT ("|DONE|")
      CALL WREOF (MTUNIT,JSTAT)
      KP=0
      CALL SAVE(2)
      CALL OVRLY (1,IER)
      CALL EXIT2
      END
```

EXEC_COM NAME: VERSATEC.EC

Author: James Fisher

Purpose of the program: **versatec.ec**, written in Multics command language, reads the tape that was created on the Data General minicomputer and creates the Versatec border, grid, State, county, and coordinate files. If there is more than one coordinate file, the files must be combined on Multics by means of an editor, and then **sort.vers.coor.ec must be run**.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: ec versatec *nnnnnn* nof file1 . . . filen

Arguments:
nnnnnn–Volume number of the tape
nof–Number of files to be copied to disk
file1 . . . filen–Name of the files when copied to disk

Subroutines called: None

Common data referenced: None

Input files: User's tape

Output files: Files copied to disk

Arrays used: None

Called by: None

Error checking and reporting: None

Constants: None

Program logic:

1. To execute this exec_com, the operator must type: **ec versatec.ec**, *tape number*, *number of files*, *seg1 seg2* . . . *segN* where *seg1*, *seg2*, . . ., *segN* are the file names that are to be created. *seg1*, *seg2*, . . ., *segN* must not exceed a total of 32 characters because of the **value** command used extensively in this exec_com. The **value** command returns a character string associated with a named item in a user symbol table segment. This enables administrative exec_com segments to reference variables.
2. The exec_com uses the **tape_ibm** command with density 800, record size 80, and ASCII character mode.
3. The first file is read from tape and written to disk under the name given by *seg1*.
4. The number of file parameters is then checked against the number of files that have been written. When they are not equal, the next file is attached by the I/O command.
5. After the segment is attached, **copy_file** is used to write the segment to disk.
6. This process continues until the number of file parameters is equal to the number of files that have been written.
7. When the parameters are equal, the tape is renamed and the execution of the exec_com is ended.

```
& ************************************************************
&
&
&        versatec.ec
&
&
& *************************************************************
&
&     use: ec versatec.ec tape no  no of files  seg1 seg2....segn
&
&          where seg1 seg2....segn must not exceed a total of 32 char
&
&
&     function:
&           This ec reads the tape that was created on the Bendix
&           minicomputer and creates the Versatec border, grid, state,
&           county, and coordinate files. If there is more than one
&           coordinate file, they must be combined on Multics and then
&           sort.vers.coor.ec must be run.
&
&command_line off
&input_line off
&
value$set_seg value_seg
value$set file_names  [string &f3]
```

```
value$set this_file_name &3
value$set increment 0
value$set tape_file_no 1
&if [equal &2 1] &then value$set all_or_none none
&else value$set all_or_none all
&
&label copy
io attach input tape_ibm_ &1 -nlb -nb [value tape_file_no] -den 800 -f
\cmt fb -rec 80 -bk 80 -mode ascii -retain [value all_or_none]
io attach output record_stream_ -target vfile_ [value this_file_name]
copy_file -isw input -osw output
io detach (input output)
&
&if [equal [value tape_file_no]  &2] &then &goto quit
value$set tape_file_no [plus [value tape_file_no] 1]
&if [equal [value tape_file_no] &2] &then value$set all_or_none none
value$set start_of_next_file_name [plus [length [value this_file_name]
\c] [value increment] 2]
value$set remnant [string [substr [string [value file_names]] [value s
\ctart_of_next_file_name]]]
value$set nxtblnk [search [string [value remnant]] " "]
&if [equal [value nxtblnk] 0] &then value$set next_file_length [length
\c [value remnant]]
&else value$set next_file_length [minus [value nxtblnk] 1]
value$set next_file_name [substr [string [value remnant] ] 1 [value ne
\cxt_file_length]]
value$set increment [plus [value increment] [length [value this_file_n
\came]] 1]
value$set this_file_name [value next_file_name]
&goto copy
&
&label quit
truncate value_seg
&quit
```

PROGRAM NAME: INDEX_VERSATEC

Author: Lawrence Balcerak

Purpose of the program: **index_versatec** plots index maps using the Versatec plotter.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: index_versatec

Arguments: None

Subroutines called: **io_call**, **ioa_$nnl**, **setup_versaplot** (Multics software), **plots**, **plot**, **newpen**, **letter**, **factor** (Versaplot software), **openf**, **rotate**, **legend**, **pattern**, **rdftur**, **srtdup**, **pltsel**, **closef**

Common data referenced: *size*, *end*, *istate*, *scale*, */paraml*, *in1*, *ipen*, *kpen*, */wordl*

Input files: *bordNM* (*file10*), *gridNM* (*file11*), *statNM* (*file12*), *counNM* (*file13*), *coorNM* (*file14*), *pverNM* (*file15*)

Output files: Versatec plot

Arrays used: *input 4*(*6*) (used to read the six input files)

Called by: None

Error checking and reporting: None

Constants: *in* = 15, *rsiz* = 0.14, *ipen* = 4, *rrsiz* = 0.04, *kpen* = 1

Program logic:

1. Set *fmt2* = (6h(a4,2h,i2,1h)). This is used to put the State number into *fmt1*. The format is those characters between the outermost parentheses.
2. Attach to the file *init_vals*. This file contains those changes to the default parameters to the Versaplot software needed for this plot. The file contains *xmas* = 50.0, which should be adequate for most of our plotting.
3. Send message to terminal:
   TYPE IN TWO DIGIT STATE NUMBER
   and read *istate.*

4. If *istate* is less than 10, then *fmt2* =(6h(a4,2h,1h0,i1,1h)).
   Do step 5 for *i* = 10, 15.
5. Concatenate the State number to *input(i)* and place into *name*. Call **openf** with *iunit* = *i*, *name* = *name*, *mode* = "si".
6. Call **setup_versaplot**. This activates the Versaplot software.
7. Call **plots** (0,0,0). This initializes the default parameters. If *init_vals* is attached, it reads this file and makes those changes.
8. Set *in1* = 10. This is the reference number for *bord-NM*.
   Set *size* = 17.99, which is the width of the plotting paper.
   Call **rotate**.
   Print value of scale.
   Call **legend**.
   Call **pattern**.
9. If *deltay* is less than 0.001, go to step 10. A value greater than 0.001 indicates that the plot is to be rotated and translated.
   Transform *xsym*, *ysym*, *xlet*, and *ylet*.
10. Call **factor**. Reset the scale to the value of *scale*.
11. Read *inum*(5) and *iwords*(5) from *pverNM*.
    Set *iangle* = 0.
    Find the starting x-values (*xsym* and *xlet*) for the text string *iword*(5). The values for the use of both text plotting subroutines (**symbol** and **letter**) are given. The visible difference in determining which one to use is in the fonts for their letters.
12. If *deltay* is less than 0.001, go to step 13.
    Transform *xsym*, *ysym*, and *xlet*, *ylet*.
    Set *iangle* = 270.
    Call **plot** (2.0,0.005,−3). This changes the software origin. Trying to plot $y = 0$ or $x = 0$ when the hardware origin is in effect has presented problems in the past. Also, at times the upper left x value is negative.
    Go to step 14.
13. Call plot (2.0, 1.18, -3). Changes the origin to allow negative y values, which are needed to plot the legends at the bottom.
14. Plot the five text strings in *iwords(i)*. The **letter** subroutine is the one presently used. The dot width for letters is set at 4 except for the date, where it is 3.
15. Read the parameter plotting values into *inparm*.
16. If *inparm(1)* is not equal to 1, go to next step. A value of 1 indicates that the neat outline is to be plotted. A value of 0 indicates do not plot.
    Set *in1* = 10, which is the reference number for the neat outline.
    Call **rdftur**.
    Rewind *in1*.
17. Repeat step 16 with *inparm(23)* and *in1* = 11 (grid).
    Go to step 18 if no plot.
18. Repeat step 16 with *inparm(45)* and *in1* = 12 (State).
    Go to step 19 if no plot.
19. Repeat step 16 with *inparm(67)* and *in1* = 13 (counties).
    Go to step 20 if no plot.
20. Repeat step 16 with *inparm(111)* and *in1* = 14 (coordinates outlines).
    Go to step 21 if no plot.
    Otherwise, go to step 22 after this is plotted.
21. If *inparm(89)* is not equal 1, go to next step.
    Set *in1* = 14.
    Call **srtdup(*in*)**.
    Read in cards with the selected outlines, sort them in ascending order, and remove duplications.
    Call **tsel(*inparm(89)*)**.
    Plot the selected outlines.
    Rewind *in1*.
22. Call **plot(0.,0.,999)** (end of plot).
    Send message to terminal:
    FINISH PLOT
23. If *inparm(133)* is equal to 1, go to step 10.
    Otherwise go to step 24.
24. Call **plot(0.,0.,−999)**.
    End all plotting.
25. Call **setup_versaplot(*"-reset"*)** (removes links to Versatec software).
    Call **closef(i)** for *i* = 10, 15 (closes and detaches all files).

```
c                     PROGRAM - INDEX_VERSATEC
c                        PLOT LAND USE ID'S
c
c                           L.L. BALCERAK
c                     U. S. GEOLOGICAL SURVEY
c
          common /rot/ size,end,xmax,ymax,istate
          common /param/ scale,deltay,rsiz,rrsiz
          common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
```

```
            common /word/ xsym(5),ysym(5),inum(5),numsta(72),
                        &xlet(5),ylet(5),iscale(5),height(5)
            character input*4(6),fmt1*10,fmt2*21,name*6,iwords*53(5)
            dimension   inparm(133)
            external io_call (descriptors),ioa_$nnl (descriptors),
                    &setup_versaplot (descriptors),letter (descriptors)
            data input /"bord","grid","stat","coun","coor","pver"/
            in=15
            ipen=4
            kpen=1
            rsiz=0.14
            rrsiz=0.04
            fmt2="(6h(a4,2h,i2,1h))"
c
            call io_call ("attach","init_vals","vfile_","init_vals")
            call ioa_$nnl ("^/TYPE IN TWO-DIGIT STATE NUMBER:  ")
            read 10, istate
10          format (i2)
            if (istate .lt. 10)  fmt2="(6h(a4,2h,1h0,i1,1h))"
            do 20 i=10,15
            encode (fmt1,fmt2) istate
            encode (name,fmt1) input(i-9)
            call openf (i,name,"si  ")
20          continue
            call setup_versaplot
c
c                SET ORIGIN ON PLOTTER
c
            call plots (0,0,0)
            in1=10
            size=17.99
            call rotate
            print ,"scale=",scale
            call legend (iwords)
            call pattern
            if (deltay .lt. .001)  go to 50
            do 40 i=1,4
            temp=xsym(i)
            xsym(i)=ysym(i)
            ysym(i)=deltay-temp
            temp=xlet(i)
            xlet(i)=ylet(i)
            ylet(i)=deltay-temp
40          continue
50          call factor (scale)
c
c                        BORDER INFORMATION
c
            read (in,60,end=180) inum(5),iwords(5)
60          format (i2,a53)
            iangle=0
            xsym(5)=(xmax-inum(5)*(height(5)+0.019))/2.
            xlet(5)=(xmax-inum(5)*(iscale(5)*(0.0625-0.0029)))/2.
            if (deltay .lt. .001)  go to 70
            iangle=270
```

```
          ysym(5)=deltay-xsym(5)
          ylet(5)=deltay-xlet(5)
          xsym(5)=-0.61
          xlet(5)=-0.61
          call plot (2.0,0.005,-3)
          go to 80
70        call plot (2.0,1.18,-3)
80        do 90 i=1,5
          j=4
          if (i .eq. 4)  j=3
          call newpen (j)
          call letter (inum(i),iscale(i),iangle,xlet(i),ylet(i),iwords(i))
90        continue
c
c              READ INPUT INFORMATION
c
          read (in,100,end=170)  inparm
100       format  (66i1/67i1)
c
c              CHECK FOR PLOT OR NO PLOT
c              ON EACH ITEM
c
c              NEAT OUTLINE
c
          if (inparm(1) .ne. 1)  go to 110
          in1=10
          call rdftur (inparm(1))
          rewind in1
c
c              GRID
c
110       if  (inparm(23) .ne. 1) go to 120
          in1=11
          call rdftur (inparm(23))
          rewind in1
c
c              STATE
c
120       if  (inparm(45) .ne. 1)  go to 130
          in1=12
          call rdftur (inparm(45))
          rewind in1
c
c              COUNTIES
c
130       if  (inparm(67) .ne. 1)  go to 140
          in1=13
          call rdftur (inparm(67))
          rewind in1
c
c              ALL FEATURES
c
140       if  (inparm(111) .ne. 1) go to 150
          in1=14
          call rdftur  (inparm(111))
          rewind in1
          go to 170
c
c               SELECTED FEATURES
c
150       if  (inparm(89) .ne. 1)  go to 170
          in1=14
          call srtdup (in)
160       call pltsel  (inparm(89))
          rewind in1
c
c              CHECK IF SOMETHING ELSE IS TO BE PLOTTED
c
170       call plot (0.,0.,999)
          print ,"finish plot"
          if  (inparm(133) .eq. 1)  go to 50
180       call plot (0.,0.,-999)
          call setup_versaplot ("-reset")
          do 190 i=10,15
          call closef(i)
190       continue
          stop
          end
```

SUBROUTINE NAME: LEGEND

Author: Lawrence Balcerak
Purpose of the program: **legend** initializes text strings and beginning text positions for the index maps.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call legend (iwords)
Arguments: ***iwords***–Five text strings containing the legends to be placed on the index maps
Subroutines called: None
Common data referenced: ***xmax***, ***ymax***, ***istate***, ***xsym***, ***ysym***, ***inum***, ***numsta***, ***xlet***, ***ylet***, ***iscale***, ***height***
Input files: None
Output files: None
Arrays used:
iwords53(5)
nstate20(72)–Array in which to read the States
Called by: **index_versatec, index_calcomp**
Error checking and reporting: None
Constants: None

Program logic:
1. Set ***numsta(i)*** = the number of letters in the State name, where ***i*** represents the FIPS code.
   Set ***iwords(1)*** and ***iwords(2)***
   Set ***nstate(i)*** = State names
   Set ***iwords(3)*** = ***nstate(istate)***
2. Find year date.
   Set ***iwords(4)*** = year date.
3. Set other variables: ***inum*** is the count of letters for each iword; ***height*** is the height of letters if using **symbol** to plot the legends; ***iscale*** is the height of letters if using subroutine **letter** to plot the legends; ***xsym***, ***ysym*** are the starting *x*, *y* coordinates if using **symbol**; ***xlet***, ***ylet*** are the starting *x*, *y* coordinates if using **letter**.
   Note: There are factors of 0.019 and −0.0029 for computing ***xsym*** and ***xlet***. The factor for using **symbol** is to take into account the slight widening of the letter where line width is greater than 1. Where **letter** is used, the supposed width of a letter is slightly reduced, but no explanation is given in the manual.

```
          subroutine legend (iwords)
          common /rot/ size,end,xmax,ymax,istate
          common /word/ xsym(5),ysym(5),inum(5),numsta(72),
                   &  xlet(5),ylet(5),iscale(5),height(5)
          character iwords*53(5),nstate*20(72),idt*6,year*4
c
          iwords(1)="UNITED STATES GEOLOGICAL SURVEY"
          iwords(2)="DEPARTMENT OF THE INTERIOR"
```

```
c
          nstate(1) ="ALABAMA"
          nstate(2) ="ALASKA"
c
          nstate(4) ="ARIZONA"
          nstate(5) ="ARKANSAS"
          nstate(6) ="CALIFORNIA"
          nstate(7) ="CANAL ZONE"
          nstate(8) ="COLORADO"
          nstate(9) ="CONNECTICUT"
          nstate(10)="DELAWARE"
          nstate(11)="DISTRICT OF COLUMBIA"
          nstate(12)="FLORIDA"
          nstate(13)="GEORGIA"
c
          nstate(15)="HAWAII"
          nstate(16)="IDAHO"
          nstate(17)="ILLINOIS"
          nstate(18)="INDIANA"
          nstate(19)="IOWA"
          nstate(20)="KANSAS"
          nstate(21)="KENTUCKY"
          nstate(22)="LOUISIANA"
          nstate(23)="MAINE"
          nstate(24)="MARYLAND"
          nstate(25)="MASSACHUSETTS"
          nstate(26)="MICHIGAN"
          nstate(27)="MINNESOTA"
          nstate(28)="MISSISSIPPI"
          nstate(29)="MISSOURI"
          nstate(30)="MONTANA"
          nstate(31)="NEBRASKA"
          nstate(32)="NEVADA"
          nstate(33)="NEW HAMPSHIRE"
          nstate(34)="NEW JERSEY"
          nstate(35)="NEW MEXICO"
          nstate(36)="NEW YORK"
          nstate(37)="NORTH CAROLINA"
          nstate(38)="NORTH DAKOTA"
          nstate(39)="OHIO"
          nstate(40)="OKLAHOMA"
          nstate(41)="OREGON"
          nstate(42)="PENNSYLVANIA"
c
          nstate(44)="RHODE ISLAND"
          nstate(45)="SOUTH CAROLINA"
          nstate(46)="SOUTH DAKOTA"
          nstate(47)="TENNESSEE"
          nstate(48)="TEXAS"
          nstate(49)="UTAH"
          nstate(50)="VERMONT"
          nstate(51)="VIRGINIA"
          nstate(52)="VIRGIN ISLANDS"
          nstate(53)="WASHINGTON"
          nstate(54)="WEST VIRGINIA"
          nstate(55)="WISCONSIN"
          nstate(56)="WYOMING"
```

```
c
          nstate(60)="AMERICAN SAMOA"
          nstate(66)="GUAM"
          nstate(72)="PUERTO RICO"
c
          iwords(3)=nstate(istate)
          call pl1_date_ (idt)
          decode (idt,10) year
10        format (a2)
          encode (iwords(4),20) year
20        format (2h19,a2)
c
          inum(1)=31
          inum(2)=26
          inum(3)=numsta(istate)
          inum(4)=4
c
          height(1)=0.18
          height(2)=0.18
          height(3)=0.18
          height(4)=0.14
          height(5)=0.28
c
          iscale(1)=3
          iscale(2)=3
          iscale(3)=3
          iscale(4)=3
          iscale(5)=5
c
          xsym(1)=0.25
          ysym(1)=ymax+0.35
          xsym(2)=0.25
          ysym(2)=ymax+0.68
          xsym(3)=xmax-inum(3)*(height(3)+0.019)-0.25
          ysym(3)=ymax+0.68
          xsym(4)=(xmax-inum(4)*(height(4)+0.019))/2.
          ysym(4)=-1.17
          ysym(5)=-0.61
c
          do 30 i=1,5
          ylet(i)=ysym(i)
30        continue
          xlet(1)=xsym(1)
          xlet(2)=xsym(2)
          xlet(3)=xmax-inum(3)*(iscale(3)*(0.0625-0.0029))-0.25
          xlet(4)=(xmax-inum(4)*(iscale(4)*(0.0625-0.0029)))/2.
          return
          end
c
c
          block data
          common /word/ xsym(5),ysym(5),inum(5),numsta(72),
     &    xlet(5),ylet(5),iscale(5),height(5)
```

```
      data numsta / 7, 6, 0, 7, 8,10,10, 8,11, 8,
     &             20, 7, 7, 0, 6, 5, 8, 7, 4, 6,
     &              8, 9, 5, 8,13, 8, 9,11, 8, 7,
     &              8, 6,13,10,10, 8,14,12, 4, 8,
     &              6,12, 0,12,14,12, 9, 5, 4, 7,
     &              8, 0,10,13, 9, 7, 0, 0, 0,14,
     &              0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0,11/
      end
```

SUBROUTINE NAME: ROTATE

Author: Lawrence Balcerak

Purpose of the program: **rotate** checks *bordNM* to see if it will fit on plotting paper at full scale. If not, it computes a scale that produces the largest possible plot on the paper. The finished plot can be upright or rotated on its side.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call rotate

Arguments: None

Subroutines called: None

Common data referenced: *size*, *end*, *xmax*, *ymax*, *scale*, *deltay*, *in1*

Input files: bordNM

Output files: None

Arrays used:

x(6)−x coordinate

y(6)−y coordinate

Called by: **index_versatec, index_calcomp**

Error checking and reporting: None

Constants: None

Program logic:

1. Reads the header card (does not use).
2. Reads data points (always six points).
3. Finds maximum x value and y value and stores in *xmax* and *ymax*.
4. Set parameters (assume upright at full scale): end = *xmax* + 8.0. In Calcomp plotting, this is the amount to move in x to start a new plot; *scale* = 1.0; *deltay* = 0.0. If plot is rotated, *deltay* is the amount of translation needed to bring plot back to plotting frame.
5. If *ymax* plus amount needed for legends at top and bottom is less than *size* (width of paper), go to step 10 (plot fits on paper).
6. If *xmax* is greater than *size*, go to step 8 (rotated plot too big to fit).
7. Rotated plot will fit at full scale.
   Set *deltay* = *xmax*
   Set *end* = *ymax* + 8.0
   Go to step 10.
8. Plot must be scaled. If plot would be larger rotated on side, go to step 9. Otherwise, compute scale in upright position.
   Go to step 10.
9. Compute scale in rotated position.
   Set *deltay* = *xmax*
   Set *end* = *ymax* + 8.0
10. Rewind data file.
    Return to calling program.

```
      subroutine rotate
      common /rot/ size,end,xmax,ymax,istate
      common /param/ scale,deltay,rsiz,rrsiz
      common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
      dimension x(6),y(6)
      read (in1,10) if
10    format (i5)
      read (in1,20) (x(i),y(i),i=1,6)
20    format (12f6.3)
      xmax=x(2)
      ymax=y(2)
      do 30 i=3,6
      if (x(i) .gt. xmax)  xmax=x(i)
      if (y(i) .gt. ymax)  ymax=y(i)
30    continue
```

```
          end=xmax+8.0
          scale=1.0
          deltay=0.0
          if ((ymax + 2.05) .le. size)  go to 60
          if (xmax .gt. size)  go to 40
          deltay=xmax
          end=ymax+8.0
          go to 60
40        if (xmax .lt. (ymax+2.05))  go to 50
          scale=size/(ymax+2.05)-0.01
          go to 60
50        scale=size/xmax-0.01
          deltay=xmax
          end=ymax+8.0
60        rewind in1
          return
          end
```

SUBROUTINE NAME: RDFTUR

Author: Lawrence Balcerak
Purpose of the program: **rdftur** reads a header card from the file being plotted and branches to the designated plotting subroutine.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call rdftur (infom)
Arguments: ***infom*** – Array of 22 elements with plotting parameters for the file being read
Subroutines called: **plotch**, **plotli**, **pllich**
Common data referenced: ***if***, ***ifno***, ***isf***, ***isfno***, ***not***, ***nor***, ***nif***, ***in1***
Input files: ***gridNM***, ***statNM***, ***counNM***, ***coorNM***
Output files: None
Arrays used: ***infom*** – Elements with plotting parameters for the file being read
Called by: **index_versatec**, **index_calcomp**
Error checking and reporting: None
Constants: None
Program logic:

1. ***infom(2)*** can have values of 0, 1, or 2 (see plotting parameter cards).
   Add 1 and store in ***item***.
2. Read a header card from file ***in1***.
   If EOF, return to calling program.
3. Call subroutine, which will plot according to the parameter stored in ***infom(2)***.
4. After return from plotting subroutine, go to step 2.

```
          subroutine rdftur  (infom)
c
c              DETERMINE WHAT IS TO BE PLOTTED FOR
c              THE SPECIFIED FEATURE
c
          common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
          dimension  infom(22)
          item=infom(2)+1
c
c              READ A FEATURE CARD
c
5         read (in1,10,end=80) if,ifno,isf,isfno,not,nor,nif,ispan
10        format (8i5)
          go to (40,30,20),item
c
c              CHARACTERS PLOTTED ONLY
c
20        call plotch  (infom)
          go to 5
```

```
c
c           LINES PLOTTED ONLY
c
30       call plotli   (infom)
         go to 5
c
c           BOTH PLOTTED
c
40       call pllich   (infom)
         go to 5
80       return
         end
```

SUBROUTINE NAME: PENCHG

Author: Lawrence Balcerak

Purpose of the program: **penchg** changes pen numbers (Calcomp) or line widths (Versatec) in a sequence predetermined by the plotting parameters.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call penchg (infom,ipen)

Arguments:
- *infom* – Array of 22 elements with plotting parameters for the file being read
- *ipen* – Counter for member of *infom* to examine

Subroutines called: **newpen** (Versaplot and Calcomp software)

Common data referenced: None

Input files: None

Output files: None

Arrays used: *infom(22)* – Elements with plotting parameters for the file being read

Called by: **plotch, plotli, pllich**

Error checking and reporting: None

Constants: None

Program logic:
1. Set *k* = *infom(ipen)*. *k* equals pen number (Calcomp) or line width (Versatec).
2. Call **newpen(k)**
3. Add 1 to *ipen*.
   If *ipen* is equal to 13, set *ipen* = 4.
   Or, if *infom(ipen)* = 0, set *ipen* = 4.
4. Return.

```
         subroutine penchg (infom,ipen)
c
c           change pens
c
         dimension infom(22)
         k=infom(ipen)
         call newpen(k)
         ipen=ipen+1
         if (ipen .eq. 13)  ipen=4
         if (infom(ipen) .eq. 0)  ipen=4
         return
         end
```

SUBROUTINE NAME: PLOTLI

Author: Lawrence Balcerak

Purpose of the program: **plotli** plots a coordinate outline without accompanying text.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call plotli (infom)

Arguments: *infom* – Array of 22 elements with plotting parameters for the file being read

Subroutines called: **plot, symbol** (Versaplot), **penchg**

Common data referenced: *deltay, rsiz, isfno, in1, ipen*

Input files: *gridNM, statNM, counNM, coorNM*

Output files: None

Arrays used:
- *xx(6)* – *x* coordinate,
- *yy(6)* – *y* coordinate
- *infom(22)* – Elements with plotting parameters for the file being read

Called by: **rdftur, pltsel**

Error checking and reporting: None
Constants: None
Program logic:
1. Call **penchg**.
2. Set *ie* = minimum of (6,*isfno*).
3. Read the first coordinate data card into *xx(i)*, *yy(i)*.
   Set *ang* = 0.
4. If *deltay* is less then 0.001, go to step 4. A value greater than 0.001 indicates that the plot is to be rotated.
   Rotate and translate the data points.
   Set *ang* = 270.0.
5. If *isfno* is greater than 2, go to step 6.
   Otherwise, plot a centered symbol (#2) scaled to 1.5 times the character height *rsiz*.
   Return.
6. Go to the first data point of outline with pen up.
   Set *k* = 2. This is the position in *xx*, *yy* at which to start plotting. The first point is the text position.
7. Plot points *k* to *ie* with pen down.
8. Subtract 6 from *isfno*.
   If *isfno* is greater then 0 (there are more points to plot), go to step 9.
   Otherwise return.
9. Set *ie* = minimum of (6,*isfno*).
   Read another data card into *xx*, *yy*.
10. If *deltay* is less then 0.001, go to step 11.
    Otherwise, rotate and translate *xx*, *yy*.
11. Set *k* = 1.
    Go to step 7.

```
          subroutine  plotli (infom)
c
c               PLOT LINES ONLY
c
          common /param/ scale,deltay,rsiz,rrsiz
          common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
          dimension infom(22),xx(6),yy(6)
c
c               CHANGE PENS
c
          call penchg (infom,ipen)
c
c               READ THE FIRST COORDINATE CARD
c
          ie=isfno
          if (isfno .gt. 6)  ie=6
          read (in1,20,end=130)  (xx(i),yy(i),i=1,ie)
20        format (12f6.3)
          ang=0.0
          if (deltay .lt. .001)  go to 40
          do 30 i=1,ie
          temp=xx(i)
          xx(i)=yy(i)
          yy(i)=deltay-temp
30        continue
          ang=270.0
c
c               CHECK FOR A SINGLE POINT
c
40        if (isfno .ge. 3)  go to 50
          call symbol (xx(2),yy(2),1.5*rsiz,2,ang,-1)
          go to 130
c
c               PLOT LINES
c
50        call plot (xx(2),yy(2),3)
          k=2
```

```
60          do 70 i=k,ie
            call plot (xx(i),yy(i),2)
70          continue
c
c               CHECK FOR MORE COORDINATES
c
            isfno=isfno-6
            if (isfno)  130,130,80
80          if  (isfno -6)  90,90,100
90          ie=isfno
100         read (in1,20,end=130)  (xx(i),yy(i),i=1,ie)
            if (deltay .lt. .001)  go to 120
            do 110  i=1,ie
            temp=xx(i)
            xx(i)=yy(i)
            yy(i)=deltay-temp
110         continue
120         k=1
            go to 60
130         return
            end
```

SUBROUTINE NAME: PLLICH

Author: Lawrence Balcerak
Purpose of the program: **pllich** plots both the feature number and outline from a coordinate file.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call pllich (infom).
Arguments: ***infom*** – Array of 22 elements with plotting parameters for the file being read
Subroutines called: **number**, **symbol**, **plot** (Versaplot), **penchg**, **shade**
Common data referenced: ***deltay***, ***rsiz***, ***rrsiz***, ***/dex/***
Input files: ***gridNM***, ***statNM***, ***counNM***, ***coorNM***
Output files: None
Arrays used:
xx(6) – x coordinate
yy(6) – y coordinate
infom(22) – Elements with plotting parameters for the file being used
Called by: **rdftur**, **pltsel**
Error checking and reporting: None
Constants: None
Program logic:

1. Call **penchg**.
2. Set *ie* = minimum of (6,*isfno*).
3. Read the first data card.
   Set *ang* = 0.
4. If *deltay* is less than 0.001, go to step 5. A value of *deltay* greater than 0.001 indicates that the plot is to be rotated.
   Rotate and translate *xx*, *yy*.
   Set *ang* = 270.0.
5. If *kpen* is greater then 0, call **newpen(3)**. *kpen* serves a dual function. For Calcomp plots, it has a value of –1. For Versatec plots, it has a value greater than 0. Also, in Versatec plots it serves as a counter for the number of the pattern last used.
6. Set *ff* = *if*.
   Call **number** to plot the feature number.
7. If the subfeature number is not to be plotted, go to next step.
   The subfeature numbers are not to be plotted on the Versatec plots, so if *kpen* is greater then 0, go to the next step.
   Set *ff* = *isf*. The starting coordinate for the subfeature number must be offset to place it directly under the feature number.
   If the plot is not rotated, subtract the letter size *rsiz* and a small amount for a gap between lines *rrsiz* from *yy*.
   If the plot is rotated, subtract *rsiz* and *rrsiz* from *xx*.
   Call **number**.
8. Repeat previous step for the span.
9. Repeat previous step for the second subfeature number.
10. If *isfno* is greater then 2, go to step 11.
    Otherwise, plot a centered symbol (#2) scaled to 1.5 times the character height *rsiz*.
    Return.
11. If this is the coordinate outline file being read (*in1* = 14) and this is a Versatec plot (*kpen* greater than 0), go to step 12.

Otherwise, go to step 13.
12. Call **shade**.
Return.
13. Go to the first data point of outline with pen up.
Set *k* = 2. This is the position in *xx*, *yy* at which to start plotting. The first point is the text position.
14. Plot points *k* to *ie* with pen down.
15. Subtract 6 from *isfno*.
If *isfno* is greater than 0 (there are more points to plot) go to step 16.
Otherwise, return.
16. Set *ie* = minimum of (6,*isfno*).
Read another data card into *xx*, *yy*.
17. If *deltay* is less than 0.001, go to step 18.
Otherwise, rotate and translate *xx*, *yy*.
18. Set *k* = 1.
Go to step 14.

```
          subroutine pllich (infom)
c
c              PLOT BOTH LINES AND CHARECTERS
c
          common /param/ scale,deltay,rsiz,rrsiz
          common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
          dimension infom(22),xx(6),yy(6)
c
c                CHANGE PENS
c
          call penchg (infom,ipen)
c
c              READ THE FIRST COORDINATE CARD
c
          ie=isfno
          if(isfno .gt. 6)  ie=6
          read (in1,10,end=230) (xx(i),yy(i),i=1,ie)
10        format (12f6.3)
          ang=0.0
          if (deltay .lt. .001)  go to 30
          do 20 i=1,ie
          temp=xx(i)
          xx(i)=yy(i)
          yy(i)=deltay-temp
20        continue
          ang=270.0
c
c              PLOT FEATURE NUMBER
c
30        if (kpen .gt. 0)  call newpen(3)
          ff=if
          call number (xx(1),yy(1),rsiz,ff,ang,-1)
c
c              CHECK FOR SUBFEATURE NUMBER
c
          if (ifno .eq. 1) go to 70
c******THE FOLLOWING LINE IS INCLUDED TO ELIMINATE THE SUBFEATURE
c******   NUMBER IN VERSATEC PLOTS
          if (kpen .gt. 0)  go to 70
          if (deltay .gt. .001)  go to 50
          yy(1)=yy(1)-rsiz-rrsiz
          go to 60
50        xx(1)=xx(1)-rsiz-rrsiz
60        ff=isf
          call number (xx(1),yy(1),rsiz,ff,ang,-1)
```

```
c
c              CHECK FOR SPAN
c
70        if (ispan .eq. 0)  go to 100
          if (deltay .gt. .001)  go to 80
          yy(1)=yy(1)-rsiz-rrsiz
          go to 90
80        xx(1)=xx(1)-rsiz-rrsiz
90        ff=ispan
          call number (xx(1),yy(1),rsiz,ff,ang,-1)
c
c              CHECK FOR SECOND SUBFEATURE NUMBER
c
100       if (not .eq. 0)  go to 130
          if (kpen .gt. 0) go to 130
          if (deltay .gt. .001)  go to 110
          yy(1)=yy(1)-rsiz-rrsiz
          go to 120
110       xx(1)=xx(1)-rsiz-rrsiz
120       ff=not
          call number (xx(1),yy(1),rsiz,ff,ang,-1)
c
c              CHECK FOR A SINGLE POINT
c
130       if (isfno .ge. 3)  go to 140
          call symbol (xx(2),yy(2),1.5*rsiz,2,ang,-1)
          return
140       if ((in1 .eq. 14) .and. (kpen .gt. 0))  go to 220
c
c               RESET PEN
c
          ipen=ipen-1
          call penchg (infom,ipen)
c
c              PLOT LINES
c
          call plot (xx(2),yy(2),3)
          k=2
150       do 160  i=k,ie
          call plot (xx(i),yy(i),2)
160       continue
c
c              CHECK FOR MORE COORDINATES
c
          isfno=isfno-6
          if (isfno)  230,230,170
170       if (isfno-6)  180,180,190
180       ie=isfno
190       read (in1,10,end=230)  (xx(i),yy(i),i=1,ie)
          if (deltay .lt. .001)  go to 210
          do 200 i=1,ie
          temp=xx(i)
          xx(i)=yy(i)
          yy(i)=deltay-temp
200       continue
210       k=1
```

```
            go to 150
220         call shade (xx,yy)
230         return
            end
```

SUBROUTINE NAME: PLTSEL

Author: Lawrence Balcerak
Purpose of the program: **pltsel** reads through a file of coordinates, identifies those that are to be plotted, and branches to the appropriate plotting subroutine.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call pltsel (infom)
Arguments: ***infom***–Array of 22 elements with plotting parameters for the file being read
Subroutines called: **plotch, plotli, pllich**
Common data referenced: ***ik, num, numsel, numsub, if, isf, isfno, in1***
Input files: ***coorNM***
Output files: None
Arrays used: ***infom(22)***–Elements with plotting parameters for the file being read
Called by: **index_versatec, index_calcomp**
Error checking and reporting: If some features cannot be located, a message appears on the screen:
SOMETHING IS WRONG WITH THIS NUMBER
along with the number.
Constants: ***io*** = 0 sets the program for terminal output
Program logic:

1. Set ***io*** = 0 (terminal output).
   Set ***item*** = ***infom(2)*** + 1. ***infom(2)*** has possible values of 0, 1, or 2.
   Set ***ik*** = 0. This is a counter for number of the outline being plotted.
   Set ***kount*** = 0. This is a counter for number of times file has been read.
2. Add 1 to ***ik***.
3. If ***ik*** is less than or equal to ***num*** go to step 4.
   Otherwise, return. ***num*** is the total number of outlines to be plotted.
4. Read a header card. If EOF, go to step 8.
5. If the feature and subfeature numbers of the header card just read match with the selected feature, go to step 7. Otherwise, go to step 6.
6. Read through to data points to reach the next header card.
   Go to step 4.
7. Call the appropriate plotting subroutine as identified by ***item***.
   Go to step 4.
8. Add 1 to ***kount***. Rewind data file:
   If ***kount*** is less than 2, go to step 4. Otherwise go to step 9.
9. Write to terminal giving error message about unlocatable features.
   Set ***kount*** = 0
   Go to step 2.

```
            subroutine pltsel (infom)
c
c                PLOT ONLY SELECTED FEATURES
c
            common /sortd/ ik,num,numsel(1500),numsub(1500),numgo(1500)
            common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
            dimension infom(22)
            io=0
            item=infom(2)+1
            ik=0
            kount=0
10          ik=ik+1
      iwro = 0
            if (ik-num) 20,20,160
20          read (in1,30,end=140) if,ifno,isf,isfno,not,nor,nif,ispan
30          format (8i5)
            if ((numsel(ik) .eq. if) .and. (numsub(ik) .eq. isf)) go to 100
       if ((numsel(ik) .eq. if) .and. (iwro .eq. 1)) go to 100
50          read (in1,60,end=140)  x
```

```
60          format  (f6.3)
            if (isfno .le. 0)  go to 20
            isfno=isfno-6
            go to 50
100         go to (130,120,110),item
110         call plotch (infom)
            go to 10
120         call plotli (infom)
            go to 10
130         call pllich (infom)
            go to 10
140         if (ik-num)  150,150,160
150         kount=kount+1
            rewind in1
      if ((iwro .eq. 1) .and. (isec .eq. 1)) go to 157
      if (iwro .eq. 1) go to 20
            if (kount .lt. 2)  go to 20
            write (io,155) numsel(ik),numsub(ik)
155         format (1x, "SOMETHING IS WRONG WITH THIS NUMBER.",2(3x,i5) )
      iwro = 1
      isec = 1
            kount=0
            go to 20
 157   kount = 0
       isec = 0
       iwro = 0
       go to 10
160         return
            end
```

SUBROUTINE NAME: PLOTCH

Author: Lawrence Balcerak

Purpose of the program: **plotch** plots the feature number of an outline and then reads through the data points for that outline.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call plotch (infom)

Arguments: ***infom*** – Array of 22 elements with plotting parameters for the file being read

Subroutines called: **number** (Versaplot), **penchg**

Common data referenced: ***deltay, rsiz, rrsiz, if, ifno, isf, isfno, not, ispan, in1, ipen***

Input files: ***gridNM, statNM, counNM, coorNM***

Output files: None

Arrays used: ***infom***(22) – Elements with plotting parameters for the file being read

Called by: **rdftur, pltsel**

Error checking and reporting: None

Constants: None

Program logic:

1. Call **penchg**.
2. Read text position from first data card into *xx* and *yy*.
3. Set *ff* = *if* (the number to be plotted).
   Set *ang* = 0.0 (the angle at which to plot).
4. If ***deltay*** is less then 0.001, go to step 5.
   Otherwise, the plot is to be transformed.
   Set *ang* = 270.0. Rotate and translate *xx, yy*.
5. Call **number** to plot the feature number.
6. If subfeature number is not to be plotted, go to next step.
   Set *ff* = *isf*.
   The starting coordinate for the subfeature number must be offset to place it directly under the feature number:
   If the plot is not rotated, subtract the letter size (*rsiz*) and a small amount for a gap between lines (*rrsiz*) from *yy*.
   If the plot is rotated, subtract *rsiz* and *rrsiz* from *xx*.
   Call **number**.
7. Repeat previous step for the span.
8. Repeat previous step for the second subfeature number.
9. If there are more data points, read through them to position the file at next header card.
   Return

```
          subroutine  plotch (infom)
c
c              PLOT CHARACTERS ONLY
c
          common /param/ scale,deltay,rsiz,rrsiz
          common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
          dimension infom(22)
c
c              CHANGE PENS
c
          call pencng (infom,ipen)
c
c              READ THE FIRST COORDINATE CARD
c
          read (in1,10,end=120)  xx,yy
10        format (12f6.3)
c
c              PLOT THE FEATURE NUMBER
c
          ff=if
          ang=0.0
          if (deltay .lt. .001)  go to 20
          ang=270.0
          temp=xx
          xx=yy
          yy=deltay-temp
20        call number (xx,yy,rsiz,ff,ang,-1)
c
c              CHECK FOR SUBFEATURE NUMBER
c
          if  (ifno .eq. 1) go to 50
          if (deltay .gt. .001)go to 30
          yy=yy-rsiz-rrsiz
          go to 40
30        xx=xx-rsiz-rrsiz
40        ff=isf
          call number (xx,yy,rsiz,ff,ang,-1)
c
c              CHECK FOR SPAN
c
50        if (ispan .eq. 0)  go to 80
          if (deltay .gt. .001)  go to 60
          yy=yy-rsiz-rrsiz
          go to 70
60        xx=xx-rsiz-rrsiz
70        ff=ispan
          call number (xx,yy,rsiz,ff,ang,-1)
c
c              CHECK FOR SECOND SUBFEATURE NUMBER
c
80        if (not .eq. 0)  go to 110
          if (deltay .gt. .001) go to 90
          yy=yy-rsiz-rrsiz
          go to 100
```

```
90          xx=xx-rsiz-rrsiz
100         ff=not
            call number (xx,yy,rsiz,ff,ang,-1)
c
c                MUST PROCESS OTHER COORDINATE CARDS
c
110         if (isfno .le.6)  go to 120
            isfno=isfno-6
            read (in1,10,end=120)  xx,yy
            go to 110
120         return
            end
```

SUBROUTINE NAME: SHADE

Author: Lawrence Balcerak
Purpose of the program: **shade** draws coordinate outlines and shades them either as a sequential variety of patterns or as a predetermined pattern. There are presently 10 patterns.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call shade (*xx,yy*)
Arguments: *xx, yy*– Pairs of coordinates from first card image read in calling subroutine
Subroutines called: **newpen, plot, tone** (Versaplot)
Common data referenced: ik, numgo, deltay, kpen, in1, isfno, ip1–ip10
Input files: coorNM
Output files: None
Arrays used:
xx(6)– *x* coordinate
yy(6)– *y* coordinate
x(2000)– Buffer area for *x* coordinates
y(2000)– Buffer area for *y* coordinates
Called by: **pllich**
Error checking and reporting: None
Constants: None
Program logic:

1. Add 1 to *kpen*. *kpen* is the counter by which the patterns rotate.
2. Set first five elements of *x, y* = to elements 2 through 6 of *xx, yy*. The first coordinate pair of *xx, yy* is the text position.
3. Read rest of coordinates into *x, y*.
4. If *deltay* is less than 0.001, go to step 5. *deltay* has a positive value if the plot is to be rotated and translated. The first five elements of *x, y* have already been transformed in the calling program.
   Rotate and translate the remaining elements of *x, y*.
5. Set the line width to two dots. Thick outlines are more conspicuous.
6. Plot the outline. Go to the first point with pen up. Go to succeeding points with pen down.
7. Set line width to one dot for the shading to follow.
8. If *numgo(ik)* is greater than 0, then change the shading pattern to that number. Otherwise, set the pattern to *kpen*.
9. Shade the outline.
   Return.

```
            subroutine shade (xx,yy)
            common /sortd/ ik,num,numsel(1500),numsub(1500),numgo(1500)
            common /param/ scale,deltay,rsiz,rrsiz
            common /dex/ if,ifno,isf,isfno,not,nor,nif,ispan,in1,ipen,kpen
            common /pat/ ip1(16),ip2(16),ip3(4),ip4(4),ip5(16),ip6(16),
                     &   ip7(16),ip8(16),ip9(16),ip10(16)
            dimension xx(6),yy(6),x(2000),y(2000)
            kpen=kpen+1
            if ( kpen .gt. 10)  kpen=1
            do 10 i=1,5
            x(i)=xx(i+1)
            y(i)=yy(i+1)
10          continue
            number=isfno-1
            if (number-5) 50,50,20
```

```
20        read (in1,30) (x(i),y(i),i=6,number)
30        format (12f6.3)
          if (deltay .lt. .001)  go to 50
          do 40 i=6,number
          temp=x(i)
          x(i)=y(i)
          y(i)=deltay-temp
40        continue
50        call newpen (2)
          call plot (x(1),y(1),3)
          do 60 i=1,number
          call plot (x(i),y(i),2)
60        continue
          call newpen (1)
          if (numgo(ik) .le. 0)  go to 70
          go to (80,90,100,110,120,130,140,150,160,170),numgo(ik)
70        go to (80,90,100,110,120,130,140,150,160,170),kpen
80        call tone (0.,0.,ip1,-16)
          go to 180
90        call tone (0.,0.,ip2,-16)
          go to 180
100       call tone (0.,0.,ip3,-4)
          go to 180
110       call tone (0.,0.,ip4,-4)
          go to 180
120       call tone (0.,0.,ip5,-16)
          go to 180
130       call tone (0.,0.,ip6,-16)
          go to 180
140       call tone (0.,0.,ip7,-16)
          go to 180
150       call tone (0.,0.,ip8,-16)
          go to 180
160       call tone (0.,0.,ip9,-16)
          go to 180
170       call tone (0.,0.,ip10,-16)
180       call tone (x,y,number,1)
          return
          end
```

SUBROUTINE NAME: SRTDUP

Author: Lawrence Balcerak
Purpose of the program: **srtdup** reads the selected feature numbers to be plotted, sorts them into ascending order, and removes all duplications.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call srtdup (in)
Arguments: *in*–File number of parameter file *pverNM*
Subroutines called: None
Common data referenced: *num*, *numsel*, *numsub*, *numgo*
Input files: *pverNM*
Output files: None
Arrays used:
numsel(1500)–Reference numbers
numsub(1500)–Reference subfeature numbers
numgo(1500)–Optional shading pattern numbers
Called by: **index_versatec**, **index_calcomp**
Error checking and reporting: None
Constants: None
Program logic:
1. Set *num* = 1. *num* counts the reference numbers as they are read in.
2. Read first:
   numsel(1) (selected reference number).

   numsub(1) (reference subfeature number).
   numgo(1) (optional shading pattern number).
  If ***numsel(1)*** = −1, return.
3. Read next line into ***itemp1, itemp2, itemp3.***
  If ***itemp1*** = −1, return.
4. If the number is a duplicate of some existing member, go to step 3.
  If the number should be inserted between two existing members at position ***k***, go to step 6.
  Otherwise:
5. Add 1 to ***num***.
  Set:
   numsel(num) = itemp1
   numsub(num) = itemp2
   numgo(num) = itemp3
  Go to step 3.
6. Shift all members of the arrays from position ***k*** through ***num*** up one element.
  Set:
   numsel(k) = itemp1
   numsub(k) = itemp2
   numgo(k) = itemp3
  Add 1 to ***num***.
  Go to step 3.

```
      subroutine srtdup (in)
c
c           READ IN THE SELECTED FEATURE NUMBERS TO BE PLOTTED.
c           SORT THEM IN ASCENDEDING ORDER AND REMOVE
c           ALL DUPLICATIONS
c
      common /sortd/ ik,num,numsel(1500),numsub(1500),numgo(1500)
      num=1
      read (in,20)  numsel(1),numsub(1),numgo(1)
20    format (i8,i2,i2)
      if (numsel(1) .eq. -1)  go to 90
c
c           SORT AND REMOVE DUPLICATE CARDS AS EACH IS READ IN
c
30    read (in,20)  itemp1,itemp2,itemp3
      if (itemp1 .eq. -1)  go to 90
      do 70 k=1,num
      if ( (itemp1 .eq. numsel(k)) .and. (itemp2 .eq. numsub(k)) )
       go to 30
      if (itemp1 .gt. numsel(k))  go to 70
      if (itemp1.lt. numsel(k))  go to 50
40    if ( (itemp2 .lt. numsub(k)) .or. (itemp1 .lt. numsel(k)) )
       go to 50
      k=k+1
      if (k .gt. num)  go to 80
      go to 40
50    l=num+1
      do 60  j=k,num
      l=l-1
      numsel(l+1)=numsel(l)
      numsub(l+1)=numsub(l)
      numgo(l+1)=numgo(l)
60    continue
      numsel(k)=itemp1
      numsub(k)=itemp2
      numgo(k)=itemp3
      num=num+1
      go to 30
70    continue
80    num=num+1
```

```
        numsel(num)=itemp1
        numsub(num)=itemp2
        numgo(num)=itemp3
        go to 30
90      return
        end
```

SUBROUTINE NAME: PATTERN

Author: Lawrence Balcerak
Purpose of the program: **pattern** sets the shading pattern variables to user-defined values.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call pattern
Arguments: None
Subroutines called: None
Common data referenced: **pat**
Input files: None
Output files: None
Arrays used: *ip1(16), ip2(16) ip3(4), ip4(4), ip5(16), ip6(16), ip7(16), ip8(16), ip9(16), ip10(16)* (shading pattern arrays)
Called by: **index_versatec, verplot**
Error checking and reporting: None
Constants: None
Program logic:
1. Set all elements of the shading pattern arrays to those interger values that will give the bit patterns desired.

```
        subroutine pattern
        common /pat/ ip1(16),ip2(16),ip3(4),ip4(4),ip5(16),ip6(16),
                   & ip7(16),ip8(16),ip9(16),ip10(16)
        do 10 I=1,4
        ip3(i)=0
        ip4(i)=0
10      continue
        do 20 i=1,16
        ip1(i)=0
        ip2(i)=0
        ip5(i)=0
        ip6(i)=0
        ip7(i)=0
        ip8(i)=0
        ip9(i)=0
        ip10(i)=0
20      continue
c
        ip1(1)=4*16**4+1
        ip1(5)=16*ip1(1)
        ip1(9)=8*16**6+2*16**2
        ip1(13)=16*ip1(9)
c
        ip2(4)=ip1(13)
        ip2(8)=ip1(9)
        ip2(12)=ip1(5)
        ip2(16)=ip1(1)
c
        ip3(1)=4*16**4+1
c
        ip4(3)=4*16**8+2*16**6+16**4+8*16
c
        ip5(1)=4*16**7
        ip5(2)=16**8+16**7
```

```
      ip5(4)=2*16**8+8*16**6
      ip5(6)=ip5(2)
      ip5(7)=ip5(1)
      ip5(9)=16**3
      ip5(10)=4*16**3+4*16**2
      ip5(12)=8*16**3+2*16**2
      ip5(14)=ip5(10)
      ip5(15)=ip5(9)
c
      ip6(1)=8*16**7+8*16**6+4*16**5+4*16**4+2*16**3+2*16**2+16+1
c
      ip7(4)=16**4+8*16
      ip7(8)=4*16**8+4*16**7+2*16**6+16**4+16**3+8*16
      ip7(12)=4*16**8+2*16**6
      ip7(16)=ip7(8)
c
      ip8(1)=ip6(1)
      ip8(9)=ip6(1)
c
      ip9(2)=ip5(2)
      ip9(4)=ip5(1)
      ip9(6)=ip5(2)
      ip9(8)=4*16**8+2*16**6+16**4+8*16
      ip9(10)=ip5(10)
      ip9(12)=ip5(9)
      ip9(14)=ip9(10)
      ip9(16)=ip9(8)
c
      ip10(2)=ip5(9)
      ip10(4)=16**4+5*16**3+4*16**2+8*16
      ip10(6)=ip10(2)
      ip10(8)=4*16**7+16**3
      ip10(10)=ip5(7)
      ip10(12)=5*16**8+5*16**7+2*16**6
      ip10(14)=ip10(10)
      ip10(16)=ip10(8)
c
      return
      end
```

EXEC_COM: SORT.VERS.COOR.EC

Author: Donald Hanson
Purpose of the program: **sort.vers.coor.ec** executes the three programs written in the Multics command language and **system_sort** that produces the sorted coordinate file.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec sort.vers.coor *NM*
Arguments: *NM*–FIPS code
Subroutines called: **pgm1.vers.exthdr**, **pgm2.vers.sequent**, **pgm3.vers.merge**
Common data referenced: None
Input files: *coorNM.unsort* used on unit 10 (*file10*)
Output files:
coorNM.unsort.hdr used on unit 12 (*file12*)
coorNM.sequent used on unit 14 (*file14*)
coorNM.sort.hdr used on unit 10 (*file10*)
coorNM
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:

1. To execute this exec com, the operator must type:
   ec sort.vers.coor *NM*
   where *NM* is the State code.

2. A message will print that the first program has started executing. This program extracts the header records from the unsorted coordinate file.
3. The unsorted coordinate file (*coorNM.unsort*) is attached to *file10*, and the unsorted header file is attached to *file12*.
4. The next step is the system sort, which sorts the header file by feature number and subfeature number. These are the first 10 characters of the header record.
5. A message will next appear that program 2 has been started. This program takes the unsorted coordinate file and converts it from a stream to a sequential file.
6. The stream coordinate file is attached to *file13* and the sequential coordinate file is attached to *file14*.
7. When program 2 is completed, program 3 starts.
8. The third program merges the sorted header file and the unsorted coordinate file in to the sorted coordinate file.
9. The sorted header file is attached to *file10*. The sequential coordinate file is attached to *file13*.
10. *file13* is opened with a mode of sequential update in order to allow records to be deleted after they have been written to *file12*.
11. The sorted coordinate output file is attached to *file12*.
12. At completion, all files are closed and a message appears:
    JOB FINISHED

```
&  ***************************************************
&
&
&          sort.vers.coor.ec
&
&
&  ***************************************************
&
&       use:    ec sort.vers.coor.ec nm
&       where nm is state code
&
&       function:
&          this ec takes the unsorted coordinate file and produces
&          a sorted coordinate file. The first program extracts the
&          header records from the unsorted coordinate file. The header
&          records are then sorted in sort_seg. The second program takes
&          the unsorted coordinate file and converts it from a stream
&          to a sequential file. The third program merges the sorted
&          header file and the unsorted coordinate file into the sorted
&          coordinate file.
&
&print program1 started
&
io attach file10 vfile_ coor&1.unsort
io attach file12 vfile_ coor&1.unsort.hdr
pgm1.vers.exthdr
io detach file10
io detach file12
&
&print sort started
&
sort_seg coor&1.unsort.hdr -sm coor&1.sort.hdr -fl 1 10
&
&print program 2 started
&
io attach file13 vfile_ coor&1.unsort
io attach file14 vfile_ coor&1.sequent
pgm2.vers.sequent
io detach file13
io detach file14
&
&print program3 started
&
io attach file10 vfile_ coor&1.sort.hdr
io attach file13 vfile_ coor&1.sequent
io open file13 squ
io attach file12 vfile_ coor&1
pgm3.vers.merge
io detach file10
io close file13
io detach file13
io detach file12
&print coordinate file merge is complete
&print job finished
```

SUBROUTINE NAME: PGM1.VERS.EXTHDR

Author: Donald Hanson
Purpose of the program: **pgm1.vers.exthdr** extracts the header records from the files that were created on the Data General minicomputer. The input is the unsorted coordinate file created in **versatec.ec** and merged together in editor **qedx**. This program is the first program executed in **sort.vers.coor.ec**.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: pgm1.vers.exthdr
Arguments: None
Subroutines called: **close_files**
Common data referenced: None
Input files: *coorNM.unsort*
Output files: *coorNM.unsort.hdr*
Arrays used:
data(12)–Coordinates
i(8)–Header records
Called by: **sort.vers.coor.ec**
Error checking and reporting: None
Constants: None
Program logic:

1. The input file is attached by the I/O switch to *file10.*
2. The output file is attached by the I/O switch to *file12.*
3. The input file consists of header and coordinate records not in order by feature number. The input file is read and the header records only are extracted and written to the output file. The output file consists of header records not in order by feature number.
4. The input file is read until the end of file is reached, at which time the STOP message appears.

```
c       this is a program to extract the header records
c       from the files that were created on the Bendix minicomputer
c       written DHanson 4/5/78
        external cf(descriptors)
          dimension i(8)
          dimension data(12)
          print,"exthdr started"
   75    read(10,100,end=300) i
  100    format(8i5)
         write(12,150) i
  150    format(8i5,40x)
  170    read(10,200,end=300) data
  200    format (12f6.3)
         i(4) = i(4) - 6
         if(i(4).gt.0) go to 170
         go to 75
  300   call cf("-all")
        stop
         end
```

SUBROUTINE NAME: PGM2.VERS.SEQUENT

Author: Donald Hanson

Purpose of the program: **pgm2.vers.sequent** changes the unsorted coordinate records from stream to sequential, which is necessary because the position parameter in **pgm3.vers.merge** must operate on a sequential file. The input is the stream format unsorted coordinate file, and the output is the sequential format unsorted coordinate file. This program is the second program executed in **sort.vers.coor.ec.**

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Arguments: None

Operating system: Multics

Calling sequence: pgm2.vers.sequent

Subroutines called: **close_files**

Common data referenced: None

Input files: coorNM.unsort

Output files: coorNM.sequent

Arrays used:

ihead(*8*) – Header records

data(12) – Coordinate records

Called by: **sort.vers.coor.ec**

Error checking and reporting: None

Constants: None

Program logic:

1. The input file *coorNM.unsort* is attached by the I/O switch to *file13.*
2. The output file is attached by the I/O switch to *file14.*
3. The input file is read in a formatted stream mode, and the output file is written in an unformatted sequential mode.
4. The number of coordinate data points that follow each header card is contained in the fourth field of the header card.
5. As each data card is read, six is subtracted from the number of points because each record contains six pairs of coordinates.
6. When this number is no longer greater than zero, the next record is a header record.
7. This process is continued until the end of file is reached.

```
c      this is a program to change the unsorted coordinate
c      records from stream to sequential
c
external cf(descriptors)
dimension ihead(8),data(12)
75 read(13,100,end=300) ihead
   write(14) ihead
240 read(13,250,end=300) data
    write(14) data
    ihead(4) = ihead(4) - 6
    if(ihead(4).gt.0) goto 240
    goto 75
100 format(8i5)
250 format(12f6.3)
300  call cf("-all")
     stop
end
```

SUBROUTINE NAME: PGM3.VERS.MERGE

Author: Donald Hanson

Purpose of the program: **pgm3.vers.merge** merges the unsorted coordinate file and the sorted header file to form the sorted coordinate file. This file is then used for input to the **index_versatec** programs. This program is the third program executed in **sort.vers.-coor.ec**.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: pgm3.vers.merge

Arguments: None

Subroutines called: **io_call, close_files**

Common data referenced: None

Called by: **sort.vers.coor.ec**

Input files: coorNM.sort.hdr, coorNM.sequent

Output files: coorNM

Arrays used:

x(*6*) – x coordinate

y(*6*) – y coordinate

Error checking and reporting: None

Constants: None

Program logic:

1. There are two input files. The first input file, *coorNM.sort.hdr*, is attached by the I/O switch to *file10*.
2. The second input file, *coorNM.sequent*, is attached by the I/O switch to *file13*.
3. The output file is identified by the name *coorNM* and is attached by the I/O switch to *file12*.
4. The records from *file10* and *file13* are read. If feature number and subfeature number of *file10* match those of *file13*, the records are written to *file12*.
5. The records in *file13* that have been matched are then deleted from *file13* by the *delete_record* feature of **io_call**. By deleting these records that are no longer needed from the input file, total processing time is drastically reduced.
6. When the feature and subfeature numbers do not match between *file10* and *file13*, *file13* is advanced until a match is found.
7. When feature numbers cannot be matched *file13* is positioned to the next header records by the *char_skip* feature of **io_call**.
8. After a match has been found and the records written to *file12*, *file13* is positioned at the beginning of file, (BOF), by this feature of **io_call**. The cycle is then repeated until the end of file is reached.

```
c     this is a program to merge the unsorted coordinate
c     file and the sorted header file for versatec
external io(descriptors)
external cf(descriptors)
integer skip
 character*3 char_skip
dimension x(6),y(6)
75 read(10,100,end=300) iif,ifno,isf,isfno,not,nor,nif,ispan
120 read(13,end=300) iif1,ifno1,isf1,isfno1,not1,nor1,nif1,ispan1
if(iif1.eq.iif.and.isf.eq.isf1) goto 220
if (mod(isfno1,6).eq.0) goto 200
skip = ifix(float(isfno1)/6.0 + 1.0)
goto 400
200 skip = isfno1/6
400 encode(char_skip,500) skip
 call io("position","file13","fwd",char_skip)
goto 120
220 write(12,230)iif,ifno,isf,isfno1,not,nor,nif,ispan
    call io ("delete_record","file13")
240 read(13,end=300) (x(i),y(i),i=1,6)
    call io ("delete_record","file13")
    write(12,250) (x(i),y(i),i=1,6)
    isfno1 = isfno1 - 6
    if(isfno1.gt.0) goto 240
    call io ("position","file13","bof")
    goto 75
100 format(8i5)
230 format(8i5,40x)
250 format(12f6.3)
500 format(i3)
300 call cf("-all")
    stop
end
```

PROGRAM NAME: MASTER

Author: Harold Johnson
Purpose of the program: **master** uses the numerical coordinate files for maps of the State and the reference outlines to calculate the areas in square kilometers, to calculate reasonable center-point coordinates for each area, to test the reasonableness of these center points, and to output a file of center points that may not be suitably located.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: master
Arguments: None
Subroutines called: **arcntr_master**, **adjust_master**, **closer**, **ftnumber**
Common data referenced: ***true***, ***sk***, ***ia***
Input files:
  areano used on unit 37 (*file37*)
    List of variables: *it* = true State area in square miles.
    Formats: (I6)
    Layout description: See **areano**.
  statNM used on unit 31 (*file31*)
  coorNM used on unit 32 (*file32*)
Output files:
  areaNM used on unit 33 (*file33*)
    Format: (2I5, F8.1)
    File created by: **adjust_master**
    Layout description: *Area* is the summed areas of all outlines having the same *if*.
  measNM used on unit 40 (*file40*)
    List of variables: *if*, *isf*, *area*
    Formats: (2I5, F8.1)
    File created by: **master**
    Layout description: *Area* is the area of the outline with the corresponding *if*, *isf*.
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. Input files are identified with Fortran numbers: *statNM*, *coorNM*, are coordinate files for plotting the State outlines; **areano** is a file of State areas. The user is prompted for only the FIPS code number *ia* and for the scale used in the maps from which *statNM*, and *coorNM* were derived. Scale is 1 to *sk*.
2. **areano** is searched for the true area of this State.
3. **arcntr_master** is called to compute the areas of the outlines for this *coorNM* file, and areas are adjusted using the State areas.
4 . **adjust_master** is called to sum areas belonging to the same reference.

```
c  ******* MASTER *******
c  **********  PROGRAM  MASTER  ***************
c  ----------------------------------------------------------------------
c  |  THIS PROGRAM DOES THE FOLLOWING:                                   |
c  |  1.  USING "ARCNTR", MASTER CALCULATES THE STATE AREA FROM FILE      |
c  |  "STATNM",                                                          |
c  |  2.  READS THE TRUE STATE AREA FROM FILE "AREANO" AND FORMS THE      |
c  |  RATION OF THESE TO GIVE A CORRECTION FACTOR "FACTOR" WHICH IS      |
c  |  MULTIPLIES EVERY SUCCESSIVE AREA CALCULATION                        |
c  |  3.  EACH MAP OUTLINE IN "COORNM" IS THEN EXAMINED AND A CENTER     |
c  |  POINT IS CONSTRUCTED ALONG WITH AN AREA.  THESE ARE WRITTEN TO     |
c  |  THE FILES "CNTRNM" AND "MEASnm", RESPECTIVELY                       |
c  |  4.  USING "CNTEST" SUBROUTINE, THESE CENTERS ARE CHECKED FOR       |
c  |  THEIR NEARNESS TO THE BOUNDARY OF THE OUTLINE WHEN THE DIAMETER   |
c  |  OF THE OUTLINE IS LARGE, AND FOR THEIR POSITION INSIDE OR OUT-     |
c  |  SIDE OF THE BOUNDARY.  CENTERS WHICH FAIL THE TEST ARE WRITTEN     |
c  |  A FILE CALLED "DOUBT" AND TO FILE 6.                                |
c  |  5.  FINALLY THE SUBROUTINE "ADJUST" SUMS ALL AREAS HAVING THE      |
c  |  SAME "IF" NUMBER.  THE OUTPUT FILE IS "AREANM".                     |
c  |                                                                      |
c  |     THE FOLLOWING INPUT FILES ARE REQUIRED:                          |
c  |  FILE 5 = TO INPUT AREA CODE AND MAP SCALE.                          |
c  |  FILE 37 = LIST OF STATE CODES AND TRUE STATE AREAS, =AREANO"       |
c  |  FILE 31 = "STATNM"                                                  |
c  |  FILE 32 = "COORNM"                                                  |
c    |                                                                  |
c    |      THE FOLLOWING OUTPUT FILES ARE REQUIRED:                    |
c    |   FILE 33 = "AREANM" (SUMMED AREAS)                              |
c    |   FILE 34 = "CNTRNM" (CENTER POINTS)                             |
c    |   FILE 15 = "DOUBT" (DOUBTFUL CENTER POINTS)                     |
c    |   FILE 40 = "MEASnm" (ALL AREAS BY SEPARATE IF, ISF)             |
c    |   FILE 6 = MESSAGES TO TERMINAL                                  |
c    |                                                                  |
c    |      NOTE:  WHEN FINISHED, CHANGE "MEASnm" TO "MEASNM" FOR FUTURE |
c    |   USE.                                                           |
c    |                                                                  |
c    ----------------------------------------------------------------
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c converted to multics May 7, 1977 by H Johnson.
c
c
c PROGRAM TO CALCULATE AREAS AND CENTERS FOR OUTLINES, DETERMINE WHEN
c CENTERS ARE INSIDE THE OUTLINES AND WHEN PROPERLY CENTERED.
c IT ALSO SUMS AREAS WITH THE SAME IF.
c
   common true,sk,ia
  character file*4, state*2, outfile*6, mode*4
c
    write(6,880)
880 format(" THIS PROGRAM USES THE FILES coorNM AND statNM TO"/
" PRODUCE NEW FILES areaNM, cntrNM, measNM AND doubt."/
"0THERE WILL BE ERRORS IF THESE FILES ALREADY EXIST FROM "/
" PREVIOUS RUNS."/
" IF areaNM, measNM  ALREADY EXIST, HIT BREAK. "/
" DELETE THESE FILES AND RUN MASTER AGAIN.")
c
write(6,900)
900 format("0ENTER IN THE 2-DIGIT CODE FOR THE STATE BEING STUDIED")
read(5,910) state
910 format(a2)
 decode(state,913) ia
913 format(i2)
write(6,914)
914 format(" YOUR MAP HAS SCALE 1 TO WHAT - F8.0 -?")
read(5,916) sk
916 format(f8.0)
outfile = "areano"
mode = "si"
  call ftnumber(37,outfile,mode)
file = "coor"
encode(outfile,920) state
920 format("coor",a2)
  call io ("attach","file32","vfile_",outfile,"-append","-ssf")
   call io("open","file32","si")
encode(outfile,922) state
922 format("stat",a2)
  call ftnumber(31,outfile,mode)
mode = "so"
encode(outfile,924) state
924 format("area",a2)
  call ftnumber(33,outfile,mode)
```

```
mode = "sio "
encode(outfile,927) state
927 format("meas",a2)
  call ftnumber(40,outfile,mode)
outfile = "doubt"
mode = "so  "
  call ftnumber(15,outfile,mode)
do 7 jj=1,ia
read(37,930)it
7 continue
930 format(i6)
true = float(it)
     call arcntr_master
rewind 40
     call adjust_master
  call closer(37)
  call closer(32)
   call closer(31)
   call closer(33)
   call closer(40)
   call closer(15)
c
c NOW SORT THE MEASNM FILE
c
encode(outfile,952) state
952 format("meas",a2)
   call sort_seg(outfile,"-fl","1","10")
c
stop
end
```

SUBROUTINE NAME: ARCNTR_MASTER

Author: Harold Johnson

Purpose of the program: **arcntr_master** calculates the areas, corrected, of the outlines whose coordinates are contained in *coorNM*

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call arcntr_master

Arguments: None

Subroutines called: **weight_master**

Common data referenced: *true*, *sk*, *ia*

Input files:

statNM used on unit 31 (*file31*)
coorNM used on unit 32 (*file32*)

Output files: *measNM* used on unit 40 (*file40*)

Arrays used: *xx(6)*, *yy(6)*

Called by: **master**

Error checking and reporting: True State area is reported to the user, along with the area calculated for each part of the possibly multiconnected State boundary. An estimate is given for each of these closed segments. The ratio of calculated to true State area is also given to the user. Where this differs markedly from 1, an error may have occurred.

Constants: None

Program logic:

1. The first file processed is *areaNM*, the coordinate file for the State boundary. A header card is read, noting the total number of data points, *isfno*. The successive data cards are read, and a calculation is made. Area is incremented by the amount
$$((x_0 - x_1)(y_0 + y_1))/2$$
for each segment in the coordinate file running from (x_0, y_0) to (x_n, y_n).
2. While the State coordinates are being processed, all areas are summed and used to give a *factor*, the ratio of the true State area to the State area calculated for this map projection; this ratio will be used in all successive calculations as a factor to modify those calculations.
3. The second file, *coorNM*, is now processed as in 1, and the calculated centers and areas are written to file *measNM*.

```
c ******* SUBROUTINE ARCNTR_MASTER *******
           subroutine arcntr_master
c**********  WEIGHTED AVERAGE CENTER AND AREA PROGRAM  ***********
c SUBROUTINE OF THE MAIN PROGRAM "MASTER".
c UPDATED AS OF DEC. 27,1976  H. JOHNSON
c converted to Multics May 7, 1977 by H Johnson
c
           common true,sk,ia
           dimension xx(6),yy(6)
c
c EQUATE 30 TO THE SOURCE FILE TAREA, CONTAINING THE TRUE AREA IN
c   SQUARE MILES
           write(6,9799)true
9799        format(" true area = ",f10.3)
c EQUATE 31 TO THE STATE BOUNDARY FILE, STAT--
c EQUATE 32 TO THE COORDINATE FILE, COOR--
c EQUATE 40 TO THE TEMPORARY AREA FILE ARTEMP
c EQUATE 34 TO THE NEW CENTER FILE, CNTR--
c
           factor = 1.0
          item=25
           scale=645.16
901        format(12f6.3)
900        format(8i5)
c
c FIRST, COMPUTE THE STATE MAP AREA IN SQUARE KILOMETRES.
c
           in=31
           ioarea=6
           iocntr=6
           nrun=1
           totar=0.0
100        read(in,900,end=99)  if,ifno,isf,isfno,not,nor,nif,ispan
c
c  WHEN ISFNO IS LESS THAN 4 WE DON'T HAVE A REGION AT ALL
c
          if(isfno.gt.3)go to 102
           read(in,901,end=99)(xx(i),yy(i),i=1,6)
c
c IN THE CASE OF A SINGLE POINT, CALL THAT POINT CNTR.
c
           ixc=xx(2)*1000. +.5
           iyc=yy(2)*1000. +.5
           isfno=2
           go to 100
102        continue
c
c CALCULATE NCARDS, THE NUMBER OF DATA CARDS ON THIS MAP
c
           ncards=isfno/6
           if(6*ncards .lt. isfno) ncards=ncards+1
c
c
           area=0.
           xc=0.
```

```
            yc=0.
            dt=0.
c
c   XC IS GOING TO BE THE X-COORDINATE OF THE CENTER
c   YC IS GOING TO BE THE Y/COORDINATE OF THE CENTER
c   DT IS THE ACCUMULATED NORMED DISTANCE BETWEEN POINTS
c
c   READ IN THE FIRST DATA CARD
c
            read(in,901,end=99)(xx(i),yy(i),i=1,6)
            xstart=xx(2)
            ystart=yy(2)
            ie=6
            if(ncards .eq. 1) ie=isfno
            do 200 j=3,ie
            j1=j-1
            area = area + (xx(j)-xx(j1))*(yy(j)+yy(j1))*0.5
            call weight_master(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
200         continue
            if(ncards .eq. 1) go to 500
            xlast=xx(6)
            ylast=yy(6)
c
c
            if(ncards .eq. 2) go to 400
c
c   READ IN THE MIDDLE CARDS, BETWEEN THE FIRST AND LAST.
c
            kl=ncards-1
            do 300 k=2,kl
            read(in,901)(xx(i),yy(i),i=1,6)
            area=area+(xx(1)-xlast)*(yy(1)+ylast)*0.5
            call weight_master(xlast,ylast,xx(1),yy(1),xc,yc,dt)
             do 301 j=2,6
            j1=j-1
            area = area + (xx(j)-xx(j1))*(yy(j)+yy(j1))*0.5
            call weight_master(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
301         continue
            xlast=xx(6)
            ylast=yy(6)
300         continue
400         continue
c
c   NOW READ IN THE LAST CARD
c
            read(in,901)(xx(i),yy(i),i=1,6)
            ie=isfno-6*(ncards-1)
            if(ie .eq. 0) ie=6
            area=area+(xx(1)-xlast)*(yy(1)+ylast)*0.5
            call weight_master(xlast,ylast,xx(1),yy(1),xc,yc,dt)
            if(ie .eq. 1) go to 500
            do 401 j=2,ie
            j1=j-1
            area = area + (xx(j)-xx(j1))*(yy(j)+yy(j1))*0.5
            call weight_master(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
401         continue
```

```
500          continue
c
c  WHEN THE REGION IS NOT CLOSED, WE MUST ADD THE LAST DATA POINT
c
            test=(xx(ie)-xstart)**2 + (yy(ie)-ystart)**2
            if(test.lt..01) go to 501
            call weight_master(xx(ie),yy(ie),xstart,ystart,xc,yc,dt)
            area=area+(xstart-xx(ie))*(ystart+yy(ie))*0.5
501         continue
            if(nrun.gt.1)go to 503
            fisfno=isfno-1
            error=dt*.001+.000001*(fisfno)
            error=error*scale*((sk/1000000.)**2)
            write(6,903)if,isf,error
903         format(" THE AREA CALCULATION FOR IF =",i5," ISF =",i5,
           " HAS ERROR BOUNDED BY",f10.3)
503         continue
             area=abs(area*scale*factor*((sk/1000000.)**2))
            xc=xc/dt
            yc=yc/dt
           isfno=2
905         format(8i5)
            ixc=xc*1000.
            iyc=yc*1000.
908         format(12x,2i6)
write(ioarea,907)if,isf,area
            if(if .eq. 995)go to 550
            totar=totar + area
907 format(2i5,f8.1)
550         if(ispan .eq. 0)go to 100
            ispan=-ispan
            if1=if+1
ione = 1
            do 600 j=if1,ispan
write(ioarea,907)j,ione,area
600         continue
c ON THE STATE AREA RUN, WE WANT TO COMPUTE FACTOR,
c WHICH IS THE RATIO OF THE TRUE TO CALCULATED AREAS.
c
            go to 100
99          if(nrun .gt. 1)return
c CHANGE TRUE TO SQUARE KILOMETRES AND COMPUTE RATIO.
            true=true*2.59
           write(6,9898)true
9898        format (f10.3)
            factor=true/totar
c NOW SET INPUTS TO READ THE COORDINATE FILES
            write(6,9899)factor
9899        format(" FACTOR =",f10.3)
c
            nrun=2
           totar=0.0
            in=32
            ioarea=40
            go to 100
            end
c ******* END ARCNTR_MASTER *******
```

SUBROUTINE NAME: WEIGHT_MASTER

Author: Harold Johnson

Purpose of the program: **weight_master** is used to modify the previously calculated center point by means of a weighted average of the midpoint of a new edge of the outline.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call weight_master (x1,y1,x2,y2,xc,-yc,td)

Arguments:

x1, y1—Coordinates of one end of the new segment of the outline

x2, y2—Coordinates of the other end

xc, yc—Coordinates of the center point

td—Sum of the squares of the lengths of the outline segments

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **arcntr_master**

Error checking and reporting: None

Constants: None

Program logic:

1. The square of the length of the segment is calculated, *z*.
2. The average of the x coordinates of the segment is multiplied by *z* and added to ***xc***.
3. The average of the y coordinates of the segment is multiplied by *z* and added to ***yc***.
4. *z* is added to ***td***. (In the calling program, after all calculations on an outline are completed, ***xc*** and ***yc*** are divided by ***td***.)

```
c ******* SUBROUTINE WEIGHT_MASTER *******
            subroutine weight_master(x1,y1,x2,y2,xc,yc,td)
c SUBROUTINE USED IN MAIN PROGRAM "MASTER"
c UPDATED AS OF DEC. 27, 1976   H. JOHNSON
c
c  converted to multics May 6, 1977 H. Jonnson
c
            z=abs(x2-x1) + abs(y2-y1)
            xc=xc + 0.5*(x2+x1)*z
            yc = yc + 0.5*(y2+y1)*z
            td = td + z
            return
           end
c ******* END WEIGHT_MASTER *******
```

SUBROUTINE NAME: ADJUST_MASTER

Author: Harold Johnson

Purpose of the program: **adjust_master** is used to sum the areas of all outlines having the same reference number, *if*.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call adjust_master

Arguments: None

Subroutines called: None

Common data referenced: None

Input files:

measNM used on unit 40 (file40)

List of variables: *if*, *isf*, ***area***

Formats: See **master**.

Layout description: See **master**.

Output files:

areaNM used on unit 33 (*file33*)

List of variables: *if1*, *isf1*, ***area1***

Formats: See **master**.

File located by: **adjust_master**

Layout description: See **master**.

Arrays used: None

Called by: **master**

Error checking and reporting: None

Constants: ***inarea*** = 40; ***ioarea*** = 33

Program logic:

1. The area file ***measNM*** is rewound.
2. Areas are successively read, and if two are from the same *if*, their areas are summed.
3. Each time a new *if* is found, the old area is written to ***areaNM***.

```
c ******* SUBROUTINE ADJUST_MASTER *******
          subroutine adjust_master
c
c SUBROUTINE USED IN MAIN PROGRAM "MASTER".
c UPDATED AS OF DEC. 27, 1976   H. JOHNSON
c
c converted to multics May 6, 1977 H. Johnson
c ***ADJUST AREA FILES
c
c IN CASE OF SEVERAL OUTLINES WITH THE SAME IF, THIS PROGRAM
c SUMS UP THESE AREAS
c
         inarea = 40
         ioarea = 33
c EQUATE 40 TO THE AREA SOURCE FILE ARTEMP
c EQUATE 33 TO THE OUTPUT AREA FILE AREA--
c
1         read(inarea,900,end=99)if,isf,area
900       format(2i5,f8.1)
2         read(inarea,900,end=99)if1,isf1,area1
          if(if1 .eq. if) go to 3
          write(ioarea,900)if,isf,area
          if=if1
          area=area1
          go to 2
3         area=area+area1
          go to 2
c
99        write(ioarea,900)if,isf,area
          return
         end
c ******* END ADJUST_MASTER *******
```

SUBROUTINE NAME: WORK_MASTER

Author: Harold Johnson

Purpose of the program: **work_master** calls two functions that are used in computing the distance from the center point to the boundary and in counting how many times the horizontal ray to the right of the center crosses the boundary.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call work_master (x1,y1,x2,y2,xcen,-nx,dst)

Arguments:

x1, y1 – Coordinates of one end of a segment of the boundary

x2, y2 – Coordinates of the other end of the segment

xcen, ycen – Coordinates of the center

nx – The number of times the boundary crosses the horizontal ray from the center point

dst – The minimum distance from the center point to the boundary

Subroutines called: **dist_master, ncross_master**

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **cntest_master**

Error checking and reporting: None

Constants: None

Program logic:

1. The two functions **dist** and **ncross** are called to possibly modify the values of *nx* and *dst*.

```
c ******* SUBROUTINE WORK_MASTER *******
          subroutine work_master (x1,y1,x2,y2,xcen,ycen,nx,dst)
c
c SUBROUTINE USED IN MAIN PROGRAM "MASTER"
c UPDATED AS OF DEC. 27, 1976 H. JOHNSON
c
c converted to multics May 6, 1977 by H Johnson.
c
         dst=dist_master(x1,y1,x2,y2,xcen,ycen,dst)
         nx=ncross_master(x1,y1,x2,y2,xcen,ycen,nx)
         return
        end
c ******* END WORK_MASTER *******
```

FUNCTION NAME: DIST_MASTER

Author: Harold Johnson

Purpose of the program: **dist_master** computes the distance from the center to a segment whose endpoints are given. This distance is compared with a previously calculated minimum distance, and the minimum of the two is returned as the value of ***dist***.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: dst = dist_master (x1,y1,x2,y2,x0,y0,-dold)

Arguments:

x1, y1–Coordinates of one end of the segment

x2, y2–Coordinates of the other end of the segment

x0, y0–Coordinates of the center point

dold–Previously calculated minimum distance

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **work_master**

Error checking and reporting: None

Constants: None

Program logic:

1. The segment is tested to determine whether its length is zero; if it is zero, ***dst = dold***.
2. The point is found on this one line through ***x1, y1*** and ***x2, y2*** that is closest to ***x0, y0***.
3. When this point is outside the line segment, the nearest endpoint is used as the nearest point.
4. Distance from the center to the nearest point is computed and compared with ***dold***. The smaller value is returned as the value.

```
c ******* SUBROUTINE DIST_MASTER *******
          function dist_master(x1,y1,x2,y2,x0,y0,dold)
c subroutine used in main program master.
c converted to multics May 6, 1977 by H Johnson.
c
c DOLD IS THE OLD MINIMAL DISTANCE FROM (X0,Y0) TO THE
c BOUNDARY.  DST = DOLD OR THE DISTANCE FROM (X0,Y0) TO THE SEGMENT
c WHICH RUNS FROM (X1,Y1) TO (X2,Y2) - WHICHEVER IS SMALLER.
         itest=((x2-x1)**2+(y2-y1)**2)*1000.
         if(itest .eq. 0) dist_master=dold
          if(itest .eq. 0) go to 73
          t=(x1-x0)*(x1-x2)+(y1-y0)*(y1-y2)
         test=(x2-x1)**2+(y2-y1)**2
          t=t/test
          xt=t*x2+(1.-t)*x1
          yt=t*y2+(1.-t)*y1
          dist_master=sqrt((xt-x0)**2+(yt-y0)**2)
          if(t .lt. 0.)dist_master=sqrt((x1-x0)**2+(y1-y0)**2)
          if(t .gt. 1.)dist_master=sqrt((x2-x0)**2+(y2-y0)**2)
          if(dold .lt. dist_master) dist_master=dold
73        continue
c
          return
        end
c ******* END DIST_MASTER *******
```

FUNCTION NAME: NCROSS_MASTER

Author: Harold Johnson

Purpose of the program: **ncross_master** determines whether or not the right horizontal ray from the center point crosses the line segment that has the given end points. If it does cross, ***nold*** is increased by 1.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: nx = ncross_master (x1,y1,x2,y2,x0,-y0,nold)

Arguments:

x1, y1– Coordinates of one end of the segment

y2, y2– Coordinates of the other end of the segment

x0, y0– Coordinates of the center point

nold– The number of crossings before this program

nx– The number of crossings after calling this program (increases ***nold*** by 1 if a crossing occurs in this routine)

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **work_master**

Error checking and reporting: None

Constants: None

Program logic:

1. If the segment is nearly horizontal, ***nx*** = ***nold.***
2. The horizontal coordinate where the horizontal crosses the line through the points of the segment is calculated.
3. When this coordinate is less than ***x0***, ***nx*** = ***nold.***
4. Otherwise, ***nx*** = ***nold*** + 1.

```
c ******* NCROSS_MASTER *******
          function ncross_master(x1,y1,x2,y2,x0,y0,nold)
c function used in main program master
c converted to multics May 6, 1977 H Johnson.
c
          io=6
c  NCROSS INCREASES NOLD BY 1 IF THE SEGMENT FROM (X1,Y1) TO (X2,Y2)
c SROSSES THE RIGHT HORIZONTAL RAY FROM (X0,Y0) .
          ncross_master=nold
          if(abs(y1-y2) .lt. .001) go to 7
          t=(y0-y1)/(y2-y1)
8         if(t.lt.0. .or. t.gt.1. ) go to 7
5        x=t*x2 + (1.-t)*x1
          if(x .lt. x0) go to 7
          ncross_master=ncross_master+1
7         continue
           return
         end
c ******* END NCROSS_MASTER *******
```

SUBROUTINE NAME: CNTEST_MASTER

Author: Harold Johnson

Purpose of the program: **cntest_master** tests whether the center points are actually within the map outlines. It measures their distance to the boundary of the outline and computes the diameter of the outline. When the point lies outside the outline or is too close to the boundary of a region whose diameter is not small, error messages to this effect are sent to the operator, and the center is written to a file named ***doubt.***

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics
Calling sequence: call cntest_master
Arguments: None
Subroutines called: **work_master, dm_master**
Common data referenced: None
Input files: None
Output files: None
Arrays used: xx(6), yy(6)
Called by: **master**
Error checking and reporting: When the boundary crosses the right horizontal ray from the center an even number of times, a message that the center does not lie inside the outline is sent to the operator. When the center is too close to the edge of a region, a message about this is sent to the operator. In each case, the center coordinates are placed in *doubt*.

Constants: None
Program logic:
1. Center header card and coordinates are read. The center is shifted by 0.0001 in., so that it cannot lie exactly on a line.
2. Coordinate file is searched for the corresponding header card.
3. Coordinates are read and **work_master** and **dm_master** are called to compute successive minimal distances, maximal distance from the initial coordinate point of the outline (used as an approximation to the diameter), and number of crossings that the outline makes over the right-hand ray from the center.
4. Indicated problems are reported to the user and written to *doubt*.

```
          subroutine cntest_master
c SUBROUTINE USED IN PROGRAM "MASTER"
c UPDATED AS OF DEC. 27, 1976   H. JOHNSON
c
c converted to multics May 6, 1977 by H Johnson
c$$$$$$$$$$$$$$$CENTER  TEST$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c
c  THIS PROGRAM TESTS WHETHER POINTS CALLED "CENTER POINTS"
c ARE ACTUALLY INSIDE THEIR MAP OUTLINES.
c
c IT ALSO COMPUTES THEIR DISTANCE TO THEIR MAP BOUNDARIES.
c
          dimension xx(6),yy(6)
c
          io1=15
          in=32
          io=6
          in1=34
          write(6,900)
900       format("  THE FOLLOWING CENTERS ARE IN DOUBT")
c
c EQUATE 32 TO THE FILE OF MAP OUTLINES, COOR--
c EQUATE 15 TO THE FILE  DOUBT .
c EQUATE 34 TO THE FILE OF CENTERS, CNTR--
c
          rewind 32
c  IT IS ASSUMED THAT THE CENTER DATA AND MAP OUTLINE DATA
c ARE IN THE SAME ORDER, AND THAT EACH CENTER BELONGS TO SOME
c MAP OUTLINE, BUT IT SKIPS OVER MAP OUTLINES WHICH HAVE NO DENTER.
c
1         read(in1,902,end=99)ifc,ifnoc,isfc,isfnoc,notc,norc,nifc
9902      format(1x,7i5)
          read(in1,901)inull,jnull,xcen,ycen
901       format(2i6,2f6.3)
c
c  ADD .0001 TO XCEN AND YCEN TO MAKE THEM DIFFERENT FROM
c ANY DATA POINT.
```

```
c
           xcen=xcen+.0001
           ycen=ycen+.0001
c
c  READ THE DATA HEADER CARD
2          read(in,902,end=99)if,ifno,isf,isfno,not,nor,nif
902        format(7i5)
           ncards=isfno/6
           if(6*ncards .lt. isfno) ncards=ncards+1
c
c COMPARE NUMBERS.  IF DIFFERENT,SKIP THIS DATA
           if((ifc.eq.if).and.(isfc.eq.isf).and.(notc.eq.not)) go to 200
           do 50 k=1,ncards
           read(in,903)(xx(i),yy(i),i=1,6)
903        format(12f6.3)
50         continue
           go to 2
200        continue
c
c
c IGNORE CENTER FILES FOR SINGLE POINT PLOTS.
c
           if(isfno .gt. 2) go to 210
           read(in,903)(xx(i),yy(i),i=1,6)
           go to 1
c
c  NOW READ THE FIRST DATA CARD AND BEGIN THE CALCULATIONS.
c  WE SKIP THE FIRST DATA POINT.  IT LOCATES PRINTING FOR
c THE ID NUMBERS.
c
210        dst=100.
           diam=0.0
           nx=0
           read(in,903)(xx(i),yy(i),i=1,6)
           ie=6
           if(isfno .lt. 6) ie=isfno
           xfirst=xx(2)
           yfirst=yy(2)
           do 300 k=3,ie
           call work_master(xx(k-1),yy(k-1),xx(k),yy(k),xcen,ycen,nx,dst)
           diam=dm_master(xx(k-1),yy(k-1),xfirst,yfirst,diam)
9903       format(f6.3)
300        continue
           xlast=xx(6)
           ylast=yy(6)
           if(ncards .eq. 1) go to 500
           if(ncards .eq. 2) go to 400
c
c NOW READ IN THE DATA CARDS BETWEEN THE FIRST AND LAST.
c
           n1=ncards-1
           do 380 l=2,n1
           read(in,903)(xx(i),yy(i),i=1,6)
           call work_master(xlast,ylast,xx(1),yy(1),xcen,ycen,nx,dst)
           diam=dm_master(xlast,ylast,xfirst,yfirst,diam)
```

```
             do 350 k=2,6
             call work_master(xx(k-1),yy(k-1),xx(k),yy(k),xcen,ycen,nx,dst)
             diam=dm_master(xx(k-1),yy(k-1),xfirst,yfirst,diam)
350          continue
             xlast=xx(6)
             ylast=yy(6)
380          continue
c
c NOW READ IN THE LAST CARD
c
400          read(in,903)(xx(i),yy(i),i=1,6)
             ie=isfno-6*(ncards-1)
             call work_master(xlast,ylast,xx(1),yy(1),xcen,ycen,nx,dst)
             diam=dm_master(xlast,ylast,xfirst,yfirst,diam)
             if(ie .eq. 1) go to 500
             do 401 k=2,ie
             call work_master(xx(k-1),yy(k-1),xx(k),yy(k),xcen,ycen,nx,dst)
             diam=dm_master(xx(k-1),yy(k-1),xfirst,yfirst,diam)
401          continue
c
500           continue
c
c WHEN THE REGION IS NOT CLOSED WE ADD THE LAST POINT TO CLOSE IT.
c
            test=(xx(ie)-xfirst)**2+(yy(ie)-yfirst)**2
             if(test .lt. .01) go to 501
            call work_master(xx(ie),yy(ie),xfirst,yfirst,xcen,ycen,nx,dst)
             diam=dm_master(xx(ie),yy(ie),xfirst,yfirst,diam)
c
c NOW REPORT THE RESULTS
c
501         continue
c
60           nx=(-1)**nx
c
c WHEN NX=-1 THERE ARE AN ODD NUMBER OF CROSSINGS, SO THE CENTER IS
c INSIDE THE REGION
c
c
c  WHEN NX=1 THERE ARE AN EVEN NUMBER OF CROSSINGS,SO
c  THE CENTER IS OUTSIDE THE REGION
c
9991        format(f6.3)
             if((nx .eq. -1) .and. (dst .gt. .09)) go to 1
            if(diam .lt. .3333) go to 1
write(6,9981)
9981 format("0the following center is in doubt because :")
if(nx .eq. -1) go to 70
write(6,9983)
9983 format(" it does not lie inside the boundary.")
70 if(dst .gt. .09) go to 80
write(6,9985) diam,dst
9985 format(" the region has diameter ",f6.3," and the center "/
" is ",f6.3," incnes from the boundary.")
```

```
80      write(io,905)if,ifno,isf,isfno,not,nor,nif
905        format(1x,7i5)
           write(io,906)xcen,ycen
906        format(1x,2f6.3)
            ioarea=40
           iycen=ycen*1000.
           ixcen=xcen*1000.
           isfno=1
            write(io1,902)if,ifno,isf,isfno,not,nor,nif
           write(io1,907)ixcen,iycen
907         format(12x,2i6)
           go to 1
99          return
            end
```

SUBROUTINE NAME: DM_MASTER

Author: Harold Johnson
Purpose of the program: **dm_master** is used to compute the diameter of an outline. It calculates the length on one segment and compares it with the previously calculated diameter.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: diam = dm_master (x1,y1,x2,y2,dold)
Arguments:
x1, *y1* – Coordinates for one endpoint of a segment
x2, *y2* – Coordinates for the other endpoint
dold – Previously calculated maximum diameter

Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **cntest_master**
Error checking and reporting: None
Constants: None
Program logic:
1. *diam* is set to *dold*.
2. The distance between the endpoints is calculated and compared with *dold*.
3. If this distance is more than *dold*, *diam* is set to this distance.

```
c ******* SUBROUTINE DM_MASTER *******
            function dm_master(x1,y1,x2,y2,dold)
c  function used in main program master.
c  converted to multics May 6, 1977  H. Johnson
c
             dm_master=dold
            test=abs(x1-x2)+abs(y1-y2)
            if(test .gt. dold)dm_master=test
            return
            end
c ******* END DM_MASTER *******
```

FILE NAME: AREANO

Purpose of the file: **areano** is a list of true State areas. **master** calculates a State area and then reads **areano** to find the true area. The ratio of these is used as a correcting factor in the area calculations for the outlines of the State.

Format: The *k*th record of **areano** contains the true area, format I6 (integer part), for the State with FIPS code number *k*. If no State has State code *k*, the record contains a zero.
Arguments: One integer occurs on each record, I6, and is the integral part of the true State area in square miles.
Referenced by: **master**

```
      areano

 51609
586412
0
113909
 53104
158693
0
104247
  5009
  2057
    67
 58560
 58876
0
  6450
 83557
 56400
 36291
 56290
 82264
 40395
 48523
 33215
 10577
  8257
 58216
 84068
 47716
 69686
147138
 77227
110540
  9304
  7836
121666
 49576
 52586
 70665
 41222
 69919
 96981
 45333
0
  1214
 31055
 77047
 42244
267339
 84916
  9609
 40817
0
 68192
```

```
   24181
   56154
   97914
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
    3435
```

PROGRAM NAME: STATE_OPTIMA

Author: Harold Johnson
Purpose of the program: **state_optima** reads through a *strdNM* file, finds the highest and lowest latitudes, leftmost and rightmost longitudes, and prints out this information.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: state_optima
Arguments: None
Subroutines called: **open_si.ec**, **optima**, **close.ec**
Common data referenced: None
Input files: *strdNM* used on unit 60 (*file60*)
Output files: Information is printed on the terminal.
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. Prompt:
   TYPE THE FEDERAL STATE CODE NUMBER:
2. The user's response is read into *state*.
3. The word *strd* and the value of state are concatenated.
4. Call **open_si.ec**, which opens and attaches *strdNM* to *file60* for stream input.
5. Call subroutine **optima**.
6. Call **close.ec**, which detaches and closes *file60*.
7. End.

```
c state_optima
c
c Purpose: To read through a strdNM file, find the highest
c   and lowest latitudes, hleft and right-
c   most loingitudes, and print out this information.
c
c Programmer: H Jonnso
c Date: July 20, 1978
c
c input file: strdNM
c output file: terminal
external ec(descriptors), dms
      character file*6
c
    write(6,910)
910 format(" Type the Federal State Code number:")
    read(5,920) state
```

```
920 format(a2)
   encode(file,930) state
930 format("strd",a2)
c
   call ec ("open_si","60",file)
c
c
  call optima
   call ec ("close","60")
c
c
end
```

SUBROUTINE NAME: OPTIMA

Author: Harold Johnson
Purpose of the program: **optima** reads through the radian coordinate files and determines the uppermost, lowermost, leftmost, and rightmost coordinates for each map boundary.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call optima
Arguments: None
Subroutines called: **dms**
Common data referenced: None
Input files: *strdNM*
Output files: Information is printed on the terminal.
Arrays used: *x(3), y(3), x1(4), y1(4) ix1(4), iy1(4), z(4), ideg(4), imin(4), isec(4)*
Called by: **state_optima**
Error checking and reporting: None
Constants: None
Program logic:

1. The header record of the *strdNM* file is read.
2. Set *ncards* equal to *isfno* divided by 3.
3. Read the first record.
4. Ignore the first point because it is the location of the written *if* numbers.
5. Compare the points to find the rightmost, leftmost, topmost, and lowermost points.
6. Put the coordinates into integer format for packed writing.
7. Call function **dms**.
8. The following will be output on the screen:

```
     STATE = nor, OUTLINE = if, SUB = isf
              DEGREES,MINUTES, SECONDS

NORTH
  MAXIMUM   ideg(1)   imin(1)   isec(1)
SOUTH       ideg(2)   imin(2)   isec(2)
WEST        ideg(3)   imin(3)   isec(3)
EAST        ideg(4)   imin(4)   isec(4)
```

9. This organization is as in the *comxNM* files: upper latitude, lower latitude, left longitude, and right longitude.
10. Return control to calling module.

```
subroutine optima
c
c purpose: to read through the radian coordinate files and
c   determine the upper, lower, left, and right most
c   coordinates for each map boundary.
c
c
c input file number 60 is the standrd coordinate file.
c outpue file number 30 is a file of max, mins.
c
double precision x,y,x1,y1,z
dimension x(3), y(3), x1(4), y1(4), ix1(4), iy1(4)
dimension z(4), ideg(4), imin(4), isec(4)
c
10 read(60,910,end=1000) if, ifno, isf, isfno,not,nor,nif
910 format(7i5)
```

```
c
c
ncards = isfno/3
if(3*ncards .lt. isfno) ncards = ncards + 1
c
    read(60,920)(x(j),y(j),j=1,3)
920 format(6f12.9)
c
do 15 j = 1, 4
x1(j) = x(2)
15 y1(j) = y(2)
c we don't start with the first point, because it is
c the location of the written if numbers.
j0 = 2
17 do 80 j = j0, 3
if(x(j) .eq. 0. .or. y(j) .eq. 0.) go to 90
c
c x1(1), y1(1) is the right-most point.
if(x1(1) .ge. x(j)) go to 20
x1(1) = x(j)
y1(1) = y(j)
20 continue
c
c x1(2), y1(2) is the left-most point.
if(x1(2) .le. x(j)) go to 30
x1(2) = x(j)
y1(2) = y(j)
30 continue
c
c x1(3), y1(3) is the top-most point.
if(y1(3) .ge. y(j)) go to 40
y1(3) = y(j)
x1(3) = x(j)
40 continue
c
c x1(4), y1(4) is the lowest point.
if(y1(4) .le. y(j)) go to 80
y1(4) = y(j)
x1(4) = x(j)
80 continue
c
ncards = ncards - 1
if(ncards .lt. 1) go to 90
read(60,920)(x(j),y(j),j=1,3)
j0 = 1
go to 17
c
90 isfno = 2
c
c put the coordinates into integer format for packed
c writing.
c
do 200 j = 1,4
   ix1(j) = idint(x1(j)*(10.0**9) + .5)
```

```
200 iy1(j) = idint (y1(j)*(10.0**9) + .5)
c
z(1) = y1(3)
z(2) = y1(4)
z(3) = x1(1)
z(4) = x1(2)
do 106 k = 1, 4
z(k) = z(k)*180./3.141592653
   call dms(z(k),ideg(k),imin(k),isec(k))
106 continue
c
    write(6,940) nor, if, isf
940 format(" state = ",i5,", outline = ",i5,", sub = ",i5)
    write(6,950)
950 format(14x,"degrees minutes seconds")
    write(6,960) ideg(1),imin(1),isec(1)
960 format(" north maximum  ",i3,i8,i8)
    write(6,970) ideg(2),imin(2),isec(2)
970 format(" south          ",i3,i8,i8)
    write(6,980) ideg(3),imin(3),isec(3)
980 format(" west           ",i3,i8,i8)
    write(6,990) ideg(4),imin(4),isec(4)
990 format(" east           ",i3,i8,i8)
c this organization is as in the comx files:
c upper latitude, lower latitude, lft longitude, right
c longitude.
c
930 format(4d20.9)
c
go to 10
c
1000 continue
return
```

EXEC_COM NAME: OPEN_SI.EC

Author: Harold Johnson
Purpose of the program: **open_si.ec** attaches and opens a file for stream input.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("open_si","60",file)
Arguments:
60—Unit number
file—Name *strdNM*
Subroutines called: **io_attach**, **io_open**
Common data referenced: None
Input files: *strdNM*
Output files: None
Arrays used: None
Called by: **state_optima**
Error checking and reporting: None
Constants: None
Program logic:
1. Call **io** to attach *file60* via *vfile_* to *strdNM.*
2. Call **io** to open *file60* for stream input.
3. Return control to the calling module.

```
&command_line off
io attach file&1 vfile_ &2 -append -ssf
io open file&1 si
```

EXEC_COM NAME: CLOSE.EC

Author: Harold Johnson
Purpose of the program: **close.ec** detaches and closes the file.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("close","60")
Arguments: *60*—Unit number
Subroutines called: **io_detach**, **io_close**
Common data referenced: None
Input files: ***strdNM***
Output files: None
Arrays used: None
Called by: **state_optima**
Error checking and reporting: None
Constants: None
Program logic:
1. Call **io** to detach ***file60***.
2. Call **io** to close ***file60***.
3. Return control to calling module.

```
&command_line off
io close file&1
io detach file&1
```

PROGRAM NAME: ADDRAD

Author: Harold Johnson
Purpose of the program: **addrad** inserts the correct values for the areas, the latitude and longitude coordinates for the centers and for the north, south, east, and west boundaries of the outlines for each map reference contained in ***strgNM*** files.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call addrad
Arguments: None
Subroutines called: **ibound_addrad**, **srch20_addrad**, **srch30_addrad**, **srch40_addrad**, **closer**, **ftnumber**, **center_addrad**, **optima_addrad**
Common data referenced: None
Input files:
  strgNM used on unit 10 (***file10***)
    Created by: **concat**
  measNM used on unit 20 (***file20***)
    Created by: **master**
  cordNM used on unit 60 (***file60***)
Output files:
  comxNM used on unit 30 (***file30***)
    Created by: **optima_addrad**
  ctrdNM used on unit 40 (***file40***)
    Created by: **center_addrad**
  redyNM used on unit 50 (***file50***)
    Created by: **addrad**
Arrays used: ***iarea(8)***, ***iaunit(7)***, ***inlat(12)***, ***islat(12)***, ***iwlong(12)***, ***ielong(12)***, ***iclat(12)***, ***iclong(12)***, ***ifile(1211)***
Called by: None
Error checking and reporting: In subroutines
Constants: None
Program logic:
1. The user is prompted for the 2-digit FIPS code of the State being processed.
2. Using this **assoc** will attach ***strgNM*** to Fortran ***file10***, ***measNM*** to ***file20***, ***comxNM*** to ***file30***, ***ctrdNM*** to ***file40***, ***cordNM*** to ***file60***, and ***redyNM*** to ***file50***. ***comxNM*** is the north, south, east, and west latitude and longitude file; ***ctrdNM*** is the latitude and longitude file of center points; ***redyNM*** is the output file ready for final input to the GRASP system.
3. **Center_addrad** is called to compute a center point for each outline in ***cordNM*** and store these in ***ctrdNM***.
4. **Optima_addrad** is called to determine the extreme north and south latitudes and east and west longitudes for each outline in ***cordNM***.
5. A record from ***strgNM*** is read.
6. Subroutine **ibound_addrad** is called to determine the ***if*** and ***isf*** for the map outline of this reference data.
7. **srch20_addrad** is called to search ***file20*** for the area of the corresponding outline. This is inserted in the record from ***strgNM***.
8. **srch30_addrad** and **srch40_addrad** are called to locate data in ***comxNM*** and ***ctrdNM*** belonging to this outline. This is inserted in the same record.
9. The record is written to ***redyNM***. Control returns to step 2.

```
c ******* ADDRAD *******
   character file*4, mode*4, state*2, outfile*6
c       PROGRAM ADDRAD
c       UPDATED AS OF DEC. 17, 1976  H. JOHNSON
c       THIS PROGRAM OPERATES ON THE FILES STRGNM CONTAINING THE OUTPUT
c       VECTORS OF THE CONCAT PROGRAM, BEFORE THEY ARE USED AS INPUT TO
c       THE CREATE PROGRAM.
c
c $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c
c
c       WHAT THIS PROGRAM DOES IS TO INSERT CORRECT AREAS, LATITUDE AND
c       LONGITUDE COORDINATES FOR THE CENTERS ,N,S,LATITUDES AND E,W,
c       LONGITUDES OF THE BOUNDARIES.
c       (IN REFNM ONLY ONE AREA,CENTER AND BOUNDARY IS GIVEN PER "IF")
c
c       THE FOLLOWING FILES ARE REQUIRED FOR INPUT:
c       FILE 10 = INPUT STRGNM WHICH WAS PRODUCED BY CONCAT.
c       FILE 20 = AREA FILE PRODUCED BY "MASTER" PROGRAM, CALLED MEASNM.
c       FILE 30 = N,S LATITUDE ;E,W LONGITUDE FILE CALLED COMXNM.
c       FILE 40 = CENTER LONGITUDE-LONGITUDE FILE CTRDNM.
c       THE OUTPUT FILE PRODUCED BY THIS PROGRAM FOR CREATE:
c       FILE 50 = THE OUTPUT FILE REDYNM.
c
c
c       THE FOLLOWING VALUES ARE THE STARTING POSITIONS FOR THE VALUES OF
c       IBOUND, AREA, AUNIT, NLAT,SLAT,WLONG,ELONG,CLAT, CLONG AS
c       DETERMINED BY THE FILE CALLED "MATRIX".
c
c $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
           dimension iarea(8),iaunit(7),inlat(12),islat(12),iwlong(12)
           dimension ielong(12)
           dimension iclat(12),iclong(12),ifile(1211)
          data nbound/1160/,narea/869/,naunit/877/,nnlat/884/,nslat/896/
          data nwlong/908/,nelong/920/,nclat/932/,nclong/944/,nrec/1211/
c
c       NOTICE THE DIMENSION OF IFILE IS THE SAME AS NREC, WHICH MUST BE
c       CHANGED IF NREC IS CHANGED.
c
          data iaunit/"s","q",".","  ","k","m","."/
c
write(6,910)
910 format(" ENTER THE 2-DIGIT CODE FOR THE STATE BEING PROCESSED")
read(5,920) state
920 format(a2)
encode(outfile,925)state
925 format("strg",a2)
mode = "si  "
   call ftnumber(10,outfile,mode)
encode(outfile,926)state
926 format("cord",a2)
    call ftnumber(60,outfile,mode)
encode(outfile,927)state
927 format("meas",a2)
   call ftnumber(20,outfile,mode)
```

```
encode(outfile,930)state
930 format("comx",a2)
mode = "sio "
   call ec ("open_sio","30",outfile)
encode(outfile,932)state
932 format("ctrd",a2)
    call ftnumber(40,outfile,mode)
encode(outfile,937)state
937 format("redy",a2)
mode = "so  "
   call ftnumber(50,outfile,mode)
c
   call center_addrad
c this routine computes a center point in radians for each
c outline.
c
    call optima_addrad
c this routine computes the extreme north and south latitude for
c each outline, and the extreme east and west longitude.
c
10          read(10,940,end=1000)(ifile(j),j=1,nrec)
940         format(80a1)
c
c      WE HAVE READ ONE RECORD OF LENGTH NREC FROM STRGNM.
            call ibound_addrad(ifile,nbound,if,isf,nrec)
c      THIS ROUTINE "IBOUND" READS IFILE FROM NBOUND TO NBOUND+6 TO
c      DETERMINE THE "IF" AND "ISF" OF THE MAP OUTLINE WHICH IS THE OUTLI
c      E
c      MAP FOR THIS REFERENCE, IF ONE EXISTS.  IF = 0 WHEN NONE EXISTS.
c
            if(if .gt. 0) go to 20
            go to 100
c
20          call srch20_addrad(if,isf,iarea,iflag20)
            if(iflag20 .eq. 1) go to 50
c      THIS ROUTINE SEARCHES FILE 20 TO LOCATE THE AREA OF THE OUTLINE
c      I
c      HAVING THIS IF AND ISF.  IAREA IS THE LEFT-JUSTIFIED, DECODED AREA
c      FORMAT 8A1 .
c
            do 30 k=1,8
30          ifile(narea+k-1)=iarea(k)
c
c      NEXT, INSERT "SQ.KM." IN IFILE.
            do 40 k=1,7
40          ifile(naunit+k-1)=iaunit(k)
c
50          call srch30_addrad(if,isf,inlat,islat,iwlong,ielong,iflag30)
            if(iflag30 .eq. 1) go to 70
c      THIS SEARCHES THROUGH FILE 30 FOR THE LATITUDE-LONGITUDES, DECODES
c      THEM INTO A1 FORMAT.
c
            do 60 k=1,12
            ifile(nnlat+k-1)=inlat(k)
            ifile(nslat+k-1)=islat(k)
```

```
          ifile(nwlong+k-1)=iwlong(k)
60        ifile(nelong+k-1)=ielong(k)
c
70        call srch40_addrad(if,isf,iclat,iclong,iflag40)
          if(iflag40 .eq. 1) go to 100
c     THIS ROUTINE SEARCHES FILE 40 FOR THE LATITUDE-LONGITUDE OF THE
c     CENTER.
c
          do 80 k=1,12
          ifile(nclat+k-1)=iclat(k)
80        ifile(nclong+k-1)=iclong(k)
c
100       write(50,940)(ifile(j),j=1,nrec)
          go to 10
c
 1000  call closer(10)
   call closer(20)
   call closer(30)
   call closer(40)
   endfile 50
c this is supposed to put a final blank record which might
c otherwise be lost.
c
   call closer(50)
   call closer(60)
c
stop
          end
```

SUBROUTINE NAME: OPTIMA_ADDRAD

Author: Harold Johnson
Purpose of the program: **optima_addrad** reads through the radian coordinate files and determines the uppermost, lowermost, leftmost, and rightmost coordinates for each map boundary.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call optima_addrad
Arguments: None
Subroutines called: None
Common data referenced: None
Input files: **cordNM** used on unit 60 (*file60*)
Output files: *file30*– a file of maximums and minimums
Arrays used: *x(3), y(3), x1(4), y1(4) ix1(4), iy1(4)*
Called by: **addrad**
Error checking and reporting: None
Constants: None
Program logic:
1. The header record of the coordinate file is read.
2. Set ***ncards*** equal to *isfno* divided by 3.
3. Read the first record.
4. Ignore the first point, because it is the location of the written *if* numbers.
5. Compare the points to find the rightmost, leftmost, uppermost, and lowermost points.
6. Put the coordinates into integer format for packed writing.
7. Write the upper latitude, lower latitude, left longitude, and right longitude to *file30.*
8. Continue steps 1 through 7 until the end of file is reached.

```
subroutine optima_addrad
c
c purpose: to read through the radian coordinate files and
c    determine the upper, lower, left, and right most
c    coordinates for each map boundary.
c
c
```

```
c input file number 60 is the standrd coordinate file.
c outpue file number 30 is a file of max, mins.
c
double precision x,y,x1,y1
dimension x(3), y(3), x1(4), y1(4), ix1(4), iy1(4)
rewind 60
c
rewind 30
10 read(60,910,end=1000) if, ifno, isf, isfno,not,nor,nif
910 format(7i5)
c
c
ncards = isfno/3
if(3*ncards .lt. isfno) ncards = ncards + 1
c
    read(60,920)(x(j),y(j),j=1,3)
920 format(6f12.9)
c
do 15 j = 1, 4
x1(j) = x(2)
15 y1(j) = y(2)
c we don't start with the first point, because it is
c the location of the written if numbers.
j0 = 2
17 do 80 j = j0, 3
if(x(j) .eq. 0. .or. y(j) .eq. 0.) go to 90
c
c x1(1), y1(1) is the right-most point.
if(x1(1) .ge. x(j)) go to 20
x1(1) = x(j)
y1(1) = y(j)
20 continue
c
c x1(2), y1(2) is the left-most point.
if(x1(2) .le. x(j)) go to 30
x1(2) = x(j)
y1(2) = y(j)
30 continue
c
c x1(3), y1(3) is the top-most point.
if(y1(3) .ge. y(j)) go to 40
y1(3) = y(j)
x1(3) = x(j)
40 continue
c
c x1(4), y1(4) is the lowest point.
if(y1(4) .le. y(j)) go to 80
y1(4) = y(j)
x1(4) = x(j)
80 continue
c
ncards = ncards - 1
if(ncards .lt. 1) go to 90
read(60,920)(x(j),y(j),j=1,3)
j0 = 1
```

```
go to 17
c
90 isfno = 2
    write(30,910) if,ifno,isf,isfno,not,nor,nif
c
c put the coordinates into integer format for packed
c writing.
c
do 200 j = 1,4
   ix1(j) = idint(x1(j)*(10.0**9) + .5)
200 iy1(j) = idint (y1(j)*(10.0**9) + .5)
c
    write(30,930) y1(3),y1(4),x1(1),x1(2)
c this organization is as in the comx files:
c upper latitude, lower latitude, lft longitude, right
c longitude.
c
930 format(4d20.9)
c
go to 10
c
1000 endfile 30
rewind 30
return
end
```

SUBROUTINE NAME: IBOUND_ADDRAD

Author: Harold Johnson

Purpose of the program: **ibound_addrad** reads the characters in the vector *ifile* to interpret the associated *if* and *isf* as integers.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call ibound_addrad (ifile,nbound,if,-isf,nrec)

Arguments:

ifile–A vector of characters from one record of *strgNM*

nbound–The number of the element of *ifile* that preceeds the data containing *if* and *isf* characters

if–The identification number for the current reference outline

isf–The subidentification number for the current reference outline

nrec–The length of *ifile*

Subroutines called: **decoder_addrad**

Common data referenced: None

Input files: None

Output files: None

Arrays used: *ifile*(*nrec*), *num*(6)

Called by: **addrad**

Error checking and reporting: None

Constants: None

Program logic:

1. The pertinent characters are read from *ifile*, and **decoder_addrad** is called to interpret them as integers. These are summed with suitable powers of 10 to output the corresponding *if* and *isf*.

```
c ******* SUBROUTINE IBOUND_ADDRAD *******
          subroutine ibound_addrad(ifile,nbound,if,isf,nrec)
c      SUBROUTINE USED IN MAIN PROGRAM "ADDRAD"
c      UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c converted to Multics May 21, 1977 by H Johnson
c
```

```
          dimension ifile(nrec),num(6)
          data iblank/" "/
c
c      THE PURPOSE OF THIS ROUTINE IS TO READ CHARACTERS IN IFILE
c      TO DETERMINE THE IF, ISF OF THE BOUNDARY FOR THIS MAP
c      THE CHARACTERS CONSIST OF 1-3 NUMERALS FOR "IF" FOLLOWED BY 2
c      FOR "ISF".
c
c      WE FIRST READ THESE NUMBERS INTO A VECTOR NUM(K), STOPPING WHEN
c      A BLANK IS ENCOUNTERED.
          do 10 k=1,6
          iblah=ifile(nbound+k-1)
          if(ifile(nbound+k-1) .eq. iblank)go to 20
          call decoder_addrad(ifile(nbound+k-1),num1)
10        num(k)=num1
c
20        if(k .gt. 1)go to 25
c      WHEN THE RECORD IS ENTIRELY BLANK WE RETURN 0 FOR IF AND ISF.
          if=0
          isf=0
          return
c
c      READ THE LAST TWO NUMBERS INTO ISF:
25        klast=k-1
          isf=0
          do 30 j=1,2
          isf=isf+num(klast-j+1)*10**(j-1)
30        continue
c
c      READ THE FIRST NUMBERS INTO IF:
          klast=klast-2
          if=0
          do 50 j=1,klast
50        if=if+num(klast-j+1)*10**(j-1)
          return
          end
c ******* END IBOUND_ADDRAD *******
```

SUBROUTINE NAME: DECODER_ADDRAD

Author: Harold Johnson
Purpose of the program: **decoder_addrad** is used to interpret integer characters and output them as integers.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call decoder_addrad (iblah,iy)
Arguments:
iblah-Integer character
iy-Integer number
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: ***ino(10)***
Called by: **ibound_addrad**
Error checking and reporting: When ***iblah*** is not the character for any single integer, a message to that effect is written to the user.
Constants: None
Program logic:
1. ***iblah*** is compared with each of the integer characters 0 through 9 until a match is found.
2. If no match is found, a message is sent to the user along with the character in question.
3. If a match is found on the *n*th test, ***iy*** is equated to $n - 1$.

```
subroutine decoder_addrad(iblah,iy)
c
c subroutine used in addrad
c written by H Johnson August 22, 1977
c
dimension ino(10)
data ino/"0","1","2","3","4","5","6","7","8","9"/
c
do 10 n = 1,10
if(iblah .eq. ino(n)) go to 20
10 continue
write(6,910)iblah
910 format(" there is no way to decode iblah = ",a1)
return
c
20 iy = n-1
return
end
```

SUBROUTINE NAME: SRCH20_ADDRAD

Author: Harold Johnson

Purpose of the program: **srch20_addrad** searches through the file number 20, *measNM*, to find that record with an assigned *if* and *isf*. It returns the area written there in character form.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call srch20_addrad (if,isf,iarea,iflag20)

Arguments:

if–The reference number for an outline

isf–The subreference number for an outline

iarea–An 8-character vector giving the area for an outline

iflag20–A flag that indicates by the value 1 that no area was found in *measNM* for this outline

Subroutines called: **icoder_addrad**

Common data referenced: None

Input files: *measNM* used on unit 20 (*file20*)

Output files: None

Arrays used: *iarea*(*8*)

Called by: **addrad**

Error checking and reporting: When no record in *measNM* can be located that has assigned *if* and *isf*, a message to that effect is sent to the operator.

Constants: None

Program logic:

1. A record from *measNM* is read, giving its value for *if*, *isf*, *area*.
2. When this record indicates that it is beyond the record we seek, *file20* is rewound and control begins at step 1.
3. Reading begins again, and this time if a record is read that should be beyond the assigned *if* and *isf*, the error message is written and the subroutine returns.
4. When a match is found, the area value is decoded into character format, along with the decimal point, and left-justified into *iarea*. *iflag20* is set at 0.

```
c ******* SUBROUTINE SRCH20_ADDRAD *******
         subroutine srch20_addrad(if,isf,iarea,iflag20)
c
c     SUBROUTINE USED IN MAIN PROGRAM "ADDRAD"
c     UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c
c converted to Multics May 21, 1977 H Johnson
c
c     THIS ROUTINE SEARCHES THROUGH THE AREA FILE 20 TO LOCATE AN IF, IS
c
c     TO MATCH THE NUMBERS GIVEN IN THE CALL.
c
```

```
          dimension iarea(8)
          data idot/"."/,izero/"0"/,iblank/" "/
      iflag20 = 0
10        read(20,900,end=45)if1,isf1,area
900       format(2i5,f8.1)
          if(if1 .eq. if .and. isf1 .eq. isf) go to 100
          if(if1 .gt. if .or. (if1 .eq. if .and. isf1 .gt. isf))rewind 20
          backspace 20
40        read(20,900,end=45)if1,isf1,area
          if(if .eq. if1 .and. isf.eq.isf1)go to 100
          if(if1 .lt. if .or. (if1 .eq. if .and. isf1 .lt. isf))go to 40
45        write(6,910)if,isf
910       format(" THERE IS NO AREA WITH IF = ",i5," AND ISF = ",i5)
          rewind 20
          iflag20 = 1
          return
100       area=area+.1
          do 105 k=0,6
          f=area/10.**k
          ifx=ifix((f-float(ifix(f)))*10.)
           call icoder_addrad(iblah,ifx)
105       iarea(7-k)=iblah
          iarea(8)=iarea(7)
          iarea(7)=idot
c
c     NOW, LEFT-JUSTIFY
          do 135 k=1,6
          if(iarea(1) .ne. izero)go to 140
          do 130 l=1,7
130       iarea(l)=iarea(l+1)
          iarea(8)=iblank
135       continue
140       continue
          iflag20 = 0
          return
          end
c ******* END SRCH20_ADDRAD *******
```

SUBROUTINE NAME: ICODER_ADDRAD

Author: Harold Johnson
Purpose of the program: **icoder_addrad** determines what character corresponds to a given input integer.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call icoder_addrad (iblah,iy)
Arguments:
iblah–The output character that symbolizes the integer *iy*
iy–The input integer
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: *ino(10)*
Called by: **ibound_addrad**, **rconv_addrad**, **srch20_addrad**
Error checking and reporting: When *iy* is not an integer between 0 and 9, an error message is sent to the operator.
Constants: None
Program logic:
1. In a do loop, *iy* is compared with each integer from 0 to 9. When a match is made, the correct character is placed in *iblah*.

```
subroutine icoder_addrad(iblah,iy)
c
c subroutine used in addrad
c written by h. johnson May 21, 1977
c
dimension ino(10)
data ino/"0","1","2","3","4","5","6","7","8","9"/
c
do 10 n = 1, 10
if(iy .eq. n-1) go to 20
10 continue
c
write(6,910)iy
910 format(" THERE IS NO WAY TO ENCODE IY = ",i3)
return
c
20 iblah = ino(n)
return
end
```

SUBROUTINE NAME: SRCH30_ADDRAD

Author: Harold Johnson
Purpose of the program: **srch30_addrad** searches through *file30* for a record that matches an assigned *if* and *isf*. It returns values in character format.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call srch30_addrad (if,isf,inlat,islat,-iwlong,ielong,iflag30)
Arguments:
if – Reference identification number
isf – Subidentification number
inlat – Vector containing the characters of the latitude of the most northerly point of the outline
islat – Vector containing the characters of the latitude of the most southerly point of outline
iwlong – Vector containing the characters of the longitude of the most westerly point of the outline
ielong – Vector containing the characters of the longitude of the most easterly point of the outline
iflag30 – Flag indicating whether or not the search was successful
Subroutines called: **rconv_addrad, read2_addrad**
Common data referenced: None
Input files: ***comxNM*** used on unit 30 (*file30*)
Output files: None
Arrays used: *inlat*(12), *islat*(12), *iwlong*(12), *ielong*(12)
Called by: **addrad**
Error checking and reporting: When no match is found, this is reported to the operator.
Constants: None
Program logic:

1. A record in *file30* is read to determine an *if1* and *isf1*.
2. If a match with the assigned *if* and *isf* is found, **rconv_addrad** is called to convert the data into degrees, minutes, and seconds in character form. Then control returns to the calling program.
3. If the read record in *file30* seems to be farther along than the assigned *if, isf* indicated, *file30* is rewound and step 1 is initiated, with a flag indicating that this has occurred. If that flag was already set, the error message is written to the operator that no record exists in *file30* with the *if, isf,* and the routine ends.
4. A second record is read and compared with the assigned *if, isf,* and this is repeated until the expected position of that record is passed. Then the error message is written.
5. When no record is found, *iflag30* is set to 1. Otherwise, it is set to 0.

```
c ******* SUBROUTINE SRCH30_ADDRAD *******
          subroutine srch30_addrad(if,isf,inlat,islat,iwlong,ielong,if
          lag30)
c      SUBROUTINE USED IN THE MAIN PROGRAM "ADDRAD"
c      UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c
```

```
c converted to Multics May 21, 1977 H Johnson
c
c      THIS ROUTINE SEARCHES THE LATITUDE LONGITUDE FILE 30 FOR THE RECOR
c
c      HAVING IF AND ISF THE SAME AS THE ONES SUPPLIES BY THE CALLING
c      PROGRAM.  WHEN IT FINDS THEM IT READS THE NLAT,SLAT,WLONG,ELONG
c      VALUES ON THE NEXT CARD, ENCODES THEM TO INLAT,ISLAT,IWLONG,IELONG
c      AND RETURNS.
c
          dimension inlat(12),islat(12),iwlong(12),ielong(12)
          double precision ulat,slat,wlong,elong
c
c      FIRST, SEARCH FOR THE ASSIGNED IF AND ISF.
10        call read2_addrad(if1,isf1,ulat,slat,wlong,elong,kflag)
          if(kflag .eq. 1) go to 45
c      THIS READS 2 CARDS, A HEADER CARD FOLLOWED BY COORDINATES.  IT
c      RETURNS THE IF AND ISF FROM THE HEADER CARD.
c
          if(if1 .eq. if .and. isf1 .eq. isf)go to 100
          if(if1 .gt. if .or. (if1 .eq. if .and. isf1 .gt. isf))rewind 30
40        call read2_addrad(if1,isf1,ulat,slat,wlong,elong,kflag)
          if(kflag .eq. 1) go to 45
          if(if .eq. if1 .and. isf .eq. isf1)go to 100
          if(if1 .lt. if .or. (if1 .eq. if .and. isf1 .lt. isf))go to 40
45        write(6,910)if,isf
910       format(" THERE IS NO COMX RECORD WITH IF = ",i5," ISF = ",i5)
          rewind 30
          iflag30 = 1
          return
100       continue
c      AT THIS POINT THE NUMBERS ULAT,SLAT,WLONG,ELONG MUST BE CONVERTED
c      0
c      DEGREES, MINUTES AND SECONDS AND ENCODED.
          call rconv_addrad(ulat,inlat)
          call rconv_addrad(slat,islat)
          call rconv_addrad(wlong,iwlong)
          call rconv_addrad(elong,ielong)
          iflag30 = 0
          return
          end
c ******* END SRCH30_ADDRAD *******
```

SUBROUTINE NAME: RCONV_ADDRAD

Author: Harold Johnson
Purpose of the program: **rconv_addrad** decodes a 12-digit floating point number into characters and writes them to a vector, one character put in each element of the vector.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call rconv_addrad (radian,ifile)
Arguments:
- ***radian***– A floating point number to be decoded
- ***ifile***– A 12-element vector, which is to contain the decoded digits of ***radian*** as characters

Subroutines called: **dms**, **icoder_addrad**
Common data referenced: None
Input files: None
Output files: None

Arrays used: *ifile*(*12*)
Called by: **srch30_addrad**, **srch40_addrad**
Error checking and reporting: In subroutines
Constants: None
Program logic:
1. *radian* is converted to degrees.
2. **dms** is called to convert these degrees to degrees, minutes, and seconds, I format.
3. These integers are written into *ifile* one at a time, with blanks in place of leading zeros. **icoder_addrad** is used to convert single integers to characters.

```
c ******* SUBROUTINE RCONV_ADDRAD *******
          subroutine rconv_addrad(radian,ifile)
c     SUBROUTINE USED IN MAIN PROGRAM "ADDRAD"
c     UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c converted to Multics May 21, 1977  h. johnson
c
          double precision radian,x,pi
          dimension ifile(12)
          data pi/3.14159265358979323842643383279/
          data iblank/" "/
          data izero/"0"/
c
c     THIS ROUTINE CONVERTS THE RADIAN ANGLE "RADIAN" INTO DEGREES,
c     MINUTES AND SECONDS, PACKING THEM INTO THE ARRAY IFILE.
c
c
          x=dabs(radian*180./pi)
   call dms(x,ideg,imin,isec)
991 format(1x,3i7)
do 10 j = 1,3
iy = ideg/(10**(3-j))
  call icoder_addrad(iblah,iy)
ifile(j) = iblah
iy = iy*(10**(3-j))
10 ideg = ideg - iy
if(ifile(1) .ne. izero) go to 15
ifile(1) = iblank
if(ifile(2) .ne. izero) go to 15
ifile(2) = iblank
if(ifile(3) .ne. izero) go to 15
ifile(3) = iblank
15 continue
c
do 20 j = 1, 2
iy = imin/(10**(2-j))
   call icoder_addrad(iblah,iy)
ifile(3+j) = iblah
iy = iy*(10**(2-j))
20 imin = imin -iy
if(ifile(4) .ne. izero) go to 25
if(ifile(5) .ne. izero) go to 25
25 continue
c
do 30 j = 1, 3
iy = isec/(10**(3-j))
   call icoder_addrad(iblah,iy)
ifile(5+j) = iblah
```

```
iy = iy*(10**(3-j))
30 isec = isec - iy
if(ifile(6) .ne. izero) go to 35
if(ifile(7) .ne. izero) go to 35
if(ifile(8) .ne. izero) go to 35
35 continue
c
do 40 j = 9,12
40 ifile(j) = iblank
995 format(1x,12a1)
return
end
c ******* END RCONV_ADDRAD *******
```

SUBROUTINE NAME: DMS

Author: Harold Johnson

Purpose of the program: **dms** converts a double-precision degree number to three integers representing degrees (between 0 and 360), minutes, and seconds.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call dms (x,ideg,imin,isec)

Arguments:

x– A double-precision floating point value for degrees
ideg– An integer representing the degrees in *x*
imin– An integer representing the minutes in *x*
isec– An integer representing the seconds in *x*

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **rconv_addrad, optima**

Error checking and reporting: None

Program logic:

1. *x* is reduced modulo 360.
2. The integer part of *x* is taken for *ideg*.
3. *ideg* is subtracted from *x*, the result multiplied by 60, and its integer part is *imin*.
4. *imin* is subtracted, the result is multiplied by 60, and its integer part becomes *isec*.

```
c ******* DMS FUNCTION *******
subroutine dms(x,ideg,imin,isec)
double precision x,y
y = 360.
x = dmod(x,y)
ideg = ifix(sngl(x))
x = x - dfloat(ideg)
imin = ifix(sngl(60.*x))
x = x*60. -dfloat(imin)
isec = ifix(sngl(60.*x))
return
end
c ******* END DMS FUNCTION *******
```

SUBROUTINE NAME: READ2_ADDRAD

Author: Harold Johnson

Purpose of the program: **read2_addrad** is used to read from the file *comxNM* the values for the latitudes and longitudes and the values for *if* and *isf*.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call read2_addrad (if1,isf1,ulat,slat,-wlong,elong,kflag)

Arguments:

if1—The *if* of the record from *comxNM*
isf1—The *isf* of the record
ulat—The northernmost latitude
slat—The southernmost latitude
wlong—The westernmost longitude

elong—The easternmost longitude
kflag—A flag to indicate that the end of ***comxNM*** has been sensed
Subroutines called: None
Common data referenced: None
Input files: **comxNM** used on unit 30 (***file30***)
Output files: None
Arrays used: None
Called by: **srch30_addrad**
Error checking and reporting: None
Constants: None
Program logic:
1. ***file30*** is read according to its preassigned format. When the end of the file is sensed, ***kflag*** is set to 1.

```
c ******* SUBROUTINE READ2_ADDRAD *******
          subroutine read2_addrad(if1,isf1,ulat,slat,wlong,elong,kflag)
c      SUBROUTINE CALLED BY SRCH30 IN MAIN PROGRAM "ADDRAD"
c      UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c converted to multics May 21, 1977 H Johnson
c
          double precision ulat,slat,wlong,elong
          read(30,900,end=100)if1,ifno,isf1
900       format(3i5)
          read(30,910,end=100)ulat,slat,wlong,elong
910       format(4d20.9)
          kflag = 0
          return
100        kflag = 1
          return
          end
c ******* END READ2_ADDRAD *******
```

SUBROUTINE NAME: SRCH40_ADDRAD

Author: Harold Johnson
Purpose of the program: **srch40_addrad** searches through ***file40*** for a record having an assigned ***if*** and ***isf***.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call srch40_addrad (if,isf,iclat,iclong,-jflag)
Arguments:
if—The assigned reference identification number
isf—The assigned reference subidentification number
iclat—A 12-character vector containing characters that represent the latitude of the center point in degrees, minutes, and seconds
iclong—The 12-character vector of characters for the longitude of the center
jflag—A flag that, when set to 1, indicates that no record in ***file40*** was found to correspond to the given ***if***, ***isf***
Subroutines called: **read40_addrad**, **dble**, **rconv_addrad**
Common data referenced: None
Input files: **ctrdNM** used on unit 40 (***file40***)
Output files: None
Arrays used: ***iclat***(*12*), ***iclong***(*12*)
Called by: **addrad**
Error checking and reporting: When no record is found with the prescribed ***if*** and ***isf***, a message is sent to the operator.
Constants: None
Program logic:
1. Following the same logic stream as in **srch20_addrad**, search the file for the record with the desired ***if*** and ***isf*** values.
2. When found, the latitude and longitudes are made double precision, and **rconv_addrad** is called to convert them to vectors of characters.

```
c ******* SUBROUTINE SRCH40_ADDRAD *******
          subroutine srch40_addrad(if,isf,iclat,iclong,jflag)
c      SUBROUTINE USED IN MAIN PROGRAM "ADDRAD"
c      UPDATED AS OF DEC. 27, 1976
c      H. JOHNSON
c
```

```
c
c      THIS ROUTINE SEARCHES FILE 40 FOR THE RECORD HAVING THE GIVEN IF,I
c      F
c      THE NEXT RECORD WILL CONTAIN THE LATITUDE, LONGITUDE OF THE CENTER
c      POINT
c
          dimension iclat(12),iclong(12)
          double precision radian
c
c
jflag = 0
c      FIRST, SEARCH FOR THE IF, ISF IN THE CALLING PROGRAM.
10        call read40_addrad(if1,isf1,clat,clong,iflag1)
if(iflag1 .eq. 1) go to 45
c      THIS READ 2 CARDS.  THE FIRST IS A HEADER CARD AND GIVES THE IF1,I
c      F1
c      THE SECOND IS A LATITUDE LONGITUDE CARD.
          if(if1 .eq. if .and. isf1 .eq. isf)go to 100
          if(if1 .gt. if .or. (if1 .eq. if .and. isf1 .gt. isf))rewind 40
40        call read40_addrad(if1,isf1,clat,clong,iflag1)
if(iflag1 .eq. 1) go to 45
          if(if .eq. if1 .and. isf.eq.isf1)go to 100
          if(if1 .lt. if .or. (if1 .eq. if .and. isf1 .lt. isf))go to 40
45        write(6,910)if,isf
910       format(" THERE IS NO AREA WITH IF = ",i5," AND ISF = ",i5)
          rewind 40
jflag = 1
return
c
100       radian = dble(clat)
          call rconv_addrad(radian,iclat)
          radian=dble(clong)
          call rconv_addrad(radian,iclong)
c      RCONV CONVERTS "RADIAN" TO DEGREES, MINUTES AND SECONDS AND ENCODE
c
c      THEM AS CHARACTERS INTO ICLAT AND ICLONG.
          return
          end
c ******* END SRCH40_ADDRAD *******
```

SUBROUTINE NAME: READ40_ADDRAD

Author: Harold Johnson
Purpose of the program: **read40_addrad** is used to read from ***file40***, **ctrdNM**.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call read40_addrad (if1,isf1,clat,-clong,iflag1)
Arguments:
if1–The ***if*** number of this outline
isf1–The ***isf*** number of this outline
clat–The latitude of the center point of this outline
clong–The longitude of the center point
iflag1–A flag to indicate the end of ***file40*** has been sensed
Subroutines called: None
Common data referenced: None
Input files: **ctrdNM** used on unit 40 (***file40***)
Output files: None
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. ***file40*** is read, two cards at a time, to determine the arguments. ***iflag1*** is set to 1 when the end of file is sensed.

```
c ******* READ40_ADDRAD *******
          subroutine read40_addrad(if1,isf1,clat,clong,iflag1)
c     SUBROUTINE CALLED BY SRCH40 IN MAIN PROGRAM "ADDRAD"
c     UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c
c converted to multics  May 20, 1977 by H Johnson
c
          read(40,910,end=100)if1,ifno,isf1
910       format(3i5)
          read(40,920,end=100)x0,y0,clong,clat
920       format(4f12.9)
          iflag = 0
          return
100        iflag = 1
           return
          end
c ******* END READ40_ADDRAD *******
```

SUBROUTINE NAME: WEIGHT_ADDRAD

Author: Harold Johnson
Purpose of the program: **weight_addrad** calculates a weighted center for each edge of a polygon and adds to the cumulated weighted center and total length.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call weight_addrad (x1,y1,x2,y2,xc,-yc,td)
Arguments:
x1 – Longitude in radians of the first point
y1 – Latitude in radians of the first point
x2 – Longitude in radians of the second point
y2 – Latitude in radians of the second point
xc – Longitude of weighted center
yc – Latitude of weighted center
td – Total length of curve
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **center_addrad**
Error checking and reporting: None
Constants: None
Program logic:
1. The value $z = |x2 - x1| + |y2 - y1|$ is calculated.
2. The value $0.5(x2 + x1)z$ is calculated and added to *xc*. The value $0.5(y2 + y1)z$ is calculated and added to *yc*.
3. *z* is added to *td*.

```
      subroutine weight_addrad(x1,y1,x2,y2,xc,yc,td)
c
c ******* SUBROUTINE WEIGHT_ADDRAD *******
c
c Purpose: To calculate a weighted center for an added edge.
c   by adding to each coordinate the average coordinate
c  multiplied by the length of the segment.
c
c Programmer: H Johnson
c
c Date: July 18, 1978
c
c  converted to multics May 6, 1977 H. Johnson
c
implicit double precision (a-z)
c
           z=abs(x2-x1) + abs(y2-y1)
           xc=xc + 0.5*(x2+x1)*z
```

```
          yc = yc + 0.5*(y2+y1)*z
          td = td + z
          return
         end
c ******* END WEIGHT_MASTER *******
```

SUBROUTINE NAME: CENTER_ADDRAD

Author: Harold Johnson
Purpose of the program: **center_addrad** computes a central point for each outline in *cordNM* and stores it in a file named *ctrdNM.*
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call center_addrad
Arguments: None
Subroutines called: **weight_addrad**
Common data referenced: None
Input files: *cordNM* used on unit 60 (*file60*)
Output files:
ctrdNM used on unit 40 (*file40*)
Created by: **center_addrad**
Arrays used: None
Called by: **addrad**
Error checking and reporting: None
Constants: None

Program logic:
1. A header card is read from *cordNM*, and *isfno*, the number of coordinate pairs that follow, is used to compute the number of coordinate cards.
2. The first coordinate card is read. If only one point is in the outline, that is assigned the center point. Otherwise, beginning with the second and third points, a weighted center is computed according to the formulas described for **weight_addrad**.
3. The middle cards are read, and these computations are continued with each pair of coordinate points.
4. The last card is read, and the final calculations are
   $xc = xc/dt$
   $yc = yc/dt$
5. A header card is written to *ctrdNM* with *isfno* = 2.
6. A latitude-and-longitude card is written to *ctrdNM* with *xc*, *yc*.
7. Control goes to step 1 until the *cordNM* file is finished.
8. **Endfile** and **rewind** are executed on *ctrdNM.*

```
          subroutine center_addrad
c
c**********  WEIGHTED AVERAGE CENTER    ***********
c Purpose: To compute a center for each outline in the
c   radian coordinate file  cordNM  and put it in the
c   file  ctrdNM .
c
c Programmer: H Johnson
c Date: July 18, 1978
c
c input file60: cordNM, a file of radian coordinates, giving the
c       latitudes and longitudes, format 3(2f12.9))
c
double precision xx, yy, xstart, ystart, xc, yc, dt, xlast, ylast
c
          common true,sk,ia
          dimension xx(3),yy(3)
c
c
c FIRST, COMPUTE THE STATE MAP AREA IN SQUARE KILOMETRES.
c
          in=60
          iocntr=40
100       read(in,900,end=1000)  if,ifno,isf,isfno,not,nor,nif,ispan
900 format(8i5)
c
```

```
c  WHEN ISFNO IS LESS THAN 4 WE DON'T HAVE A REGION AT ALL
c
          if(isfno.gt.3)go to 102
           read(in,901,end=1000)(xx(i),yy(i),i=1,3)
901 format(6f12.9)
c
c
c IN THE CASE OF A SINGLE POINT, CALL THAT POINT CNTR.
c
           ixc=xx(2)*(10.**9) +.5
           iyc=yy(2)*(10.**9) +.5
           isfno=2
           write(iocntr,905)if,ifno,isf,isfno,not,nor,nif,ispan
           write(iocntr,908)ixc,iyc
           go to 100
102        continue
c
c CALCULATE NCARDS, THE NUMBER OF DATA CARDS ON THIS MAP
c
           ncards=isfno/3
           if(3*ncards .lt. isfno) ncards=ncards+1
c
c
           xc=0.
           yc=0.
           dt=0.
c
c  XC IS GOING TO BE THE X-COORDINATE OF THE CENTER
c  YC IS GOING TO BE THE Y/COORDINATE OF THE CENTER
c  DT IS THE ACCUMULATED NORMED DISTANCE BETWEEN POINTS
c
c  READ IN THE FIRST DATA CARD
c
           read(in,901,end=1000)(xx(i),yy(i),i=1,3)
           xstart=xx(2)
           ystart=yy(2)
           ie=3
           if(ncards .eq. 1) ie=isfno
           do 200 j=3,ie
           j1=j-1
           call weight_addrad(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
200        continue
           if(ncards .eq. 1) go to 500
           xlast=xx(3)
           ylast=yy(3)
c
c
           if(ncards .eq. 2) go to 400
c
c  READ IN THE MIDDLE CARDS, BETWEEN THE FIRST AND LAST.
c
           kl=ncards-1
           do 300 k=2,kl
           read(in,901)(xx(i),yy(i),i=1,3)
          call weight_addrad(xlast,ylast,xx(1),yy(1),xc,yc,dt)
           do 301 j=2,3
          j1=j-1
          call weight_addrad(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
301       continue
          xlast=xx(3)
          ylast=yy(3)
300       continue
400       continue
c
c  NOW READ IN THE LAST CARD
c
          read(in,901)(xx(i),yy(i),i=1,3)
          ie=isfno-3*(ncards-1)
          if(ie .eq. 0) ie=3
          call weight_addrad(xlast,ylast,xx(1),yy(1),xc,yc,dt)
          if(ie .eq. 1) go to 500
          do 401 j=2,ie
          j1=j-1
          call weight_addrad(xx(j1),yy(j1),xx(j),yy(j),xc,yc,dt)
401       continue
500        continue
c
c  WHEN THE REGION IS NOT CLOSED, WE MUST ADD THE LAST DATA POINT
c
          test=(xx(ie)-xstart)**2 + (yy(ie)-ystart)**2
          if(test.lt..01) go to 501
          call weight_addrad(xx(ie),yy(ie),xstart,ystart,xc,yc,dt)
501       continue
503       continue
          xc=xc/dt
          yc=yc/dt
         isfno=2
          write(iocntr,905)if,ifno,isf,isfno,not,nor,nif,ispan
905       format(8i5)
          ixc= idint(xc*1000000000.+0.5)
          iyc= idint(yc*1000000000.+0.5)
          write(iocntr,908)ixc,iyc
908       format(24x,2i12)
          go to 100
          1000 endfile 40
          rewind 40
          return
          end
```

SUBROUTINE NAME: FTNUMBER

Author: Harold Johnson
Purpose of the program: **ftnumber** is used to attach and open a file on the Multics system. It allows for any file *mode*.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics

Calling sequence: call ftnumber (iunit,name,mode)
Arguments:
iunit–The Fortran number of a file being attached
name–The name of the segment being attached (This can have as many as six characters.)
mode–Type of file access method
Subroutines called: The Fortran routine *encode* and the system routine I/O

Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **concat, master, addrad, out1_bigsta, out-2_ bigsta**
Error checking and reporting: None
Constants: None

Program logic:
1. The assigned Fortran number *iunit* is inserted into the character string *fname* in the form *fileNM,* where *NM* is the number *iunit*.
2. If the *mode* is not *so*, attachment is made using arguments *-append* and *-ssf*.
3. If the *mode* is *so* attachment is made.

```
c ******* SUBROUTINE FTNUMBER *******
subroutine ftnumber(iunit,name,mode)
c
c PURPOSE: To automatically attach and open files in Fortran
c
c PROGRAMMER: H Jonnson
c DATE: Sept 30, 1977
c
c iunit = fortran i/o number
c name = up to 6-character c name of a file.
c mode = "si","so", etc.
c
c modified from the program assoc of  >udd>Grasp>RBowen>assoc
c by HJohnson May 20, 1977.
c
     character name*6,fname*6,mode*4,fmt*12
fmt = "(4hfile,i2) "
if(iunit .le. 9) fmt = "(5hfile0,i1)"
encode(fname,fmt) iunit
if (mode .eq. "so") go to 20
   call io ("attach",fname,"vfile_",name,"-append","-ssf")
      call io ("open",fname,mode)
return
c
20 call io ("attach",fname,"vfile_",name)
     call io ("open",fname,mode)
return
c
end
```

EXEC-COM NAME: COVERT.EC

Author: P. A. Fulton
Purpose of the program: **covert.ec,** written in Multics command language, reads the *redyNM* file and creates a GRASP file for the State.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec covert *NM state*
Arguments:
NM– FIPs code for the State
state– Name of the State
Subroutines called: **setmas, convert**
Common data referenced: None
Input files: dicn, mask, crfile, *indxNM, redyNM,* defn
Output files: index0
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. Turn off the COMMAND LINE.
2. Attach the exec_com to **setmas**.
3. Execute **setmas**.
4. Upon completion of **setmas**, detach the files.
5. Attach the exec_com to **convert**.
6. The exec_com will pass the names of six files– dicn, mask, crfile, *indxNM, redyNM,* and defn to be read by **convert**.

7. Detach the files.
8. Attach the exec_com to **ted**, which is a text editor.
9. Read indexO and then read *file15*, which will append *file15* to indexO.
10. Write indexO.
11. Quit and exit from the text editor.
12. Detach the files.
13. Delete *file15*.
14. Quit the exec_com.

```
&command_line off
&attach
setmas
1 18
yes
&detach
&attach
convert
dicn
mask
crfile
indx&1
n
redy&1
n
defn
reference map file for &2
&detach
&attach
ted
r index0
r file15
w index0
q
&detach
dl file15
&quit
```

FILE NAME: CRFILE

Purpose of the file: crfile is a control file for the GRASP programs. It enables GRASP to read the *redyNM* files correctly.

Format: The first record contains *nacr* and *nrec*, format I3, I5. The remaining records contain *acronm*, *itype*, and *ifirst*, format A9, I1, I5.

Arguments:

nacr–The number of acronyms to follow = remaining number of records in crfile

nrec–The total length of the *strgNM* and *redyNM* records

acronm–The acronymns used in matrix

itype–The type code used by GRASP

ifirst–The position in the records of *strgNM* and *redyNM* where this type of data begins

Referenced by: The GRASP programs, which set up GRASP files.

```
          crfile


 38 1211
id        1    1
state     3    5
author    6   25
year      1  205
title     6  209
```

```
county   6  449
publish  6  629
series   6  689
emphasi  6  809
area     2  869
aunit    6  877
nlat     1  884
slat     1  896
wlong    1  908
elong    1  920
clat     1  932
clong    1  944
omaps    6  956
avail    6 1016
base     3 1076
geology  3 1106
plate    6 1118
idstat   1 1148
scale    1 1150
idsub    1 1158
ibound   1 1160
ispan    1 1166
alsomap  6 1172
dum0     1 1202
dum1     1 1203
dum2     1 1204
dum3     1 1205
dum4     1 1206
dum5     1 1207
dum6     1 1208
dum7     1 1209
dum8     1 1210
dum9     1 1211
```

PROGRAM NAME: SETMAS

Author: P. A. Fulton

Purpose of the program: **setmas** creates the index file required by GRASP. This GRASP file is limited to 10 entries. This program accepts a list of State FIPS codes and creates files containing a GRASP entry for each State.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: setmas

Arguments: None

Subroutines called: None

Common data referenced: None

Input files: *index0* used on unit 11 (*file11*)(State codes entered interactively)

Output files: *index*, a subset of *index0*, used on unit 12 (*file12*)

Arrays used: *ir(9,2)*, *ir1(80)*

Called by: **covert.ec, gr.ec, usmerg.ec**

Error checking and reporting: System only

Constants: None

Program logic:

1. This program executes interactively. It begins by requesting the State codes:

   ENTER ONE DIGIT FOR NUMBER OF STATES TO QUERY AND FOLLOW BY LIST OF TWO DIGIT STATE CODES WITH BLANKS BETWEEN

   Reply: 3 02 16 39

   This reply indicates 3 State entries are to be put into the output file *index*. These States are coded 02 (Alaska), 16 (Idaho), and 39 (Ohio).
2. All I/O operations are handled internally by the program.
3. The program compares the codes input via the terminal to the codes in file index0. When the codes match, a State entry in proper GRASP format is placed in the output file *index*. *index* is then used by GRASP.

```
setmas
 character ir*1,ir1*1
dimension ir(9,2), ir1(80)
 call io ("attach","file11","vfile_ ","index0")
 call io ("open","file11","si")
 call io ("attach","file12","vfile_ ","index")
 call io ("open","file12","sio")
print,"enter one digit for number of states to query and follow"
print,"by list of two digit state codes with  blanks between"
read (5,101) n,(ir(i,1),ir(i,2),i=1,n)
101 format (i1,x,9(2a1,x))
221 read (11,102,end=90) ir1
102 format (80a1)
do 220 i=1,n
if (ir(i,1) .ne. ir1(5))  go to 220
if (ir(i,2) .ne. ir1(6)) go to 220
write (12, 102) ir1
220 continue
 go to 221
90 continue
       end file 12
       call io ("close","file11")
       call io ("close","file12")
 call io ("detach","file11")
 call io ("detach","file12")
stop
end
```

EXEC_COM NAME: GR.EC

Author: P. A. Fulton
Purpose of the program: **gr.ec**, written in Multics command language, sorts the State index file by scale and creates three files: *t1p* for scales LE (less than) 1:24,000, *t2p* for scales GT (greater than) 1:63,360, and *t3p* for scales BE (between) 1:24,001 and 1:63,360.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec gr *NM*
Arguments: NM– FIPS code for the State
Subroutines called: **setmas**, GRASP
Common data referenced: None
Input files: index0
Output files: ***t1p***, ***t2p***, ***t3p***, ***t1***, ***t2***, ***t3***, and ***output_file***
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None

```
gr.ec
&command_line off
&attach
fo
setmas
1 31
```

Program logic:
1. Turn off the command line.
2. Attach the exec_com to the program **setmas**.
3. Designate the user output to a segment by the FO command.
4. Execute subroutine **setmas**.
5. The exec com supplies answers to queries in **setmas**.
6. The file is detached.
7. ***input_line*** is turned off.
8. Attach the exec_com to the program GRASP.
9. The exec_com contains responses to prompts in the GRASP program.
10. At the end of GRASP, the file is detached.
11. User output is directed to the console by the RO command.
12. The file called ***output_file*** is deleted.
13. The three files ***t1p***, ***t2p***, and ***t3p*** are automatically dprinted.
14. Quit the exec_com.

```
yes
&detach
& input_line off
&attach
grasp
cond
scale le 24000
scale gt 63360
scale be 24001,63360

logic
a
search

t1
list
t1
50
c
y
t1p
ibound
id
idsub

logic
b
search

t2
list
t2
50
c
y
t2p
n
logic
c
search

t3
list
t3
50
c
y
t3p
n
quit
no
&detach
co
```

```
dl output_file
dp t1p
dp t2p
dp t3p
&quit
```

EXEC_COM NAME: INPLOT.EC

Author: P. A. Fulton
Purpose of the program: **inplot.ec**, written in Multics command language, plots the three files created by **gr.ec**–that is, ***t1p***, ***t2p***, and ***t3p***. It provides a visual check of the integrity of the plot files.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec inplot *NM state*
Arguments:
NM–FIPS code for the State
state–State name
Subroutines called: **pn16**
Common data referenced: None
Input files: ***bordNM***, ***coorNM***, ***statNM***, ***counNM***, ***gridNM***

Output files: Plots on the Tektronix screen
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:

1. ***command_line*** is turned off.
2. ***page_length*** is set to 0 to disable end-of-page checking.
3. ***&input_lines*** off disables the computer from accepting input from the terminal.
4. ***&attach*** attaches the arguments in the exec_com directly to **pn16**.
5. Response to prompts in **pn16** are fulfilled within the exec_com.
6. ***&detach*** detaches the exec_com from **pn16**.
7. ***page_length*** is set to 114.
8. Quit.

```
inplot.ec
&command_line off
stty -modes pl0
&input_lines off
&attach
pn16
n
&1
y
t1p
y
scale ge 1:24000 &2
1
0
1
0
c
n
y
y
t2p
y
scale lt 1:63360 &2
1
0
1
0
c
```

```
n
y
y
t3p
y
scale between 1:24000,1:63360 &2
1
0
1
0
c
n
y
n
counties for &2
0
1
0
0
c
n
n
yes
&detach
stty -modes pl114
&quit
```

PROGRAM NAME: PN16

Author: P. A. Fulton

Purpose of the program: **pn16** plots a State index map interactively on a Tektronix CRT screen. This is a two-step process. First, a GRASP retrieval is executed wherein a disk file is created that contains the links to the coordinate Geoindex files. This GRASP file is identifed as unit 13. However, the program is constructed so that the user has the option of plotting any combination of the input files.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: pn16

Arguments: None

Subroutines called: **initt**, **anmode**, **erase**, **movabs**, **dwindo**, **swindo**, **movea**, **drawa**, **ancho**, **bell**, **hdcopy**, **vcursr**, **finitt** (Tektronix routines), **plod**, **pos**, **plo**, **ploc6**, **assoc**

Common data referenced: *x1*, *y1*, *x2*, *y2*

Input files:
Input from terminal used on unit 5 (*file5*)
coorNM used on unit 10 (*file10*)
bordNM used on unit 11 (*file11*)
GRASP file used on unit 13 (*file13*)
statNM used on unit 14 (*file14*)
counNM used on unit 15 (*file15*)
gridNM used on unit 16 (*file16*)
skod used on unit 17 (*file17*)
Superimposed file used on unit 20 (*file20*)

Output files: Output to terminal used on unit 6 (*file6*)

Arrays used: *lead*(*20*), *xx*(*6*), *yy*(*6*), *mort*(*300*), *ista*(*6*)

Called by: Can be executed by **inplot.ec**

Error checking and reporting: When the array size of 300 entries for the input plot file (*file13*) is exceeded, the program returns the error message:

BUFFER EXCEEDED

When the number of x, y coordinates does not match the number given on the header card (*isfno*), the program returns the error message:

REC COUNT ERR

All error messages concerned with I/O handling are identified by system messages and system returns.

Constants: *ingd* = 10 and *ing* = 13 are file numbers; *rsiz* = 0.3 size in inches for numbers shown on plot; *ichar* = 43 character (+) for points plots.

Program logic:
1. The program prompts the user for all the specific input and output desired. The first phase requires the user to identify the single, specific State.
2. The program does all the file handling internally to relieve the user of all systems duties. However, all

the files listed above must be available because they are attached and opened here at the beginning of the program.

3. The program requests the input file (***file13***), which is to be plotted. If necessary, the program sorts the data into ascending order by keying on the ***id*** number.
4. The plot window is computed from the coordinates of the border files, ***bordNM***.
5. The annotation and exact data files that are to be plotted are elicited from the user.
6. The actual plotting is initiated in the following order: border, State, county, grid, and the input file, which was created by GRASP (***file13***) and which is to be plotted. The program reads the value of ***ibound*** from this input file and assigns to it the variable name ***imap***.
7. The program reads through the ***coorNM*** file. From the header cards, it obtains the identification and subidentification numbers, which it combines into the variable ***if*** and compares to ***imap***.
8. When these numbers from the two different files match, the outline is plotted. First the identification number is plotted, its position determined by the first pair of x, y coordinates. Then the rest of the x, y coordinates are plotted to form the outline. The program uses the number of points (***isfno***) read from the header card to compute the end of a feature outline.
9. For very small areas, there is only one x, y coordinate pair for the outline, and the symbol (+) is plotted at that point.
10. After the entire input file (***file13***) has been plotted, the program tests to see whether another file should be superimposed. If so, it is plotted by subroutine **ploc6**. **ploc6** plots this superimposed file similar to the way **pn16** plots the ***coorNM*** file. The exception is that **ploc6** plots the entire file that is to be superimposed but does not compare it to any other input file from GRASP.
11. After the plot is completed, the bell rings to alert the user to make hard copies of the plot and (or) to continue the execution of the program.
12. The next phase of the program permits the user to enlarge any part of the plot. The new plot window is defined by use of the crosshair cursor.
13. The program then loops back so that the appropriate annotation and files are used for the enlarged part of the plot. This enlargement cycle can be continued until the user is satisfied and decides to go on to the next part of the program.
14. The next part of the program enables the user to designate another input file from GRASP or simply to loop back through the program and plot any of the base sheet files. However, to plot another State, the program must be exited and reinitiated.
15. The final phase of the program closes the files. It then writes the message

    GOOD

    to the screen to indicate that the program terminated successfully.

```
c     pn16
c     program to plot map indices on tektronix
c
c     u. s. geological survey
c
c          june 1977
c
c
c
c
c              input/output files
c     5 input from terminal
c     6 output to terminal
c     10 coordinate files
c     11 bord
c     13 file from grasp contains three items:ibound,id,idsub
c     14 stat
c     15 coun
c     16 grid
c     17 skod - file with state names, numeric and alphabetic fips codes
c     20 file to superimposed same format as coor file
```

```
c
c
      common x1,y1,x2,y2
      character skud*8,fname*8,fmt*12,ifile*8
      dimension lead(20),xx(6),yy(6),mort(300),ista(6)
      data jes/"y"/,kop/"c"/,ibnk/"        "/
      data skud/"skod    "/
c
      call initt(960)
c
c
      ingd=10
      rsiz=0.3
      ichar=43
      ing=13
c
c     assign files
      call anmode
      print,"need state codes (enter y for yes)"
      read (5,130) irep
      if (irep .ne. jes) go to 383
      fmt="(a8)"
      encode (fname,fmt) skud
      call assoc (17,fname,"si  ")
      do 381 i=1,54
      read (17,141) ista,ile,nmb
      write (6,142) ista,ile,nmb
142   format (1x,6a4,a2,5x,i2)
141   format (6a4,a2,i2)
381   continue
      print,"type 1 and hit return key when ready"
      read,ready
      call closer (17)
383   call erase
      call movabs (30,725)
      call anmode
      print,"enter state id number"
      read(5,140) istate
140   format (a4)
c
      fmt="(4hcoor,a4)"
      encode (fname,fmt) istate
      call assoc (10,fname,"si  ")
      fmt="(4hbord,a4)"
      encode (fname,fmt) istate
      call assoc (11,fname,"si  ")
      fmt="(4hstat,a4)"
      encode (fname,fmt) istate
      call assoc (14,fname,"si  ")
      fmt="(4hcoun,a4)"
      encode (fname,fmt) istate
      call assoc (15,fname,"si  ")
      fmt="(4hgrid,a4)"
      encode (fname,fmt) istate
      call assoc (16,fname,"si  ")
```

```
c
c
c       request input file for plotting
c
380     continue
        kk=0
c
        call movabs (30,715)
        call anmode
        print,"if a coordinate file is to be plotted, enter y"
        read (5,130) icor
        if (icor .ne. jes)   go to 343
        print,"enter name of file to be plotted"
        read (5,131) ifile
131     format(a8)
        call assoc (13,ifile,"si  ")
c
c       eliminate duplicate id numbers
c         and sort id numbers into ascending order
c
        print,"if input should be sorted reply with a y for yes"
        read (5,130) irep
130     format(a1)
        if (irep .ne. jes)   go to 343
c
        rewind ing
        im=0
c
        im1=0
339     read (ing,124,end=340) imap
124     format (i10)
        im=im+1
        mort(im)=imap
        if (im .lt. 300) go to 339
        print,"buffer exceeded"
340     continue
        call closer (13)
        k=0
c
        do 338 i=1,im
        les=mort(i)
        do 364 j=i,im
        if (les .le. mort(j))  go to 364
        less=mort(j)
        mort(j)=les
        les=less
364     continue
        mort(i)=les
338     continue
c
        rewind ing
        call assoc (13,ifile,"sio ")
        imap1=mort(1)
        write (ing,124) imap1
        k=k+1
```

```
      do 363 i=2,im
      imap=mort(i)
      if (imap .eq. imap1) go to 363
      write (ing,124) imap
      imap1=imap
      k=k+1
363   continue
      end file ing
      rewind ing
343   continue
c
c     set origin on plotter
c
      call pos(11,xx,yy)
      x1=amin1(xx(2),xx(3),xx(4),xx(5),xx(6))
      x2=amax1(xx(2),xx(3),xx(4),xx(5),xx(6))
      y1=amin1(yy(2),yy(3),yy(4),yy(5),yy(6))
      y2=amax1(yy(2),yy(3),yy(4),yy(5),yy(6))
      dx=x2-x1
      dy=y2-y1
c
c           use the bord file to compute the plot window
c
      call dwindo (0.,dx,0.,dy)
553   x=(dx*780.)/dy
      ix=x
      ix1=1023-ix
      if (dx .gt. dy) go to 550
      call swindo (ix1,ix,0,780)
      go to 551
550   y=(dy*1023.)/dx
      iy=y
      if (iy .gt. 780)   go to 552
      call swindo (0,1023,0,iy)
      go to 551
552   ix1=iy-780
      ix=1023-ix1
      call swindo (ix1,ix,0,780)
551   continue
c
c
c     border information
c
303   continue
      call movabs(30,650)
      call anmode
  302 print,"enter title for map"
      read (5,122) lead
122   format (20a4)
      print,"to plot state enter 1"
      read (5,160) istat
  160 format(i1)
      print,"county plot-enter 1 for solid line, 2 for dotted, else 0"
      read (5,160) icoun
      print,"to plot grid enter 1"
      read (5,160) igrid
      print,"to superimpose another file,enter 0 for no, "
      print,"1 for lines only, 2 for lines and characters"
      read (5,160) isup
      if (isup .eq. 0) go to 304
      print,"enter file name"
      read (5,131) ifile
      call assoc (20,ifile,"si  ")
  304 continue
      call erase
      call movabs (30,750)
      call anmode
      write (6,123) lead
123   format (1x,20a4)
c
c          draw neat line
c
      if (kk .eq. 0)   go to 419
      call movea(x1,y1)
      call drawa (x2,y1)
      call drawa (x2,y2)
      call drawa (x1,y2)
      call drawa (x1,y1)
c
c          plot base map
c
419   continue
c
      call plo(11)
      if (istat .ne. 1)  go to 470
      call plo(14)
470   if (icoun .ne. 1)  go to 476
      call plo (15)
476   if (icoun .ne. 2)  go to 471
      call plod (15)
471   if (igrid .ne. 1)  go to 472
      call ploco(16)
c
  472 if (icor .ne. jes)   go to 89
c
c     plot coor file
c
      rewind ing
      rewind ingd
c
c
319   continue
      read (ing,124,end=89) imap
320   read (ingd,111,end=89) if,ifno,isf,isfno,if1,jstat,jgrat,jspan
      if=(if*100)+isf
111   format (8i5)
      if (if-imap)  335,337,360
c
360   read (ing,124,end=89) imap
      if (if-imap) 335,337,360
```

```
c
335   do 336 j=1,isfno,6
      read (ingd,126) (xx(i),yy(i),i=1,6)
336   continue
      go to 320
c
337   continue
c
c
      ie=isfno
      if (isfno .ge. 6)    ie=6
      read (ingd,126) (xx(i),yy(i),i=1,ie)
126   format (12f6.3)
c
      if (isfno .ge.3)   go to 321
c
      if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 320
      if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 320
      call movea (xx(1),yy(1))
      call anmode
      write (6,137) if
137   format (1x,i5)
      if (jspan .eq. 0) go to 520
      ry=yy(1)-rsiz
      call movea (xx(1),ry)
      call anmode
      write (6,137) jspan
520   call movea (xx(2),yy(2))
      call ancho(ichar)
      if (if1 .eq. 0)  go to 320
      ry=ry-rsiz
      call movea (xx(1),ry)
      call anmode
      write (6,137) if1
      go to 320
c
321   continue
      if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 449
      if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 449
      call movea (xx(1),yy(1))
      call anmode
      write (6,137) if
      if (jspan .eq. 0)   go to 521
      ry=yy(1)-rsiz
      call movea (xx(1),ry)
      call anmode
      write (6,137) jspan
c
521   if (if1 .eq. 0)  go to 449
      ry=ry-rsiz
      call movea (xx(1),ry)
      call anmode
      write (6,137) if1
```

```
c
      go to 449
c
c
447   continue
      call movea (xx(1),yy(1))
      call anmode
      write (6,137) if
c
449   continue
      call movea (xx(2),yy(2))
c
      do 341 k=2,ie
      call drawa (xx(k),yy(k))
341   continue
c
c
537   isfno=isfno-6
      if (isfno) 320,320,345
345   if (isfno-6) 322,322,333
322   ie=isfno
333   read (ingd,126,end=88) (xx(i),yy(i),i=1,ie)
c
      do 342 k=1,ie
      call drawa (xx(k),yy(k))
342   continue
c
      if (isfno-6) 320,320,537
c
88    write (6,128) if,ifno,isf,isfno
128   format (1x,i3,2i2,i5)
      write (6,129)
129   format (1x,"rec count err")
      go to 99
c
c              superimpose another file
c
89    continue
      if (isup-1) 305,505,506
505   call plo (20)
      call closer (20)
      go to 305
506   call ploc6 (20)
      call closer (20)
c
c
c          copy and/or exit
c
  305 continue
      call bell
      call anmode
      read (5,130) icopy
      if (icopy .ne. kop)  go to 452
      call hdcopy
452   continue
c
c     selected portion of plot can be enlarged
c
      call movabs(30,730)
      call anmode
      print,"for an enlargement of a part of this plot, type y"
      read (5,130) irep
      kk=kk+1
      if (irep .ne. jes)    go to 477
      call movabs (30,720)
      call anmode
      print,"position cursor at lower left of desired area, type c"
      call vcursr(ichar,x1,y1)
      call movabs (30,710)
      call anmode
      print,"position cursor at upper right of desired area, type c"
      call vcursr(ichar,x2,y2)
      call dwindo (x1,x2,y1,y2)
      call erase
      go to 303
c
c
c     more data or exit
c
477   call movabs (30,710)
      call anmode
      print,"to plot another file enter y for yes"
      read (5,130) irep
      if (irep .ne. jes)    go to 99
      if (icor .ne. jes)   go to 306
      call closer (13)
  306 continue
      call erase
      go to 380
c
99    continue
      call closer (10)
      call closer (11)
      call closer (14)
      call closer (15)
      call closer (16)
      if (icor .ne. jes)   go to 307
      call closer (13)
  307 write (o,106)
106   format (7h   good)
      call finitt (0,0)
c
      stop
c
      end
```

SUBROUTINE NAME: POS

Author: P. A. Fulton
Purpose of the program: **pos** reads the ***bordNM*** data file, which consists of a header card and a card with six *x*, *y* coordinate pairs. ***bordNM*** is the neat line around the map plot, and it serves to define the plot window.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call pos (ingd,xx,yy)
Arguments:
ingd– The unit number assigned to the ***bordNM*** file (This is an input value to the subroutine.)
xx– An array containing six elements (The *x* coordinates are stored in it and passed as output from the subroutine.)
yy– An array dimensioned 6 (The *y* coordinates are stored in it and passed as output from the subroutine.)
Subroutines called: None
Common data referenced: None
Input files: ***bordNM***
Output files: None
Arrays used: *xx*(6), *yy*(6)
Called by: **pn16**
Error checking and reporting: None
Constants: None
Program logic:
1. The program picks up the device number via the argument and reads the header and data images; then, it passes the *x*, *y* coordinates to the calling routine via the arguments and returns. This subroutine only accesses the file. The file opening, closing, and all other manipulations are done in the calling routine.

```
      SUBROUTINE POS(INGD,XX,YY)
      DIMENSION XX(6),YY(6)
      IE=6
      REWIND INGD
320   READ (INGD,111,END=89) IF,IFNO,ISF,ISFNO,IF1
 111  FORMAT (5I5)
      READ (INGD,126) (XX(I),YY(I),I=1,IE)
126   FORMAT (12F6.3)
  89  RETURN
      END
```

SUBROUTINE NAME: PLO

Author: P. A. Fulton
Purpose of the program: **plo** plots solid outlines and points for spatial data files that are structured the same as the ***coorNM*** files. These are ***bordNM***, ***gridNM***, ***statNM***, and ***counNM***.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call plo (in)
Arguments: *in*– The unit number assigned to the file to be plotted
Subroutines called: Tektronix plot routines– **movea, ancho, drawa**
Common data referenced: None
Input files: ***bordNM***, ***statNM***, ***counNM***, ***gridNM***
Output files: None
Arrays used: *xx*(6), *yy*(6)
Called by: **pn16**
Error checking and reporting: None
Constants: *ichar* = 43, *rsiz* = 0.3
Program logic:
1. This subroutine only reads the file and plots the data. The file opening and closing procedures are done in the calling routine.

```
      SUBROUTINE PLO(IN)
C
      DIMENSION XX(6),YY(6)
      ICHAR=43
      RSIZ=0.3
      REWIND IN
      IPEN=0
```

```
C       PLOT DATA FROM FIRST SOURCE
  419   CONTINUE
420     READ (IN,111,END=319) IF,IFNO,ISF,ISFNO,IF1
111     FORMAT (5I5)
132     FORMAT (1X,5I5)
        IPEN=IPEN+1
        IF (IPEN .GT. 4)   IPEN=1
        NL=NL+1
C
        IE=ISFNO
        IF (ISFNO .GE. 6)    IE=6
        READ (IN,  126) (XX(I),YY(I),I=1,IE)
  126   FORMAT (12F6.3)
127     FORMAT (1X,12F6.3)
        IF (ISFNO .GE. 3)  GO TO 448
        CALL MOVEA (XX(2),YY(2))
        CALL ANCHO(ICHAR)
        GO TO 420
C
  448   CONTINUE
        CALL MOVEA (XX(2),YY(2))
        DO 441 K=2,IE
        CALL DRAWA (XX(K),YY(K))
  441   CONTINUE
437     ISFNO=ISFNO-6
        IF (ISFNO) 419,419,445
445     IF (ISFNO-6) 422,422,433
  422   IE=ISFNO
433     READ (IN,  126,END=88)  (XX(I),YY(I),I=1,IE)
C
        DO 442 K=1,IE
        CALL DRAWA (XX(K),YY(K))
  442   CONTINUE
C
        IF (ISFNO-6) 419,419,437
C
   88   CONTINUE
  319   CONTINUE
        RETURN
        END
```

SUBROUTINE NAME: PLOD

Author: P. A. Fulton

Purpose of the program: **plod** plots points and dotted outlines for spatial data files that are structured the same as the *coorNM* files.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call plod (in)

Arguments: in—The unit number assigned to the file to be plotted

Subroutines called: Tektronix plot routines—**movea, ancho, dasha**

Common data referenced: None

Input files: counNM

Output files: None

Arrays used: xx(6), *yy*(6)

Called by: **pn16**

Error checking and reporting: None

Constants: ichar = 43, *rsiz* = 0.3

Program logic:

1. This subroutine only reads the file and plots the data. The file opening and closing procedures are done in the calling routine.

```
      SUBROUTINE PLOD(IN)
C
      DIMENSION XX(6),YY(6)
      RSIZ=0.3
      ICHAR=43
      REWIND IN
      IPEN=0
C     PLOT DATA FROM FIRST SOURCE
  419 CONTINUE
  420 READ (IN,111,END=319) IF,IFNO,ISF,ISFNO,IF1
  111 FORMAT (5I5)
  132 FORMAT (1X,5I5)
      IPEN=IPEN+1
      IF (IPEN .GT. 4) IPEN=1
      NL=NL+1
C
      IE=ISFNO
      IF (ISFNO .GE. 6) IE=6
      READ (IN,  126) (XX(I),YY(I),I=1,IE)
  126 FORMAT (12F6.3)
  127 FORMAT (1X,12F6.3)
      IF (ISFNO .GE. 3) GO TO 448
      CALL MOVEA (XX(2),YY(2))
      CALL ANCHO(ICHAR)
      GO TO 420
C
  448 CONTINUE
      CALL MOVEA (XX(2),YY(2))
      DO 441 K=2,IE
      CALL DASHA (XX(K),YY(K),1)
  441 CONTINUE
  437 ISFNO=ISFNO-6
      IF (ISFNO) 419,419,445
  445 IF (ISFNO-6) 422,422,433
  422 IE=ISFNO
  433 READ (IN,  126,END=88) (XX(I),YY(I),I=1,IE)
C
      DO 442 K=1,IE
      CALL DASHA (XX(K),YY(K),1)
  442 CONTINUE
C
      IF (ISFNO-6) 419,419,437
C
   88 CONTINUE
  319 CONTINUE
      RETURN
      END
```

SUBROUTINE NAME: PLOC6

Author: P. A. Fulton

Purpose of the program: **ploc6** plots solid outlines, points, and identification numbers for spatial data files that are structured the same as the *coorNM* files.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call ploc6 (in)

Arguments: *in*– The unit number assigned to the file to be plotted

Subroutines called: Tektronix plot routines– **movea, anmode, ancho, drawa**
Common data referenced: ***x1, y1, x2, y2***
Input files: ***gridNM*** used on unit 16 (***file16***)
Output files: None
Arrays used: ***xx***(6), ***yy***(6)
Called by: **pn16**
Error checking and reporting: None
Constants: ***ichar*** = 43, ***rsiz*** = 0.3, ***ipen*** = 0
Program logic:

1. This subroutine only reads and plots the data. The file opening and closing procedures are done in the calling routine.

```
      SUBROUTINE PLOC6 (IN)
      DIMENSION XX(6),YY(6)
      COMMON X1,Y1,X2,Y2
      ICHAR=43
      RSIZ=0.3
      IPEN=0
      REWIND IN
  419 CONTINUE
420   READ (IN,111,END=319) IF,IFNO,ISF,ISFNO,IF1
111   FORMAT (5I5)
132   FORMAT (1X,5I5)
      IPEN=IPEN+1
      IF (IPEN .GT. 4)   IPEN=1
      NL=NL+1
C
      IE=ISFNO
      IF (ISFNO .GE. 6)    IE=6
      READ (IN,  126) (XX(I),YY(I),I=1,IE)
  126 FORMAT (12F6.3)
127   FORMAT (1X,12F6.3)
      IF (ISFNO .GE. 3)   GO TO 421
C
      IF ((XX(1) .GE. X2) .OR. (YY(1) .GE. Y2))  GO TO 420
      IF ((IFNO .EQ. 1) .AND. (ISF .EQ. 1))  GO TO 450
      CALL MOVEA (XX(1),YY(1))
      CALL ANMODE
      WRITE (6,137) IF
137   FORMAT (1X,I5)
      RY=YY(1)-RSIZ
      CALL MOVEA (XX(1),RY)
      CALL ANMODE
      WRITE (6,137) ISF
      CALL MOVEA (XX(2),YY(2))
      CALL ANCHO(ICHAR)
      GO TO 420
C
  450 CONTINUE
      CALL MOVEA (XX(1),YY(1))
      CALL ANMODE
      WRITE (6,137) IF
      CALL MOVEA (XX(2),YY(2))
      CALL ANCHO(ICHAR)
      GO TO 420
C
  421 CONTINUE
      IF ((XX(1) .LE. X1) .OR. (YY(1) .LE. Y1)) GO TO 448
      IF ((XX(1) .GE. X2) .OR. (YY(1) .GE. Y2)) GO TO 448
      IF ((IFNO .EQ. 1) .AND. (ISF .EQ. 1))  GO TO 446
      CALL MOVEA (XX(1),YY(1))
      CALL ANMODE
      WRITE (6,137) IF
      RY=YY(1)-RSIZ
      CALL MOVEA (XX(1),RY)
      CALL ANMODE
      WRITE (6,137) ISF
      IF (IF1 .EQ. 0)  GO TO 448
      RY=RY-RSIZ
      CALL MOVEA (XX(1),RY)
      CALL ANMODE
      WRITE (6,137) IF1
C
      GO TO 448
C
  446 CONTINUE
      CALL MOVEA (XX(1),YY(1))
      CALL ANMODE
      WRITE (6,137) IF
C
  448 CONTINUE
      CALL MOVEA (XX(2),YY(2))
      DO 441 K=2,IE
      CALL DRAWA (XX(K),YY(K))
  441 CONTINUE
C
437   ISFNO=ISFNO-6
      IF (ISFNO) 419,419,445
445   IF (ISFNO-6) 422,422,433
  422 IE=ISFNO
433   READ (IN,  126,END=88)  (XX(I),YY(I),I=1,IE)
      DO 442 K=1,IE
      CALL DRAWA (XX(K),YY(K))
  442 CONTINUE
C
      IF (ISFNO-6) 419,419,437
C
   88 CONTINUE
  319 CONTINUE
      RETURN
      END
```

SUBROUTINE NAME: ASSOC

Author: R. W. Bowen
Purpose of the program: **assoc** performs the I/O functions necessary to access a file. These functions are ***attach***, ***open***, ***close***, and ***detach***.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call assoc (iunit,name,mode)

Arguments:
iunit– Fortran I/O unit
name– Name of disk file data set (may be passed as a character string literal with 8 characters, or as a double precision variable, or as a character string variable)
mode– "si" for formatted input, "so" for formatted output, "sqi" for unformatted input, "sqo" for unformatted output, "di" for keyed input, "do" for keyed output

Subroutines called: **io_call**
Common data referenced: None
Input files: Any file needed by user
Output files: None
Arrays used: None
Called by: **pn16**
Error checking and reporting: None
Constants: None
Program logic:
1. This subroutine performs the I/O functions of attach and open on the file passed to it as the *name* parameter when called by the entry **assoc**. When called via the entry point, **closer**, the subroutine closes and detaches the *name* file.

```
      subroutine assoc (iunit,name,mode)
c
c   iunit= fortran i/o number
c   name=  name of disk file data set.  May be passed as a character
c          string literal with 8 characters or as a double precision
c          variable or as a character string variable
c   mode=  "si   " for formatted input
c          "so   " for formatted output
c          "sqi  " for unformatted input
c          "sqo  " for unformatted output
c          "di   " for keyed input
c          "do   " for keyed output
c
      character name*8,fname*6,mode*4,fmt*12
      fmt="(4hfile,i2) "
      if (iunit .le. 9)  fmt="(5hfile0,i1)"
      encode (fname,fmt) iunit
c
      call io ("attach",fname,"vfile_ ",name,"-append")
      call io ("open",fname,mode)
      return
c
      entry closer(iunit)
      endfile iunit
      fmt="(4hfile,i2) "
      if (iunit .le. 9)  fmt="(5hfile0,i1)"
      encode (fname,fmt) iunit
      call io ("close",fname)
      call io ("detach",fname)
      return
      end
```

PROGRAM NAME: BIGSTA

Author: Harold Johnson
Purpose of the program: **bigsta** compiles statistics on the reference and coordinate files.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: bigsta
Arguments: None
Subroutines called: **ftnumber**, **out1_bigsta**, **out2_bigsta**, **bigcal_bigsta**
Common data referenced: *ncd*
Input files: *coorNM*, *cordNM*, *statNM*, *strdNM*, *counNM*, *curdNM*, *cntrNM*, *ctrdNM*, *gridNM*, *refNM*, *areaNM*, *redyNM*, *measNM*, *bordNM*
Output files: Listing
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. The user is asked for the FIPS for the State whose files are being processed.
2. This number is concatenated with *coorNM* and *cordNM* that are then passed to **out_bigsta**,

which calculates the number of records in these files, the number of header cards, the number of data points, and the sum of the perimeters for these outlines. These data are written to the user.

3. The number of header cards is used by **bigsta** to compute the number of cards in *comxNM*. This is written to the user.
4. The State code is concatenated with *statNM* and *strdNM* and passed to **out1_bigsta**, which makes the same calculations for these files, writing the results to the user.
5. The same steps are used to process *counNM* and *curdNM*.
6. The same steps are used to process *cntrNM* and *ctrdNM*.
7. **bigcal_bigsta** is called directly to process *gridNM*.
8. The State code is concatenated with *refNM* and passed to **out2_bigsta**, which counts the cards in the reference file.
9. The user is asked whether there are more files to be processed. If she names one, **out2_bigsta** is called to count its cards.
10. When no more files are to be processed, the program gives a grand total of the number of cards in all these files.

```
c******* BIGSTA *******
c
external ftnumber(descriptors), closer
c
          common ncd
           character state*2, iblank*1, filename*6, mode*4, outfile*
\c6, no*6
c  PROGRAM UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c
c converted to Multics May 23, 1977 by h johnson
           data no/"no    "/
          data iblank/" "/
c THIS PROGRAM COMPILES STATISTICS ON THE FOLLOWING FILES:
c COOR,CORD,COMX,STAT,STRD,COUN,CURD,CNTR,CTRD,GRID,PARA,REF,
c FOR A USER-DESIGNATED STATE.  IT ALSO ALLOWS
c THE USER TO SPECIFY OTHER DESIRED FILES.
c
c NCD WILL BE THE TOTAL NUMBER OF CARDS IN ANY OF THESE FILES.
c
           write(6,900)
900        format(" TYPE THE 2-DIGIT STATE CODE FOR THE STATE",
          &" BEING STUDIED. HIT RETURN.")
           read(5,910)state
910        format(a2)
c
          ncd=0
c
c THIS ENABLES THE OPERATOR TO SELECT THE STATE.
c
c THE FIRST FILE IS COOR, FROM WHICH CORD AND  COMX
c CAN BE EVALUATED.
c
           encode(outfile,912)state
912        format("coor",a2)
           encode(filename,913)state
913        format("cord",a2)
           mode = "si  "
c
          nfile = 10
          call out1_bigsta(outfile,filename,mode,nheadr,nfile)
```

```
c OUT1 USES OUTFILE FILE TO WRITE THE OUTPUT FOR THE
c COORDINATE FILE AND ITS RADIAN ANALOGUE.  NHEADR
c IS THE NUMBER OF HEADER CARDS IN THE FILE.
c NHEADR IS USED NEXT TO FIND THE SIZE OF COMX
c
          ncd=ncd+nheadr
c
          write(6,920)state,nheadr
920       format("0THE FILE comx",a2," IS ON ",i5," CARDS.")
c
c NOW DO STAT AND STRD.
10        nfile = nfile + 2
           encode(outfile,923)state
923       format("stat",a2)
         encode(filename,924)state
924       format("strd",a2)
         call out1_bigsta(outfile,filename,mode,nheadr,nfile)
c
c NOW DO COUN AND CURD.
20        nfile=nfile+2
         encode(outfile,925)state
925       format("coun",a2)
         encode(filename,926)state
926       format("curd",a2)
         call out1_bigsta(outfile,filename,mode,nheadr,nfile)
c
c NOW DO CNTR AND CTRD.
30        nfile=nfile+2
         encode(outfile,927)state
927       format("cntr",a2)
         encode(filename,928)state
928       format("ctrd",a2)
         call out1_bigsta(outfile,filename,mode,nheadr,nfile)
c
c NOW DO GRID.
         encode(outfile,929)state
929       format("grid",a2)
40        nfile=nfile+2
          call ftnumber(nfile,outfile,mode)
  call bigcal_bigsta(nfile,perim,ncards,nrads,npoint,nheadr,outfile)

         write(6,930)outfile,ncards,npoint,nheadr,perim
930       format("0THE FILE ",a6," IS ON ",i5," CARDS AND INVOLVES
\c",
         &i6," DATA POINTS"/"    FOR ",i4,"MAP OUTLINES OF TOTAL ",
         &"LENGTH ,",f10.3," INCHES.")
c
         ncd=ncd+ncards
c
c NOW COUNT THE CARDS IN THE REMAINING FILES.
c
         encode(outfile,932)state,iblank
```

```
932         format("ref",a2,a1)
43          nfile=nfile+2
            call out2_bigsta(outfile,nfile,mode)
c THIS SUBROUTINE RUNS THROUGH THE FILE OUTFILE
c AND COUNTS THE CARDS.  IT THEN WRITES THE TOTAL.
c
encode(outfile,939)state
939 format("area",a2)
nfile = nfile + 2
   call out2_bigsta(outfile,nfile,mode)
c
encode(outfile,933)state
933 format("redy",a2)
nfile = nfile + 2
   call out2_bigsta(outfile,nfile,mode)
c
encode(outfile,934)state
934 format("meas",a2)
nfile = nfile + 2
   call out2_bigsta(outfile,nfile,mode)
c
encode(outfile,935)state
935 format("bord",a2)
nfile = nfile + 2
   call out2_bigsta(outfile,nfile,mode)
c
c
c
c NOW ASK THE USER IF THERE ARE ANY OTHER CARD FILES
c WHICH HE WOULD LIKE COUNTED.
c
50          write(6,940)
940         format("0IF THERE ARE MORE CARD FILES TO BE COUNTED",/
           &",TYPE THE NAME OF ONE. OTHERWISE, TYPE ""no"" AND",
           &" HIT RETURN")
           read(5,950)outfile
950         format(a6)
            if(outfile .eq. no) go to 100
60          nfile=nfile+2
            call out2_bigsta(outfile,nfile,mode)
            go to 50
c
100         write(6,960)ncd
960         format("0****** THE TOTAL NUMBER OF CARDS IN THESE FILES
\cIS",i10)
c
do 120 k = 10, nfile, 2
    call closer(k)
120 continue
c
           end
c ******* END BIGSTA *******
```

SUBROUTINE NAME: OUT1_BIGSTA

Author: Harold Johnson
Purpose of the program: **out1_bigsta** associates Fortran numbers with two file names. It also calls **bigcal_bigsta** to perform calculations on these files. It reports the results by writing a message to the user.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call out1_bigsta (outfile,filename,-mode,nheadr,nfile)
Arguments:
 outfile–The name of a coordinate file, such as *coorNM*, *statNM*, *counNM*, or *cntrNM*
 filename–The name of the radian file that corresponds to the coordinate file *outfile*
 nheadr–The number of header cards in *outfile*
 nfile–A Fortran file number
Subroutines called: **ftnumber**, **bigcal_bigsta**
Common data referenced: *ncd*
Input files: None
Output files: None
Arrays used: None
Called by: **bigsta**
Error checking and reporting: None
Constants: None
Program logic:
1. **Assoc** is called to associate *nfile* to *outfile*.
2. **bigcal_bigsta** is called to compute the total perimeter of the outlines in *outfile*, the number of cards in *outfile*, the number of cards in *filename*, and the number of header cards in *outfile*.
3. The results are reported to the user.

```
c ******* SUBROUTINE OUT1_BIGSTA *******
          subroutine out1_bigsta(outfile,filename,mode,nheadr,nfile)
           character outfile*6, filename*6, mode*4
           common ncd
c
c SUBROUTINE USED IN BIGSTA PROGRAM
c Converted to Multics May 24, 1977 by H Johnson
c
           call ftnumber(nfile,outfile,mode)
  call bigcal_bigsta(nfile,perim,ncards,nrads,npoint,nheadr,outfile)
           write(6,900)outfile,ncards,npoint,nheadr,perim
900        format("0THE FILE ",a6," IS ON ",i5," CARDS AND INVOLVES ",
          i6," DATA POINTS "/" FOR ",i4," MAP OUTLINES OF TOTAL LENGTH ",
          f10.3," INCHES.")
           write(6,930)filename,nrads,npoint
930        format("0THE FILE ",a6," IS ON ",i5," CARDS AND INVOLVES ",
          i6," DATA POINTS.")
c
           ncd=ncards+nrads+ncd
c
110        return
          end
c ******* END OUT1_BIGSTA *******
```

SUBROUTINE NAME: OUT2_BIGSTA

Author: Harold Johnson
Purpose of the program: **out2_bigsta** is used to count the records in a file.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call out2_bigsta (outfile,nfile,mode)
Arguments:
 outfile– The name of a file
 nfile– A fortran number that is to be associated with outfile *mode*– Specifies input or output
Subroutines called: **ftnumber**
Common data referenced: *ncd*
Input files: None
Output files: None

Arrays used: None
Called by: **bigsta**
Error checking and reporting: Any read error is reported and control is returned to the calling program.
Constants: None
Program logic:
1. ***Outfile*** is associated with the fortran number ***nfile***.
2. Cards are successively read into (A1) format, one character per card, and counted.

```
c  *******  SUBROUTINE OUT2_BIGSTA *******
             subroutine out2_bigsta(outfile,nfile,mode)
            character outfile*6, mode*4
             common ncd
c
             call ftnumber(nfile,outfile,mode)
             kount = 0
1            read(nfile,900,end=100,err=110)a
900          format(a1)
             kount=kount+1
             go to 1
c
100          continue
             write(6,910)outfile,kount
910          format("0THE FILE ",a6," IS ON ",i5," CARDS.")
c
             ncd=ncd+kount
return
c
110 write(6,920) outfile
920 format("0THERE SEEMS TO BE AN ERROR IN THE FILE :",a6)
return
c
            end
```

SUBROUTINE NAME: BIGCAL_BIGSTA

Author: Harold Johnson
Purpose of the program: **bigcal_bigsta** compiles statistics on the coordinate files and on their radian counterparts.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call bigcal_bigsta (nfile,perim,ncards,-nrads,npoint,nheadr,outfile)
Arguments:
- ***nfile***– A fortran file number
- ***perim***– The total length in inches of the outlines in the coordinate file with number ***nfile***
- ***ncards***– The total number of cards in ***nfile***
- ***nrads***– The total number of cards in the radian file that corresponds to this coordinate file
- ***npoint***– The total number of data points in the coordinate file or radian file
- ***nheadr***– The total number of header cards found in these files (which is used to tell the number of outlines)

Subroutines called: **prim_bigsta**, **icards_bigsta**, **irads_bigsta**
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **out1_bigsta, bigsta**
Error checking and reporting: On a read error, control returns to the calling program.
Constants: None
Program logic:
1. A record is read as a header card from ***nfile***.
2. **icards_bigsta** is called to calculate the number of data cards that should follow for this header card.
3. **irads_bigsta** is called to calculate the number of data cards that should follow this header in the radian file.
4. ***ncards***, ***nheadr***, ***nrads***, and ***npoint*** are updated.
5. **prim_bigsta** is called to calculate the length of the outline whose coordinate points follow. ***perim*** is updated.

```
c ******* SUBROUTINE BIGCAL_BIGSTA *******
 subroutine bigcal_bigsta(nfile,perim,ncards,nrads,npoint,nheadr,outfile)
  character outfile*6
c SUBROUTINE USED IN MAIN PROGRAM "BIGSTA"
c UPDATED AS OF DEC. 27, 1976  H. JOHNSON
c
c
c THIS SUBROUTINE COMPILES STATISTICS ON THE FILE NFILE AND
c THE CORRESPONDING RADIAN FILE, IF ONE EXISTS.
c
c INITIATE
c
c subroutine used in program bigsta.
c converted to multics  May 23, 1977, H Johnson
c
          perim=0.
          ncards=0
          nrads=0
          npoint=0
          nheadr=0
c
1         read(nfile,900,end=1000,err=1100)if,ifno,isf,isfno,not,nor,nif
900       format(7i5)
          ic=icards_bigsta(isfno)
          ncards=ncards+1+ic
          ir=irads_bigsta(isfno)
          nrads=nrads+1+ir
          npoint=npoint+isfno
          nheadr=nheadr+1
          call prim_bigsta(nfile,isfno,dist)
          perim=perim+dist
          go to 1
1000 return
c
1100 write(6,910)outfile
910 format("0THERE SEEMS TO BE AN ERROR IN FILE :",a6)
return
c
c
         end
```

FUNCTION NAME: IRADS_BIGSTA

Author: Harold Johnson

Purpose of the program: **irads_bigsta** calculates the number of cards in a radian file that must be used to contain *isfno* data points.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: ir = irads_bigsta (isfno)

Arguments:

irads—The number of cards needed to hold *isfno* data points in a radian file

isfno—A certain number of data points

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **bigcal_bigsta**

Error checking and reporting: None

Constants: None

Program logic:

1. Similar to **icards_bigsta**, except that only three data points can occur on radian coordinate files.

```
c ******* FUNCTION IRADS_BIGSTA *******
          function irads_bigsta(isfno)
          irads_bigsta=isfno/3
          if(3*irads_bigsta .lt. isfno)irads_bigsta=irads_bigsta+1
          return
c
          end
c ******* END IRADS_BIGSTA *******
```

SUBROUTINE NAME: PRIM_BIGSTA

Author: Harold Johnson
Purpose of the program: **prim_bigsta** calculates the length in inches of an outline in one of the coordinate files (*coorNM, statNM, counNM, gridNM*).
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call prim_bigsta (in,isfno,finch)
Arguments:
in—A fortran file number
isfno—The number of data points on the next and following cards that describes an outline
finch—The length in inches of the outline described by the data points
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: xx(6), yy(6)
Called by: **bigcal_bigsta**
Error checking and reporting: None
Constants: None
Program logic:
1. First, the number of data cards that contain *isfno* points is calculated.
2. The first data card is read. If *isfno* is 2, control returns because the first data point is used to position characters that name the outline.
3. The length of the segments described by this first card is calculated.
4. If more cards remain in this outline file, they are read one by one, and their outlines are calculated and added to the running total.

```
c ******* SUBROUTINE PRIM_BIGSTA *******
          subroutine prim_bigsta(in,isfno,finch)
c
c subroutine used in main statistics program "bigsta"
c converted to multics May 23, 1977 H Johnson.
c
          dimension xx(6),yy(6)
          finch = 0.
          ncard1=isfno/6
          if(6*ncard1 .lt. isfno) ncard1 = ncard1 + 1
          read(in,910)(xx(i),yy(i),i=1,6)
910       format(12f6.3)
          if(isfno .eq. 2) return
          ilast = 6
          if(isfno .lt. 6) ilast = isfno
          do 10 k=3,ilast
          finch=finch+sqrt((xx(k)-xx(k-1))**2+(yy(k)-yy(k-1))**2)
10         continue
          xlast=xx(6)
          ylast = yy(6)
          if(ncard1 .eq. 1) return
          if (ncard1 .eq. 2) go to 25
          nl=ncard1 - 1
          do 20 j=2,nl
          read(in,910)(xx(i),yy(i),i=1,6)
          finch=finch+sqrt((xx(1)-xlast)**2+(yy(1)-ylast)**2)
          do 15 k = 2,6
          finch=finch+sqrt((xx(k)-xx(k-1))**2+(yy(k)-yy(k-1))**2)
15        continue
          xlast=xx(6)
          ylast=yy(6)
20        continue
25        read(in,910)(xx(i),yy(i),i=1,6)
          ilast = isfno - (ncard1-1)*6
          finch=finch+sqrt((xx(1)-xlast)**2+(yy(1)-ylast)**2)
          if(ilast .eq. 1) return
          do 30 k = 2,ilast
          finch=finch+sqrt((xx(k)-xx(k-1))**2+(yy(k)-yy(k-1))**2)
30        continue
          return
          end
c ******* END PRIM_BIGSTA ******
```

FUNCTION NAME: ICARDS_BIGSTA

Author: Harold Johnson

Purpose of the program: **icards_bigsta** is used to compute from the *isfno* number in a coordinate file the number of data cards that should follow in order to contain the indicated number of points.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: ic = icards_bigsta(isfno)

Arguments:

ic—The number of cards that are required to hold *isfno* data points

isfno—A certain number of data points

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **bigcal_bigsta**

Error checking and reporting: None

Constants: None

Program logic:

1. A maximum of six data points are on each card, so **icards_bigsta** is *isfno*/6 unless 6 does not divide *isfno*, in which case one more card must be used.

```
c ******* FUNCTION ICARDS_BIGSTA *******
          function icards_bigsta(isfno)
          icards_bigsta=isfno/6
          if(6*icards_bigsta .lt. isfno)icards_bigsta=icards_bigsta+1
          return
c
        end
c ******* END ICARDS_BIGSTA *******
```

EXEC_COM NAME: USMERG.EC

Author: P. A. Fulton

Purpose of the program: **usmerg.ec**, written in the Multics command language, takes as input a newly created *indxNM* file and appends it to the existing indxus. indxus is the GRASP file that contains all the States. The output file is named usall. At the end of the run it is dprinted for checking.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: ec usmerg *NM*

Arguments: *NM*—FIPS code for the State

Subroutines called: **setmas**, GRASP

Common data referenced: None

Input files: indxus, *indxNM*

Output files: usall

Arrays used: None

Called by: None

Error checking and reporting: None
Constants: None
Program logic:
1. ***command_line*** is turned off.
2. Attach the exec_com to the program **setmas**.
3. The exec_com contains two responses to prompts in the program **setmas**.
4. When **setmas** is terminated, the program is detached from the exec_com.
5. The exec_com is attached to the program GRASP.
6. GRASP is executed. The exec_com contains responses to prompts made in GRASP.
7. Detach the files.
8. Quit.

```
&command_line off
&attach
setmas
2 us &1
yes
&detach
&attach
grasp
indxus
append
usall
indxus
indx&1
y
quit
y
1,2
&detach
&quit
```

PROGRAM NAME: STATE_TO_TAPE

Author: Harold Johnson
Purpose of the program: **state_to_tape** enables a user to copy all the State files for one State onto a backup tape, using IBM tape characteristics.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: state_to_tape
Arguments: None
Subroutines called: **sts_begin**, **heading_state_to_tape**, **up_file_number**, **disk_to_tape_fb_retain.ec**, **disk_to_tape_vbs_retain.ec**, **list_state_tape.ec**, **date_time**
Common data referenced: None
Input files: ***bginNM***, ***coorNM***, ***cordNM***, ***statNM***, ***strdNM***, ***counNM***, ***curdNM***, ***gridNM***, ***bordNM***, ***cntrNM***, ***paraNM***, ***redyNM***
Output files: None
Arrays used: ***file_name(12)***
Called by: None
Error checking and reporting: If the user has not sent a message to the operator to locate his tape, the program will abort.
Constants: None
Program logic:
1. The user is asked to type 1 if he sent a message to the operator to find his tape; otherwise, 0. The user's response is read into ***ians***.
2. If ***ians*** does not equal 1, the program stops.
3. Prompt:
   TYPE YOUR TAPE ID.
   The response is read into ***tape_number***.
4. Prompt:
   TYPE THE 2-DIGIT FEDERAL STATE NUMBER CODE FOR THIS STATE, USING FORMAT A2
   Response is read into ***state_number***.
5. Prompt:
   TYPE THE NAME OF THIS STATE.
   Response is read into ***state_name***.
6. The user is asked which file number to use. The response is read into ***file_number***.

7. Subroutine **sts_begin** is called to set up a temporary disk file that will contain a description of the records to be read into the *bginNM* file.
8. Subroutine **heading_state_to_tape** is called to set up *file81* containing a description of the files being copied.
9. Call subroutine **up_file_number**.
10. Call **disk_to_tape_fb_retain.ec**.
11. Subroutines **up_file_number and disk_to_tape_vbs_retain.ec** are called for the last three input files as they are recorded with the VBS tape option.
12. Call **list_state_tape.ec**, which prints out the tape label and the file name and numbers.
13. Call **date_time**.
14. End.

```
c PROGRAM state_to_tape
c
c PURPOSE: To enable a user to copy all the state files for one
c  state onto a backup tape, using IBM tape characteristics.
c
c PROGRAMMER: H Johnson
c DATE: Jan 13, 1978
c
    character tape_number*6, file_name*4(12), file_number*2
    character state_number*2, state_name*36, file*6
c
data file_name/"bgin","coor","cord","stat","strd","coun","curd",
"grid","bord","cntr","para","redy"/
c
c
   write(6,910)
910 format("OIF YOU HAVE SENT A MESSAGE TO SYS OP TO FIND YOUR"/
" TAPE, TYPE 1; OTHERWISE, TYPE 0")
    read(5,915) ians
915 format(i1)
   if(ians .ne. 1) stop
c
    write(6,920)
920 format("OTYPE YOUR TAPE ID:")
    read(5,925) tape_number
925 format(a6)
c
    write(6,927)
927 format("OTYPE THE 2-DIGIT FEDERAL STATE NUMBER CODE"/
" FOR THIS STATE, USING FORMAT A2")
    read(5,928) state_number
928 format(a2)
c
    write(6,930)
930 format("OTYPE THE NAME OF THIS STATE:")
    read(5,935) state_name
935 format(a36)
c
   write(6,940)
940 format("OTYPE THE FILE-NUMBER OF THE TAPE FILE JUST"/
" BEFORE THE PLACE WHERE YOU WANT TO BEGIN WRITING THIS STATE."/
"OIF YOU ARE WRITING THE FIRST STATE ON THIS TAPE, TYPE 00;"/
" IF ADDING TO A PREVIOUS STATE, TYPE THE LAST FILE NUMBER;"/
" IF WRITING OVER A PREVIOUSLY WRITTEN VERSION, TYPE THE FILE"/
" NUMBER JUST AHEAD OF THE PLACE WHERE YOU WANT TO BEGIN."/
"OPLEASE TYPE THIS NUMBER FORMAT A2:")
c
    read(5,945) file_number
945 format(a2)
c
c NOW SET UP A TEMPORARY DISK FILE WHICH WILL CONTAIN A
c DESCRIPTION OF THESE RECORDS, TO BE READ INTO THE BGIN FILE.
c
  call sts_begin(state_number,state_name,file_number)
c THIS SETS UP A FILE 81 CONTAINING A DESCRIPTION OF
c THE FILES BEING COPIED.
c
   call heading_state_to_tape(state_number,tape_number)
c
   do 50 i=1, 9
encode(file,950) file_name(i), state_number
950 format(a4,a2)
c
   call up_file_number(file_number)
c
   call ec ("disk_to_tape_fb_retain",tape_number,file,file_number)
c
50 continue
c
   do 60 i=10, 12
c
encode(file,950) file_name(i), state_number
   call up_file_number(file_number)
c
   call ec ("disk_to_tape_vbs_retain",tape_number,file,file_number)
c
60 continue
c
c THE LAST 3 FILES ARE RECORDED WITH THE VBS TAPE OPTION.
c
  call ec ("list_state_tape",tape_number)
c
   call date_time
c
c
end
```

SUBROUTINE NAME: HEADING_STATE_TO_TAPE

Author: Harold Johnson
Purpose of the program: **heading_state_to_tape** writes headings for the output of the **state_to_tape** program.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call heading_state_to_tape (state,tape_number)
Arguments:
state–Two-digit FIPS State code
tape_number–Six-position volume number
Subroutines called: None

Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **state_to_tape**
Error checking and reporting: None
Constants: None
Program logic:
1. Write to the terminal:
   THE FOLLOWING DISK FILES FROM THE STATE WITH FEDERAL CODE *NM* HAVE BEEN STORED ON TAPE *XXXXXX* FOR BACK-UP:
2. Return control to **state_to_tape**.

```
c the name of this file is:  heading_state_to_tape.fortran
   subroutine heading_state_to_tape(state,tape_number)
c PURPOSE: TO WRITE HEADINGS FOR THE OUTPUT OF THE
c STATE-TO-TAPE PROGRAM.
c
c programmer: H Johnson
c date: August 27, 1977
c
   character state*2, tape_number*6
c
write(6,910)
910 format("0*****************************************************")
c
write(6,920)state,tape_number
920 format("0",5X,"THE FOLLOWING DISK FILES FROM THE STATE ",
"WITH FEDERAL CODE ",a2,/"   HAVE BEEN STORED ON TAPE ",a6,
" FOR BACK-UP:")
c
c
return
end
```

SUBROUTINE NAME: UP_FILE_NUMBER

Author: Harold Johnson
Purpose of the program: **up_file_number** increments ***file_number***, given in character format, by 1 and returns the new value in character format.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call up_file_number (file_number)
Arguments: ***file_number***–Numerical sequence on tape
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **state_to_tape**, **pull_off**
Error checking and reporting: None
Constants: None
Program logic:
1. Using the **decode** statement, the program reads ***ifile*** from ***file_number*** and stores it into a format of (i2).
2. Add 1 to ***ifile***.
3. Using the **encode** statement, the program transmits the value of ***ifile*** to ***file_number***.
4. Control is returned to the calling module.

```
c FILE NAME: up_file_number.fortran
c
c PURPOSE: to increment a number, file_number, given in character format,
c          by 1 and return the new value in character format.
c
c PROGRAMMER:H Johnson
c DATE: August 27, 1977
c
   subroutine up_file_number(file_number)
```

```
c
   character state*2, tape_number*6, file_number*2
9990 format("file_number = ",a4)
c
decode(file_number,910)ifile
910 format(i2)
c
ifile = ifile + 1
c
encode(file_number,910) ifile
c
return
end
```

SUBROUTINE NAME: STS_BEGIN

Author: Harold Johnson
Purpose of the program: **sts_begin** sets up a temporary disk file that will contain a description of the input records and will give the user the file number of each file written to tape.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call sts_begin (state, state_name,- file_number)
Arguments:
  state–Two-digit FIPS code
  state_name–Name of the State
  file_number–Number of the tape file
Subroutines called: **ftnumber**, **up_file_number**
Common data referenced: None
Input files: None
Output files: File number: ***file81***
Arrays used: None
Called by: **state_to_tape**
Error checking and reporting: None
Constants: None
Program logic:
1. The file name ***bgin*** and the State code are concatenated by the **encode** statement to form ***bginNM***.
2. Call **ftnumber** to open and attach ***file81*** to ***bginNM*** for sequential input and output.
3. ***n*** = ***file_number*** + 1.
4. Prompt to ***file81***:
   THIS IS THE FIRST FILE FOR STATE NUMBER *NM* NAME *STATE_NAME*.
5. Prompt to ***file81***:
   BEGINNING IN TAPE FILE NUMBER ***n*** THE FOLLOWING STATE FILES ARE WRITTEN ON THIS TAPE!
6. Prompt to ***file81***:
   coorNM IS IN FILE NUMBER ***n***.
7. Add 1 to ***n***.
8. Subroutine writes a message for the next file ***cord-NM*** and repeats steps 7 and 8 until a message has been written for each input file.
9. Rewind 81.
10. Return control to **state_to_tape**.

```
    subroutine sts_begin(state,state_name,file_number)
c
   character state*2, state_name*36, file_name*6, mode*4
  character n*2, file_number*2
c
   encode(file_name,910) state
910 format("bgin",a2)
c
mode="so  "
   call ftnumber(81,file_name,mode)
c
n = file_number
  call up_file_number(n)
  call up_file_number(n)
c
    write(81,912) state, state_name
```

```
912 format("This is the first file for state number ",a2,
" name ",a36)
c
    write(81,920) n
920 format("Beginning in tape file number ",a2," the following state "/
"files are written on this tape:")
   write(81,925) state, n
925 format("coor",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,935) state, n
935 format("cord",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,940) state, n
940 format("stat",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,945) state, n
945 format("strd",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,950) state, n
950 format("coun",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,955) state, n
955 format("curd",a2," is in file number ",a2)
   call up_file_numoer(n)
c
    write(81,960) state, n
960 format("grid",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,965) state, n
965 format("bord",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,970) state, n
970 format("cntr",a2," is in file number ",a2)
   call up_file_number(n)
c
c
    write(81,975) state, n
975 format("para",a2," is in file number ",a2)
   call up_file_number(n)
c
    write(81,980) state, n
980 format("redy",a2," is in file number ",a2)
c
endfile 81
  call closer(81)
c
return
end
```

EXEC_COM NAME: DISK_TO_TAPE_FB_RETAIN.EC

Author: Harold Johnson
Purpose of the program: **disk_to_tape_fb_retain.ec** writes files to tape using fixed block format.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("disk_to_tape_fb_retain",-tape_number,file,file_number)
Arguments:
tape_number–Six-position volume number
file–Name of input file
file_number–Number of the file on tape
Subroutines called: None
Common data referenced: None
Input files: ***bginNM***, ***coorNM***, ***cordNM***, ***statNM***, ***strdNM***, ***counNM***, ***curdNM***, ***gridNM***, ***bordNM***
Output files: Input files are put on tape.
Arrays used: None
Called by: **state_to_tape**
Error checking and reporting: None
Constants: None
Program logic:
1. Using the COPY_FILE command and the I/O module, **tape_ibm**, the program writes the file to tape with a fixed block format, record length of 80. It is written in the file number designated by ***file_number*** with the name designated by ***file***.
2. Control is returned to the calling module.

```
disk-to-tape-fb-retain.ec
cpf -ids "record_stream_ -target vfile_ &2" -ods "tape_ibm_ &1 -nb &3
-nm &2  -fmt fb -rec 80 -bk 8000  -den 800 -cr -ret all -rg"
```

EXEC_COM NAME: DISK_TO_TAPE_VBS_RETAIN.EC

Author: Harold Johnson
Purpose of the program: **disk_to_tape_vbs_retain.ec** writes files to tape using spanned record format
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("disk_to_tape_vbs_retain", tape number,file,file_number)
Arguments:
tape_number– Six-position volume number
file– Name of the input file
file_number– Number of the file on tape
Subroutines called: None
Common data referenced: None
Input files: ***cntrNM***, ***paraNM***, ***redyNM***
Output files: The input files are written to tape.
Arrays used: None
Called by: **state_to_tape**
Error checking and reporting: None
Constants: None
Program logic:
1. Using the COPY_FILE command and the I/O module, **tape_ibm**, the program writes ***cntrNM***, ***paraNM*** and ***redyNM*** to tape using the spanned record format. The input description specifies a record length of 100.
2. Control is returned to the calling module.

```
disk-to-tape-vbs-retain.ec
cpf -ids "record_stream_ -length 100 -target vfile_ &2" -ods "tape_ibm_
&1 -nb &3 -nm &2    -fmt fb -rec 100 -bk 8000  -den 800 -cr -ret
all  -rg"
```

EXEC_COM NAME: LIST_STATE_TAPE.EC

Author: Harold Johnson
Purpose of the program: **list_state_tape.ec** lists the contents of the tape.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("list_state_tape", tape_number)
Arguments: *tape_number*– Six-position volume number
Subroutines called: None
Common data referenced: None
Input files: Tape used in **state_to_tape** module
Output files: None
Arrays used: None
Called by: **state_to_tape**, **pull off**
Error checking and reporting: None
Constants: None
Program logic:
1. The command is LIST_TAPE_CONTENTS &1 -LONG -IOM TAPE_IBM. The information

printed by this command is extracted from the tape labels.
2. The *-long* argument prints the file identifier (*id*), the file sequence number (*number*), the record format (*format*), the physical block size (*blksize*), the logical record length (*lrecl*), the encoding mode (*mode*), the file creation data (*created*), the file expiration date (*expires*), the file-set section number (*section*), the file version number (*version*), the file generation number (*generation*), and the operating system that recorded the tape (*system*).
3. The *-iom* argument invokes a system I/O module to attach and read the specified tape volume. The **tape_ibm_** subroutine is specified in order to list OS standard labeled tapes.
4. Control is returned to the calling program.

```
list_tape_contents s1 -long -iom tape_ibm_
```

PROGRAM NAME: PULL_OFF

Author: Harold Johnson
Purpose of the program: **pull_off** enables the user to pull off files from the Geoindex State files, and writes the selected files to disk.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: pull_off
Arguments: None
Subroutines called: **state_pull_off**, **separate_pull_off**
Common data referenced: None
Input files: User tape containing State files
Output files: Files retrieved from the input tape
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None

Program logic:
1. The user is informed that she must know the tape number, State number code, and file names, or she has the option of pulling off all the files for one State. If only selected files are to be pulled off, the user must know their file numbers on the tape, and the names and file numbers must be entered in ascending order according to the order on the tape.
2. Prompt:
   NOW TYPE THE TAPE ID:
3. The user's response is read into *tape_number*.
4. Prompt:
   IF YOU WANT THE ENTIRE SET OF FILES FOR A STATE, TYPE A 1; OTHERWISE, TYPE 0.
5. The users response is read into *ians*.
6. If *ians* is not equal to 1, go to step 8.
7. Call subroutine **state_pull_off**. Upon return, go to step 9.
8. Call subroutine **separate_pull_off**.
9. Stop.

```
c PROGRAM: pull_off
c
c PROGRAMMER: H Johnson
c DATE: February 14, 1978
c
c INPUT FILES: A user tape, containing state files.
c OUTPUT FILES: Whatever files were retrieved from the tape.
c
  character tape_number*32
c
  write(6,910)
910 format(" THIS PROGRAM ENABLES YOU TO PULL OFF FILES FROM A TAPE"/
" BACK TO DISK."/
" *** WARNING: be sure these files do not already exist"/
" in your directory or in links to another directory!!!"/
" "/
"   IT IS ASSUMED THAT THESE ARE FILES FROM OUR STATE FILES."/
"   YOU MUST KNOW THE TAPE NUMBER, STATE NUMBER CODE, AND FILE NAMES"/
"   OR, YOU CAN PULL OFF ALL THE FILES FOR ONE STATE."/
"  IF YOU WANT ONLY SOME STATE FILES, YOU MUST KNOW THEIR FILE NUMBER"/
" ON THE TAPE.  THEN YOU MUST ENTER THE NAMES AND NUMBERS IN "/
" INCREASING ORDER ACCORDING TO THE ORDER ON THE TAPE."/
"     "/
" NOW TYPE THE TAPE ID :")
c
    read(5,920) tape_number
920 format(a32)
c
c NOW DETERMINE WHETHER OR NOT THE USER WANTS A WHOLE STATE.
c
    write(6,930)
930 format(" IF YOU WANT THE ENTIRE SET OF FILES FOR A STATE,"/
" TYPE A 1; OTHERWISE, TYPE 0")
    read(5,940) ians
940 format(i1)
   if (ians .ne. 1) go to 20
c
   call state_pull_off(tape_number)
go to 30
c
20 call separate_pull_off(tape_number)
c
30 continue
end
```

SUBROUTINE NAME: STATE_PULL_OFF

Author: Harold Johnson
Purpose of the program: **state_pull_off** determines which States are to be retrieved from the tape and then writes them to disk.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call state_pull_off (tape_number)
Arguments: *tape_number*–The six-position volume number
Subroutines called: **tape_to_disk_fb_retain.ec, tape_to_disk_vbs_retain.ec, up_file_number, list_state_tape.ec**
Common data referenced: None
Input files: A user tape containing State files
Output files: *bginNM, coorNM, cordNM, statNM, strdNM, counNM, curdNM, gridNM, bordNM, cntrNM, paraNM, redyNM*
Arrays used: None
Called by: **pull_off**
Error checking and reporting: None
Constants: None
Program logic:

1. Prompt:
   TO RETRIEVE ALL THE FILES FOR ONE STATE, YOU MUST KNOW THE STATE NUMBER CODE AND THE FILE NUMBER OF THE FIRST FILE FOR THE STATE AS IT OCCURS ON THE TAPE.
2. Prompt:
   WHAT IS THE STATE NUMBER CODE? TYPE IT FORMAT I2.
3. Read the response into *state*.
4. Prompt:
   NOW TYPE THE FIRST FILE NUMBER OF THE BGIN FILE FOR THIS STATE, FORMAT A2.
5. Read the response into *file_number*.
6. Using the **encode** statement, concatenate the name of the output file and the State code.
7. Call **tape_to_disk_fb_retain.ec**.
8. Call **up_file_number** subroutine.
9. Repeat steps 6–8 for the first nine output files.
10. Using the **encode** statement, concatenate the name of the file and the State number.
11. Call **tape_to_disk_vbs_retain.ec**.
12. Call subroutine **up_file_number**.
13. Repeat steps 10–12 for the last three output files.
14. Call **list_state_tape.ec** to list the contents of the tape.
15. Return control to the calling program.

```
subroutine state_pull_off(tape_number)
c
c SUBROUTINE USED IN pull_off
c PURPOSE: To determine what state's files are to be retrieved
c   from a tape, and then retrieve them.
c
  character tape_number*32, file_number*2, name*4(12),file*6,state*2
data name/"bgin","coor","cord","stat","strd","coun","curd","grid",
"bord","cntr","para","redy"/
c
    write(6,910)
910 format(" TO RETRIEVE ALL THE FILES FOR ONE STATE, YOU MUST"/
" KNOW THE STATE NUMBER CODE AND THE FILE NUMBER"/
" OF THE FIRST FILE FOR THE STATE AS IT OCCURS ON THE TAPE."/
"OWHAT IS THE STATE NUMBER CODE? TYPE IT FORMAT i2")
c
   read(5,920) state
920 format(a2)
  write(6,924)
924 format(" NOW TYPE THE FIRST FILE NUMBER OF THE BGIN FILE FOR"/
" THIS STATE, FORMAT A2")
   read(5,927) file_number
927 format(a2)
c
  do 20 k = 1, 9
encode(file,930) name(k), state
930 format(a4,a2)
c
   call ec("tape_to_disk_fb_retain",tape_number,file,file_number)
  call up_file_number(file_number)
c
20 continue
c
   do 50 k = 10,12
c
encode(file,930) name(k), state
c
c
   call ec ("tape_to_disk_vbs_retain",tape_number,file,file_number)
c
  call up_file_number(file_number)
50 continue
c
   call ec ("list_state_tape",tape_number)
c
   call date_time
c
return
end
```

SUBROUTINE NAME: SEPARATE_PULL_OFF

Author: Harold Johnson

Purpose of the program: **separate_pull_off** determines which State files are to be retrieved from tape and then writes them to the disk.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call_separate_pull_off (tape_-number)

Arguments: ***tape_number***–Six-position volume number
Subroutines called: **tape_to_disk_fb_retain.ec**, **tape_to_disk_vbs_retain.ec**, **list_state_tape.ec**
Common data referenced: None
Input files: A user tape containing State files
Output files: Any one or more of the following: ***bginNM***, ***coorNM***, ***cordNM***, ***statNM***, ***strdNM***, ***counNM***, ***curdNM***, ***gridNM***, ***bordNM***, ***cntrNM***, ***paraNM***, ***redyNM***.
Arrays used: ***file_name 6(25)***, ***file_number 2(25)***
Called by: **pull_off**
Error checking and reporting: None
Constants: None
Program logic:

1. Initialize k equal to 0.
2. $k = k + 1$.
3. Prompt:
   TYPE THE NAME OF THE NEXT FILE YOU WANT:
4. Read the response into ***file_name(k)***.
5. Prompt:
   TYPE THE FILE NUMBER OF THIS FILE, FORMAT A2
6. Read the user's response into ***file_number(k)***.
7. Prompt:
   TO CONTINUE WITH MORE FILES, TYPE 1, TO STOP, TYPE 0
8. Read user's response into ***ians***.
9. If ***ians*** is equal to 1 go to step 2.
10. Using the **decode** statement, separate the file name from the State number.
11. If the file name is ***cntrNM***, ***paraNM***, or ***redyNM***, go to step 14.
12. Call **tape_to_disk_fb_retain.ec**, which will write the files to disk using fixed block format.
13. Go to step 15.
14. Call **tape_to_disk_vbs_retain.ec**, which will write the files to disk using spanned record format.
15. Repeat steps 10–14 until all files have been written to disk.
16. Call **list_state_tape.ec** to list the contents of the tape.
17. Return control to the calling program.

```
  subroutine separate_pull_off(tape_number)
c
c PURPOSE: TO DETERMINE WHAT STATE FILES ARE TO BE RETRIEVED.
c    THEN RETRIEVE THEM.
c
c
   character tape_number*32, file_name*6(25), file_number*2(25)
c
   character name*4, number*2
  character ipara*4,icntr*4,iredy*4
data icntr/"cntr"/, ipara/"para"/, iredy/"redy"/
c
 k = 0
10 k = k + 1
c
    write(6,910)
910 format(" TYPE THE NAME OF THE NEXT FILE YOU WANT:")
    read(5,920) file_name(k)
920 format(a6)
    write(6,930)
930 format(" TYPE THE FILE NUMBER OF THIS FILE, FORMAT A2")
    read(5,940) file_number(k)
940 format(a2)
c
   write(6,950)
950 format(" TO CONTINUE WITH MORE FILES, TYPE 1,"/
" TO STOP, TYPE 0")
    read(5,960) ians
960 format(i1)
   if (ians .eq. 1) go to 10
```

```
c
  do 50 j = 1, k
c
   decode(file_name(j),970) name, number
970 format(a4,a2)
   if (name .eq. icntr) go to 40
   if (name .eq. ipara) go to 40
   if (name .eq. iredy) go to 40
c
   call ec ("tape_to_disk_fb_retain",tape_number,file_name(j),file_
   number(j))
go to 50
c
40   call ec ("tape_to_disk_vbs_retain",tape_number,file_name(j),file_
     number(j))
50 continue
c
   call ec("list_state_tape",tape_number)
return
end
```

EXEC_COM NAME: TAPE_TO_DISK_FB_RETAIN.EC

Author: Harold Johnson
Purpose of the program: **tape_to_disk_fb_retain.ec** writes files to disk using fixed block format.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("tape_to_disk_fb_-retain",tape number,file,file number)
Arguments:
tape_number–Six-position volume number
file–Name of the file
file_number–Number of the file on tape
Subroutines called: None
Common data referenced: None
Input files: User's tape of the State files
Output files: Any one or all of the following: *bginNM*, *coorNM*, *cordNM*, *statNM*, *strdNM*, *counNM*, *curdNM*, *gridNM*, and *bordNM*
Arrays used: None
Called by: **state_pull_off**, **separate_pull_off**
Error checking and reporting: None
Constants: None
Program logic:
1. Using the COPY_FILE command and the I/O module, **tape_ibm_**, the program writes the selected file to disk, as determined by *file* and *file_number*.
2. Control is returned to the calling module.

```
tape-to-disk-fb-retain.ec
cpf -ods "record_stream_ -target vfile_ &2" -ids "tape_ibm_ &1 -nb &3
-nm &2    -fmt fb -rec 80 -bk 8000  -den 800  -ret all "
```

EXEC_COM NAME: TAPE_TO_DISK_VBS_RETAIN.EC

Author: Harold Johnson
Purpose of the program: **tape_to_disk_vbs_retain.ec** writes files to disk using spanned record format.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("tape_to_disk_vbs_-retain",tape_number,file,file_number)
Arguments:
tape_number–Six-position volume number
file–Name of the input file
file_number–Number of the file on tape
Subroutines called: None

Common data referenced: None
Input files: User's tape of State files
Output files: cntrNM, paraNM, redyNM
Arrays used: None
Called by: **state_pull_off, separate_pull_off**
Error checking and reporting: None
Constants: None
Program logic:
1. Using the COPY_FILE command and the I/O module, **tape_ibm_**, program writes *cntrNM, paraNM*, and *redyNM* to disk if specified by *file* and *file_number*. The output description specifies no new line.
2. Control is returned to the calling module.

```
tape-to-disk-vbs-retain.ec
copy_file -ids "tape_ibm_ &1 -nb &3 -nm &2 -fmt fb -rec 100 -bk 8000
-den 800 -ret all" -ods "record_stream_ -nnl -target vfile_ &2"
```

PROGRAM NAME: BACKUP

Author: Harold Johnson
Purpose of the program: **backup** enables the user to dump various segments and (or) whole directories to a tape. They can have any file characteristic such as ASCII, binary, or whatever happens to be in the directory to be dumped.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: backup
Arguments: None
Subroutines called: **backup.ec**
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: None
Error checking and reporting: None
Constants: None
Program logic:
1. The program calls **backup.ec**.

```
c PROGRAM backup
c
c PURPOSE: TO ENABLE A USER TO DUMP VARIOUS SEGMENTS AND WHOLE
c    DIRECTORIES TO A TAPE
c
c PROGRAMMER: H Johnson
c DATE: Dec 29, 1977
c
   call ec ("backup")
c
c THIS IS THE EXEC COMMAND:
c
c backup1
c io close file10
c io detach file10
c backup2
c
c
end
```

EXEC_COM NAME: BACKUP.EC

Author: Harold Johnson
Purpose of the program: **backup.ec** calls two subroutines and then closes and detaches the file.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("backup")
Arguments: None
Subroutines called: **backup1, backup2**
Common data referenced: None
Input files: None

Output files: None
Arrays used: None
Called by: **backup**
Error checking and reporting: None
Constants: None
Program logic:
1. This exec_com executes **backup1**.
2. Control is then returned to the exec_com and *file10* is closed and detached.
3. It then executes **backup2**.

```
backup.ec
backup1
io close file10
io detach file10
backup2
```

EXEC_COM NAME: DUMP.EC

Author: Harold Johnson
Purpose of the program: **dump.ec** places segments or directories, specified in **backup1**, on tape.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call ec ("dump","control.dump","hhj",-tape)
Arguments:
control.dump—Dump control file
hhj—Operator
tape—Tape number
Subroutines called: None
Common data referenced: None
Input files: **control.dump**
Output files:
**.dump.map*
possibly an error file ending with *.ef*
Arrays used: None
Called by: **backup2**
Error checking and reporting: None
Constants: None
Program logic:
1. *&ATTACH* allows the arguments of the exec_com to be passed directly to **complete_dump**.
2. **complete_dump** requires a minimum of three arguments as in the command: **complete_dump:** *control.dump*, *hhj-debug*, where *control.dump* is the name of a control segment, *hhj* represents the author's initials, and *-debug* disables calls to highly privileged system subroutines normally used when **complete_dump** is used by the operators during the weeky system backup session. The argument *tape* is the volume identifier of the desired dump tape. One 2,400 ft. tape at 1,600 bpi can hold approximatley 7,500 disk pages (records).
3. The segments or directories stored in *control.dump* are written to tape.
4. Quit.

```
dump.ec
&attach
complete_dump &1 &2 -debug
&3
&quit
```

SUBROUTINE NAME: BACKUP1

Author: Harold Johnson
Purpose of the program: **backup1** allows a user to dump various files and directories to a tape.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call backup1
Arguments: None
Subroutines called: None
Common data referenced: None
Input files: None
Output files: *control.dump* used on unit 10 (*file10*)
Arrays used: None
Called by: **backup.ec**
Error checking and reporting: The user will be asked whether he has sent a message to the operator to locate his tape. If the user answers with other than 1, the program will abort.
Constants: None
Program logic:
1. The user is asked whether a message was sent to locate his tape. If the user responds with 1, processing continues. Otherwise the program halts.
2. The program prints a message describing its purpose.
3. *file10* is attached to control dump and opened for stream output.

4. The user is informed that segments or directories to be dumped must be absolute path names and in alphabetical order.
5. Prompt:
   NOW TYPE IN THE ABSOLUTE PATH NAME OF THE NEXT SEGMENT OR DIRECTORY YOU WANT TO BACKUP.

   Type its absolute path name.
6. The user's response is read from the terminal and written to *file10*.
7. Prompt:
   IF YOU WANT TO DUMP MORE PATHS, TYPE 1; OTHERWISE, 0
8. The user's response is read into *ians*.
9. If *ians* is equal to 1, go to 5.
10. Stop.

```
c PROGRAM  backup1
c
c PURPOSE: TO ALLOW A USER TO DUMP VARIOUS FILES AND DIRECTORIES TO
c     A TAPE.
c
c PROGRAMMER: H Johnson
c DATE: Dec. 29, 1977
c
  character path*60
c
write(6,900)
900 format(" DID YOU SEND A MESSAGE TO THE OPERATOR TO "
"FIND YOUR TAPE?"/" IF YOU DID, TYPE A 1 ")
    read(5,920) ians
if(ians .ne. 1) stop
c
    write(6,910)
910 format("0THIS PROGRAM ENABLES YOU TO DUMP ONE OR MORE SEGMENTS"/
" TO YOUR TAPE. YOU CAN EVEN DUMP WHOLE DIRECTORIES. "/
" THE PROGRAM CREATES A FILE NAMED 'CONTROL.DUMP'")
c
920 format(i1)
c
   call io ("attach","file10","vfile_","control.dump")
   call io ("open","file10","so")
c
c
    write(6,945)
945 format("0YOU MUST TYPE IN ALL THE ABSOLUTE PATH NAMES OF"/
" THE SEGMENTS OR DIRECTORIES YOU WANT TO DUMP TO TAPE."/
"0*** THESE MUST BE ENTERED IN ALPHABETICAL ORDER !******")
10  write(6,950)
950 format("0      NOW TYPE IN THE ABSOLUTE PATH NAME OF"/
" THE NEXT SEGMENT OR DIRECTORY YOU WANT TO BACKUP."/
"0TYPE ITS ABSOLUTE PATH NAME :")
   read(5,960) path
960 format(a60)
c
    write(6,970)
970 format("0IF YOU WANT TO DUMP MORE PATHS, TYPE 1; OTHERWISE, 0")
   read(5,920) ians
    write(10,960) path
if(ians .eq. 1) go to 10
c
c
end
```

SUBROUTINE NAME: BACKUP2

Author: Harold Johnson
Purpose of the program: **backup2** allows the user to dump various files and directories to a tape.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call backup2
Arguments: None
Subroutines called: **dump.ec**
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **backup.ec**
Error checking and reporting: None
Constants: None
Program logic:
1. Prompt:
   TYPE YOUR TAPE NUMBER, FORMAT A6
2. The user's response is read into tape.
3. The program calls **dump.ec**.
4. The user receives a message that 1 or 2 message files have been added to her directory and will automatically be dprinted. These should be picked up and saved; any old ***dump.maps*** for this tape should be discarded because they are obsolete.
5. Stop.

```
c PROGRAM  backup2
c
c PURPOSE: TO ALLOW A USER TO DUMP VARIOUS FILES AND DIRECTORIES TO
c     A TAPE.
c
c PROGRAMMER: H Johnson
c DATE: Dec. 29, 1977
c
  character tape*6
c
c
   write(6,910)
910 format("0TYPE YOUR TAPE NUMBER, FORMAT A6")
c
c
    read(5,940)tape
940 format(a6)
c
c
   call ec ("dump","control.dump","hhj",tape)
c
c THIS IS THE EXEC COM BEING CALLED:
c &attach
c complete_dump &1 &2 -debug
c &3
c &quit
c
50  write(6,980)
980 format("0THIS ROUTINE ADDS 1 OR 2 MESSAGE FILES TO "/
" YOUR DIRECTORY WHICH ARE AUTOMATICALLY DPRINTED."/
" THEY ARE VERY IMPORTANT AND SHOULD BE PICKED UP AND SAVED"/
" IN A SAFE PLACE. "/
"  THEY ARE THE DUMP.MAP AND POSSIBLE ERROR MESSAGE."/
"0SAVE THEM IN A SAFE PLACE. THROW AWAY ANY OLD DUMP.MAPS"/
" FOR THIS TAPE, SINCE THEY ARE COMPLETELY OBSOLETE.")
c
c
end
```

PROGRAM NAME: RESTORE

Author: Harold Johnson
Purpose of the program: **restore** allows the user to restore files that she has previously dumped to tape using **complete_dump** or **backup**.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: restore
Arguments: None
Subroutines called: **retrieve.ec**
Common data referenced: None
Input files: User's backup tape
Output files: ***control.retrieve*** used on unit 10 (***file10***)
Arrays used: None
Called by: None
Error checking and reporting: The user is asked whether she sent a message to the operator asking her to locate the tape. If the user responds with other than 1, the program aborts.
Constants: None
Program logic:

1. ***file10*** is attached and opened for sequential output.
2. Prompt:
   DID YOU SEND A MESSAGE TO THE OPERATOR TO FIND YOUR BACKUP TAPE? IF YOU DID, TYPE A 1.
3. The program prints a message describing its purpose.
4. Prompt:
   NOW TYPE THE ABSOLUTE PATH NAME OF THE NEXT SEGMENT OR DIRECTORY YOU WANT TO RESTORE WHICH IS ON YOUR BACKUP TAPE. USE ITS ABSOLUTE PATH NAME.
5. The response is read from the terminal and then written to ***file10***. You can request as many as 50 absolute path names.
6. Prompt:
   IF YOU WANT TO RESTORE MORE PATHS, TYPE 1; OTHERWISE 0
7. If ***ians*** is equal to 1, go to step 4.
8. Close and detach ***file10***.
9. Prompt:
   TYPE THE NUMBER OF YOUR BACKUP TAPE, FORMAT A6
   Read the response into ***tape***.
10. Call **ec ("retrieve","control.retrieve",tape)**.
11. Message to the user:
    THIS ROUTINE AUTOMATICALLY DPRINTS A 'RETRIEVE' MAP WHICH YOU SHOULD OBTAIN. CHECK THAT THE REQUESTED FILES ARE IN YOUR DIRECTORY.
12. Stop.

```
c restore program
c
c PURPOSE: To allow a user to restore files which he has
c   previously dumped to a tape using compete_dump or  backup
c
c PROGRAMMER: H Johnson
c DATE: Jan 6, 1978
c
   character tape*6, path*60
c
  call io ("attach","file10","vfile_","control.retrieve")
  call io ("open","file10","so")
c
    write(6,910)
910 format("0DID YOU SEND A MESSAGE TO THE OPERATOR TO"/
" FIND YOUR BACKUP TAPE? IF YOU DID, TYPE A 1")
   read(5,920) ians
920 format(i1)
c
    write(6,930)
930 format("0THIS PROGRAM ENABLES YOU TO RETRIEVE ONE OR MORE"/
" FILES FROM A BACKUP TAPE, WHICH WAS PROCESSED USING complete_dump"/
" OR backup. "/
"0YOU MUST KNOW THE COMPLETE PATH NAMES OF SEGMENTS YOU WANT"/
" TO RESTORE. *** THESE MUST BE TYPED IN IN ALPHABETICAL ORDER ***")
```

```
c
10  write(6,940)
940 format("ONOW TYPE THE ABSOLUTE PATH NAME OF THE NEXT"/
" SEGMENT OR DIRECTORY YOU WANT TO RESTORE WHICH IS ON"/
" YOUR BACKUP TAPE."/
" USE ITS ABSOLUTE PATH NAME :")
   read(5,950) path
950 format(a60)
c
   write(10,950) path
c
    write(6,970)
970 format("OIF YOU WANT TO RESTORE MORE PATHS, TYPE 1; OTHERWISE, 0")
    read(5,920) ians
if(ians .eq. 1) go to 10
c
   endfile 10
c THIS IS A MICKEY MOUSE STATEMENT TO GET A NL CHARACTER ON
c THE END OF THE LAST RECORD.
c
  call io ("close","file10")
   call io ("detach","file10")
c
   write(6,980)
980 format("OTYPE THE NUMBER OF YOUR BACKUP TAPE, FORMAT A6")
c
    read(5,990) tape
990 format(a6)
c
   call ec ("retrieve","control.retrieve",tape)
c
c THIS IS THE FOLLOWING EXEC COMMAND:
c
c $attach
c retrieve &1 -debug
c &2
c no
c dprint -dl &1.retrieve.map
c &quit
c
    write(6,9910)
9910 format("OTHIS ROUTINE AUTOMATICALLY DPRINTS A 'RETRIEVE"/
" MAP' WHICH YOU SHOULD OBTAIN. CHECK THAT THE REQUESTED FILES"/
" ARE IN YOUR DIRECTORY.")
c
stop
end
```

EXEC_COM NAME: RETRIEVE.EC

Author: Harold Johnson
Purpose of the program: **retrieve.ec** consists of a Multics command that retrieves the path names given by the user from the tape specified by the user.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: ec ("retrieve","control.retrieve",tape)
Arguments:
control.retrieve– File of path names
tape– Tape number

Subroutines called: None
Common data referenced: None
Input files: *control.retrieve*
Output files: *control.retrieve.map*
Arrays used: None
Called by: **restore**
Error checking and reporting: None
Constants: None
Program logic:
1. The file is attached.
2. The Multics command, **retrieve**, is executed.
3. The path names listed on the file *control.retrieve* are restored to the user's directory from the tape.
4. The **retrieve** command asks whether more tapes are to be reloaded, and the exec_com gives an automatic NO response.
5. The *control.retrieve.map* is automatically dprinted and deleted.
6. Quit.

```
retrieve.ec
&attach
retrieve &1 -debug
&2
no
dprint -dl &1.retrieve.map
&quit
```

PROGRAM NAME: VERPLOT

Author: Lawrence Balcerak
Purpose of the program: **verplot** generates the System Status Map for the Geoindex system. This program reads a file of commands and creates a Versatec plot file using the instructions from that file.
Data base: Geoindex
Computer: Honeywell series 60 (level 68)
Operating system: Multics
Calling sequence: verplot
Arguments: None
Subroutines called: **change_origin, change_symbol, change_width, closef, newpattern, openf, pattern, pattern_verplot, plotfile, plotlegend, plotoutline, scaleplot, cf, dprint, io_call, ioa_$nnl, plot, plots, setup_versaplot**
Common data referenced: *icom(80), in1*
Input files: *init_vals* used on unit 15 (*file15*) (initial value file for the command file)
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *iwhat(16,7)*
Called by: None
Error checking and reporting: The program checks for invalid commands. Any invalid command causes an error message to be written along with the erroneous command line.
Constants:
in1 = 15 (input reference number for the command file)
in2 = 16 (input reference number for any plotting file used)
Program logic:
1. Set initial values:
   iwhat(16,7) (contains the names of the eight different kinds of commands in both uppercase and lowercase)
   lsym = 2 (the number of the default symbol to be used for plotting single points)
   name = " " (blank space)
2. The program sends a message to the terminal:
   WHAT IS THE NAME OF YOUR COMMAND FILE??
   USE NO MORE THAN SIX CHARACTERS!
   Read the character string sent back into the variable name.
3. Call **openf** to attach and open the command file for input. Use *in1* as the reference number.
   Call **openf** to attach and open *temp10* for output. Use reference number 10.
   Call **io_call** to attach the initial values file to the name *init_vals*.
4. Call **setup_versatec**. This links to the Versatec software.
5. Call **pattern**.
   Call **pattern_verplot.**
   These subroutines initialize the arrays for the 13 shading patterns used.
6. Call **plots** to initialize the Versatec routines.
   Call **plot** to position the software origin.
7. Place blanks into the input line *icom(80)*.
   Read a command line into *icom(80)*.
   If EOF, go to step 27.
8. Find the semicolon in the input line and set *ll* equal to this position.
   If one is found, go to step 10. All commands must have a semicolon in the first eight positions.
9. The program prints the error message:
   THIS LINE CANNOT BE IDENTIFIED AS A COMMAND.

along with the erroneous line.
Go back to step 7 to read another command.

10. If *ll* is equal to 1, 2, 3, or 4, go to step 9, because there are no commands with a semicolon in these positions.
Depending on the value of *ll*, go to step 11 (*ll* = 15), step 13 (*ll* = 6), step 17 (*ll* = 7), or step 23 (*ll* = 8).
11. If the command is not *PLOT* or *plot*, go to step 9. This is the only four-letter command.
12. Call **plotfile**.
Go to step 7 to read another command line.
13. If the command is not *SCALE* or *scale*, go to step 15.
14. Call **scaleplot**.
Go to step 7 to read another command line.
15. If the command is not *REORG* or *reorg*, go to step 9.
16. Call **change_origin**.
Go to step 7 to read another command line.
17. If the command is not *LEGEND* or *legend*, go to step 19.
18. Call **plotlegend**.
Go to step 7 to read another command line.
19. If the command is not *SYMBOL* or *symbol*, go to step 21.
20. Call **change_symbol**.
Go to step 7 to read another command line.
21. If the command is not *LINWID* or *linwid*, go to step 9.
22. Call **change_width**.
Go to step 7 to read another command line.
23. If the command is not *OUTLINE* or *outline*, go to step 25.
24. Call **plotoutline**.
Go to step 7 to read another command line.
25. If the command is not *PATTERN* or *pattern*, go to step 9.
26. Call **newpattern**.
Go to step 7 to read another command line.
27. The program sends a message to the terminal:
PLOT FINISHED
28. Call **plot** (0.,0.,999) (frame finished).
Call **plot** (0.,0.,−999) (all plotting finished).
29. Call **setup_versaplot** ("-reset") (unlinks from the Versatec software).
30. Call **closef** to close and detach from files *in1* and *file10*.
Call **io_call** to detach the initial values file.
31. Dprint the message file with the delete option.
32. Call **cf** to close all files.

```
c         PROGRAM VERPLOT
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character name*o,fmt*16,icom*1,iwhat*1(16,7),init*10,id*1
      external io_call (descriptors),ioa_$nnl (descriptors),
               & cf (descriptors),setup_versaplot (descriptors),
               & dprint (descriptors)
      data ((iwhat(i,j),j=1,7),i=1,16)
           &/"S","C","A","L","E"," "," ",
           & "L","E","G","E","N","D"," ",
           & "O","U","T","L","I","N","E",
           & "P","A","T","T","E","R","N",
           & "S","Y","M","B","O","L"," ",
           & "L","I","N","W","I","D"," ",
           & "P","L","O","T"," "," "," ",
           & "R","E","O","R","G"," "," ",
           & "s","c","a","l","e"," "," ",
           & "l","e","g","e","n","d"," ",
           & "o","u","t","l","i","n","e",
           & "p","a","t","t","e","r","n",
           & "s","y","m","b","o","l"," ",
           & "l","i","n","w","i","d"," ",
           & "p","l","o","t"," "," "," ",
           & "r","e","o","r","g"," "," "/
      in1=15
      in2=16
      isym=2
      name=" "
```

```
c
      call ioa_$nnl ("^/WHAT IS THE NAME OF YOUR COMMAND FILE??")
      call ioa_$nnl ("^/USE NO MORE THEN SIX CHARACTERS!        ")
      read 10, name
   10 format (a6)
      call openf (in1,name,"si  ")
      call openf (10,"temp10","so  ")
      call io_call ("attach","init_vals","vfile_ ","init_vals")
      call setup_versaplot
      call pattern
      call pattern_verplot
      call plots(0,0,0)
      call plot (2.0,0.005,-3)
c
   25 do 30 i=1,80
   30 icom(i)=" "
      read (in1,40,end=260)  icom
   40 format (80a1)
      do 50 ll=1,8
      if (icom(ll) .eq. ";")  go to 70
   50 continue
   55 write (10,60)  icom
   60 format (" THIS LINE CANNOT BE IDENTIFIED AS A COMMAND.",/,1x,80a1)
      go to 25
   70 go to (55,55,55,55,80,100,140,220),ll
c
   80 do 90 i=1,4
   90 if ((icom(i) .ne. iwhat(7,i))  .and.
     &    (icom(i) .ne. iwhat(15,i)))  go to 55
      call plotfile
      go to 25
c
  100 do 110 i=1,5
  110 if ((icom(i) .ne. iwhat(1,i))  .and.
     &    (icom(i) .ne. iwhat(9,i)))  go to 120
      call scaleplot
      go to 25
  120 do 130  i=1,5
  130 if ((icom(i) .ne. iwhat(8,i))  .and.
     &    (icom(i) .ne. iwhat(16,i)))  go to 55
      call change_origin
      go to 25
c
  140 do 150  i=1,6
  150 if ((icom(i) .ne. iwhat(2,i))  .and.
     &    (icom(i) .ne. iwhat(10,i)))  go to 160
      call plotlegend
      go to 25
  160 do 170  i=1,6
  170 if ((icom(i) .ne. iwhat(5,i))  .and.
     &    (icom(i) .ne. iwhat(13,i)))  go to 200
      call change_symbol
      go to 25
  200 do 210  i=1,6
```

```
 210   if ((icom(i) .ne. iwhat(6,i))  .and.
     &     (icom(i) .ne. iwhat(14,i)))  go to 55
       call change_width
       go to 25
c
 220   do 230 i=1,7
 230   if ((icom(i) .ne. iwhat(3,i))  .and.
     &     (icom(i) .ne. iwhat(11,i)))  go to 240
       call plotoutline
       go to 25
 240   do 250 i=1,7
 250   if ((icom(i) .ne. iwhat(4,i))  .and.
     &     (icom(i) .ne. iwhat(12,i)))  go to 55
       call newpattern
       go to 25
c
 260   call ioa_$nnl ("^/PLOT FINISHED")
       call plot (0.,0.,999)
       call plot (0.,0.,-999)
       call setup_versaplot ("-reset")
       call closef (in1)
       call closef (10)
       call io_call ("detach","init_vals")
       call dprint ("-dl","temp10")
       call cf ("-all")
       stop
       end
```

SUBROUTINE NAME: CHANGE_ORIGIN

Author: Lawrence Balcerak
Purpose of the program: **change_origin** identifies and evaluates the *x* and *y* distances for a change in the origin and moves the origin that distance.
Data base: Geoindex
Computer: Honeywell series 60 (level 68)
Operating system: Multics
Calling sequence: call change_origin
Arguments: None
Subroutines called: **find_number**, **plot**
Common data referenced: *icom(80)*, *id(15)*
Input files: None
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *icom(80)*, *id(15)*
Called by: **verplot**
Error checking and reporting: The subroutine checks for a blank keyword field, an error in a keyword, and an error in a data value. If any are found, an appropriate message is written to a file, which is dprinted at the end of the run.
Constants: None
Program logic:

1. Set initial values. The keywords for *x* and *y* are not necessarily both present. The default values for both are 0. The first position to look for a keyword (*iplace*) is 7, because the command takes up 6 spaces. The last nonblank character of the record is in position *ilast*, which is initialized to 81 to start.
2. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 4.
3. The subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS
   and returns to the calling program.
4. Find the first nonblank character in this field and set *iplace* equal to this position.
5. If *ic* = *iplace* to *ilast*, then find the end of the next keyword by looking for a comma.
   Set *ic* equal to one less than the position of the comma or to *ilast* if none exists. This is the position of the last character in this field.
6. If the keyword is *x*= or *X*= go to step 11.
7. If the keyword is *y*= or *Y*= go to step 12.
8. The subroutine prints the error message:
   THIS FIELD IS NOT RECOGNIZED AS A KEYWORD!!
   and the string involved.
9. Set *iplace* = *ic* + 2, which is the first possible position for the next keyword.

10. If there are more characters to check (compare the present position, *iplace*, with the last possible position, *ilast*), go to step 4.
    Otherwise, go to step 20.
11. Set the switch *key* equal to 1.
    Go to step 13.
12. Set the switch *key* equal to 2.
13. Add 2 to *iplace*. This is the first possible position for the data string.
14. Check for a nonexistent data string. If data string is nonexistent, go back to step 9.
15. Find the first nonblank character in this data string, and if found go to step 16.
    If the whole string is blank, set *rnum* = 0, and go to step 19.
16. Set *num* equal to the number of characters in the string.
    If *num* is greater than 15, go to step 18.
17. Place the character string in the array *id*.
    Call **find_number** to translate the string into the real number *rnum*.
    If the error return code, *istat*, equals 0, go to step 19.
18. The subroutine prints the error message:
    THIS STRING HAS AN UNRECOGNIZABLE CHARACTER
    and the string involved. Then it returns to the calling program.
19. Depending upon the value of key, set *xx* (*key* = 1) or *yy* (*key* = 2) equal to *rnum*.
    Go back to step 10 to check for more data.
20. Call **plot** to change the origin.
21. The subroutine prints the message:
    THE ORIGIN HAS BEEN MOVED BY X = *nnn*
    Y = *nnn*

Return

```
      subroutine change_origin
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1
c
      xx=0.
      yy=0.
      iplace=7
      ilast=81
c
c    FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
      do 10 i=iplace,80
      ilast=ilast-1
   10 if (icom(ilast) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS",/,1x,80a1)
      go to 240
c    FIND THE FIRST NON-BLANK CHARACTER IN THIS FIELD
   30 do 40 m=iplace,ilast
   40 if (icom(m) .ne. " ")  go to 50
      m=ilast
   50 iplace=m
c    FIND THE LAST POSITION FOR THIS FIELD
      do 60 ic=iplace,ilast
   60 if (icom(ic) .eq. ",")  go to 70
      ic=ilast
      go to 80
   70 ic=ic-1
c    IDENTIFY THE KEYWORD
   80 if (icom(iplace+1) .ne. "=")  go to 90
      if ((icom(iplace)   .eq. "X")  .or.
     &    (icom(iplace)   .eq. "x"))  go to 110
      if ((icom(iplace)   .eq. "Y")  .or.
     &    (icom(iplace)   .eq. "y"))  go to 120
   90 write (10,100) (icom(i),i=iplace,ic)
  100 format (" THIS FIELD IS NOT RECOGNIZED AS A KEYWORD!!",/,1x,74a1)
```

```
c     FIND THE FIRST POSTION IN THE NEXT FIELD
 105   iplace=ic+2
       if (ilast-iplace)  220,220,30
 110   key=1
       go to 130
 120   key=2
 130   iplace=iplace+2
       if (ic-iplace)  105,135,135
c     FIND THE FIRST NON-BLANK CHARACTER FOR THIS DATA FIELD
 135   do 140  num=iplace,ic
 140   if (icom(num) .ne. " ")  go to 150
       rnum=0.
       go to 190
 150   iplace=num
       num=ic-iplace+1
       if (num .gt. 15)  go to 165
       l=0
       do 160  i=iplace,ic
       l=l+1
 160   id(l)=icom(i)
       call find_number (id,num,rnum,istat)
       if (istat .eq. 0)  go to 190
 165   write (10,170) (icom(i),i=iplace,ic)
 170   format (" THIS STRING HAS AN UNRECOGNIZABLE CHARACTER",/,1x,74a1)
       go to 240
c
 190   go to (200,210),key
 200   xx=rnum
       go to 105
 210   yy=rnum
       go to 105
c
 220   call plot (xx,yy,-3)
       write (10,230) xx,yy
 230   format (" THE ORIGIN HAS BEEN MOVED BY X=",f10.3,/,30x,"Y=",f10.3)
 240   return
       end
```

SUBROUTINE NAME: CHANGE_WIDTH

Author: Lawrence Balcerak
Purpose of the program: **change_width** evaluates the data string given and changes the line dot width to that value.
Data base: Geoindex
Computer: Honeywell series 60 (level 68)
Operating system: Multics
Calling sequence: call change_width
Arguments: None
Subroutines called: **find_number, newpen**
Common data referenced: *icom(80), id(15)*
Input files: None
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *icom(80), id(15)*
Called by: **verplot**
Error checking and reporting: The subroutine checks for a blank data field and an error in a data value. If either are found, an appropriate message is written to a file, which is dprinted at the end of the run.
Constants: None
Program logic:

1. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 3.
2. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS!
   and goes to step 7.
3. Find *m*, the first nonblank character in the field.
4. Set *inum* equal to the number of characters in the string. If *inum* is greater than 15, go to step 6.

5. Place the character string in the array *id*.
   Call **find_number** to translate the string into the real number *rnum*.
   If the error return code, *istat*, equals 0, go to step 8.
6. Subroutine prints the error message:
   THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!
7. Subroutine prints the message:
   THE DEFAULT VALUE WILL BE USED!!
   Set *rnum* = 1, which is the default line width.
8. Change the real number *rnum* to the integer *ivis*.
9. Call **newpen** to change the line width to *ivis*.
10. Subroutine prints the message:
    THE LINE WIDTH HAS BEEN CHANGED TO *nn* DOTS WIDE

Return

```
      subroutine change_width
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1
c
c     FIND THE LAST NON-BLANK CHARACTER ON THE RECORD
      do 10 j=8,80
      num=88-j
   10 if (icom(num) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS!  ",/,1x,80a1)
      go to 75
c     FIND THE FIRST NON-BLANK CHARACTER IN THE DATA FIELD
   30 do 40  m=8,num
   40 if (icom(m) .ne. " ")  go to 50
      m=num
   50 inum=num-m+1
      if (inum .gt. 15)  go to 75
      l=0
      do 60  i=m,num
      l=l+1
   60 id(l)=icom(i)
      call find_number (id,inum,rnum,istat)
      if (istat .eq. 0)  go to 90
      write (10,70)  (icom(i),i=8,num)
   70 format (" THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!",/,1x,73a1)
   75 write (10,80)
   80 format(" THE DEFAULT VALUE WILL BE USED!!")
      rnum=1.
   90 ivis=rnum+0.5
      call newpen (ivis)
      write (10,100) ivis
  100 format (" THE LINE WIDTH HAS CHANGED TO ",i2," dots wide")
      return
      end
```

SUBROUTINE NAME: CHANGE_SYMBOL

Author: Lawrence Balcerak
Purpose of the program: **change_symbol** evaluates the data string given and changes the number of the symbol (used in plotting all single points) to that value.
Data base: Geoindex
Computer: Honeywell series 60 (level 68)
Operating system: Multics
Calling sequence: call change_symbol
Arguments: None
Subroutines called: **find_number**
Common data referenced: *icom(80)*, *id(15)*, *isym*
Input files: None
Output files: *temp10* used on unit 10 (*file10*) (for messages)

Arrays used: **icom(80)**, **id(15)**
Called by: **verplot**
Error checking and reporting: The subroutine checks for a blank data field and an error in the data value. If either are found, an appropriate message is written to a file, which is dprinted at the end of the run.
Constants: None
Program logic:

1. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 3.
2. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS!
   and goes to step 7.
3. Find *m*, the first nonblank character in this field.
4. Set *inum* equal to the number of characters in the string. If *inum* is greater than 15, go to step 18.
5. Place the character string in the array *id*.
   Call **find_number** to translate the string into a real number *rnum*.
   If the error return code, *istat*, equals 0, go to step 8.
6. Subroutine prints the error message:
   THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!
7. Subroutine prints the message:
   THE DEFAULT VALUE WILL BE USED!!
   Set *rnum* = 2, which is the number of the default symbol.
8. Change the real number *rnum* to the integer *isym*.
9. Subroutine prints the message:
   THE SYMBOL NUMBER HAS BEEN CHANGED TO *nn*

Return

```
      subroutine change_symbol
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1
c
c     FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
      do 10 j=8,80
      num=88-j
   10 if (icom(num) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS!  ",/,1x,80a1)
      go to 75
c     FIND THE FIRST NON-BLANK CHARACTER IN THE DATA FIELD
   30 do 40  m=8,num
   40 if (icom(m) .ne. " ")  go to 50
      m=num
   50 inum=num-m+1
      if (inum .gt. 15)  go to 75
      l=0
      do 60  i=m,num
      l=l+1
   60 id(l)=icom(i)
      call find_number (id,inum,rnum,istat)
      if (istat .eq. 0)  go to 90
      write (10,70)  (icom(i),i=8,num)
   70 format (" THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!",/,1x,73a1)
   75 write (10,80)
   80 format(" THE DEFAULT VALUE WILL BE USED!!")
      rnum=2.
   90 isym=rnum+0.5
      if ((isym .lt. 0) .or. (isym .gt. 127) )  isym=2.
      write (10,100)  isym
  100 format (" THE SYMBOL NUMBER HAS BEEN CHANGED TO ",i4)
      return
      end
```

SUBROUTINE NAME: SCALEPLOT

Author: Lawrence Balcerak
Purpose of the program: **scaleplot** evaluates the data string given and changes the scale to that value.
Data base: Geoindex
Computer: Honeywell series 60 (level 68)
Operating system: Multics
Calling sequence: call scaleplot
Arguments: None
Subroutines called: **find_number, factor**
Common data referenced: *icom*(*80*), *id*(*15*)
Input files: None
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *icom*(*80*), *id*(*15*)
Called by: **verplot**
Error checking and reporting: The subroutine checks for a blank data field and an error in the data value. If either are found, an appropriate message is written to a file, which is dprinted at the end of the run.
Constants: None

Program logic:
1. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 3.
2. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS!
   and goes to step 7.
3. Find *m*, the first nonblank character in the field.
4. Set *inum* equal to the number of characters in the string. If *inum* is greater than 15, go to step 6.
5. Place the character string in the array *id*.
   Call **find_number** to translate the string into the real number scale.
   If the error return code, *istat*, equals 0, go to step 8.
6. Subroutine prints the error message:
   THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!
7. Subroutine prints the message:
   THE DEFAULT VALUE WILL BE USED!!
   Set *scale* = 1.
8. Call factor to change the scale to the new value.
9. Subroutine prints the message:
   SCALE CHANGED TO *nnn*

Return

```
      subroutine scaleplot
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1
c
c     FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
      do 10 j=7,80
      num=87-j
   10 if (icom(num) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS!  ",/,1x,80a1)
      go to 75
c     FIND THE FIRST NON-BLANK CHARACTER IN THE DATA FIELD
   30 do 40  m=7,num
   40 if (icom(m) .ne. " ")  go to 50
      m=num
   50 inum=num-m+1
      if (inum .gt. 15)  go to 75
      l=0
      do 60  i=m,num
      l=l+1
   60 id(l)=icom(i)
      call find_number (id,inum,scale,istat)
      if (istat .eq. 0)  go to 90
      write (10,70)  (icom(i),i=7,num)
   70 format (" THIS STRING HAS AN UNRECOGNIZABLE CHARACTER!!",/,1x,74a1)
   75 write (10,80)
   80 format(" THE DEFAULT VALUE WILL BE USED!!")
      scale=1.
```

```
c
   90   call factor (scale)
        write (10,100)  scale
  100   format ( " SCALE CHANGED TO ",f10.3)
        return
        end
```

SUBROUTINE NAME: PATTERN_VERPLOT

Author: Lawrence Balcerak
Purpose of the program: **pattern_verplot** initializes some of the shading pattern arrays used in **verplot.**
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: pattern_verplot
Arguments: None
Subroutines called: None
Common data referenced: None
Input files: None
Output files: None
Arrays used: None
Called by: **verplot**
Error checking and reporting: None
Constants: None
Program logic:
1. The first three arrays are set to values whose bit patterns form the patterns needed. All other arrays are set to 0. These are used for the extra patterns that are defined by the user.

```
        subroutine pattern_verplot
c
        common /userpat/ ip11    ,ip12    ,ip13(4) ,ip14(16),ip15(16),
                       & ip16(16),ip17(16),ip18(16),ip19(16),ip20(16)
c
        do 10 i=1,4
        ip13(i)=0
   10   continue
c
        do 20 i=1,16
        ip14(i)=0
        ip15(i)=0
        ip16(i)=0
        ip17(i)=0
        ip18(i)=0
        ip19(i)=0
        ip20(i)=0
   20   continue
c
        ip11= 4*16**8 +2*16**6 +16**4 +8*16
c
        ip12= ip11 +ip11/16
c
        ip13(1)= 7*16**8 +11*16**6 +13*16**4 +14*16**2 +
              & 15*(16**7 +16**5 +16**3 +16 +1)
        ip13(2)=ip12
        ip13(3)=ip13(2)
        ip13(4)=ip13(2)
c
        return
        end
```

SUBROUTINE NAME: INTERPRET_DATA

Author: Lawrence Balcerak

Purpose of the program: **interpret_data** interprets numeric data strings on a record and returns the total number of data values on the record and the numeric value of each.

Data base: Geoindex

Computer: Honeywell series 60 (level 68)

Operating system: Multics

Calling sequence: call interpret_data (kind, numback,realval)

Arguments:

kind– Indicates type of data to be read on the cards

numback– Number passed back to the calling program

realval– The real value of the number

Subroutines called: **find_number, find_octal_number**

Common data referenced: *icom(80), id(15)*

Input files: None

Output files: None

Arrays used: *realval(20)*

Called by: **newpattern**

Error checking and reporting: The subroutine checks for blank data fields, empty data fields, data fields too long, and errors in a data field. If such data fields are found, the number for that field is set to 0 and the program continues.

Constants: None

Program logic:

1. Set initial values.
   key = 1 (branching flag to show type of data); 1 indicates a real or integer value that is the default.
   numback = 0 (the number of values returned to calling program).
   iplace = 1 (position to start processing).
   ilast = 81 (the last nonblank position of the record).
2. If the data is in octal, *kind* = "O", set *key* = 2.
3. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 5.
4. Set *realval(1)* = 0.
   Return to calling program.
5. Find the first nonblank character in this field.
   Set *iplace* equal to this position.
6. Find the last position for this field. It will be just before the next comma or, if no commas are left, just before the last character, *ilast*.
7. Check for the possibility of two commas in succession, and if not found, go to step 8. Otherwise, go to step 15.
8. Set *num* equal to the number of characters in this field.
9. If *num* is greater than 15, go to step 15. This is an error and the default value will be used.
10. Place the character string in the array *id*.
11. If this is a real or integer number (*key* = 1), go to step 12.
    If this is an octal number (*key* = 2), go to step 13.
12. Call **find_number** to translate the character string into the real number *rnum*.
    Go to step 14.
13. Call **find_octal_number** to translate the character string into the integer *knum*.
    Change the integer *knum* into the real number *rnum*.
14. If the error return code, *istat*, is equal to 0, go to step 16. A nonzero value indicates some kind of error in translation.
15. Set *rnum* = 0, which is the default value when an error occurs.
16. Add 1 to *numback*, the number of data fields translated and also the index for the array *realval*. Store the number *rnum* in *realval*.
17. Set *iplace* = *ic* + 2, which is the next place past the comma (if there was one). If there is another data field, go to step 5 to continue.

Return.

```
      subroutine interpret_data (kind,numback,realval)
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1,kind*1
      dimension realval(20)
c
c    KIND INDICATES THE TYPE OF DATA TO BE READ ON THE CARDS
c      DEFAULT VALUE=1 -EITHER INTEGER OR REAL
      key=1
      numback=0
      iplace=1
      ilast=81
      if (kind .eq. "o")  key=2
c
c   FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
      do 10 i=iplace,80
      ilast=ilast-1
   10 if (icom(ilast) .ne. " ")  go to 20
      realval(1)=0.
      return
c
c    FIND THE FIRST NON-BLANK CHARACTER IN THIS FIELD
   20 do 30  m=iplace,ilast
   30 if (icom(m) .ne. " ")  go to 40
      m=ilast
   40 iplace=m
c
c    FIND THE LAST POSITION FOR THIS FIELD
      do 50  ic=iplace,ilast
   50 if (icom(ic) .eq. ",")  go to 60
      ic=ilast
      go to 70
   60 ic=ic-1
   70 if (ic-iplace) 140,90,90
c
   90 num=ic-iplace+1
      if (num .gt. 15)  go to 140
      l=0
      do 100 i=iplace,ic
      l=l+1
  100 id(l)=icom(i)
      go to (110,120),key
  110 call find_number (id,num,rnum,istat)
      go to 130
  120 call find_octal_number (id,num,knum,istat)
      rnum=knum
  130 if (istat .eq. 0)  go to 150
  140 rnum=0.
  150 numback=numback+1
      realval(numback)=rnum
      iplace=ic+2
      if (ilast-iplace) 160,160,20
  160 return
      end
```

SUBROUTINE NAME: NEWPATTERN

Author: Lawrence Balcerak

Purpose of the program: **newpattern** reads and interprets data values representing some shading pattern, which is then stored in an array and which can be accessed by the program at a later time.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call newpattern

Arguments: None

Subroutines called: **find_number, interpret_data**

Common data referenced: *icom(80)*, *id(15)*, *in1*; some entries in */pat/* and */userpat/*, depending on the shading patterns used

Input files: Command statements used on unit 15 (*file15*)

Output files: *temp10* used on unit 10 (*file10*) (for messages)

Arrays used: ***keyword(10,7)***, ***new(112)***, ***newval(16)***, ***realval(20)***

Called by: **verplot**

Error checking and reporting: The subroutine checks for a blank data field, an invalid keyword, data value error, invalid reference number, wrong number of words, missing keyword, too many data values, and not enough data values. If any such errors are found, an appropriate error message is printed with, sometimes, the character string involved.

Constants: None

Program logic:

1. Set ***new(1)*** equivalent to ***ip14(1)***. This causes the array ***new*** to overlay some entries of the common block ***/userpat/*** and thereby enables the program to access these entries by just changing the index for the array.
2. Set initial values. ***keyword(10,7)*** contains the different keywords possible. There are five keywords, and each is in both uppercase and lowercase.
   Set ***ir*** = 0. This is the flag for processing of the reference number keyword.
   Set ***in*** = 0. This is the flag for processing of the number of data points keyword.
   Set ***it*** = 0. This is the flag for processing of the type of data keyword.
   Set ***key*** = 0. This is the branching switch.
   Set ***iplace*** = 9. This is the first position that a keyword can start.
   Set ***ilast*** = 81. This is the position of the last nonblank character of the record.
3. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 5.
4. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS
   and returns to the calling program.
5. Find the first nonblank character in this field and set ***iplace*** equal to this position.
6. For ***ic*** = ***iplace*** to ***ilast***, find the end of the next keyword by looking for a comma.
   Set ***ic*** equal to one less than the position of the comma or to ***ilast*** if no comma is found. This is the position of the last character in this field.
7. Search the field for the character "=". If one is found, go to step 10.
   Any keyword must have this character.
8. Subroutine prints the error message:
   PATTERN: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD
   along with the erroneous field.
9. Set ***iplace*** = ***ic*** + 2, which is one position past the comma.
   If there are more characters to examine, go to step 5. Otherwise, go to step 33.
10. Set ***ll*** equal to the count of characters including the = of the keyword.
    If ***ll*** is greater than 8, go to step 8. No keyword has more than 8 characters.
11. If ***ll*** is equal to 1, 2, 3, 4, or 6, go to step 8. There is no such keyword.
    If ***ll*** is equal to 5, go to step 12.
    If ***ll*** is equal to 7, go to step 19.
    If ***ll*** is equal to 8, go to step 21.
12. To reach this step, the keyword contains 5 characters including the =.
    If the characters are not ***TYPE*** or ***type***, there is an error, so go to step 8.
13. Add 5 to ***iplace***, one character past the =.
    Check for a comma immediately following the = or for no more characters in the string.
    If either condition is found, go to step 9 to examine the next field.
14. Check for a blank data field. If found, go to step 9 to look at the next field. Otherwise, set ***iplace*** equal to the first nonblank character and to go to next step.
15. If the characters are not ***INTEGER*** or ***integer***, go to step 17.
16. Set ***it*** = 1 (flag for type keyword processed).
    Set ***kind*** = "i". This indicates integer data when calling **interpret_data** later.
    Go to step 9 to examine the next data field.
17. If the characters are not ***OCTAL*** or ***octal***, go to step 8. This is an error message of some sort.
18. Set ***it*** = 1. This is the flag for type keyword processed.
    Set ***kind*** = "O" (indicates octal data).
    Go to step 9 to examine the next field.
19. If the characters are not ***REFNUM*** or ***refnum***, go to step 8. This is an error of some kind.
20. Set ***key*** = 1. This is the branching switch used later to indicate the reference number.
    Add 7 to ***iplace*** (one postion past the =).
    Go to step 23 to interpret the number.
21. If the characters are not ***NUMWORD*** or ***numword***, go to step 8. This is an error of some sort.
22. Set ***key*** = 2, which is the branching switch used later to indicate the number of data points.
    Add 8 to ***iplace***, which is one position past the =.
23. Check for a comma immediately following the = character or for no more characters in the string. If either condition is true, go to step 9 to examine the next field.
24. Check for a blank data field. If found, go to step 9 to examine the next data field. Otherwise, set ***iplace*** equal to the first nonblank character and go to next step.

25. Set ***num*** equal to the number of characters in the field. If ***num*** is greater than 15, go to step 27. The field is too large.
26. Place the characters in the array ***id***.
    Call **find_numbers** to evaluate the data. The returned value is in ***rnum***.
    If the error return code, ***istat***, equals 0, go to step 28. A nonzero value indicates an error of some type when interpreting.
27. Subroutine prints the error message:
    PATTERN: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER
    along with the erroneous field.
    Go to step 9 to examine the next field.
28. Depending on the value of ***key***, go to step 29, ***key*** = 1, or step 31, ***key*** = 2.
29. Set ***numref*** = ***rnum***.
    Set ***ir*** = 1. The flag indicates that the reference number has been processed. If the reference number, ***numref***, is equal to 14 through 20, go to step 9 to examine the next field.
30. The reference number is invalid. The subroutine prints the error message:
    PATTERN: THE REFERENCE NUMBER MUST BE FROM 14 to 20
    and returns to the calling program.
31. Set ***numword*** = ***rnum***.
    Set ***in*** = 1. The flag indicates the number of data words that have been processed.
    If numword is equal to 1, 2, 4, 8, or 16, go to step 9 to examine the next field. These values are all even divisors of 16.
32. The subroutine prints the error message:
    PATTERN: THE NUMBER OF WORDS MUST BE 1,2,4,8 or 16
    and returns to the calling program. Note: At this point all keywords have been evaluated.
33. If all keywords have been processed, go to step 35.
34. Subroutine prints the error message:
    THIS PATTERN WILL NOT BE PROCESSED THERE IS SOME KEYWORD MISSING!!
    and returns to the calling program.
35. Set ***iplace*** = 1, which is the index for ***newval*** (always indicates the next value).
36. Read a data record into ***icom***. If EOF, go to step 41.
    Call **interpret_data** to evaluate all data fields on the new record.
37. If the number of values already processed plus the number just received from **interpret_data** is less than or equal to the total number of data values required, ***numword***, go to to step 39.
38. The subroutine prints the error message:
    PATTERN: THERE ARE TOO MANY DATA VALUES!!
    and returns to the calling program.
39. Place the returned values into the array ***newval*** using ***iplace*** as an index counter. Add ***numback*** to ***iplace***. It is now equal to the total number of data values interpreted.
40. If there are more data values to be interpreted, go to step 36 to read another record. If there are too many data values, go to step 38. If the number of values interpreted equals ***numword***, to to step 42.
41. Subroutine prints the error message:
    PATTERN: EOF REACHED WHEN TRYING TO READ ANOTHER RECORD!!
42. Place the new data values stored in ***newval*** into the common block by placing them into ***new***(***iplace***), where ***iplace*** now takes on values based on the reference number, ***numref***. There must be 16 values placed into new.
    If ***numword*** is less than 16, repeat the sequence of values until 16 values have been exchanged.
43. Subroutine prints the message:
    NEW PATTERN ASSIGNED TO REFERENCE NUMBER *nn*
44. Return.

```
      subroutine newpattern
c
      common /comand/ icom(80),id(15),isym,in1,in2
      common /pat/  ip1(16),ip2(16),ip3(4) ,ip4(4) , ip5(16),
     &  ip6(16),ip7(16),ip8(16),ip9(16),ip10(16)
      common /userpat/ ip11    ,ip12    ,ip13(4) ,ip14(16),ip15(16),
     &  ip16(16),ip17(16),ip18(16),ip19(16),ip20(16)
      character  icom*1,id*1,keyword*1(10,7),kind*1
      dimension new(112),newval(16),realval(20)
c
      equivalence (new(1),ip14(1))
c
      data ((keyword(i,j),j=1,7),i=1,10)
     &/"R","E","F","N","U","M"," ",
     & "N","U","M","W","O","R","D",
                         & "T","Y","P","E"," "," "," ",
                         & "I","N","T","E","G","E","R",
                         & "O","C","T","A","L"," "," ",
                         & "r","e","f","n","u","m"," ",
                         & "n","u","m","w","o","r","d",
                         & "t","y","p","e"," "," "," ",
                         & "i","n","t","e","g","e","r",
                         & "o","c","t","a","l"," "," "/
c
      ir=0
      in=0
      it=0
      key=0
      iplace=9
      ilast=81
c
c     FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
      do 10 i=iplace,80
      ilast=ilast-1
   10 if (icom(ilast) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS",/,1x,80a1)
      return
c
c     FIND THE FIRST NON-BLANK CHARACTER IN THIS FIELD
   30 do 40 m=iplace,ilast
   40 if (icom(m) .ne. " ")  go to 50
      m=ilast
   50 iplace=m
c
c     FIND THE LAST POSITION FOR THIS FIELD
      do 60  ic=iplace,ilast
   60 if (icom(ic) .eq. ",")  go to 70
      ic=ilast
      go to 80
   70 ic=ic-1
c
c     IDENTIFY THE KEYWORD
   80 do 90  ll=iplace,ic
   90 if (icom(ll) .eq. "=")  go to 120
   95 write (10,100)  (icom(i),i=iplace,ic)
  100 format (" PATTERN: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD",
     &       /,1x,73a1)
  110 iplace=ic+2
      if (ilast-iplace) 340,340,30
c
  120 ll=ll-iplace+1
      if (ll .gt. 8)  go to 95
      go to (95,95,95,95,130,95,200,220),ll
c
c     FIND THE TYPE OF DATA
  130 do 140  i=1,4
  140 if ((icom(iplace+i-1) .ne. keyword(3,i) )  .and.
     &    (icom(iplace+i-1) .ne. keyword(8,i)))  go to 95
      iplace=iplace+5
      if (ic-iplace)  110,145,145
```

```
 145   do 150  k=iplace,ic
 150   if (icom(k) .ne. " ")  go to 160
       go to 110
 160   iplace=k
       do 170  i=1,7
 170   if ((icom(iplace+i-1) .ne. keyword(4,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(9,i)))  go to 180
       it=1
       kind="i"
       go to 110
c
 180   do 190 i=1,5
 190   if ((icom(iplace+i-1) .ne. keyword(5,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(10,i)))  go to 95
       it=1
       kind="o"
       go to 110
c
c     FIND THE REFERENCE NUMBER FOR THE NEW PATTERN
 200   do 210 i=1,6
 210   if ((icom(iplace+i-1) .ne. keyword(1,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(6,i)))  go to 95
       key=1
       iplace=iplace+7
       go to 240
c
c     FIND THE NUMBER OF DATA POINTS
 220   do 230  i=1,7
 230   if ((icom(iplace+i-1) .ne. keyword(2,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(7,i)))  go to 95
       key=2
       iplace=iplace+8
c
 240   if (ic-iplace) 110,250,250
 250   do 260  k=iplace,ic
 260   if (icom(k) .ne. " ")  go to 270
       go to 110
 270   iplace=k
       num=ic-iplace+1
       if (num .gt. 15)  go to 290
       l=0
       do 280  i=iplace,ic
       l=l+1
 280   id(l)=icom(i)
       call find_number (id,num,rnum,istat)
       if (istat .eq. 0)  go to 310
 290   write (10,300) (icom(i),i=iplace,ic)
 300   format (" PATTERN: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER",
             & /,1x,73a1)
       go to 110
c
 310   go to (320,330),key
 320   numref=rnum+0.5
       ir=1
       if ((numref .ge. 14) .and. (numref .le. 20) )  go to 110
       write (10,325)
 325   format (" PATTERN: THE REFERENCE NUMBER MUST BE FROM 14 TO 20")
       return
 330   numword=rnum+0.5
       in=1
       if   ((numword .eq. 1) .or. (numword .eq. 2) .or. (numword .eq. 4)
     & .or. (numword .eq. 8) .or. (numword .eq. 16) )  go to 110
       write (10,335)
 335   format (" PATTERN: THE NUMBER OF WORDS MUST BE 1,2,4,8 OR 16")
       return
c
c
 340   if ((ir .eq. 1) .and. (in .eq. 1) .and. (it .eq. 1) )  go to 360
       write (10,350)
 350   format (" THIS PATTERN WILL NOT BE PROCESSED",/,
              &" THERE IS SOME KEYWORD MISSING!!")
       return
c
 360   iplace=1
 370   read (in1,380,end=420)  icom
 380   format (80a1)
       call interpret_data (kind,numback,realval)
       if ((iplace+numback-1) .le. numword)  go to 400
 385   write (10,390)
 390   format (" PATTERN: THERE ARE TOO MANY DATA VALUES!!")
       return
c
 400   do 410  i=1,numback
 410   newval(iplace+i-1)=realval(i)+0.5
       iplace=iplace+numback
       if(iplace-1-numword)  370,440,385
c
 420   write (10,430)
 430   format (" PATTERN: EOF REACHED WHEN TRYING TO READ ANOTHER
       RECORD!!")
       stop
 440   k=16/numword
       iplace=(numref-14)*16+1
       do 460  i=1,k
       do 450  j=1,numword
       new(iplace)=newval(j)
 450   iplace=iplace+1
 460   continue
       write (10,470) numref
 470   format (" NEW PATTERN ASSIGNED TO REFERENCE NUMBER",i3)
       return
       end
```

SUBROUTINE NAME: PLOTOUTLINE

Author: Lawrence Balcerak

Purpose of the program: **plotoutline** reads and evaluates keywords describing how a list of data points immediately following should be plotted. It will then read and interpret the data points and plot them.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call plotoutline

Arguments: None

Subroutines called: **find_number**, **interpret_data**, **plot**, **set_shade**, **tone**

Common data referenced: *icom(80)*, *id(15)*, *in1*

Input files: Command statements used on unit 15 (***file15***)

Output files: ***temp10*** used on unit 10 (***file10***) (for messages)

Arrays used: ***keyword(6,6)***, ***jshade(16)***, ***xx(20)***, ***yy(20)***

Called by: **verplot**

Error checking and reporting: The subroutine checks for a blank data field, an invalid keyword, data value error, too many data values, end of file, and missing keyword. If any such errors are found, an appropriate error message is printed with, sometimes, the character string involved.

Constants: None

Program logic:

1. Set initial values. ***keyword(6,6)*** contains the different keywords possible. There are three keywords, each in both uppercase and lowercase.
   Set ***noline*** = 0, which is the flag for the noline option.
   Set ***numpoint*** = 0, which is the number of points in the outline being plotted. It will be evaluated later.
   Set ***numref*** = 0, which is the reference number of the pattern for the shading option.
   Set ***key*** = 0, which is the branching switch.
   Set ***kind*** = "r", which indicates real number. It is used when calling interpret data.
   Set ***iplace*** = 9, which is the first position of record that keywords can start.
   Set ***ilast*** = 81, which is the position of the last nonblank character of the record.
   Set the arrays ***xx*** and ***yy*** = 0.
2. Starting at the last position of the record (80), check in descending order for a nonblank character. If there is one, go to step 4.
3. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS
   and returns to the calling program.
4. Find the first nonblank position in this field and set ***iplace*** equal to this position.
5. For ***ic*** = ***iplace*** to ***ilast***, find the end of the next keyword by searching for a comma.
   Set ***ic*** equal to one less than the position of the comma or to ***ilast*** if no comma is found. This is the position of the last character in this field.
6. If the characters are not ***NPOINT*** or ***npoint***, go to step 8.
7. Add 7 to ***iplace***, the character position past the =.
   Set ***key*** = 1. This is the branching switch used later to indicate the number of points in the outline. Go to step 14 to interpret the data value.
8. If the characters are not ***SHADE*** or ***shade***, go to step 10.
9. Add 6 to ***iplace***, which is the character position past the =.
   Set ***key*** = 2. This is the branching switch used later to indicate the reference number of shading. Go to step 14 to interpret the data value.
10. If the characters are not ***NOLINE*** or ***noline***, go to step 12.
11. Set ***noline*** = 1. This is the flag to indicate that the outline is not to be plotted. Go to step 13.
12. Subroutine prints the error message:
    OUTLINE: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD!
    and prints the erroneous field.
13. Add 2 to ***iplace***. This is the position where the next keyword could start. If there are more characters to check, go to step 4.
14. If the comma immediately follows the =, go to step 13.
15. Find the first nonblank character in this data field.
    If there is one, set ***num*** equal to this position and go to step 16.
    If only blanks are in the data field, then set ***rnum*** = 0, and go to step 20.
16. Set ***iplace*** = ***num***, which is the position of the first nonblank character in this data field.
    Find ***num***, which is the number of characters in the data field. If ***num*** is greater than 15, go to step 19. The field is too long.
17. Place the character string in the array ***id***.
    Call **find_number** to find ***rnum***, which is the numerical equivalent of the string.
18. If the error return code, ***istat***, is equal to 0, go to step 20. A nonzero value indicates an error of some kind in translation.
19. Subroutine prints the error message:
    OUTLINE: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER
    along with the erroneous data field.
    Go to step 13 to examine the next data field, if any.
20. On the basis of the value of ***key***, go to step 21 (***key*** = 1) or go to step 22 (***key*** = 2).
21. Set ***numpoint*** = ***rnum***. This is the number of data points.
    Go to step 13 to examine the next data field (if any).
22. Set ***numref*** = ***rnum***. This is the reference number for the new pattern.
    Go to step 13 to examine the next data field (if any).
23. If ***numpoint*** is greater than 0, go to step 25. A zero value indicates that the ***npoint*** keyword was not present.
24. Subroutine prints the error message:
    OUTLINE: THE NUMBER OF POINTS WAS NOT GIVEN! THE OUTLINE WILL NOT BE PLOTTED
    and returns to the calling program.
25. Set ***iplace*** = 1, which is index for ***realval***. It always indicates the next value to use.

26. Read a data record into *icom*. If EOF, go to step 31. Call **interpret_data** to evaluate all data fields on the new record.
27. If the number of values already processed plus the number just received from **interpret_data** is less than or equal to the total number of data values required, *numpoint*, go to step 29.
28. Subroutine prints the error message:
    OUTLINE: THERE ARE TOO MANY DATA VALUES!!
    and returns to the calling program.
29. Place the returned values into the arrays *xx* and *yy* using *iplace* as an index counter.
30. If there are more data values to be interpreted, go to step 26 to read another record. If there are too many data values to be interpreted, go to step 28. If the number of values interpreted equals *numpoint*, go to step 32.
31. Subroutine prints the error message:
    OUTLINE: EOF REACHED WHEN TRYING TO READ ANOTHER DATA RECORD!!
    along with a list of the arrays *xx* and *yy*. Return to the calling program.
32. If *noline* is equal to 1, go to step 35. The outline is not to be plotted.
33. Call **plot** to move with pen up to the first coordinate.
    Call **plot** to move with pen down from point to point through the coordinate arrays.
34. Subroutine prints the message:
    OUTLINE PLOTTED
35. If *numref* is equal to 0, go to step 38. A zero value signifies no shading is to be done.
36. Call **set_shade** to place the 16 words that correspond to the shading pattern identified by the reference number, *numref*, into the array *jshade*.
37. Call **tone** to set the shading pattern to that contained in *jshade*.
    Call **tone** to shade the outline.
    Subroutine prints the message:
    OUTLINE SHADED
38. Return

```
      subroutine plotoutline
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1,keyword*1(6,6),kind*1
      dimension jshade(16),xx(20),yy(20),realval(20)
      data ((keyword(i,j),j=1,6),i=1,6)
     &/"N","P","O","I","N","T",
     & "S","H","A","D","E"," ",
     & "N","O","L","I","N","E",
     & "n","p","o","i","n","t",
     & "s","h","a","d","e"," ",
     & "n","o","l","i","n","e"/
      noline=0
      numpoint=0
      numref=0
      key=0
      kind="r"
      iplace=9
      ilast=81
      do 5 i=1,20
      xx(i)=0.
    5 yy(i)=0.
c
c    FIND THE LAST NON-BLANK CHARACTER ON THE RECORD
      do 10 i=iplace,80
      ilast=ilast-1
   10 if (icom(ilast) .ne. " ")  go to 30
      write (10,20) icom
   20 format (" THE FIELD CONTAINS ALL BLANKS",/,1x,80a1)
      return
c
c    FIND THE FIRST NON-BLANK CHARACTER IN THIS FIELD
   30 do 40 m=iplace,ilast
```

```
  40   if (icom(m) .ne. " ")  go to 50
       m=ilast
  50   iplace=m
c
c     FIND THE LAST NON-BLANK CHARACTER IN THIS FIELD
       do 60 ic=iplace,ilast
  60   if (icom(ic) .eq. ",")  go to 70
       ic=ilast
       go to 80
  70   ic=ic-1
c
c     IDENTIFY THE KEYWORD
  80   do 90  i=1,6
  90   if ((icom(iplace+i-1) .ne. keyword(1,i))  .and.
      &    (icom(iplace+i-1) .ne. keyword(4,i)))  go to 100
       iplace=iplace+7
       key=1
       go to 170
c
 100   do 110  i=1,5
 110   if ((icom(iplace+i-1) .ne. keyword(2,i))  .and.
      &    (icom(iplace+i-1) .ne. keyword(5,i)))  go to 120
       iplace=iplace+6
       key=2
       go to 170
c
 120   do 130  i=1,6
 130   if ((icom(iplace+i-1) .ne. keyword(3,i))  .and.
      &    (icom(iplace+i-1) .ne. keyword(6,i)))  go to 140
       noline=1
       go to 160
c
 140   write (10,150) (icom(i),i=iplace,ic)
 150   format (" OUTLINE: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD!",
        & /,1x,73a1)
 160   iplace=ic+2
       if (ilast-iplace)  260,260,30
c
c     FIND THE FIRST POSITION FOR THIS DATA FIELD
 170   if (ic-iplace) 160,180,180
 180   do 190  num=iplace,ic
 190   if (icom(num) .ne. " ")  go to 195
       rnum=0.
       go to 230
c
 195   iplace=num
       num=ic-iplace+1
       if (num .gt. 15)  go to 210
       l=0
       do 200  i=iplace,ic
       l=l+1
 200   id(l)=icom(i)
       call find_number (id,num,rnum,istat)
       if (istat .eq. 0)  go to 230
 210   write (10,220)  (icom(i),i=iplace,ic)
 220   format (" OUTLINE: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER",
                 & /,1x,73a1)
       go to 160
c
 230   go to (240,250),key
 240   numpoint=rnum+0.5
       go to 160
 250   numref=rnum+0.5
       go to 160
c
 260   if (numpoint .gt. 0)  go to 280
       write (10,270)
 270   format (" OUTLINE: THE NUMBER OF POINTS IS NOT GIVEN!",/,
              &"            THE OUTLINE WILL NOT BE PLOTTED.")
       return
c
 280   iplace=1
 285   read (in1,290,end=340)  icom
 290   format (80a1)
       call interpret_data (kind,numback,realval)
       if ( (numback+iplace-1) .le. (2*numpoint) )  go to 320
 300   write (10,310)
 310   format (" OUTLINE: THERE ARE TOO MANY DATA VALUES!!")
       return
c
 320   do 330  i=1,numback,2
       xx(iplace)=realval(i)
       yy(iplace)=realval(i+1)
 330   iplace=iplace+1
       if (iplace-1-numpoint)  285,360,300
 340   write (10,350)  (xx(i),yy(i),i=1,20)
 350   format (" OUTLINE: EOF REACHED WHEN TRYING TO READ",/,
                &"            ANOTHER DATA RECORD!!",/,20(2f10.3))
       return
c
 360   if (noline .eq. 1)  go to 380
       call plot (xx(1),yy(1),3)
       do 370  i=1,numpoint
 370   call plot (xx(i),yy(i),2)
       write (10,375)
 375   format (" OUTLINE PLOTTED")
 380   if (numref .eq. 0)  go to 390
       call set_shade (numref,jshade)
       call tone (0.,0.,jshade,-16)
       call tone (xx,yy,numpoint,1)
       write (10,385)
 385   format (" OUTLINE SHADED")
 390   return
       end
```

SUBROUTINE NAME: FIND_OCTAL_NUMBER

Author: Larry Balcerak

Purpose of the program: **find_octal_number** finds the numeric value of a string of ASCII characters that represent an octal number.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call find_octal_number (id,num,-knum,istat)

Arguments:

id – The array of characters to be changed to a number
num – The number of characters to be changed
knum – The integer number that is returned
istat – The error return code (0 = no error; 1 = some kind of error in translation.)

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used:

id(15) – Contains the characters to be used
karacter(8) – Contains all possible legitimate octal characters

Called by: **interpret_data**

Error checking and reporting: Checks for too many characters and invalid characters. Any error causes the error return code to be set to 1 and a return to the calling program.

Constants: None

Program logic:

1. Set inital values:
   isign = 0 (flag for negation).
   istat = 0 (the error return code).
   knum = 0 (the returned integer number).
2. If *num* is greater than 12, go to step 3. If *num* is equal to 12, go to step 4. If num is less than 12, go to step 6.
3. There is an error – either an invalid character or too many characters.
   Set *istat* = 1 (indicates an error). Return to the calling program.
4. If the octal character is not equal to 4, 5, 6, or 7, go to step 6.
5. Set *isign* = 1. An octal character value of 4 or more indicates a negative value because it has a 1 in the leftmost bit.
   Do steps 6 through 10 for each character in turn (j = 1,*num*).
6. Set k = *num* $-j$ (the exponent for the octal number that represents the number times 8 raised to the kth power).
7. Search the array *karacter* for a match. No match indicates an error; go to step 3. If there is a match, i is one more than the value of the character being examined.
8. If *isign* is greater than 0, go to step 10.
9. Add the numeric value represented to *knum*.
   Go back to step 6 to examine the next character (if any).
10. This is a negative number and is stored in two's complement. The two's complement of an octal number n is $(7 - n)$. Add the complementary value to *knum*. Go back to step 6 to examine the next character (if any).
11. If *isign* is equal to 0, go to step 13.
12. This must be a negative number. Add 1 to *knum* (two's complement).
    Change the sign of *knum*.
13. Return

```
      subroutine find_octal_number (id,num,knum,istat)
c
      character id*1(15),karacter*1(8)
c
      data (karacter(i),i=1,8)
     &     /"0","1","2","3","4","5","6","7"/
      isign=0
      istat=0
      knum=0
c
      if (num-12)  30,20,10
c
c    TOO MANY CHARACTERS FOR AN OCTAL NUMBER
   10 istat=1
      return
c
```

```
c     TWELVE CHARACTERS, CHECK FOR NEGITIVE NUMBER
  20   if ( (id(1) .ne. "4") .and. (id(1) .ne. "5") .and. (id(1) .ne. "6")
      & .and. (id(1) .ne. "7") )  go to 30
       isign=1
  30   do 70  j=1,num
       k=num-j
       do 40  i=1,8
  40   if (id(j) .eq. karacter(i))  go to 50
c
c     UNKNOWN CHARACTER
       go to 10
c
  50   if (isign .gt. 0)  go to 60
       knum=knum+(i-1)*8**k
       go to 70
  60   knum=knum+(8-i)*8**k
  70   continue
       if (isign .eq. 0)  go to 80
       knum=knum+1
       knum=-knum
  80   return
       end
```

SUBROUTINE NAME: FIND_NUMBER

Author: Lawrence Balcerak

Purpose of the program: **find_number** finds the numeric value of a string of ASCII characters representing either a real or integer value.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call find_number (id,num,rnum,istat)

Arguments:

id—The array of characters to be changed to a number, each element containing one character

num—The number of characters to be changed; maximum number of characters, 15

rnum—The real number that is returned

istat—The error return code: 0 = no error; 1 = some kind of error in translation

Subroutines called: None

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **change_origin**, **change_symbol**, **change_width**, **interpret_data**, **newpattern**, **plotfile**, **plotlegend**, **plotoutline**, **scale**

Error checking and reporting: The subroutine checks for invalid characters, too many signs (+ or −), and too many decimal points. Any error causes the error return code to be set to 1 and a return to the calling program.

Constants: None

Program logic:

1. Set intital values:
   rnum = 0 (the returned real number).
   nsign = 0 (the number of sign characters).
   isign = 0 (flag for negation).
   idot = 0 (the number of decimal points).
   istat = 0 (the error return code).
   ifactor = 0 (the number of decimal places in the final value).

   Do steps 2 through 9 for i = 1, *num*. These steps identify the character and set appropriate flags, counters, and values.
2. Search through *jascii* for a character match. If there is a match, set *key* equal to the index number and go to step 3. Otherwise, this is an invalid character; go to step 15.
3. If *key* is greater than 10, go to step 5. This separates the numeric characters from the others.
4. Subtract 1 from *key*. *key* now has the value of the character.
   Multiply *rnum* by 10 and add *key*. This gives the value of one more digit to *rnum*).
   Go to step 2 to examine the next character (if any).
5. Subtract 10 from *key*.
   Go to step 4, 6, 7, or 8, depending on the value of key.

6. Set *isign* = 1, which is the flag for negative value.
7. Add 1 to *nsign*, the number of signs found. If *nsign* is greater than 1, go to step 15. Otherwise, go back to step 2 to examine the next character, if any.
8. Add 1 to *idot*, the number of decimal points found. If *idot* is greater than 1, go to step 15.
9. Set *ifactor* = *num* − *i*, which is the place for the decimal point in the final value. Go back to step 2 to look at the next character, if any.
10. If *ifactor* equals 0, go to step 12. The decimal point does not have to be moved.
11. Move the decimal point in *rnum* to the left by *ifactor* places.
12. If *isign* equals 0, go to step 14. Check for negation.
13. Change the sign of *rnum*.
14. Return to the calling program.
15. Set *istat* = 1, which indicates some kind of error in translating the data string.
16. Return.

```
      subroutine find_number(id,num,rnum,istat)
c
      character id*1(15),jascii*1(14)
c
      data (jascii(i),i=1,14) /"0","1","2","3","4","5","6",
     &"7","8","9"," ","-","+","."/
      rnum=0.
      nsign=0
      isign=0
      idot=0
      istat=0
      ifactor=0
c     IDENTIFY THE CHARACTER
      do 100 i=1,num
      do 30 key=1,14
   30 if (id(i) .eq. jascii(key))  go to 40
      go to 130
c
   40 if (key .gt. 10)  go to 50
   45 key=key-1
      rnum=rnum*10.+key
      go to 100
   50 key=key-10
      go to (45,60,70,80),key
   60 isign=1
   70 nsign=nsign+1
      if (nsign .ge. 2)  go to 130
      go to 100
   80 idot=idot+1
      if (idot .ge. 2)  go to 130
      ifactor=num-i
  100 continue
c
      if (ifactor .eq. 0)  go to 110
      rnum=rnum/(10.**ifactor)
  110 if (isign .eq. 0)  go to 120
      rnum=-rnum
  120 return
c     ERROR FLAG
  130 istat=1
      return
      end
```

SUBROUTINE NAME: SET_SHADE

Author: Lawrence Balcerak

Purpose of the program: Given the reference number of a shading pattern, **set_shade** will return an array that will produce that pattern.

Data base: Geoindex

Computer: Honeywell series 60 (level 68)

Operating system: Multics

Calling sequence: call set_shade (numref,jshade)

Arguments:

numref–The reference number being processed

jshade(16)–The 16-word array containing values whose bit arrangement forms some pattern.

Subroutines called: None

Common data referenced: Some entry in */pat/* or */userpat/* depending on the value of *numref*

Input files: None

Output files: None

Arrays used: *new(118)*, *kold(136)*

Called by: **set_shade**

Error checking and reporting: A check is made for *numref* to range from 1 to 20. Any value outside this range gives the default value of 1 for *numref*.

Constants: None

Program logic:

1. *new(1)* is set equivalent to *ip11*, and *kold(1)* is set equivalent to *ipl(1)*.
   This causes one array to overlay each entry in a common block and thereby enables the program to access the whole block by just changing the index for the array.
2. If *numref* is outside the range 1-20, set *numref* = 1, which is the default value.
3. If *numref* is greater than 10, go to step 8. If *numref* has a value from 5 to 10, go to step 6. If *numref* has a value of 3 or 4, go to step 5.
4. To get here, *numref* must be equal to 1 or 2.
   Compute the index for *iplace*. Set *numword* = 16, which is the number of words in the sequence that completes one pattern. Go to step 7.
5. The reference number must be 3 or 4 to reach this step.
   Compute *iplace*. Set *numword* = 4. Go to step 7.
6. The reference number must be 5 through 10 to reach this step.
   Compute *iplace*. Set *numword* = 16.
7. Fill *jshade* with the pattern. If the pattern does not take 16 words, repeat the pattern until all 16 words are given a value. The correct pattern is found in *kold* by using the index *iplace* as a starting point and reading *numword* words. Return to calling program.
8. If *numref* has a value from 14 to 20, go to step 11. If *numref* equals 13, go to step 10.
9. The reference number must be 11 or 12 to reach this step.
   Compute *iplace*. Set *numword* = 1. Go to step 12.
10. The reference number must be 13 to reach this step.
    Set *iplace* = 3. Set *numword* = 4. Go to step 12.
11. The reference number must be 14 through 20 to reach this step.
    Compute *iplace*. Set *numword* = 16.
12. Fill *jshade* with the pattern. If the pattern does not take 16 words, repeat the pattern until all 16 words are given a value. The correct pattern is found in *new* by using the index *iplace* as a starting point and reading *numword* words.
    Return to the calling program.

```
      subroutine set_shade (numref,jshade)
      common /pat/   ip1(16),ip2(16),ip3(4) ,ip4(4) ,ip5(16),
             &       ip6(16),ip7(16),ip8(16),ip9(16),ip10(16)
      common /userpat/ ip11    ,ip12    ,ip13(4) ,ip14(16),ip15(16),
                     &  ip16(16),ip17(16),ip18(16),ip19(16),ip20(16)
      dimension jshade(16),new(118),kold(136)
      equivalence (new(1),ip11),(kold(1),ip1(1))
c
      if ( (numref .lt. 1) .or. (numref .gt. 20))  numref=1
      if (numref .gt. 10)  go to 60
      if (numref .ge. 5)   go to 20
      if (numref .ge. 3)   go to 10
c    THE REFERENCE NUMBER IS 1 OR 2.
      iplace=(numref-1)*16+1
      numword=16
      go to 30
c    THE REFERENCE NUMBER IS 3 OR 4.
   10 iplace=(numref-3)*4+33
```

```
      numword=4
      go to 30
c    THE REFERENCE NUMBER IS 5 THRU 10.
  20  iplace=(numref-5)*16+41
      numword=16
c
  30  k=16/numword
      l=0
      do 50 i=1,k
      do 40  j=1,numword
      l=l+1
      jshade(l)=kold(iplace)
  40  iplace=iplace+1
      iplace=iplace-numword
  50  continue
      return
c
  60  if (numref .ge. 14)  go to 80
      if (numref .eq. 13)  go to 70
c   THE REFERENCE NUMBER IS 11 OR 12.
      iplace=(numref-11)+1
      numword=1
      go to 90
c    THE REFERENCE NUMBER IS 13.
  70  iplace=3
      numword=4
      go to 90
c    THE REFERENCE NUMBER IS 14 THRU 20.
  80  iplace=(numref-14)*16+7
      numword=16
c
  90  k=16/numword
      l=0
      do 110  i=1,k
      do 100  j=1,numword
      l=l+1
      jshade(l)=new(iplace)
 100  iplace=iplace+1
      iplace=iplace-numword
 110  continue
      return
      end
```

SUBROUTINE NAME: PLOTLEGEND

Author: Lawrence Balcerak
Purpose of the program: **plotlegend** reads a character string and plots the string using parameters given by the keywords of the command.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call plotlegend
Arguments: None
Subroutines called: **find_number**, **letter**, **newpen**
Common data referenced: *icom*(*80*), *id*(*15*), *in1*
Input files: Command statements used on unit 15 (*file15*)
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *keyword*(*8,6*), *itext*(*60*)
Called by: **verplot**

Error checking and reporting: The subroutine checks for a blank data field, invalid keywords, data field too long, data-value error, and missing keyword. If any such errors are found, an appropriate error message is printed with, in certain circumstances, the character string involved.

Constants: None

Program logic:

1. Set initial values. ***keyword(8,6)*** contains the four possible keywords.
   Each of the four is in both uppercase and lowercase.
   Set ***iangle*** = 0. This is the default angle for plotting the legend.
   Set ***lwidth*** = 1. This is the default line width in dots.
   Set ***ix*** = 0. This is the flag to indicate that the x-coordinate has been processed.
   Set ***iy*** = 0. This is the flag to indicate that the y-coordinate has been processed.
   Set ***ih*** = 0. This is the flag to indicate that the height has been processed.
   Set ***in*** = 0. This is the flag to indicate that the number of characters has been processed.
   Set ***key*** = 0. This is the branching switch.
   Set ***iplace*** = 8. This is the first position that a keyword can start.
   Set ***ilast*** = 81. This is the position of the last nonblank character of the record.
2. Starting at the last position of the record, 80, check in descending order for a nonblank character. If there is one, go to step 4.
3. The subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS
   and returns to the calling program.
4. Find the first nonblank character in this field, and set ***iplace*** equal to this position.
5. For ***ic*** = ***iplace*** to ***ilast***, find the end of the next keyword by searching for a comma.
   Set ***ic*** equal to one less than the position of the comma or to ***ilast*** if no comma is found. This is the position of the last character in this field.
6. Search the field for the character =.
   If one is found, go to step 9. Any keyword must have this character.
7. Subroutine prints the error message:
   LEGEND: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD
   along with the erroneous field.
8. Set ***iplace*** = ***ic*** + 2, one position past the comma.
   If there are more characters to examine, go to step 4. Otherwise, go to step 35.
9. Set ***ll*** equal to the count of characters including the = of the keyword.
   If ***ll*** is greater than 7, go to step 7. No keyword is greater than 8 characters.
10. If ***ll*** is equal to 1, 3, 4, or 5, go to step 7. If ***ll*** is equal to 2, go to step 11. If ***ll*** is equal to 6, go to step 15. If ***ll*** is equal to 7, go to step 19.
11. If the characters are X or x, go to step 12.
    If the characters are Y or y, go to step 13. Otherwise, go to step 7.
12. Set key = 1. This is the branching switch for the x coordinate.
    Set ***ix*** = 1. This is the flag to indicate processing of the x coordinate.
    Go to step 14.
13. Set ***key*** = 2. This is the branching switch for the y coordinate.
    Set ***iy*** = 1. This is the flag to indicate processing of the y coordinate.
14. Add 2 to ***iplace***, first position past the = character.
    Go to step 23 to find the data value.
15. If the characters are not equal to ***NCHAR*** or ***nchar***, go to step 17.
16. Set ***in*** = 1. This is the flag to indicate processing of the number of characters.
    Set ***key*** = 3. This is the branching switch for number of characters.
    Add 6 to ***iplace***, first position past the = character.
    Go to step 23 to find the data value.
17. If the characters are not equal to ***ANGLE*** or ***angle***, go to step 7.
18. Set ***key*** = 4. This is the branching switch for the angle.
    Add 6 to ***iplace***, first position past the = character.
    Go to step 21 to find the data value.
19. If the characters are not equal to ***HEIGHT*** or ***height***, go to step 21.
20. Set ***h*** = 1. This is the the flag to indicate processing of the height.
    Set ***key*** = 5. This is the branching switch for the height.
    Add 7 to ***iplace***, first position past the = character.
    Go to step 23 to find the data value.
21. If the characters are not equal to ***LWIDTH*** or ***lwidth***, go to step 7.
22. Set ***key*** = 6. This is the branching switch for the line width.
    Add 7 to ***iplace***, first position past the = character.
23. Check for a comma immediately following the = character.
    If there is one, go to step 8 to check for the next keyword.
24. Find the position of the first nonblank character in the data field.

If the field is all blank, go to step 8. Otherwise, set *iplace* equal to this position.

25. Compute *num*. This is the the number of characters in this data field.
    If *num* is greater than 15, go to step 27. The field is too long.
26. Place the characters in the array *id*.
    Call **find_number** to evaluate the data. The returned value is in *rnum*2.
    If the error return code, *istat*, is equal to 0, go to step 28. A nonzero value indicates an error of some type when interpreting.
27. Subroutine prints the error message:
    LEGEND: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER
    Go to step 8 to examine the next keyword, if any.
28. Depending on the value of key, go to:
    step 29 (*key* = 1); step 30 (*key* = 2); step 31 (*key* = 3); step 32 (*key* = 4); step 33 (*key* = 5); step 34 (*key* = 6).
29. Set *xx* = *rnum* (the x coordinate).
    Go to step 8.
30. Set *yy* = *rnum* (the y coordinate).
    Go to step 8.
31. Set *nchar* = *rnum* (the number of characters in the legend).
    Go to step 8.
32. Set *iangle* = *rnum* (the angle of the legend).
    Go to step 8.
33. Set *height* = *rnum* (the height of a character).
    Go to step 8.
34. Set *lwidth* = *rnum* (the width of a line in dots).
    Go to step 8.
35. Read the next record, which contains the text to be plotted.
36. If the four keywords (x coordinate, y coordinate, height, and number of characters) were processed, go to step 38.
37. Subroutine prints the error message:
    THIS LEGEND CANNOT BE PLOTTED!
    THERE IS SOME KEYWORD MISSING!!
    It prints the line of text involved and returns to the calling program.
38. Call **newpen** to set the line width (dots) to the new value.
39. Set *iscale* equal to the number of sixteenths in the height.
    Call **letter** to plot the character string.
40. Subroutine prints the message:
    LEGEND PLOTTED
41. Return.

```
      subroutine plotlegend
c
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1,keyword*1(8,6)
      dimension itext(20)
c
      data ((keyword(i,j),j=1,6),i=1,8)
     &/"N","C","H","A","R"," ",
     & "A","N","G","L","E"," ",
     & "H","E","I","G","H","T",
     & "L","W","I","D","T","H",
     & "n","c","h","a","r"," ",
     & "a","n","g","l","e"," ",
     & "h","e","i","g","h","t",
     & "l","w","i","d","t","h"/
      iangle=0
      lwidth=1.
      ix=0
      iy=0
      ih=0
      in=0
      key=0
      iplace=8
      ilast=81
c
c     FIND THE LAST NON-BLANK CHARACTER OF THE RECORD
```

```
      do 10  i=iplace,80
      ilast=ilast-1
 10   if (icom(ilast) .ne. " ")  go to 30
      write (10,20) icom
 20   format (" THE FIELD CONTAINS ALL BLANKS",/,1x,80a1)
      return
c
c     FIND THE FIRST NON-BLANK IN THIS FIELD
 30   do 40  m=iplace,ilast
 40   if (icom(m) .ne. " ")  go to 50
      m=ilast
 50   iplace=m
c     FIND THE LAST POSITION FOR THIS FIELD
      do 60 ic=iplace,ilast
 60   if (icom(ic) .eq. ",")  go to 70
      ic=ilast
      go to 80
 70   ic=ic-1
c     IDENTIFY THE KEYWORD
 80   do 90  ll=iplace,ic
 90   if (icom(ll)  .eq. "=")  go to 120
 95   write (10,100) (icom(i),i=iplace,ic)
 100  format (" LEGEND: THIS FIELD IS NOT RECOGNIZED AS A KEYWORD",
     & /,1x,73a1)
 110  iplace=ic+2
      key=0
      if (ilast-iplace)  390,390,30
c
 120  ll=ll-iplace+1
      if (ll .gt. 7)  go to 95
      go to (95,130,95,95,95,170,210),ll
c    FIND THE X-VALUE OR THE Y-VALUE
 130  if ((icom(iplace) .eq. "X")  .or.
     &    (icom(iplace) .eq. "x"))  go to 140
      if ((icom(iplace) .eq. "Y")  .or.
     &    (icom(iplace) .eq. "y"))  go to 150
      go to 95
 140  key=1
      ix=1
      go to 160
 150  key=2
      iy=1
 160  iplace=iplace+2
      go to 250
c     FIND THE ANGLE OR THE NUMBER OF CHARACTERS
 170  do 180  i=1,5
 180  if ((icom(iplace+i-1) .ne. keyword(1,i))  .and.
     &    (icom(iplace+i-1) .ne. keyword(5,i)))  go to 190
      in=1
      key=3
      iplace=iplace+6
      go to 250
 190  do 200  i=1,5
 200  if ((icom(iplace+i-1) .ne. keyword(2,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(6,i)))    go to 95
      key=4
      iplace=iplace+6
      go to 250
c     FIND THE HEIGHT OR THE LINE WIDTH
 210  do 220  i=1,6
 220  if ((icom(iplace+i-1) .ne. keyword(3,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(7,i)))  go to 230
      ih=1
      key=5
      iplace=iplace+7
      go to 250
 230  do 240 i=1,6
 240  if ((icom(iplace+i-1) .ne. keyword(4,i))  .and.
     &     (icom(iplace+i-1) .ne. keyword(8,i)))  go to 95
      key=6
      iplace=iplace+7
 250  if (ic-iplace) 110,260,260
c     FIND THE FIRST NON-BLANK CHARACTER FOR THIS DATA FIELD
 260  do 270  num=iplace,ic
 270  if (icom(num) .ne. " ")  go to 280
      go to 110
 280  iplace=num
      num=ic-iplace+1
      if(num .gt. 15)  go to 300
      l=0
      do 290  i=iplace,ic
      l=l+1
 290  id(l)=icom(i)
      call find_number (id,num,rnum,istat)
      if (istat .eq. 0)  go to 320
 300  write (10,310) (icom(i),i=iplace,ic)
 310  format (" LEGEND: THIS STRING HAS AN UNRECOGNIZABLE CHARACTER",
            & /,5x,73a1)
      go to 110
c
 320  go to (330,340,350,360,370,380),key
 330  xx=rnum
      go to 110
 340  yy=rnum
      go to 110
 350  nchar=rnum+0.5
      go to 110
 360  iangle=rnum+0.5
      go to 110
 370  height=rnum
      go to 110
 380  lwidth=rnum+0.5
      go to 110
c
c
 390  read (in1,395,end=450)  itext
 395  format (20a4)
```

```
      if ( (ix .eq. 1) .and. (iy .eq. 1) .and. (ih .eq. 1) .and.
        & (in .eq. 1) )  go to 430
      write (10,400)
 400  format (" THIS LEGEND CANNOT BE PLOTTED!",/,
             & " THERE IS SOME KEYWORD MISSING!!")
      write (10,420)  itext
 420  format (1x,20a4)
      return
c
 430  call newpen (lwidth)
      iscale=height*16+0.5
      call letter (nchar,iscale,iangle,xx,yy,itext)
      write (10,440)
 440  format (" LEGEND PLOTTED")
 450  return
      end
```

SUBROUTINE NAME: PLOTFILE

Author: Lawrence Balcerak
Purpose of the program: **plotfile** reads the name of a file, opens that file, and plots it.
Data base: Geoindex
Computer: Honeywell 60 (series 68)
Operating system: Multics
Calling sequence: call plotfile
Arguments: None
Subroutines called: **closef, find_number, io_call, letter, plot, set_shade, symbol, tone**
Common data referenced: *icom(80), id(15), isym, in1, in2*
Input files:
Command statements used on unit 15 (*file15*)
File to be plotted used on unit 16 (*file16*)
Output files: *temp10* used on unit 10 (*file10*) (for messages)
Arrays used: *keyword(18,9), ipat(20), ne(2), jwhat(200), xx(2000), yy(2000), kplotfield(8), jshade(16), kfield(8), char(8,5), text(5), iwhat (200,2)*
Called by: **verplot**
Error checking and reporting: The subroutine checks for a blank data field, blank file name, file name too long, data field too long, invalid data character, invalid keyword, missing keyword, and end of file reached. If any such error is found, an appropriate error message is printed with, in certain circumstances, the character string involved.
Constants: None
Program logic:

1. Set initial values. *keyword(18,9)* contains the nine keywords possible. Each is in both uppercase and lowercase.

Set *name* equal to blanks. This will be the name of the file to be opened for plotting.
Set *height* = 0.14, which is the default height of each plotted character.
Set *space* = height divided by 5, which is the space between lines of character.
Set *numpat* = 0, which is the number of the pattern to use for shading.
Set *noline* = 0, which is the flag for the *noline* option.
Set *noname* = 0, which is the flag for the processing of the *name* keyword.
Set *noselect* = 0, which is the flag for the *select* option.
Set *noclear* = 0, which is the flag for clearing the space around characters.
Set *noshade* = 0, which is the flag for the *shade* option.
Set *nochar* = 0, which is the flag for character plotting.
Set *noselshade* = 0, which is the flag for selecting shades.
Set *item* = 0, which is the index counter used to rotate through the different patterns.
Set *iplace* = 6, which is the first position that a keyword can be found.
Set *ilast* = 81, which is the position of the last nonblank character (initialized to one past the end of the record).
Set *ipat(i)* = *i* for *i* = 1, 10, which is the default sequence of patterns to rotate through.
Set *ipat(i)* = 0 for *i* = 11, 20, which indicates that these patterns are not used in the rotation.
Set *kplotfield(i)* = 0 for *i* = 1, 8, to indicate which

of the eight character fields from the header card are to be plotted and in what order.

2. Starting at the last position of the record (80), check in descending order for a nonblank character.
   If there is one, go to step 4.
3. Subroutine prints the error message:
   THE FIELD CONTAINS ALL BLANKS
   and returns to the calling program.
4. Find the first nonblank character in this field and set ***iplace*** equal to this position.
5. For ***ic*** = ***iplace to ilast***, find the end of the next keyword field by searching for a comma.
   Set ***ic*** equal to one less than the position of the comma or to ***ilast*** if no comma is found. This is the position of the last character in this field.
6. If the characters in the keyword are not ***NAME*** or ***name***, go to step 22.
7. Add 5 to ***iplace***, the first position past the = character.
8. Set ***iplace*** equal to the first nonblank character in the data field.
   If the field is all blank, set ***iplace*** equal to ***ic***, which is the last character in the field.
9. Compute m, the number of characters in the data field.
   Compute k, the number of characters in the record that lie before this data field.
   If m is greater than 0, go to step 11. A zero value would occur with an all blank field or when a comma immediately follows the = character.
10. Subroutine prints the error message:
    NAME HAS NO CHARACTERS!!
    Go to step 18 to examine the next keyword, if any.
11. If m is less than or equal to 20, go to step 13.
12. Subroutine prints the error message:
    NAME IS MORE THAN 20 CHARACTERS LONG!!
    along with the erroneous field.
    Go to step 18 to examine the next keyword, if any.
13. Backspace the command file.
14. Compute ***fmt1***, the format to be used in reading the name of the file. This format must skip spaces and read m characters from the record into ***name(k)***.
15. Read the file name from the record.
16. Call **io_call** to attach and open the file for input. Use the value of ***in2*** as the file number.
17. Set ***noname*** = 1. This is the flag to indicate that the file name has been processed.
18. Set ***iplace*** equal to ***ic*** + 2, the first position past the comma. This would be the first possible position for the next keyword.
19. If there are more characters in the record to check, go to step 4 to examine the next keyword.
20. If there is not another record containing keywords, go to step 80 to read the select record, if any.
21. Read the next record into ***icom(80)***.
    If EOF, go to step 128. Set ***ilast*** = 81. Set ***iplace*** = 1. Go to step 2 to interpret this record.
22. If the characters in the keyword are not ***HEIGHT*** or ***height***, go to step 31.
23. Add 7 to ***iplace***, the first position past the = character.
24. If there are not any characters in this data field, go to step 18 to examine the next keyword, if any.
25. Find the first nonblank character in this data field and set ***iplace*** equal to this position. If the field is all blank, go to step 18 to examine the next keyword, if any.
26. Compute ***num***, which is the number of characters in the data field.
    If ***num*** is less than or equal to 15, go to step 28 to interpret the data.
27. Subroutine prints the error message:
    PLOT-HEIGHT: THIS FIELD HAS TOO MANY CHARACTERS
    along with the erroneous field.
    Go to step 18 to examine the next keyword, if any.
28. Place the string of characters into the array ***id***.
    Call **find_number** to evaluate the data. The returned value is in ***rnum***.
29. If the return error code, ***istat***, is not equal to 0, go to step 18 to examine the next keyword, if any. A nonzero value indicates an error of some type when translating.
30. Set ***height*** = ***rnum***.
    Set ***space*** = ***height*** divided by 5. Go to step 18 to examine the next keyword, if any.
31. If the characters of the keyword are not ***PATTERN*** or ***pattern***, go to step 51.
32. Add 8 to ***iplace***, which is the first position past the = character.
    Set ***kount*** = 0, the counter for the number of pattern reference number being interpreted.
33. If the data field has no length, a comma follows the =, go to next step. Otherwise, go to step 35.
34. The subroutine prints the error message:
    PLOT-PATTERN: THE PATTERN COUNT HAS AN ERROR!!
    along with the erroneous field.
    Go to step 18 to examine the next keyword, if any.
35. Find ***num***, the position of the first nonblank character in this field.
    If there is a nonblank character, go to step 38.

36. If *kount* = 0, go to step 34. This indicates an all-blank field for the count of numbers following.
37. The subroutine prints the error message:
    PLOT-PATTERN: THE FIELD CONTAINS AN ERROR AND WILL BE SET TO THE DEFAULT VALUE
    along with the erroneous field.
    Go to step 46.
38. Set *iplace* = *num*, which is the first nonblank position of the data field.
39. Compute the number of characters in the data field and store in *num*.
    If *num* is less than or equal to 15, go to step 41.
40. If *kount* is equal to 0, go to step 34. This should be the pattern count.
    Go to step 37. There is an error in a pattern reference number.
41. Place the character string into the array *id*.
    Call **find_number** to evaluate the data. The returned value is in *rnum*.
    If the return error code, *istat*, = 0, go to step 43. A nonzero value indicates an error of some sort during the interpretation.
42. If *kount* = 0, go to step 34. This is the pattern count that has an error. Otherwise, go to step 37.
43. If *kount* is greater than 0, go to step 45.
44. Set *numpat* = *rnum*. This is the count of the pattern reference numbers that follow.
    Go to step 46.
45. Set *ipat(kount)* = *rnum*. Store the pattern reference number just translated.
46. Add 1 to *kount*.
    If *kount* is less than or equal to *numpat*, go to step 48.
47. Zero out the rest of the *ipat* array.
    Go to step 18 to examine the next keyword, if any.
48. There are more numbers to translate.
    Set *iplace* = *ic* + 2, the first position past the comma (if there was one).
    If *ilast* is less than or equal to *iplace*, go to step 47 because there are no more characters to interpret on this record.
    If *ilast* is greater than *iplace*, there is an error in the command file because the plot option and value must be on the same record.
49. Set *iplace* equal to the first nonblank character of this data field. There must be at least one.
50. For *ic* = *iplace* to *ilast*, find the end of this keyword by searching for a comma.
    Set *ic* equal to one less than the position of the comma or to *ilast*, if no comma is found. This is the position of the last character in this field.
    Go to step 35 to interpret the next data field.
51. If the characters of the keyword are not ***TEXTFIELD*** or ***textfield***, go to step 69.
52. Add 10 to *iplace*, first position past the = character.
    Set *kount* = 0, which is the counter for the number of text fields being interpreted.
53. If the data field has no length (a comma follows the =), go to the next step. Otherwise, go to step 55.
54. Subroutine prints the error message:
    PLOT-TEXTFIELD: THE FIELD COUNT HAS AN ERROR!!
    along with the erroneous field.
    Go to step 18 to examine the next keyword, if any.
55. Search for the first nonblank character in this data field, and set *num* equal to this position. If such a character is found, go to step 58.
56. If *kount* equals 0, go to step 54. This would be an all-blank data field for the number of text fields count.
57. Subroutine prints the error message:
    PLOT-TEXTFIELD: THIS FIELD CONTAINS AN ERROR AND WILL NOT BE PLOTTED!!
    along with the erroneous field.
    Go to step 65 to examine the next number, if any.
58. Set *iplace* = *num*, the first nonblank character in this field.
    Compute *num*, which is the number of characters in the data field.
    If *num* is less than or equal to 15, go to step 60.
59. If *kount* = 0, go to step 54. Otherwise, go to step 57.
60. Place the characters in the array *id*.
    Call **find_number** to evaluate the data. The returned value is in *rnum*.
    If the error return code, *istat* = 0, go to step 62. A nonzero value indicates an error of some type.
61. If *kount* = 0, go to step 54. Otherwise, go to step 57.
62. If *kount* is greater than 0, go to step 64.
63. Set *numfield* = *rnum*, which is the count of text-field numbers that follow.
    Go to step 65.
64. Set *num* = *rnum*.
    Set *kplotfield(num)* = *kount*. The array *kplotfield* contains numbers indicating the order in which the character fields from the header card will be plotted. If *kplotfield(num)* is blank or zero, there will be no plotting. If value is other than blank or zero, the text field will be plotted.
65. Add 1 to *kount*, which is the next sequence number.
    If *kount* is less than or equal to *numfield*, go to

step 67. There are more numbers in this sequence to interpret.

66. Set ***nchar*** = 1, which is a flag indicating that the textfield option is to be used. In other words, there are character fields to be plotted.
Go to step 18 to examine the next keyword, if any.

67. Set ***iplace*** = ***ic*** + 2, the first position past the comma.
If there are no more characters left in this record, go to step 66.

68. Search the remainder of the string for a comma.
Set ***ic*** equal to one less than the position of the comma or to ***ilast*** if no comma is found.
Go to step 55 to interpret this data field.

69. If the characters are not ***NOLINE*** or ***noline***, go to step 71.

70. Set ***noline*** = 1. This is a flag to turn on the ***noline*** option.
Go to step 18 to examine the next keyword, if any.

71. If the characters are not ***SELECT*** or ***select***, go to step 73.

72. Set ***noselect*** = 1. This is a flag to turn on the ***select*** option.
Go to step 18 to examine the next keyword, if any.

73. If the characters are not ***REFCLEAR*** or ***refclear***, go to step 75.

74. Set ***noclear*** = 1. This is a flag to indicate the clearing of the area around reference number is to be done.
Go to step 18 to examine the next keyword, if any.

75. If the characters are not ***SHADEALL*** or ***shadeall***, go to step 77.

76. Set ***noshade*** = 1. This is a flag to turn on the shading options for all outlines.
Go to step 18 to examine the next keyword, if any.

77. If the characters are not ***SELSHADE*** or ***selshade***, go to step 79.

78. Set ***noshade*** = 1.
Set ***noselshade*** = 1. These two flags will tell the program to shade only those outlines that have pattern reference numbers listed in the selected outlines.
Go to step 18 to examine the next keyword, if any.

79. The variable did not match any valid keyword.
The subroutine prints the error message:
PLOT: THIS KEYWORD IS NOT VALID!!
along with the erroneous field.
Go to step 18 to examine the next keyword, if any.

At this point in the program, all keywords have been read and evaluated. Next, the file of selected outlines is read and then plotted.

80. Set ***kount*** = 1. This is the counter for the number of feature numbers read in.

81. Read the next record from the command file as characters.
If EOF, go to step 128.

82. If these characters are ***END PLOT*** or ***end plot***, this signifies the end of the information for plotting this file.
Go to step 84.

83. Add 1 to ***kount***.
Go to step 81 to read another record.

84. Subtract 1 from ***kount***. The ***END FILE*** record is not to be used.

85. If ***noname*** = 1, go to step 87. A zero value indicates that the ***name*** keyword was not present or had an error in it.

86. The subroutine prints the error message:
PLOT: NO PLOT FILE???
and returns to the calling program.

87. Read a header card from the plot file. The eight fields are read here as a character string. This is needed for comparison with the select outline file.
If the EOF reached, go to step 128.

88. Backspace the plot file.

89. Read the header card as separate characters. This is needed later in the program when plotting the characters.

90. Backspace the plot file.

91. Read ***isfno***, the number of pairs of coordinate points.

92. Subtract 1 from ***isfno***. The first position is a text position.

93. Read the x text position, the y text position, and the outline points.

94. Set ***isel*** = 0, which is a flag to indicate if outline is in select file.
Set ***key*** = 0. If the outline is in the select file, ***key*** will take on the value of the pattern reference number given for that outline.

95. If ***kount*** equals 0, go to step 98. No outlines were listed in the select outline file.

96. If this outline is in the file of selected outlines go to step 97.
Otherwise, go to step 98.

97. Set ***isel*** = 1. This is a selected outline.
Set ***key*** = ***jwhat(i)***, which takes on the value of the pattern reference number listed.

98. If ***noselect*** = 1 and ***isel*** = 0, go to step 87 to read the points for another outline. This outline will not be plotted.

99. If ***isfno*** is greater than 1, go to step 101.

100. This is a single point that has some character plotted at that point.

Call **symbol** to plot the character.
Go to step 118 to plot the text, if any.

101. If *noline* equals 1, go to step 103. This indicates that the outline will not be plotted.
102. Use the subroutine **plot** to plot the outline.
103. If *noshade* is not equal to 1, go to step 118. A value of 1 indicates that the outline is to be shaded.
104. If *noselshade* equals 1, and *isel* equals 0 or key equals 0, go to step 118. A value of 1 for *noselshade* indicates that the selective shading option is in effect, and the shading pattern used will be in the selective outline file. A value of 0 for *isel* indicates that this outline is not in the selective file. A value of 0 for *key* indicates that a 0 value was in the pattern location for this outline.
105. Set *ne(1)* = *isfno*; set *ne(2)* = 0.
This array contains the number of points in an outline(s) when using the subroutine **tone** for shading. If there is more than one outline, the subroutine will alternate the shading with blank areas, depending on the overlapping of the outlines. This will be used to clear areas around the text if needed.
Set *numarea* = 1, one area to start with.
106. If *noclear* equals 0 or *nochar* equals 0, go to step 114. Either the clearing option was not used or no characters are wanted.
107. Set *numvert* = 0; set *numhorz* = 0. These are counters for the number of characters that will be plotted both vertically and horizontally. Do steps 108-111 for *i* = 1, 8.
108. If *kplotfield*(*i*) equals 0, the ith field of the header card will not be plotted; skip to the next value of *i*.
109. Add 1 to *numvert*. There is one more line of text.
110. Set *icheck* = 5. A maximum of five characters is in a field.
Check each character in this field. For each leading blank or zero, subtract 1 from *icheck*. All characters following nonzero characters are to be considered significant, even a blank.
111. Set *numhorz* equal to the maximum of *icheck* and *numhorz*. After checking all eight text fields, *numhorz* will hold the maximum number of characters in any line.
112. If *numhorz* equals 0, go to step 114. There are no lines of text to plot.
113. Set *k* = *isfno* + 1. This is the first index position used to store the outline to be cleared.
Compute the coordinates of the four corners of the rectangle to be cleared and store in *xx* and *yy* immediately after the main outline.
Set *ne(2)* = 4. Four points are in the cleared rectangle.
Set *numarea* = 2, two outlines.
114. Add 1 to *item*. *item* is used as an index counter to rotate through all the different patterns used.
If *item* is greater than *numpat*, set *item* = 1. The total number of patterns used is *numpat*.
115. If *ipat*(*item*) is less than or equal to 0 or if *ipat*(*item*) is greater than 20, go back to step 114. These would be invalid reference numbers.
116. Set *num* = *ipat*(*item*). This is the reference number that comes from the sequence of patterns.
If *key* has a value that represents a valid reference number, set *num* = *key*. This reference number takes the place of the default value.
117. Call **set_shade** to get the pattern values for this reference number.
Call **tone** to set the pattern. Call **tone** to shade the outline. It will also clear the text area if wanted.
118. If *nochar* equals 0, go back to step 87 to read the next header card.
Do steps 119–125 for *i* = 1, 8.
119. Search *kplotfield*(*j*) for a value equal to *i*. This will give the next text field to plot.
If there is a match, go to the next step. Otherwise, search for the next value of *i*.
120. Count the number of leading blanks in the character string, and set equal to *num*.
121. If there are all blanks or the numeric value of the field is 0, go back to step 119 to search for the next value of *i*.
122. Set *n* = 5 − *num*. This is the number of characters to be plotted. Store the characters to be plotted in *itext*.
123. Set *posx* = *xpos*. This is the x coordinate of the first line of text.
Set *iscale* equal to the number of sixteens in the height.
124. For each character in turn, call **letter** to plot the character.
Add the height of a letter and an interletter space to the x coordinate to locate the next character.
125. Subtract enough room from the y coordinate to correctly position the next line of text.
126. Go to step 87 to read the next header card.
127. Subroutine prints the error message:
PLOT: END OF FILE REACHED WHEN TRYING TO READ A DATA RECORD
Go to step 129.
128. Subroutine prints the message:
FINISHED PLOTTING
129. Call **closef** to close and detach the plot file. Return.

```
      subroutine plotfile
      common /comand/ icom(80),id(15),isym,in1,in2
      character icom*1,id*1,keyword*1(18,9),name*20,fmt1*21,fmt2*21,
     &  kfield*5(8),name1*6,char*1(8,5),itext*1(5),iwhat*5(200,2)
      external io_call (descriptors)
      dimension ipat(20),ne(2),jwhat(200),xx(2000),yy(2000),
     & kplotfield(8),jshade(16)
      data ((keyword(i,j),j=1,9),i=1,18)
     &/"N","A","M","E"," "," "," "," "," ",
     & "H","E","I","G","H","T"," "," "," ",
     & "P","A","T","T","E","R","N"," "," ",
     & "T","E","X","T","F","I","E","L","D",
     & "N","O","L","I","N","E"," "," "," ",
     & "S","E","L","E","C","T"," "," "," ",
     & "R","E","F","C","L","E","A","R"," ",
     & "S","H","A","D","E","A","L","L"," ",
     & "S","E","L","S","H","A","D","E"," ",
     & "n","a","m","e"," "," "," "," "," ",
     & "h","e","i","g","h","t"," "," "," ",
     & "p","a","t","t","e","r","n"," "," ",
     & "t","e","x","t","f","i","e","l","d",
     & "n","o","l","i","n","e"," "," "," ",
     & "s","e","l","e","c","t"," "," "," ",
     & "r","e","f","c","l","e","a","r"," ",
     & "s","h","a","d","e","a","l","l"," ",
     & "s","e","l","s","h","a","d","e"," "/
      name=" "
      height=0.14
      space=height/5.
      numpat=10
      noline=0
      noname=0
      noselect=0
      noclear=0
      noshade=0
      nochar=0
      noselshade=0
      item=0
      iplace=6
      ilast=81
      do 10 i=1,10
   10 ipat(i)=i
      do 20 i=11,20
   20 ipat(i)=0
      do 30 i=1,8
   30 kplotfield(i)=0
c
c    FIND THE LAST NON-BLANK CHARACTER ON THE RECORD
   35 do 40 i=iplace,80
      ilast=ilast-1
   40 if (icom(ilast) .ne. " ")  go to 60
      write (10,50)  icom
   50 format (" THE FIELD CONTAINS ALL BLANKS",/,1x,80a1)
      return
```

```
c
c     FIND THE FIRST NON-BLANK CHARACTER IN THIS FIELD
  60  do 70 m=iplace,ilast
  70  if (icom(m) .ne. " ")  go to 80
      m=ilast
  80  iplace=m
c     FIND THE LAST NON-BLANK CHARACTER IN THIS FIELD
      do 90  ic=iplace,ilast
  90  if (icom(ic) .eq. ",")  go to 100
      ic=ilast
      go to 110
 100  ic=ic-1
c
c     IDENTIFY THE KEYWORD
c
c     NAME OF PLOT FILE
 110  do 120  i=1,4
 120  if ((icom(iplace+i-1) .ne. keyword(1,i))   .and.
     &   (icom(iplace+i-1) .ne. keyword(10,i)))   go to 200
      iplace=iplace+5
      do 130  m=iplace,ic
 130  if (icom(m) .ne. " ")  go to 135
      m=ic
 135  iplace=m
      k=iplace-1
      m=ic-k
      if (m .gt. 0)  go to 145
      write (10,140)
 140  format ("NAME HAS NO CHARACTERS!!")
      go to 170
 145  if (m .le. 20)  go to 160
      write (10,150) (icom(i),i=k,m)
 150  format (" NAME IS MORE THEN 20 CHARACTERS LONG!!",/,1x,80a1)
      go to 170
 160  backspace in1
      fmt2="(1h(,i2,3hx,a,i2,1h))"
      if (m .lt. 10)  fmt2="(1h(,i2,3hx,a,i1,1h))"
      encode (fmt1,fmt2) k,m
      read (in1,fmt1)  name
      fmt1="(4hfile,i2)"
      if (in2 .le. 9)  fmt1="(5hfile0,i1)"
      encode (name1,fmt1)  in2
      call io_call ("attach",name1,"vfile_ ",name)
      call io_call ("open",name1,"si  ")
      noname=1
c
c     LOOK AT THE NEXT KEYWORD
 170  iplace=ic+2
      if (ilast-iplace)  180,180,60
 180  if (icom(ilast) .ne. ",")  go to 790
      read (in1,190)  icom
 190  format (80a1)
      ilast=81
      iplace=1
      go to 35
c
c     HEIGHT OF THE CHARACTERS
 200  do 210  i=1,6
 210  if ((icom(iplace+i-1) .ne. keyword(2,i))  .and.
      &   (icom(iplace+i-1) .ne. keyword(11,i)))   go to 280
      iplace=iplace+7
c
c     CHECK FOR A VALID DATA WORD
      if (ic-iplace)  170,220,220
 220  do 230 num=iplace,ic
 230  if (icom(num) .ne. " ")  go to 240
      go to 170
 240  iplace=num
      num=ic-iplace+1
      if (num .le. 15)  go to 260
      write (10,250) (icom(i),i=iplace,ic)
 250  format (" PLOT-HEIGHT: THIS FIELD HAS TOO MANY CHARACTERS",
               & /,1x,70a1)
      go to 170
 260  l=0
      do 270  i=iplace,ic
      l=l+1
 270  id(l)=icom(i)
      call find_number (id,num,rnum,istat)
      if (istat .ne. 0)  go to 170
      height=rnum
      space=height/5.
      go to 170
c
c     PATTERN SEQUENCE FOR SHADING
 280  do 290  i=1,7
 290  if ((icom(iplace+i-1) .ne. keyword(3,i))  .and.
       &  (icom(iplace+i-1) .ne. keyword(12,i)))   go to 470
      iplace=iplace+8
      kount=0
      if (ic-iplace)  300,320,320
 300  write (10,310) (icom(i),i=iplace,ic)
 310  format(" PLOT-PATTERN: THE PATTERN COUNT HAS AN ERROR!!",
             & /,1x,70a1)
      go to 170
c
 320  do 330  num=iplace,ic
 330  if (icom(num) .ne. " ")  go to 350
      if (kount .eq. 0)  go to 300
 335  write (10,340) (icom(i),i=iplace,ic)
 340  format (" PLOT-PATTERN: THE FIELD CONTAINS AN ERROR AND WILL
              &                   BE SET TO THE DEFAULT VALUE",/,1x,70a1)
      go to 400
 350  iplace=num
      num=ic-iplace+1
      if (num .le. 15)  go to 360
      if (kount .eq. 0)  go to 300
      go to 335
c
 360  l=0
      do 370  i=iplace,ic
      l=l+1
 370  id(l)=icom(i)
      call find_number (id,num,rnum,istat)
      if (istat .eq. 0)  go to 380
      if (kount .eq. 0)  go to 300
      go to 335
 380  if (kount .gt. 0)  go to 390
      numpat=rnum+0.5
      go to 400
 390  ipat(kount)=rnum+0.5
 400  kount=kount+1
      if (kount .le. numpat)  go to 420
 405  do 410  i=kount,20
 410  ipat(i)=0
      go to 170
c
 420  iplace=ic+2
      if (ilast .le. iplace)  go to 405
      do 430  m=iplace,ilast
 430  if (icom(m) .ne. " ")  go to 440
      m=ilast
 440  iplace=m
      do 450  ic=iplace,ilast
 450  if (icom(ic) .eq. ",")  go to 460
      ic=ilast
      go to 320
 460  ic=ic-1
      go to 320
c
c    TEXT FIELDS TO BE PLOTTED
 470  do 480 i=1,9
 480  if ((icom(iplace+i-1) .ne. keyword(4,i))  .and.
       &  (icom(iplace+i-1) .ne. keyword(13,i)))   go to 670
      iplace=iplace+10
      kount=0
      if (ic-iplace) 490,510,510
 490  write (10,500) (icom(i),i=iplace,ic)
 500  format (" PLOT-TEXTFIELD: THE FIELD COUNT HAS AN ERROR!!")
      go to 170
c
 510  do 520  num=iplace,ic
 520  if (icom(num) .ne. " ")  go to 550
      if (kount .eq. 0)  go to 490
 530  write (10,540) (icom(i),i=iplace,ic)
 540  format (" PLOT-TEXTFIELD: THIS FIELD CONTAINS AN ERROR AND
     &                         WILL NOT BE PLOTTED!!")
      go to 600
 550  iplace=num
      num=ic-iplace+1
      if (num .le. 15)  go to 560
      if (kount .eq. 0)  go to 490
      go to 530
c
 560  l=0
      do 570  i=iplace,ic
      l=l+1
 570  id(l)=icom(i)
      call find_number (id,num,rnum,istat)
      if (istat .eq. 0)  go to 580
      if (kount .eq. 0)  go to 490
      go to 530
 580  if (kount .gt. 0)  go to 590
      numfield=rnum+0.5
      go to 600
 590  num=rnum+0.5
      kplotfield(num)=kount
 600  kount=kount+1
      if (kount .le. numfield)  go to 620
 610  nochar=1
      go to 170
c
 620  iplace=ic+2
      if (ilast .le. iplace)  go to 610
      do 630 m=iplace,ilast
 630  if (icom(m) .ne. " ") go to 640
      m=ilast
 640  iplace=m
      do 650  ic=iplace,ilast
 650  if (icom(ic) .eq. ",")  go to 660
      ic=ilast
      go to 510
 660  ic=ic-1
      go to 510
c
c     NOLINE OPTION
 670  do 680  i=1,6
 680  if ((icom(iplace+i-1)  .ne. keyword(5,i))  .and.
       &  (icom(iplace+i-1)  .ne. keyword(14,i)))  go to 690
      noline=1
      go to 170
c
c     SELECT OPTION
 690  do 700 i=1,6
 700  if ((icom(iplace+i-1) .ne. keyword(6,i))  .and.
       &  (icom(iplace+i-1) .ne. keyword(15,i)))   go to 710
      noselect=1
      go to 170
c
c     CLEAR REFERENCE NUMBER
 710  do 720  i=1,8
 720  if ((icom(iplace+i-1) .ne. keyword(7,i))  .and.
      &   (icom(iplace+i-1) .ne. keyword(16,i)))  go to 730
```

```
       noclear=1
       go to 170
c
c      SHADE ALL OPTION
 730   do 740  i=1,8
 740   if ((icom(iplace+i-1) .ne. keyword(3,i))  .and.
     &  (icom(iplace+i-1) .ne. keyword(17,i)))  go to 750
       noshade=1
       go to 170
c
c      SHADE SELECTIVELY OPTION
 750   do 760  i=1,8
 760   if ((icom(iplace+i-1) .ne. keyword(9,i))  .and.
     &  (icom(iplace+i-1) .ne. keyword(18,i)))  go to 770
       noshade=1
       noselshade=1
       go to 170
c
c      NO MATCH FOR A KEYWORD
 770   write (10,780) (icom(i),i=iplace,ic)
 780   format (" PLOT: THIS KEYWORD IS NOT VALID!!")
       go to 170
c
c      READ SELECT RECORDS AND END PLOT KEYWORD
 790   kount=1
 800   read (in1,810) (iwhat(kount,i),i=1,2),jwhat(kount)
 810   format (2a5,i5)
       if (((iwhat(kount,1) .eq. "END P") .and. (iwhat(kount,2) .eq.
       "LOT; ")) .or.((iwhat(kount,1) .eq. "end p") .and. (iwhat(kount,2)
       .eq. "lot; "))   go to 820
       kount=kount+1
       go to 800
c
c      START TO PLOT THE FILE
 820   kount=kount-1
       if (noname .eq. 1)  go to 840
       write (10,830)
 830   format (" PLOT: NO PLOT FILE???")
       return
c
c      READ IN DATA VALUES FOR AN OUTLINE
 840   read (in2,850,end=1050) (kfield(i),i=1,8)
 850   format (8a5)
       backspace in2
       read (in2,855) ((char(i,j),j=1,5),i=1,8)
 855   format (8(5a1))
       backspace in2
       read (in2,860) isfno
 860   format (15x,i5)
       isfno=isfno-1
       read (in2,870,end=1030) xpos,ypos,(xx(i),yy(i),i=1,isfno)
 870   format (12f6.3)
c
```

```
c     CHECK FOR A MATCH WITH THE SELECTED FEATURES
       isel=0
       key=0
       if (kount .eq. 0)  go to 900
       do 880  i=1,kount
 880   if ((kfield(1) .eq. iwhat(i,1)) .and. (kfield(3) .eq. iwhat(i,2)))
     &   go to 890
       go to 900
 890   isel=1
       key=jwhat(i)
c
c     CHECK IF THIS OUTLINE IS TO BE PLOTTED
 900   if ((noselect .eq. 1) .and. (isel .eq. 0))  go to 840
c
c     CHECK FOR SINGLE POINT
       if (isfno .gt. 1)  go to 905
       call symbol (xx(1),yy(1),height,isym,0.,-1)
       go to 960
c
c     CHECK FOR THE NOLINE OPTION
 905   if (noline .eq. 1)  go to 920
       call plot (xx(1),yy(1),3)
       do 910 i=1,isfno
 910   call plot (xx(i),yy(i),2)
c
c     CHECK FOR SHADING
 920   if (noshade .ne. 1)  go to 960
       if ((noselshade .eq. 1) .and.
     &  ((isel .eq. 0) .or. (key .eq. 0)))  go to 960
c
c     CHECK FOR CLEARING THE REFERENCE AREA AROUND THE CHARACTERS
       ne(1)=isfno
       ne(2)=0
       numarea=1
       if ((noclear .eq. 0) .or. (nochar .eq. 0))  go to 950
       numvert=0
       numhorz=0
       do 940  i=1,8
       if (kplotfield(i) .eq. 0)  go to 940
       numvert=numvert+1
       icheck=5
       do 930 j=1,5
       if ((char(i,j) .ne. " ") .and. (char(i,j) .ne. "0")) go to 935
 930   icheck=icheck-1
 935   if (icheck .gt. numhorz)  numhorz=icheck
 940   continue
       if (numhorz .eq. 0)  go to 950
       k=isfno+1
       xx(k)=xpos-0.04
       xx(k+1)=xx(k)
       xx(k+2)=xpos+numhorz*height+0.02+space
       xx(k+3)=xx(k+2)
       yy(k)=ypos-0.04
```

```
      yy(k+1)=ypos+numvert*(height+space)-space+0.06
      yy(k+2)=yy(k+1)
      yy(k+3)=yy(k)
      ne(2)=4
      numarea=2
c
c     SHADE THE OUTLINE
 950  item=item+1
      if (item .gt. numpat)  item=1
      if ((ipat(item) .le. 0) .or. (ipat(item) .gt. 20))  go to 950
      num=ipat(item)
      if ((key .gt. 0) .and. (key .le. 20))  num=key
      call set_shade (num,jshade)
      call tone(0.,0.,jshade,-16)
      call tone (xx,yy,ne,numarea)
c
c     CHECK FOR CHARACTER PLOTTING
 960  if (nochar .eq. 0)  go to 840
      do 1025  i=1,8
      do 970  j=1,8
 970  if (kplotfield(j) .eq. i)  go to 980
      go to 1025
 980  num=0
      do 990 k=1,5
      if (char(j,k) .ne. " ")  go to 1000
 990  num=num+1
1000  if ((num .eq. 0) .or. (kfield(j) .eq. "    0"))  go to 1025
      n=5-num
      if (n .eq. 5)  go to 1015
      do 1010 m=1,n
1010  itext(m)=char(j,m+num)
1015  posx=xpos
      iscale=height*16+0.5
      do 1020 k=1,n
      call letter (1,iscale,0,posx,ypos,itext(k))
1020  posx=posx+height+space
      ypos=ypos-height-space
1025  continue
      go to 840
c
c     END OF FILE REACHED
1030  write (10,1040)
1040  format ("PLOT: END OF FILE REACHED
     &                WHEN TRYING TO READ A DATA RECORD")
      go to 1070
c
c  ALL FINISHED
1050  write (10,1060)  name
1060  format ("FINISHED PLOTTING ",a20)
1070  call closef (in2)
      return
      end
```

SUBROUTINE NAME: PATTERN

Author: Lawrence Balcerak
Purpose of the program: **pattern** sets the shading pattern variables to user-defined values.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call pattern
Arguments: None
Subroutines called: None
Common data referenced: ***/pat/***
Input files: None
Output files: None
Arrays used: ***ip1**(16)*, ***ip2**(16)*, ***ip3**(4)*, ***ip4**(4)*, ***ip5**(16)*, ***ip6**(16)*, ***ip7**(16)*, ***ip8**(16)*, ***ip9**(16)*, ***ip10**(16)* (shading pattern arrays)
Called by: **index_versatec**, **verplot**
Error checking and reporting: None
Constants: None
Program logic:
1. Set all elements of the shading pattern arrays to those integer values that will give the bit patterns desired.

```
              subroutine pattern
              common /pat/ ip1(16),ip2(16),ip3(4),ip4(4),ip5(16),ip6(16),
                         & ip7(16),ip8(16),ip9(16),ip10(16)
              do 10 I=1,4
              ip3(i)=0
              ip4(i)=0
10            continue
              do 20 i=1,16
              ip1(i)=0
              ip2(i)=0
              ip5(i)=0
              ip6(i)=0
              ip7(i)=0
              ip8(i)=0
              ip9(i)=0
              ip10(i)=0
20            continue
c
              ip1(1)=4*16**4+1
              ip1(5)=16*ip1(1)
              ip1(9)=8*16**6+2*16**2
              ip1(13)=16*ip1(9)
c
              ip2(4)=ip1(13)
              ip2(8)=ip1(9)
              ip2(12)=ip1(5)
              ip2(16)=ip1(1)
c
              ip3(1)=4*16**4+1
c
              ip4(3)=4*16**8+2*16**6+16**4+8*16
c
              ip5(1)=4*16**7
              ip5(2)=16**8+16**7
              ip5(4)=2*16**8+8*16**6
              ip5(6)=ip5(2)
              ip5(7)=ip5(1)
              ip5(9)=16**3
              ip5(10)=4*16**3+4*16**2
              ip5(12)=8*16**3+2*16**2
              ip5(14)=ip5(10)
              ip5(15)=ip5(9)
c
              ip6(1)=8*16**7+8*16**6+4*16**5+4*16**4+2*16**3+2*16**2+16+1
```

```
c
          ip7(4)=16**4+8*16
          ip7(8)=4*16**8+4*16**7+2*16**6+16**4+16**3+8*16
          ip7(12)=4*16**8+2*16**6
          ip7(16)=ip7(8)
c
          ip8(1)=ip6(1)
          ip8(9)=ip6(1)
c
          ip9(2)=ip5(2)
          ip9(4)=ip5(1)
          ip9(6)=ip5(2)
          ip9(8)=4*16**8+2*16**6+16**4+8*16
          ip9(10)=ip5(10)
          ip9(12)=ip5(9)
          ip9(14)=ip9(10)
          ip9(16)=ip9(8)

          ip10(2)=ip5(9)
          ip10(4)=16**4+5*16**3+4*16**2+8*16
          ip10(6)=ip10(2)
          ip10(8)=4*16**7+16**3
          ip10(10)=ip5(7)
          ip10(12)=5*16**8+5*16**7+2*16**6
          ip10(14)=ip10(10)
          ip10(16)=ip10(8)

          return
          end
```

PROGRAM NAME: PIN90

Author: Pearl Porter

Purpose of the program: **pin90** plots map indices interactively on the Tektronix terminal. The user has several options: plotting the entire United States, plotting as many as 10 individual States, plotting the grid file, plotting input files from GRASP, and getting an enlargement of a specific area.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: pin90

Arguments: None

Subroutines called: **initt**, **anmode**, **erase**, **movabs**, **dwindo**, **swindo**, **movea**, **drawa**, **hdcopy**, **finitt** (all Tektronix routines), **io** (Multics), **grid**, **plocv**, **enlrg**, **indiv**

Common data referenced: *x1*, *y1*, *x2*, *y2*

Input files:

stat90 used on unit 14 (*file14*), file of x, y coordinates

Files from GRASP used on unit 15 (*file15*), which plot numbers, symbols, and outlines

Grid file used on unit 16 (*file16*), which is used to plot the grid on the map

Output files: None

Arrays used: None

Called by: None

Error checking and reporting: If program requests a digit (1 or 2) from the user, it will loop until a number (1 or 2) is typed in by the user.

Constants: None

Program logic:

1. Prompt:

   NEED SYMBOL CODES? (ENTER Y FOR YES)

   If no symbol codes are needed, program goes to step 4.
2. The symbols and their corresponding numbers will be printed on the screen.
3. Prompt:

   TAP 1 AND RETURN KEY WHEN READY
4. The screen will be erased.
5. Prompt:

   ENTER SYMBOL NUMBER AND FILE TO BE PLOTTED:

   This refers to the input files from GRASP. As many as five files and symbols may be entered. They will be read into *file15*. Eight is the maximum number of characters for a file name.

6. In reference to the files just read, the user has three options.
   Prompt:
   FOR SYMBOL AND NUMBERS (WITH PLOTTING), TYPE 1
   FOR SYMBOL AND/OR OUTLINE (NO NUMBERS), TYPE 2
   FOR NUMBERS ONLY (NO SYMBOLS OR PLOTTING), TYPE 3
7. Screen is erased.
8. Define screen and virtual window.
9. The user is prompted by a series of questions at this time because no communication can take place between computer and user after the plotting starts without destroying the screen. See steps 10–14.
10. Prompt:
    ENTER TITLE FOR MAP:
11. Prompt:
    TO PLOT INDIVIDUAL STATES, ENTER 1–FOR ENTIRE U.S. ENTER 2
12. Prompt:
    TO PLOT COUNTIES ENTER 1 FOR SOLID LINE, 2 FOR DOTTED LINE, ELSE ZERO
13. Prompt:
    TO PLOT GRID, ENTER 1
14. Prompt:
    IF YOU WANT A HARD COPY UPON COMPLETION, TYPE C
15. If the user entered 1 in response to step 11, the program calls subroutine **indiv**. The program then goes to step 19.
16. If the user did not enter 1 or 2 for step 11, a message is printed on the screen:
    AT THE PRESENT TIME, THE FILE YOU WISH DOES NOT EXIST PLEASE ENTER 1 OR 2 FOR SECOND STATEMENT
    Program returns to step 10. If 2 was entered, program goes to next step.
17. The program writes the title for the map, draws the borders, and uses ***stat90***, ***file14***, to plot the States.
18. If 1 was entered in response to step 13, program calls **grid** subroutine and returns to next step.
19. If *iji* = 0, program goes to 20 (*iji* = number of files entered in step 5)
    If *iji* is not equal to 0, read first file name into ***file15*** and call subroutine **plocv**.
    Continue this step until the number of files entered in step 5 *iji* = number of files read into ***file15*** *iji*.
20. If C was entered in step 14 or 23, a hard copy will be made automatically.
21. If *nl* = 1, go to step 26. If *nl* does = 1, this indicates an enlargement or individual plotting has already been completed or the option to do so has already occurred.
22. Prompt:
    FOR AN ENLARGEMENT OF PART OF THIS PLOT, TYPE Y
23. Prompt:
    FOR A HARD COPY AFTER ENLARGEMENT, TYPE C
24. *nl* = 1 indicates that an enlargement option has been found.
25. If Y was entered for step 22, call subroutine **enlrg** and upon return, go to step 19.
26. Detach and close files.

```
c*****                PIN90                    *****

cPLOT MAP INDICES ON TEKTRONIX****
c       U.S. Geological Survey
c        Program name - pin90
c        INPUT:
c          stat90.pat = file14
        dimension lead(20),xx(6),yy(6)
        dimension jsym(9),isymb(5)
        external io (descriptors)
        character filename*8(5)
        common x1,y1,x2,y2
c
        data jes/"y"/,kop/"c"/,iblk/"  "/
        data jsym/35,36,37,38,42,43,45,79,111/
        data izero/00/
c
```

```
c
       call initt(960)
c
c
       ichar=43
c
c      ASSIGN AND OPEN FILES
       call io ("attach","file14","vfile_ ","stat90.pat","-append")
       call io ("open","file14","si")
c
       call anmode
       print ,"Need symbol codes? (enter y for yes) "
       read (0,100)irep
100    format (a1)
      if (irep .ne. jes)go to 250
       do 200 i=1,9
       istb = jsym(i)
       ile = jsym(i)*2**27
c   The above computation was made to shift the symbol to the leftmo
\cst position.
       call anmode
       write (0,150) istb,ile
150    format (2x,i3,3x,a1)
200    continue
c   Pause in execution so user can look at symbols and corresponding
\c numbers.
       print ,"Tap 1 and return key when ready"
       read (0,600)iredy
250    call erase
c      call movabs(30,725)
c      call anmode
c      print ,"Enter state id number"
c      read(0,300)istate
c300   format(a4)
c
c
c      REQUEST INPUT FILE FOR PLOTTING
c
350    continue
       kk=0
       ijj=0
       iji=0
       call anmode
       call movabs(30,725)
400    print ,"Enter symbol number and file to be plotted: "
c iji will equal 1 more than the number of files read.
       iji = iji+1
       read(0,450)isym,filename(iji)
450    format (i2,a8)
       if (isym .eq. izero) go to 500
       isymb(iji)=isym
       go to 400
c
500    continue
```

```
c
       call movabs(30,625)
       call anmode
c   User is given an option as to what he wants on the map.
       print ,"For symbol and numbers (with plotting), type 1/
             & For symbol and/or outline (no numbers), type 2/
             & For numbers only (no symbols or plotting), type 3"
       read (0,600) idec
       call erase
c
c      SET ORGIN ON PLOTTER
c
       x1=.5
       x2=23.
       y1=1.
       y2=16.
c      Define virtual window
       call dwindo(x1,x2,y1,y2)
c
c      Define Screen window
       call swindo(0,1023,0,780)
c
550    n1=0
       call movabs(30,750)
       call anmode
560    print ,"Enter title for map: "
       read(0,570)lead
570    format (20a4)
       print ,"To plot individual states enter 1--for entire U.S. en
\cter 2"
       read(0,600)istat
c      print ,"To plot counties enter 1 for solid line,/
c      2 for dotted line, else zero "
c      read (0,600) icoun
c      print ,"To plot grid enter 1 "
c      read (0,600)igrid
600    format (i1)
       print ,"If you want a hard copy upon completion, type c "
       read (0,100) icopy
       call erase
c   If istat=1, user will choose up to 10 states to be plotted.
       if (istat .eq. 1) go to 1300
       call movabs(30,760)
       call anmode
c   Lead contains title for map.
       write (0,610)lead
610    format (1x,20a4)
c
       if (istat .ne. 2) go to 1400
c   If istat=2, the entire U.S. will be plotted.
c
c
c    DRAW BORDER FOR MAP
       call movea(x1,y1)
```

```
        call drawa(x2,y1)
        call drawa(x2,y2)
        call drawa(x1,y2)
        call drawa(x1,y1)
c
c       PLOT DATA FROM FIRST SOURCE.
c
c    Read header information.
700     read (14,750,end=1100)if,ifno,isf,isfno,ifl
750     format (5i5)
c    ISFNO is the number of x-y coordinates.
        ie=isfno
        if (isfno .ge. 6) ie=6
        call anmode
c    Read x-y coordinates.
        read (14,800) (xx(i),yy(i),i=1,ie)
800     format (12f6.3)
c
825     call movea (xx(1),yy(1))
        do 850 k=1,ie
        call drawa(xx(k),yy(k))
850     continue
        isfno=isfno-6
        if (isfno)700,700,900
900     if (isfno-6)950,950,1000
950     ie = isfno
1000    read (14,800,end=1100) (xx(i),yy(i),i=1,ie)
        do 1050 k=1,ie
        call drawa(xx(k),yy(k))
c    Draw until isfno(no. of coordinates) has been exhausted.
1050    continue
        if (isfno-6)700,700,850
c
c
c   ATTENTION: When we get grid file, change end=1100 to end=1075
c1075   if (igrid .ne. 1) go to 1100
c       call io ("attach","file16","vfile_","grid90","-append")
c       call io ("open","file16","si")
c
c
c    iji = number of files to be plotted.
1100    if (iji .eq. 0) go to 1200
c
c   PLOT MAIN FILE
1150    ijj = ijj+1
        if (ijj .eq. iji) go to 1200
        isym = isymb(ijj)
        call io ("attach","file15","vfile_",filename(ijj),"-append")
        call io ("open","file15","si")
        call plocv(isym,n1,idec)
        go to 1150
c
c
```

```
1200    if (icopy .ne. kop) go to 1250
        call hdcopy
c   If nl=1, the enlargement or individual plotting has already been
\c completed.
1250    if (nl .eq. 1) go to 1500
        call movabs(30,730)
        call anmode
        print ,"For an enlargement of part of this plot, type y "
        read (0,100)irep
        print ,"For a hard copy after enlargement, type c "
        read (0,100) icopy
        nl = 1
c   Set ijj=0 as it has been incremented previously.
        ijj = 0
        if (irep .ne. jes) go to 1500
        call enlrg(14)
        go to 1100
c
1300    nl = 1
        ijj = 0
        call indiv(lead)
        go to 1100
c
c
c
1400    print ,"At the present time, the file you wish does not exist
\c"
        print ,"Please enter 1 or 2 for the second statement"
        go to 560
1500    call io ("close","file14")
        call io ("detach","file14")
c       call io ("close","file16")
c       call io ("detach","file16")
        call finitt(0,0)
        end
```

SUBROUTINE NAME: ENLRG

Author: Pearl Porter
Purpose of the program: **enlrg** enlarges a part of the plotting on the screen that the user defines by means of the crosshair cursor.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call enlrg
Arguments: None
Subroutines called: **movabs**, **anmode**, **vcursr**, **dwindo**, **swindo**, **erase**, **movea**, **drawa**, **ancho** (all Tektronix routines)
Common data referenced: *x1*, *y1*, *x2*, *y2*
Input files: ***stat90*** used on unit 14 (***file14***)
Output files: None
Arrays used: *xx*(*6*), *yy*(*6*), *ist*(*10*)
Called by: **pin90**
Error checking and reporting: Located in program **pin90**
Constants: None
Program logic:
1. The virtual screen is redefined by means of a Tektronix plotting routine called **vcursr**. The user positions the cursor at the lower left of the desired area, and its screen coordinates are transmitted to the computer by typing C. Then the user positions the cursor at the upper right of the desired area and again types C.
2. The screen is automatically erased.
3. A border is drawn for the map.
4. ***file14*** is again read, and the States within the limits of the specified virtual screen are plotted.

```
c   **** SUBROUTINE ENLRG.FORTRAN ****
        subroutine enlrg
c   This routine enlarges a portion of the screen as
c       defined by the cross-hair cursors.
c
c
        dimension xx(6),yy(6),ista(10)
        common x1,y1,x2,y2
        data jes/"y"/,iblk/"  "/,kop/"c"/
        ichar = 43
10      rewind 14
        call movabs(30,690)
        call anmode
c    Redefine graphic area by using vcursr routine.
        write (0,20)
20      format("Position cursor at lower left of desired area, type c")
        call vcursr(ichar,x1,y1)
        call movabs(30,680)
        call anmode
        write (0,30)
30      format("Position cursor at upper right of desired area,type c ")
        call vcursr(ichar,x2,y2)
        call dwindo(x1,x2,y1,y2)
        call swindo(0,1023,0,780)
c
c
        call erase
c
c   DRAW BORDER FOR MAP
        call movea(x1,y1)
        call drawa(x2,y1)
        call drawa(x2,y2)
        call drawa(x1,y2)
        call drawa(x1,y1)
c
        call anmode
c    Read header information.
70      read (14,80,end=180)if,ifno,isf,isfno,if1
80      format (5i5)
c
90      ie = isfno
        if (isfno .ge. 6) ie=6
        call anmode
c    Read x-y coordinates.
        read (14,100) (xx(i),yy(i),i=1,ie)
100     format (12f6.3)
        if (isfno .ge. 3) go to 105
        call movea(xx(2),yy(2))
        call ancho(ichar)
        go to 70
105     call movea(xx(1),yy(1))
        do 110 k=1,ie
        call drawa(xx(k),yy(k))
110     continue
```

```
      isfno=isfno-6
      if (isfno)70,70,120
120   if (isfno-6)130,130,140
130   ie = isfno
c   Read x-y coordinates.
140   read (14,100,end=160) (xx(i),yy(i),i=1,ie)
      do 150 k=1,ie
      call drawa(xx(k),yy(k))
150   continue
c   Continue drawing until isfno(no. of coordinates) has been exhausted.
      if (isfno-6)70,70,110
160   continue
c
180   return
      end
```

SUBROUTINE NAME: INDIV

Author: Pearl Porter
Purpose of the program: **indiv** is called when 1-10 States are to be plotted. This routine negates plotting the entire United States. If will enlarge the maps of individual States as an option.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call indiv (lead)
Arguments: *lead*–Contains the title of the map
Subroutines called: **movabs**, **anmode**, **erase**, **movea**, **drawa** (all Tektronix routines), **min_max**
Common data referenced: None
Input files: ***stat90*** used on unit 14 (*file14*)
Output files: None
Arrays used: *xx*(6), *yy*(6), *ista*(10), *lead*(20)
Called by: **pin90**
Error checking and reporting: Located in **pin90**
Constants: None
Program logic:
1. An array *ista* is loaded with blanks.
2. The user is asked to type the corresponding 2-digit number for the States to be plotted (limit of 10 in ascending order).
3. The numbers are stored in *ista*.
4. Call **min_max** subroutine to find the minimum and maximum range for the x and y coordinates.
5. The screen is erased.
6. The title of the map is written on the screen and the border is drawn.
7. *file14* is read until *ista(j)* equals *ifno* (found in header record). The coordinates are then plotted until *isfno* (number of x, y coordinates) has been exhausted. The program continues to read through the file until the end of file is reached or *ista(j)* equals blanks.

```
c ****SUBROUTINE INDIV.FORTRAN****
      subroutine indiv(lead)
c  This routine is used when only 1-10 states are to be plotted.
c     The user requests up to 10 states by number and these are
c     stored in an array called ista.  In order to use the entire
c     screen, min_max routine is called to redefine the virtual
c     screen thus giving an enlargement of the states requested.
c
      dimension xx(6),yy(6),ista(10),lead(20)
      data iblk/"  "/,kop/"c"/
      data izero/00/
c
      nn3 = 993
c
c   Load ista with blanks
```

```
10      do 20 j=1,10
        ista(j) = iblk
20      continue
c
        call movabs(30,690)
        call anmode
        write (0,30)
30      format ("Give code number of each state to be plotted
                 Limit of 10 codes in ASCENDING order
                 Must be a 2 digit number, 01-51")
        read (0,40) (ista(j),j=1,10)
40      format (10i2)
c    The min_max routine will find the minimum and maximum coordinates
c       for the states requested and redefine the virtual window.
        call min_max(ista,x1,x2,y1,y2)
        call erase
        call movabs(500,750)
        call anmode
c    Lead is the title of the map to be plotted.
        write (0,45) lead
45       format (1x,20a4)
c
c    DRAW BORDER FOR MAP
        call movea(x1,y1)
        call drawa(x2,y1)
        call drawa(x2,y2)
        call drawa(x1,y2)
        call drawa(x1,y1)
c
50      do 160 j=1,10
c    Read header information.
60      read (14,70,end=180)if,ifno,isf,isfno,if1
70      format (5i5)
c
        if (ista(j) .eq. izero) go to 180
        if (ista(j) .ne. ifno) go to 60
        if (if .ne. nn3) go to 60
c    Check if=993 to insure information from header record is
c       being compared rather than erronously matching the
c       state number against the coordinates.
80      ie = isfno
        if (isfno .ge. 6) ie=6
c    Read x-y coordinates.
        read (14,90) (xx(i),yy(i),i=1,ie)
90      format (12f6.3)
        if (isfno .ge. 3) go to 100
        call movea(xx(2),yy(2))
        go to 60
100     call movea(xx(1),yy(1))
        do 110 k=1,ie
        call drawa(xx(k),yy(k))
110     continue
        isfno=isfno-6
        if (isfno)160,160,120
```

```
120     if (isfno-6)130,130,140
130     ie = isfno
140     read (14,90,end=160) (xx(i),yy(i),i=1,ie)
        do 150 k=1,ie
        call drawa(xx(k),yy(k))
150     continue
c   Continue drawing ntil isfno(no. of coordinates) has been exhaustea.
        if (isfno-6)160,160,110
160     continue
c
180     return
        end
```

SUBROUTINE NAME: MIN_MAX

Author: Pearl Porter

Purpose of the program: **min_max** routine reads through the individual States requested for plotting and compares each x and y coordinate to find the minimum and the maximum x and y coordinates. The program then redefines the virtual window.

Data base: Geoindex

Computer: Honeywell Series 60 (level 68)

Operating system: Multics

Calling sequence: call min_max (ista,x1,x2,y1,y2)

Arguments:

ista – Array containing the numbers of the States to be plotted

x1 – Used to return the minimum x coordinate

x2 – Used to return the maximum x coordinate

y1 – Used to return the minimum y coordinate

y2 – Used to return the maximum y coordinate

Subroutines called: **dwindo** (Tektronix routine)

Common data referenced: None

Input files: None

Output files: None

Arrays used: None

Called by: **indiv**

Error checking and reporting: Located in **pin90**

Constants: *nn3*

Program logic:

1. Load *x1* with the maximum x coordinate and *x2* with the minimum x coordinate.
   Load *y1* with the maximum y coordinate and *y2* with the minimum y coordinate.
2. Read through *file14* until *ista(j)* equals *ifno*.
3. Compare each x coordinate to *x1* and *x2*.
   If it is less than *x1*, load *x1* with the value of the x coordinate.
   If the x coordinate is greater than *x2*, load *x2* with the value of the x coordinate.
   The same logic applies to the y coordinate.
4. Compare each x and y coordinate until the number in *isfno* (header record) has been exhausted.
5. If *ista(j)* is blank, go to next step. Otherwise, go to step 3.
6. Call **dwindo** (Tektronix routine) using new values from step 3 for *x1*, *x2*, *y1*, *y2* to define the virtual window.
7. Return

```
c   ****SUBROUTINE MIN_MAX.FORTRAN****
       subroutine min_max(ista,x1,x2,y1,y2)
c
c   This routine reads through the coordinates for the inaiviaual
c       states requested for plotting and compares each xx coordinate
c       against x1 and x2 to find the minimum and maximum coor.
c       It does the same to the yy coordinates.  x1 and y1 are set to
c       the maximum and x2 and y2 are set to the minimum before the
c       compares are maae.
c
       dimension xx(6),yy(6),ista(10)
        data izero/00/
c
c
```

```
        nn3 = 993
        x1 = 23.
        x2 = .5
        y1 = 16.
        y2 = 1.
        do 110 j=1,10
10      read (14,20,end=120) if,ifno,isf,isfno,if1
20      format(5i5)
c    If ista = 0, all the requested data has been read.
        if (ista(j) .eq. izero) go to 120
        if (ista(j) .ne. ifno) go to 10
        if (if .ne. nn3) go to 10
c    Check if=993 to insure this record is a header record.
c
        ie = 6
c    Read x-y coordinates.
30      read (14,40,end=120) (xx(i),yy(i),i=1,ie)
40      format (12f6.3)
c
c    Read each xx and yy coordinate and compare x1-2 and
c       y1-2 to find the minimum and maximum.
        do 50 i=1,ie
        if (xx(i) .lt. x1)  x1 = xx(i)
        if (xx(i) .gt. x2)  x2 = xx(i)
        if (yy(i) .lt. y1)  y1 = yy(i)
        if (yy(i) .gt. y2)  y2 = yy(i)
50      continue
c
        isfno = isfno-6
        if (isfno)110,110,60
60      if (isfno-6)70,70,30
70      ie = isfno
        go to 30
c
110     continue
c    Define virtual window from coordinates stored during min_max routine.
120     call dwindo(x1,x2,y1,y2)
        rewind 14
        return
        end
```

SUBROUTINE NAME: GRID

Author: Pearl Porter
Purpose of the program: **grid** will plot the grid file, *file16*, on the map drawn by **pin90**
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call grid
Arguments: None
Subroutines called: **movea**, **anmode**, **ancho**, **drawa** (all Tektronix routines)
Common data referenced: *x1*, *y1*, *x2*, *y2*
Input files: Grid file used on unit 16 (*file16*)
Output files: None
Arrays used: *xx(6)*, *yy(6)*
Called by: **pin90**
Error checking and reporing: Located in program **pin90**
Constants: None
Program logic:

1. *iskp* is set to 9, and a count is kept so that only every 10th record is plotted.
2. The header record and first set of coordinates are read.

3. If the first *xx* and *yy* coordinates are less than *x1* or *y1* or greater than *x2* or *y2*, respectively, read the next set of coordinates (step 2)
4. If number of sets of coordinates is less than 3, write *if*, *isf*, and *ichar*. Go to step 2.
5. Plot the coordinates until *isfno* (number of coordinates) has been exhausted.
6. Read the next header record, add 1 to *j* and continue until *j* equals *iskp*, which equals 9. Go to step 2.

```
c  ****SUBROUTINE GRID****
c   THIS ROUTINE WILL PLOT THE GRID FILE
c         It has been determined to plot only every 10th set
c           of grid records since the Tektronix plot is so small.
       subroutine grid
c
       dimension xx(6),yy(6)
       common x1,y1,x2,y2
c
       iskp = 9
c        Set iskp=9 so as to plot only every tenth record
c          of the grid file.
       j = 0
       ichar = 43
       rsiz = 0.3
       rewind 16
c
c        Read header information.
10     read(16,20,end=270) if,ifno,isf,isfno,if1
20     format (5i5)
c
       ie = isfno
       if (isfno .ge. 6) ie=6
c        Read x-y coordinates.
       read(16,30) (xx(i),yy(i),i=1,ie)
30     format (12f6.3)
       if (isfno .ge. 3) go to 80
c
c        If coordinates are out of range, read next set of coordinat
\ces.
       if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 10
       if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 10
       if ((ifno .eq. 1) .and. (isf .eq. 1)) go to 60
       call movea(xx(1),yy(1))
       call anmode
c        If there's less than 3 sets of coordinates, write if, isf a
\cnd ichar.
       write (0,40) if
40     format (1x,i5)
       ry=yy(1)-rsiz
       call movea(xx(1),ry)
       call anmode
       write (0,40) isf
       call movea(xx(2),yy(2))
       call ancho(ichar)
       go to 10
c
```

```
60      call movea(xx(1),yy(1))
        call anmode
        write (0,40) if
        call movea(xx(2),yy(2))
        call ancho(ichar)
        go to 10
c
80      continue
        if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 120
        if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 120
        if ((ifno .eq. 1) .and. (isf .eq. 1)) go to 100
        call movea(xx(1),yy(1))
        call anmode
        write (0,40) if
        ry = yy(1)-rsiz
        call movea(xx(1),ry)
        call anmode
        write (0,40) isf
        if (ifl .eq. 0) go to 120
        ry = ry-rsiz
        call movea(xx(1),ry)
        call anmode
        write (0,40) ifl
        go to 120
c
100     continue
        call movea(xx(1),yy(1))
        call anmode
        write (0,40) if
c
120     continue
        call movea(xx(2),yy(2))
        do 130 k=2,ie
        call drawa(xx(k),yy(k))
130     continue
c
140     isfno=isfno-6
        if (isfno)200,200,150
150     if (isfno-6)160,160,170
160     ie = isfno
170     read (16,30,end=270) (xx(i),yy(i),i=1,ie)
        do 180 k=1,ie
        call drawa(xx(k),yy(k))
c         Draw until isfno(no. of coordinates) has been exhausted.
180     continue
c
        if (isfno-6)200,200,140
c
200     j = j+1
c         J is the count of header records read.
        read(16,20,end=270) if,ifno,isf,isfno,ifl
210     ie = isfno
        if (isfno .ge. 6) ie=6
220    read(16,30) (xx(i),yy(i),i=1,ie)
```

```
c        Read through x-y coordinates until isfno has been exhausted
\c.
         isfno = isfno-6
         if (isfno)250,250,230
230      if (isfno-6)210,210,220
c
250      if (j .eq. iskp) go to 10
c          The above compare is made to determine when the 10th record
\c is reached.
         go to 200
c
270      continue
         return
         end
```

SUBROUTINE NAME: PLOCV

Author: Pearl Porter
Purpose of the program: **plocv** will plot the symbols, numbers, and outline from the GRASP input files (*file15*) depending on the value of *idec*, which was supplied by the user.
Data base: Geoindex
Computer: Honeywell Series 60 (level 68)
Operating system: Multics
Calling sequence: call plocv (ichar,nl,idec)
Arguments:
ichar–Symbol designated by the user to be used with a specific GRASP input file
nl–Code specifying that this routine was called owing to an enlargement option
idec–Code used to determine which of three options will be used:
If *idec* = 1, symbol, number, and outline plotted
If *idec* = 2, symbol or outline (with no numbers)
If *idec* = 3, numbers only (no plotting or symbols)
Subroutines called: **movea, anmode, ancho, drawa** (all Tektronix routines), **io** (Multics)
Common data referenced: *x1*, *y1*, *x2*, *y2*
Input files: GRASP file
Output files: None
Arrays used: *xx*(6), *yy*(6)
Called by: **pin90**
Error checking and reporting: Located in **pin90**
Constants: None
Program logic:
The subroutine does the following:

1. *rsiz* is loaded with 0.3. Later in the program, *rsiz* will be subtracted from the *y* coordinate in order to print the *isf* below the *if*.
2. If *nl* is equal to 1, loads *rsiz* with 0.05. If this is an enlargement of a section of the screen, the number to be subtracted has to be decreased to compensate for the change in the size of the screen.
3. Reads the header record.
4. Reads first *x*, *y* coordinate record.
5. If sets of coordinates are more than three, goes to 18.
6. If the first *x*, *y* coordinates are not within the range of *x1*, *y1*, *x2*, and *y2*, goes to 3.
7. If *ifno* = 1 and *isf* = 1, goes to 13.
8. If *idec* = 2, goes to 12.
9. Writes the *if*.
10. Subtracts *rsiz* from the *y* coordinate and uses this computed coordinate to write *isf*.
11. If *idec* = 3, goes to 3.
12. Plots the symbol. Goes to 3.
13. If *idec* = 2, goes to 17.
14. If *nl* does not equal 1, goes to 15.
    Subtracts *rsiz* (either 0.3 or 0.05) from the first *y* coordinate.
    Moves the cursor to the *x* and computed *y* coordinates. Goes to 16.
15. Moves cursor to position designated by *x* and *y* coordinates.
16. If *idec* = 3, write *if*. Goes to 3.
17. Plots symbol. Goes to 3.
18. If the first *x*, *y* coordinates are not within the range of *x1*, *y1*, *x2*, and *y2*, goes to 20.
    If *ifno* = 1 and *isf* = 1, goes to 19.
    Writes *if*, *isf*, and *if1* on map. Goes to 20.
19. Writes *if*.
20. If *idec* = 3, there will be no plotting.
    Otherwise, continues drawing until *isfno* (number of coordinates) has been exhausted. Goes to 3.
21. At EOF, closes and detaches *file15*.

```
c   ****SUBROUTINE  PLOCV ****
c
        subroutine plocv(ichar,nl,idec)
c
        dimension xx(6),yy(6)
        common x1,y1,x2,y2
c
c     If idec = 1, symbol, number and outline will be plotted.
c     If idec = 2, symbol or outline (with no numbers).
c     If idec = 3, numbers only (no plotting or symbols).
c
        rsiz=0.3
        if (nl .eq. 1) rsiz = 0.050
        rewind 15
10      continue
c         Read header information.
20      read (15,30,end=160) if, ifno,isf,isfno,if1
30      format (5i5)
c
        ie = isfno
        if (isfno .ge. 6)   ie=6
c         Read x-y coordinates.
        read (15,40) (xx(i),yy(i),i=1,ie)
40      format (12f6.3)
c
        if (isfno .ge. 3) go to 70
c
        if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 20
        if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 20
        if ((ifno .eq. 1) .and. (isf .eq. 1)) go to 60
        if (idec .eq. 2) go to 55
        call movea(xx(1),yy(1))
        call anmode
c         This routine writes if, isf and ichar.
        write (0,50) if
50      format (1x,i5)
        ry = yy(1)-rsiz
        call movea (xx(1),ry)
        call anmode
        write (0,50) isf
        if (idec .eq. 3) go to 20
55      call movea(xx(2),yy(2))
        call ancho(ichar)
        go to 20
c
60      continue
c         This routine will write if and ichar depending on idec.
        if (idec .eq. 2) go to 67
        if (nl .ne. 1) go to 62
        ry = yy(1)-rsiz
        call movea(xx(1),ry)
        go to 65
62      call movea(xx(1),yy(1))
65      call anmode
```

```
        write(0,50) if
        if (idec .eq. 3) go to 20
67      call movea(xx(2),yy(2))
        call ancho(ichar)
        go to 20
c
70      continue
        if ((xx(1) .le. x1) .or. (yy(1) .le. y1)) go to 90
        if ((xx(1) .ge. x2) .or. (yy(1) .ge. y2)) go to 90
        if ((ifno .eq. 1) .and. (isf .eq. 1)) go to 80
        if (idec .eq. 2) go to 90
        call movea(xx(1),yy(1))
        call anmode
c         This routine will print out if, isf and if1.
       write (0,50) if
        ry = yy(1)-rsiz
        call movea(xx(1),ry)
        call anmode
        write (0,50) isf
        if (if1 .eq. 0) go to 90
        ry = ry-rsiz
        call movea(xx(1),ry)
        call anmode
        write (0,50) if1
        go to 90
c
80      continue
        if (idec .eq. 2) go to 90
        call movea(xx(1),yy(1))
        call anmode
        write (0,50) if
c
90      continue
        call movea(xx(2),yy(2))
        do 100 k=2,ie
        if (idec .eq. 3) go to 100
c          If idec = 3, bypass any plotting.
        call drawa(xx(k),yy(k))
100     continue
c
110     isfno=isfno-6
        if (isfno) 10,10,120
120     if (isfno-6) 130,130,140
130     ie = isfno
c
140     read(15,40,end=160) (xx(i),yy(i),i=1,ie)
        do 150 k=1,ie
        if (idec .eq. 3) go to 150
        call drawa(xx(k),yy(k))
150     continue
c
        if (isfno-6) 10,10,110
c
```

```
160     continue
        call io ("close","file15")
        call io ("detach","file15")
        return
        end
```

APPENDIX D. FORMATS AND NOTES

FORMAT OF REF*NM* FILES

Item No.	Item name	Character type	Maximum field length
1	Id	Integer-in automatic	4
2	State	Dictionary character	20
3	Author 1	Embedded character string	60
4	Author 2	do	60
5	Author 3	do	60
6			
7			
8	Year	Integer	4
9	Title 1	Embedded character string	60
10	Title 2	do	60
11	Title 3	do	60
12	County 1 or region 1	do	60
13	County 2 or region 2	do	60
14	County 3 or region 3	do	60
15	County 4 or region 4		
16	County 5 or region 5		
17	Publisher	Embedded character string	60
18	Scale 1	Integer	8
19	Scale 2	do	8
20	Scale 3	do	8
21	Scale 4	do	8
22	Scale 5	do	8
23	Series 1	Embedded character string	60
24	Emphasis	do	60
25	Area of coverage	Real	8
26	Unit for area of coverage	Embedded character string	7
27	Extreme north latitude	Integer *DDDMMSS S*[1]	12
28	Extreme south latitude	do	12
29	Extreme west longitude	do	12
30	Extreme east longitude	do	12
31	Center-point latitude	do	12
32	Center-point longitude	do	12
33	Boundary id	Integer not used.	
34	Other map not included	Embedded character string	60
35	Depositories	do	60
36	Base	Dictionary character string	30
37	Title 4	Embedded character string	60
38	Geology or geochemistry	do	12
39	Plate 1 map plate name	do	30
40	Plate 2 map plate name	do	30
41	Plate 3 map plate name	do	30
42	Plate 4 map plate name	do	30
43	Plate 5 map plate name	do	30
44	Idstat--State code	Integer	2
45	Id sub 1	do	2
46	Id sub 2	do	2
47	Id sub 3	do	2
48	Id sub 4	do	2
49	Id sub 5	do	2
50	Bound 1	do	6
51	Bound 2	do	6
52	Bound 3	do	6
53	Bound 4	do	6
54	Bound 5	do	6
55	Span 1	do	6
56	Span 2	do	6

Item No.	Item name	Character type	Maximum field length
57	Span 3	do	6
58	Span 4	do	6
59	Span 5	do	6
60	Series 2	Embedded character string	60
61	Scale 6	Integer	8
62	Scale 7	do	8
63	Scale 8	do	8
64	Scale 9	do	8
65	Scale 10	do	8
66	Plate 6	do	30
67	Plate 7	do	30
68	Plate 8	do	30
69	Plate 9	do	30
70	Plate 10	do	30
71	Id sub 6	do	2
72	Id sub 7	do	2
73	Id sub 8	do	2
74	Id sub 9	do	2
75	Id sub 10	do	2
76	Bound 6	do	6
77	Bound 7	do	6
78	Bound 8	do	6
79	Bound 9	do	6
80	Bound 10	do	6
81	Span 6	do	6
82	Span 7	do	6
83	Span 8	do	6
84	Span 9	do	6
85	Span 10	do	6
86	Also other maps	Character	30
87	Dum 0 Flag for expansion. 1 prototype. 2 additional maps.	Integer	1
88	Dum 1	Integer	1
89	Dum 2	do	
90	Dum 3	do	1
91	Dum 4	do	1
92	Dum 5	do	1
93	Dum 6	do	1
94	Dum 7	do	1
95	Dum 8	do	1
96	Dum 9	do	1

[1] A space and S indicate a decimal point followed by one digit.

FORMAT FOR REFERENCE FILE

Columns 1,2 State	Columns 3,4,5 Reference No.	Columns 6,7 Item No.	Columns 8 to 67 Data
I2	I3	I2	60 characters maximum

NOTES FOR ENTERING CARD DATA

1. Do not put a comma after the year.
2. Item 12: All counties or regions [for Item 12] are typed in small letters.
3. Item 12: All counties or regions are typed on the same line and a comma and a space separates each. Counties or regions cannot exceed 60 characters. Continue on the next line, creating Item 13.
4. Item 24 (emphasis) is always typed in lowercase. If there is more than one emphasis, add a comma and a space between each one.
5. Items 18-22, 61-65: Omit the period or semicolon after scale.
6. Omit the one digit and colon before the scale.
7. Omit commas between scales.
8. Title can be not more than 60 characters of data; together with the numeric data (State, ref, item number), 67 characters is maximum number for any one line.
9. Item 38 (geology) is always in lowercase.
10. Items 3-5: Omit spaces between authors initials.
11. Item 17: Omit space between U. and S. in U.S. Geol. Survey.
12. Item 2 (the State name) always has first letter capitalized and the rest in lower case, as in Missouri.
13. Item 18-22, 62-65: Omit the (a), (b), (c), and so on, between scales.
14. Item 39-43, 66-70: The first letter in the name of the first plate is always capitalized, names of all other plates begin with small letters.
15. If there is no series (Item 23), place a period after name of publisher.
16. Do not underscore entered data.
17. Item 35: Type as shown in "Abbreviations for depositories." Use no more than 60 characters.

Abbreviations for depositories

USBM	= U.S. Bureau of Mines
BM&G	= Bureau of Mines and Geology
GS	= Geological Survey
WYGS	= Wyoming Geological Survey