U.S. DEPARTMENT OF THE INTERIOR
U.S. GEOLOGICAL SURVEY

Program Development Tools For General Earthquake Observation System(GEOS) In C language

by

Meei-You Lee
Gary L. Maxwell

Open-File Report OF95-258

1995

TABLE OF CONTENTS

1. INTRODUCTION

General Earthquake-Observation System (GEOS) [1] is a portable, digital seismic data acquisition system. Its design is focused on adaptability and versatility with respect to a wide variety of seismologic and engineering experiments. Its functions include signal conditioning of analog sensor outputs, conversion of analog signals to digital, temporary data storage for preprocessing and online external decision capability based on the incoming data stream; data storage, retrieval and transmission; time reference; system calibration and operator interface. Modular hardware components are used to isolate major system functions on corresponding hardware with the central control of each module produced by the microcomputer via the general computer bus. General segmented software controls the various hardware modules, performs system functions, facilitates system debugging and replacement of hardware modules.

All GEOS system software is stored in Erasable-Programmable Read Only Memory(EPROM) and executed from Random-Access Memory (RAM) with transfer occurring at the time of system start-up. The system software was developed using layers. At the bottom layer is the control panel. On top of the control panel is the kernel. The kernel software is a fundamental part of the operating system developed for a microcomputer. On top of the kernel are the application packages. The application packages have been developed to both record and playback data via the GEOS. All control panel, kernel and software packages are written in PAL assembly language and were developed on the host minicomputer PDP-11/73. Development and debug tools were developed to load and transfer the GEOS programs from the host machine to the GEOS and interactively debug GEOS programs. These tools were written in assembly language, FORTRAN or Pascal and run on PDP-11/73. These tools include Downline Loader (program DOWNLINE), On-line Debugger(program GDT), Data I/O EPROM Programmer loader (program DATAIO), Memory Map Generator (programs GETFLD and GETMAP), and Binary Files Comparison(program BINCMP32).

General purpose computer language C is used to rewrite all tools mentioned above. C is a language not tied to any particular hardware or system. Conversion of the programs to C permits the programs to run without changes on any machine that supports ANSI-standard C. All tools mentioned above have been converted to C and ported successfully to PC MS-DOS. Porting to other systems requires only minor changes in tools DOWNLINE and DATAIO which use communication functions.

Each section of this document describes one GEOS system development tool. It describes the function of each program, procedures for compiling, linking and executing each program, and the error messages generated by each program. A complete listing for each of the programs is provided in the appendices.

1.1 Command Syntax

This section describes the general syntax of running each program. A UNIX-like command line is used to start the execution of programs. The format is as follows:

program-name [options] file

where program-name is the name of the program to be executed. File is the name of input file. The options are used to modify and control how the program will be executed. Options begin with a dash '-' and contain one or two letters. Uppercase or lowercase letters mean the same. The format of options is as follows :

-l -a1 -FP87

When the options are followed by a value, options are used to select one of many choices. When options are not followed by a value , options are used to control to do or not to do a task. The order of options is irrelevant.

1.2 Program-Maintenance Tool

The Microsoft Program Utility (MAKE) is used to generate the software. MAKE automates program development and maintenance. It can update an executable file automatically whenever changes are made to one of its source or object files. It can also update any related file whenever changes are made to other files. Before MAKE can be run, a file known as MAKE description file containing information needed to run MAKE is created. The description file documents the interdependencies of all modules used to generate the executable application file. It documents the dependency between object file and its source files and other related files such as include files and other source files. If one of the source file has changed, MAKE automatically recompiles and generates a new object file. It also documents the dependency between the executable file and object files and library files . It includes commands to compile the source modules, and commands to link object files and library files to generate the final executable file. One MAKE description file is created for each program here. MAK is always used as file extension for the MAKE description file name. Only common commands are used in the make description file for each program. The make file can be run on any computer system that supports the MAKE utility program.

2. PROGRAM DOWNLINE

This loader program loads and transfers the binary file to the GEOS. The binary file generated by the CPAL cross-assembler is loaded to the proper program memory area, by resolving the address difference between RAM and EPROM for each segment it encounters. The loader also loads the needed segments to execute the binary file to the RAM area. Then this loaded memory area is transferred to the GEOS via a user selected serial port. All unused memory areas are initialized to a special value. The reader software in the GEOS system loads the data to the memory space by scanning the incoming data to determine when to start to load and continues to load the data until it reads a stop indicator. The communications settings between PC and GEOS are one stop bit, eight data bits , no parity and a user selected baud rate. Checksum method is used to check for communication errors. No handshake or any other error checking method is used .

NOTE: This program is used to test a new version of a GEOS application program without an actual burn-in of the application to the EPROMs. The GEOS program memory configurations should be all in RAM when the DOWNLINE program is used. In normal configuration, the GEOS program memory area is configured in 6 EPROM chips and 2 RAM chips.

2.1 **Execution Procedure**

SYNTAX:

DOWNLINE [-C{1|2|3|4}] [-B{1|2|3|4|5|6|7}] [-H{2|3|4|5|6}] finm

FINM is the name of binary file to be transferred. No extension is allowed in the name

OPTIONS:

-C serial communication ports used

valid values are:
1 : COM1
2 : COM2
3 : COM3
4 : COM4

default value: 2

-B baud rate used

valid values are:
1: 150 bits/sec.
2: 300 bits/sec.
3: 600 bits/sec.
4: 1200 bits/sec.
5: 2400 bits/sec.
6: 4800 bits/sec.
7: 9600 bits/sec.

default value : 7

-H highest field to be transferred

valid values are: 2,3,4,5,6

default value : the actual highest memory field used by the application program.

EXAMPLES

DOWNLINE record (cr)
load and transfer file record.bin up to field 6 using com2 and 9600 baud rate.

DOWNLINE -c3 -b6 record
load and transfer file record.bin up to field 6 using com3 and 4800 baud rate.

2.2 Compilation and Linkage Procedure

MAKE DOWNLINE.MAK

The above command automatically recompiles source programs and relinks the object files when any of the source programs or include files have been changed.
downline.mak is the make file of program downline.
The listing of downline.mak is :

```
all       = downline.obj assign.obj send.obj proc_cmd.obj
compile   =  cl /FPi /c /AM /W3 /Fs          (cl is the compile and link command)

incpath   = m:\c\include                     (m is the name of virtual drive)
libpath   = m:\c\lib

downline.obj: downline.c downcm.h
        $(compile) downline.c

assign.obj:  assign.c com_port.h $(incpath)\asynch_1.h
        $(compile) assign.c

proc_cmd.obj:  proc_cmd.c
        $(compile) proc_cmd.c

send.obj:  send.c com_port.h downcm.h $(incpath)\asynch_1.h
        $ (compile) send.c

downline.exe:  $(all) $(libpath)\asynch.obj  $(libpath)\asy_mcm.lib
        link  $(all)  $(libpath)\asynch.obj, , ,$(libpath)\asy_mcm.lib
```

2.3 Format of Binary Input File

The binary input file is the output of the CPAL cross-assembler. The file starts with a string of special binary numbers, the start of text, and ends with a string of special binary numbers, the end of text. Any bytes between the start and the end of text are meaningful data. The last word in the data is the checksum. The checksum is used for error checking. Data can be interpreted in one of three ways: change of segment, change of RAM location and actual data.

The data types in the binary file are :

Meaning	Representation
Start of text:	a string of bytes with octal value 2xx. x can be any octal number. Currently 200 is used.
End of text:	a byte with octal value 2xx and the byte not in the start of text or the second byte of a data word. Currently 200 is used.
Change of segment:	a byte with a octal value 3xx. Where xx represents the new segment number. The byte is not the second byte of a word.
Change of location:	a word has a octal value 1xxxx. Where xxxx represents the new location in the RAM .
Data :	any word between the start and the end of text and not interpreted as change of segment or change of location. Only the right most 6 bits of a byte are used. Two bytes form a word to be transferred. The GEOS memory word contains 12 bits.

2.4 Program Portability

Since communication functions are not included in the ANSI-C library, the software communication package, C ASYNCH MANAGER, is used to transfer data from the PC using a serial port. Function "assign" opens a communication port and sets the communications settings of the port. DOS system functions are called in "assign". Function "send" also calls DOS system functions to transfer bytes through the opened serial port to the GEOS. When program DOWNLINE is ported to a different system, different system function calls are needed to open the communication port, set the communication settings of the port and transfer bytes.

3. PROGRAM GDT

This program lets a user interactively display a saved RAM memory image file of a GEOS application program. The program reads the saved file into memory and builds a symbol table by reading the symbol table file of the application program into memory. The user can display memory in octal using direct or indirect addressing mode , search for a symbol and find

the symbol's memory location and its value, and also find all memory locations with a specified value.

Note: This program is used to debug GEOS application programs. At the time of a GEOS crash, the RAM is saved to a tape file. The tape file is read back and saved in a disk file. Program GDT can read back the saved disk file and show contents of various memory locations and values of symbols at the time of system crash .

3.1 Execution Procedure

SYNTAX:

GDT [-S=symfinm] [-D=dmpfinm]

OPTIONS:

-s, -S	symbol table file used "symfinm" is the name of user specified symbol table file. The default file name is RECORD.STB
-d, -D	memory image file used. "dmpfinm" is the name of memory image file. The default file name is MEM.DMP.

EXAMPLES:

GDT

GDT brings up the prompt (?) for user commands to let user examine the default memory image file mem.dmp using the default symbol table file record.stb

GDT -d=test.dmp -s=test.stb

GDT brings up the prompt (?) for user commands to let user examine the saved memory image file test.dmp using the symbol table file test.stb

INTERACTIVE COMMANDS:

1. octal number N :display the content of memory location N using direct addressing mode.

2. /# : change the memory field to #. The valid # value is 1 or 0.

3. $S# : display all memory locations which have the user entered octal value #.

4. @# : display the content of memory location # using indirect addressing mode.

5. CR : increment the current memory location by 1 and display the content of current location.

3.2 Compilation and Linkage Procedure

MAKE GDT.MAK

The above command automatically recompiles source programs and relinks the object files when any of the source programs or include files have been changed.
gdt.mak is the make description file of program gdt.
gdt.mak listing is as follows:

```
all      = gdt.obj readsym.obj memdisk.obj intact.obj
compile =  cl /FPi /c /AL /W3 /Fs

incpath   = m:\c\include
libpath   = m:\c\lib

gdt.obj: gdt.c gdtcom.h
        $(compile) gdt.c

readsym.obj:  readsym.c gdtcom.h
        $(compile) readsym.c

memdisk.obj:  memdisk.c gdtcom.h
        $(compile) memdisk.c

intact.obj:  intact.c gdtcom.h
        $(compile) intact.c

gdt.exe:  $(all)
        link /ST:10000  $(all), , , ,
```

3.3 Program Portability

Since only ANSI-standard C language is used , program GDT can be ported to any computer system without any changes.

4. PROGRAM DATAIO

The program DATAIO loads and transfers the user specified binary file of a GEOS application program to the DATA I/O EPROM PROGRAMMER. The DATAIO program loads the binary file into the proper memory area using the same rules as described in the program DOWNLINE. The DATAIO program does not load the needed segment to RAM memory area in order to start the execution of the binary file. The DATAIO program transfers the loaded memory area to the DATA I/O PROGRAMMER. The I/O PROGRAMMER will burn the data into the EPROM memory chips. Data is transferred to the PROGRAMMER with a record size of 43 characters. Only 16 bytes in a record are data bytes. Checksum, record type, start address and bytes count are also stored in the record. The user has options to set up the communications settings between the sender and the DATA I/O PROGRAMMER. The communications settings follow the same rules as described in the program DOWNLINE.

During the execution, the DATAIO program prompts the user for the memory area to be transferred. It transfers 4k of bytes at a time.

4.1 Execution Procedure

SYNTAX:

DATAIO [-C{1|2|3|4|}] [-B{1|2|3|4|5|6|7}] [-S=symfinm] filenm

Filenm is the name of a binary input file generated by the cross assembler. The file extension is optional in the input file name. The default file extension is bin.

OPTIONS:

-C, -c serial communication ports used

valid values are:
1 : COM1
2 : COM2
3 : COM3
4 : COM4

default value: 2

-B ,-b baud rate used

valid values are:
1: 150 bits/sec.
2: 300 bits/sec.

3: 600 bits/sec.
4: 1200 bits/sec.
5: 2400 bits/sec.
6: 4800 bits/sec.
7: 9600 bits/sec.
default value : 7

-S, -s =symfinm the symbol table file used.

"symfinm" is the name of symbol table file. The default symbol table file name is the concatenation of the principal part of the binary file name "filenm" and the file extension "stb".

EXAMPLES

DATAIO -c1 -b4 -s= test.stb record.bin

The program DATAIO loads and transfers binary file record.bin to the DATA I/O EPROM PROGRAMMER using communication port 1 at 1200 baud rate. The symbol table file used is test.stb.

DATA record

The program DATAIO loads and transfers binary file record.bin to the PROGRAMMER using the default setting. Record.stb is the symbol table file, COM 2 is the serial port used to transfer at 9600 baud rate.

4.2 Compilation and Linkage Procedure

MAKE DATAIO.MAK

The above command automatically recompiles source programs and relinks the object files when any of the source programs or include files have been changed. DATAIO.MAK is the make description file of the program DATAIO. DATAIO.MAK listing is as follows:

```
all     = dataio.obj  esim8.obj  assign.obj  proc_cmd.obj  sendit.obj
compile =   cl /FPi /c /AL /W3 /Fs

incpath     = m:\c\include
libpath      =m:\c\lib
```

```
dataio.obj: dataio.c dataio.c
        $(compile) dataio.c

esim8.obj : esim8.c  dataio.h
        $(compile) esim8.c

assign.obj:   assign.c com_port.h $(incpath)\asynch_1.h
        $(compile) assign.c

proc_cmd.obj:  proc_cmd.c
        $(compile) proc_cmd.c

sendit.obj:  sendit.c  dataio.h
        $(compile) sendit.c

dataio.exe: $(all)  $(libpath)\asynch.obj  $(libpath)\asy_mcm.lib
        link  $(all)  $(libpath)\asynch.obj,  ,  ,$(libpath)\asy_mcm.lib
```

4.3 Program Portability

Since communication functions are not included in the ANSI-C library, the software communication package, C ASYNCH MANAGER, is used to transfer data from the PC using a serial port. Function "assign" opens a communication port and sets the communications settings of the port. DOS system functions are called in "assign". Function "send" also calls DOS system functions to transfer bytes through the opened serial port to the GEOS. When program DATAIO is ported to a different system, different system function calls are needed to open the communication port, set the communication settings of the port and transfer bytes.

5. Program GETFLD

The program GETFLD generates an EPROM memory map of a GEOS application program. The EPROM has memory fields from 2 to 6. For each memory field, the map includes the segment number, the description of the segment , and the segment starting address, the size and the end address for all segments used in the memory field. The map also shows which memory area is not currently in use. The GETFLD reads a description file of the GEOS application program to get descriptions for all segments used and reads the application program's symbol table file to get the symbols of all segments used. A segment, like a function in C language, breaks the main task into small pieces.

Note: The GEOS memory is configured as follows: memory field 0 and 1 in RAM, memory field 2 to 6 in EPROM and memory field 7 in RAM. Each memory field has 4K words. The EPROM is used to store the kernel and an application program. At start-up and execution time, computer code is moved from EPROM to RAM as needed. The memory field 7 is used as a window to map data from data buffer area to the 4K RAM.

5.1 Execution Procedure

SYNTAX:

GETFLD [-D = descriptor] [-M =mapname] [-S = symname] filenm

filenm is the name of GEOS application program. The file extension is optional in the name.

OPTIONS

- d, -D	the description file used. "descriptor " is the name of description file. If the description file does not exist, the EPROM memory map will be generated without having descriptions for all segments. The default file name is the concatenation of the main part of "filenm" with the file extension "fld".
-m, -M	The map file used. "mapname" is the name of map file. The default name is formed in the same way as described in the option "d" except the extension is "map".
-s, -S	the symbol table file name used. "symname" is the name of symbol table file. The default name is formed in the same way as described in the option "d" except the extension is "stb".

EXAMPLES

GETFLD TEST

GETFLD generates the EPROM memory map of GEOS program TEST to the file test.map using the default descriptor "test.map" and the default symbol table file "test.stb".

GETFLD -D=seg.fld -M=seg.map TEST

GETFLD generates the EPROM memory map of GEOS program TEST to the file "seg.map" using the descriptor "seg.fld" and the default symbol table file "test.stb"

5.2 Compilation and Linkage Procedure

MAKE GETFLD.MAK

The above command automatically recompiles source programs and relinks the object files when any of the source programs or include files have been changed.
getfld.mak is the make description file of program getfld.
getfld.mak listing is as follows:

```
compile = cl /FPi /c /W3 /AS /Fs
all    = getfld.obj readsym.obj initfi.obj

getfld.obj: getfld.c getfld.h
        $(compile) getfld.c

readsym.obj:  readsym.c getfld.h
        $(compile) readsym.c

initfi.obj: initfi.c getfld.h
        $(compile) initfi.c

getfld.exe: $(all)
        link $(all), , , ,
```

5.3 Program Portability

Since only ANSI-C language is used , program GETFLD can be ported to any computer system with ANSI-C compiler without any changes.

6 Program GETMAP

Program GETMAP is the RAM version of program GETFLD. It generates the RAM memory map of GEOS program. Running and maintaining GETMAP follow the same procedures as described in the GETFLD sections [5.1 & 5.2].

7 Program BINCMP32

The program BINCMP32 compares two binary files generated by cross-assembler and displays the differences. BINCMP32 reads these two binary files and loads each of them into the proper EPROM memory area . Then BINCMP32 compares the memory contents of these two files for every memory location in the EPROM area and displays the differences to a list file.

The differences show the memory field, memory location in the field and memory values of these two files at this location .The program BINCMP32 is used to compare a new version of GEOS application programs such as RECORD or PLAYBACK to the current version and decide which memory fields need to be burned again.

7.1 Execution Procedure

SYNTAX:

BINCMP32 [-l=listf] file1 file2

File1 is the name of first binary input file and file2 is the name of the second binary input file. The default file name extension is "bin" and the file name extension is optional.

OPTIONS:

-l list file used.

listf is name of the user specified list file. The default list file is the standard output.

7.2 Compilation and Linkage Procedure

MAKE bincmp32.mak

The above command automatically recompiles source programs and relinks the object files when any of the source programs or include files have been changed.
BINCMP32.mak is the make description file of program bincmp32.
BINCMP32.mak listing is as follows:

```
compile = cl /FPi /c /W3 /AL /Fs
all    = bincmp32.obj esim8.obj

bincmp32.obj: bincmp32.c esim8.c bincmp32.h
        $(compile) bincmp32.c

esim8.obj:   esim8.c bincmp32.h
        $(compile) esim8.c

bincmp32.exe: bincmp32.obj esim8.obj
        link $(all), , , ,
```

7.3 Program Portability

Since only ANSI-C language is used , program BINCMP32 can be ported to any computer system with ANSI-C compiler without any changes.

8 References

[1] Borcherdt, R, D., J. B. Fletcher,E. G. Jensen,G. L. Maxwell, J. R.Van Schaack, R.E.Warrick, E. Cranswick, M. J. S. Johnston, and R. McClearn, 1985, A general earthquake observation system (GEOS): Bulletin of the Seismological Society of America 75, no.6 1783-1826.

[2] Harris Corp., 1979, Harris CMOS Microprocessor Data Book: Harris Semiconductor, Melbourne, Florida.

[3] Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language : Prentice Hall, Englewood Cliffs, N.J.

APPENDIX A Error Messages

This appendix lists the error messages you may encounter when you run the programs documented in sections 2-7, and gives a brief explanation of each message. The explanations are listed in italic .

A.1 Program Downline

Invalid command format , try again
Downline [-c] [-b] [-h] filenam
The user entered invalid format. The valid format is shown in the above line.

No extension allowed in filename
Filename can not have extension in it.

Binary file xxx not there
Binary file xxx can not be found .

Symbol table xxx not there
Symbol table file xxx can not be found.

Checksum error: internal= xxx, actual = yyy
The computed checksum xxx is different from the actual checksum yyy.

Incomplete specification of segment xxx
Segment xxx does not have EPROM field, RAM field, RAM base address or EPROM base address.

Can not open port xxx. Error = yyy
Port xxx can not be opened . The error number is yyy.

Can not set the baud rate of port yyy to xxx. Error = zzz
The communication settings of port yyy can not have baud rate xxx. The error code is zzz.

Can not set no parity of port xxx. Error= yyy
The communication settings of port xxx can not be no parity. The error code is yyy.

Can not set 8 data bits of port xxx. Error= yyy
The communication settings of port xxx can not be 8 data bits. The error code is yyy.

Can not set one stop bit of port xxx. Error= yyy
The communication settings of port xxx can not be one stop bit. The error code is yyy.

Remote flow control error xxx
Can not disable the remote flow control of current port. The error code is xxx.

Can not disable local flow control of port xxx . Error= yyy
Can not disable the local flow control of port xxx. The error code is yyy.

Invalid port number xxx
The port number xxx is not valid. The valid number for PC is 1 to 4.

Invalid baud rate xxx
The baud rate is not valid.

Invalid field xxx
The filed number xxx is not valid. The valid number is from 0 to 6.

Can not call function oqsize successfully. Error = xxx
The call to C asynch manager function "oqsize" failed. The error code is xxx.

Can not call wrtst_a1 successfully. Error= xxx
The call to C asynch manager function "wrtst_a1" failed. The error code is xxx.

A.2 Program Gdt

Invalid format.
Gdt [-s=symfinm] [-d=dmpfinm]
The user entered invalid format. The valid format is shown in the above line.

Invalid value xxx
The valid value is 0 to 7.

Word not found xxx
Word xxx is not found in the memory.

Invalid field specified xxx
The user entered invalid field number xxx. The valid number is 0 or 1.

Symbol is undefined xxx
The user entered symbol xxx is not found in the symbol table file. Try to enter a different symbol.

Invalid command xxx
User entered invalid command. Refer 3.1 for a list of valid commands.

Can not open symbol file xxx
Symbol table file xxx can not be opened. The file may be opened already.

Error in malloc xxx
C library function malloc failed to allocate memory space for string xxx.

A.3 Program Dataio

Invalid input, enter N,C, or Q to try again .
The user input can only be N,C, or Q.

Invalid command format , try again
Dataio [-c] [-b] [-s=symname] filenam
The user entered invalid format. The valid format is shown in the above line.

No extension allowed in filename
The filename can not have extension in it.

Binary file not there
Binary file can not be found .

Symbol table not there
Symbol table file can not be found.

Checksum error: internal= xxx actual = yyy
The computed checksum xxx is different from the actual checksum yyy.

Incomplete specification of segment xxx
Segment xxx does not have EPROM field, RAM field, RAM base address or EPROM base address.

Segment xxx can not map to a valid EPROM
The EPROM field of segment xxx is not in one of 2,3,4,5,or 6.

Invalid port number xxx
The port number xxx is not valid. The valid number for PC is 1 to 4.

Invalid baud rate xxx
The baud rate xxx is not valid.

Invalid field xxx
The filed number xxx is not valid. The valid number is from 0 to 6.

Can not open dbg_out
File dbg_out can not be opened Dbg_out may be opened already.

Bad field alignment. Starting field must be in one of following: 2 - 6
User entered memory field that is not in 2 to 6.

Invalid input, enter L or Q to try again
The valid input is L or Q.

Can not open port xxx. Error = yyy
Port xxx can not be opened . The error number is yyy.

Can not set the baud rate of port yyy to xxx . Error = zzz
The communication settings of port yyy can not have baud rate xxx. The error code is zzz.

Can not set no parity of port xxx. Error= yyy
The communication settings of port xxx can not be no parity. The error code is yyy.

Can not set 8 data bits of port xxx. Error= yyy
The communication settings of port xxx can not be 8 data bits. The error code is yyy.

Can not set one stop bit of port xxx. Error= yyy
The communication settings of port xxx can not be one stop bit. The error code is yyy.

Remote flow control error xxx
Can not disable the remote flow control of current port. The error code is xxx

Can not disable local flow control of port xxx . Error= yyy
Can not disable the local flow control of port xxx. The error code is yyy.

Can not call function oqsize successfully. Error = xxx
The call to C asynch manager function "oqsize" failed. The error code is xxx.

Can not call wrtst_a1 successfully. Error= xxx
The call to C asynch manager function "wrtst_a1" failed. The error code is xxx.

A.4 Program Getfld

Command format : getfld [-D=xxx -M=yyy] finm
D-descriptor file name, M-map file name
User entered invalid command format. Valid command format is show above. Try again.

Can not open symbol table file xxx
Symbol table file xxx can not be opened. File xxx may be opened already.

Can not open map file xxx
Map file xxx can not be opened. File xxx may be opened already.

Can not open descriptor file xxx
Descriptor file xxx can not be opened. File xxx may be opened already.

Error in function call malloc
The call to C library function malloc failed.

A.5 Program Getmap

Command format : getmap [-D=xxx -M=yyy] finm
D-descriptor file name, M-map file name
User entered invalid command format. Valid command format is show above. Try again.

Can not open symbol table file xxx
Symbol table file xxx can not be opened. File xxx may be opened already.

Can not open map file xxx
Map file xxx can not be opened. File xxx may be opened already.

Can not open descriptor file xxx
Descriptor file xxx can not be opened. File xxx may be opened already.

Error in function call malloc
The call to C library function malloc failed.

A.6 Program Bincmp32

Checksum error: internal= xxx actual = yyy
The computed checksum xxx is different from the actual checksum yyy.

Incomplete specification of segment xxxx
Segment xxxx does not have EPROM field, RAM field, EPROM base address, RAM base address or segment length

Can not open binary file xxx
Binary file xxx can not be opened. The file may be opened already.

Can not open symbol table file xxx
Symbol file xxx can not be opened. The file may be opened already.

Files have conflicting EMA's definitions
Two files have different memory fields for comparison.

Invalid format
Bincmp32 [-l=listfile] infile1 infile2
The user entered invalid format. The valid format is shown in the above line.

APPENDIX B ERROR CODES

This appendix lists the error code you may encounter in the error messages , and gives a brief explanation of each code.

Code	Description
2	Invalid port specified. It must be between 1 and 4
3	The port is not current open.
4	Invalid option requested
6	No port found at the specified address.
7	The output queue is full.
9	The port is already open.
16	The specified buffer address or queue sizes are invalid.

APPENDIX C Program Listings

This appendix contains the listing of programs documented in section 2-7.

C.1 Program Downline

C.1.1 Function Downline

```
/*    ] -- Load and transfer a PAL-8 binary file
c
c    This is an upgraded version of several previous programs.
c    Support is included in this version for:
c      1. Up to six memory fields which can be written to magtape.
c      2. Support for FIELD definitions and segment loading support.
c      3. Support for PAL-8 symbol table files.
c      4. Support for filling uninitialized locations with "ERROR"
c         symbol values (which should trap to a crash location).
c
c    Modifications:
c
c      17-Sep-85      Allow complete RS232 dump              (v2.03)
c
c      10-Apr-87      Allow RS232 dump to same terminal.     (V02.04)
c                     Load Segment 1 into its RAM space.
c                     Downline to device XL0:.
c
c      14-May-87      Allow output device specification.
c      10-jul-87      Fixup RS232 low level code             (V02.06)
c
c      19-Jan-89      Disable write retries                  (V02.07)
c
c      20-July-92     convert to C and port on PC            (V03.00)
*/



#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "downcm.h"
FILE *binfp, *symfp;
int Mem[7][010000];
/*
c... Declare arrays which will hold field definitions as they
c... are encountered. ESIM8 needs this information to copy the
c... EPROM image into the RAM image.
*/

int         Sgrbas[0100]; /* RAM base address */
```

```
int     Sgrfld[0100]; /* RAM field ID (not fully supported)*/
int     Sgebas[0100]; /* EPROM base address */
int     Sgefld[0100]; /* EPROM field ID */
int     Sgleng[0100]; /* Length of segment */



main(  int argc, char *argv[])
{
    int ercode;             /* error code */
    int port_no= 2;
    int baud_rate=7 ;
    int Hifld=1 ;
    int c, i;
    char  verdat[10], vrsion[7];
    unsigned char   inbuf[80] ;

while (--argc > 0 && (*++argv)[0] == '-')
{
    c = *++argv[0];
    switch(c){

            case 'C':
            case 'c':
                        port_no= process_port(argv);
                        break;
            case 'B':
            case 'b':
                        baud_rate=process_rate(argv);
                        break;
            case 'h':
            case 'H':
                        Hifld = process_field(argv);
                        break;
            default:
                        break;

      }  /* end of switch */
}
    if ((argc >1 ) || (argc < 1 ))
    {
            printf("invalid format , try again \n");
            printf("downline [-c] [-b] [-h] filenam\n");
            exit(1);
    }
    else
    {
            strcpy(inbuf,argv[0]);
            process_filenm(inbuf);
    }

    strcpy(verdat,"10-02-92");
    strcpy(vrsion,"v03.00");
    printf("********GEOS RS232 downline loader %s %s ***** \n", vrsion,verdat);
```

```
ercode =esim8();                /* load the memory  */

if (Hifld == 1)
{
        for( i = 0; i< HISEG ; i++)
        {
                if (Sgleng[i] > 0)
                {
                        if (Sgefld[i] > Hifld) Hifld = Sgefld[i];
                        if (Sgrfld[i] > Hifld) Hifld = Sgrfld[i];
                }
         }
}

if (ercode == 0)
{
        ercode = assign(port_no,baud_rate);          /* Assign to the correct device */
        if (ercode == 0)
        {
                send(port_no,Hifld);                /* send memory */
        }
        else
                printf("error in function assign \n");
}
else
                printf("error in function esim8 \n");
}

void process_filenm(char *inbuf)
{
 char *p;
 unsigned char binfil[81], symtab[81];

 p = strchr (inbuf,'.');

 /* a period is found */
 if (p != NULL)
 {
      printf ("*** no extension allowed in filename **** \n");
      exit(1);
 }
 else
 {
  /*  open the binary file */
      strcpy(binfil,inbuf);
      strcat(binfil,".bin");
      if ((binfp = fopen(binfil,"rb")) == NULL)
      {
                      printf(" \n Binary file not there \n ");
                      exit(1);
      }
      else
```

```
        {   /*... Open symbol table file */
                strcpy(symtab, inbuf);
                strcat (symtab, ".stb");
                if ((symfp =fopen(symtab,"r")) == NULL)
                {
                        printf( "Symbol table not there...proceeding without (gulp!) \n");
                      }
        }       /* end if */
  }          /* end if */
}


#include <stddef.h>
#include <math.h>

unsigned char   binbuf[81];
int error;
char ans[3];

/*
c    ESIM8 -- Read binary loader format file and symbol table file.
c
c    This subroutine performs the following functions:
c
c      1. User is prompted for binary/symbol table file (no extension)
c      2. The symbol table file is read, with all segment definitions
c         stored in the segment arrays.
c      3. The binary file is read, with all segments loaded into their
c         respective areas in the EPROM space.
c      4. Segments 00, 01, and 77 are loaded into its proper place in
c         RAM space. Note that these segments must contain all code
c         to bring the rest of the GEOS system up (such as loading segments).
c         (If no code has been defined for segment 00 or 77, then no
c         RAM space loading occurs for that particular segment.)
c
c
c
c    Notes:
c
c      For RS232 dumps, only the first RAM field (0) will be
c      transferred. Therefore, only segments 00 and 77 are ever loaded
c      and transferred for RS232 dumps.
c
c      For compatibility with previous versions, binary files with no
c      segment information will still be loaded into field zero.
c
c
c    Output:     ierr:  0 => User requested termination
c                      -1 => Checksum, read, or symbol table error
c                       1 => Successful load
c-
*/
int esim8()
```

```
{  int        ierr=0;        /* error return code */
    int memfld ;         /* memory field */
    int i;
/*
c... Initialize segment tables and other things
*/
    for (i=0; i< HISEG; i++)    /* Initialize segment tables*/
    {
            Sgrbas[i] = -1;
            Sgrfld[i] =  0;        /* Set this to zero since not used yet*/
            Sgebas[i] = -1;
            Sgefld[i] = -1;
            Sgleng[i] = -1;
    }
/*

*... *** Finished reading symbol table file ***
*/
    ierr = rd_sym_tbl (symfp);
    if (ierr == 0)
    {
/*.. Determine fill values in null locations
*/
            if (error == 0)
            {
                    printf("\n Do you want HLT opcodes in unused locations? \n" );
                    scanf("%s", ans);
                    if ((ans[0] == 'Y')|| (ans[0] == 'y')) error = 07402;   /* HLT opcode */
            }
/*... Fill in all memory buffers with the fill value
*/

            for (memfld = 0; memfld <= MAXFLD; memfld++)
            {
                    for( i = 0; i<= 07777; i++)
                    Mem[memfld][i] = error;
            }
/*
*... Start the loading of memory
*/

            load_mem(binfp);

    }
    return (ierr);

}

int rd_sym_tbl (FILE *symfp)
/*
*... This program reads the symbol table file. We are looking for segment
*.... definition.
*....
*... symbols of with the following values:
```

```
*...   SnnRBS       Base of segment in RAM
*...   SnnRFL       RAM field ID
*...   SnnPBS       Base of segment in EPROM
*...   SnnFLD       EPROM field ID
*...   SnnLEN       Length of segment in words
*...
*... In addition, we search for a symbol with the name "ERROR". This
*... symbol is assumed to be assigned a value which when executed by
*... the GEOS will generate a fatal error (crash). It is used to
*... fill in null words so that software errors are more likely to be
*... caught.
0*/
{
      char symnam[7];
      int  symval ;
      int  num ;
      int jseg ;

      while((num=fscanf(symfp,"%s %*x %*x %x",symnam,&symval))!= EOF )
      {

            if (strcmpi(symnam,"error") ==0 )
            {
            error =symval;              /* error opcode */
            }
            else
            {
          if ((symnam[0] == 'S')   &&
                (symnam[1] >= '0') &&   ( symnam[1]<= '7' ) &&
                (symnam[2] >= '0')  &&   ( symnam[2]<= '7' ))
           {   /* the segment # is in octal, convert to integer */
                jseg = (symnam[1]-'0')*8 + (symnam[2]-'0');

                if (strcmpi(symnam+3, "RBS") == 0)
                Sgrbas[jseg] = symval ;   /* RAM base address */

                if (strcmpi(symnam+3, "RFL") == 0)
                Sgrfld[jseg] = symval ;   /* RAM field ID */

                if (strcmpi(symnam+3, "PBS") == 0)
                Sgebas[jseg] = symval ;   /* EPROM base address */

                if (strcmpi(symnam+3, "FLD") == 0)

                Sgefld[jseg] = symval ;   /* EPROM field ID */

                if (strcmpi(symnam+3, "LEN") == 0)
                Sgleng[jseg] = symval ;   /* length of segment */
            }   /* end if */
          } /*end else */

} /* end while */

return (0);
```

```
} /* end rd_sym_tbl */

int memact = TRUE;     /* Memory loading enabled*/
int memfld = 0 ;          /* memory field */
int segact = FALSE;    /* No active valid segment yet*/
int ipc = 0;          /* Set default initial program counter*/
int isegmt = 0;        /* Set default segment number*/
int iwd1 ;            /* right 6  of 12 bits */
int iwd2 ;            /* left 6 of 12 bits */
int isum = 0;         /* Checksum of input file*/
unsigned char ichar;          /* byte storatge */
int idata = FALSE;     /* Data not pending now */
int iwdx;

int load_mem(FILE *binfp)
{
     int i ;               /* index variable */
     int ibeg = TRUE;      /* Start of file flag*/
     int nxtbyt = FALSE;   /* Byte polarity flag (0 => first byte)*/
     int num ;             /* number of fields read */
     int ierr=0;           /* error return code */

/*
*... Read next record from input file
*/
     while ((num=fread(binbuf,1,80,binfp)) != EOF && (num != 0))
     {

               for (i=0; i < num; i++)
               {

/*
*... Get next character from current buffer
*/
               ichar = binbuf[i] ;
               if (nxtbyt)     /* Process second half of word */
               {
                         iwd2 = ichar & 0377;
                         iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;        /*form full word */
                         if ((iwdx & 010000) != 0)          /* origin specification */
                         {
                                   ipc = iwdx & 07777 ;
                                   if (segact)                /* relocate to EPROM */
                                   {

                                             ipc =ipc +Sgebas[isegmt] -Sgrbas[isegmt];
                                   }
                                   isum = (iwd1+iwd2+isum ) & 07777 ;       /*compute check sum */
                         }
                         else
                                   idata = TRUE ;
                         nxtbyt = FALSE ;
```

```
}
else
        {
        switch ( (ichar & 0300)){
        case (0200):
                if (! ibeg)  /* end of input file, check checksum */
                {
                        iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;
                        if (isum != iwdx)
                        {
                                printf ("\n  checksum error: internal=  %o actual = %o \n",
                                isum , iwdx);
                                ierr = -1 ;   /* return to the caller */
                        }
                else
                        {
                        /*
                         *  Now copy Segments 00, 01, and 77 into their
                         *  RAM space. We first guarantee that each segment
                         *  is valid before copying.
                         */
                                segcpy(0);
                                segcpy(1);
                                segcpy(077);
                                ierr = 0;
                          } /* end if */
                        goto exit ;
                                } /* end if */
                break;
                case (0300):
                        process_new_segn();
                        idata = FALSE ;
                        ibeg = FALSE ;
                        nxtbyt = FALSE ;
                        break;
                default:
                        if (idata )
                        { /* pending data, flush it out  */
                                idata = FALSE ;
                                if (memact) Mem[memfld][ ipc]= iwdx ;
                                /* (fp, "iwd1 %o iwd2 %o memfld %d ipc %d Mem[memfld][ipc] %o
\n",
                                iwd1,iwd2,memfld, ipc, Mem[memfld][ipc]); */
                                ipc = (++ipc) & 07777;
                                isum = (iwd1+iwd2+isum) & 07777 ;
                        }
                        iwd1 = ichar & 0377;          /* Get first half data */
                        ibeg = FALSE ;          /* Make sure leader switch off */
                        nxtbyt = TRUE;          /* Set for second half */
                        break;
                } /*end switch */
        }/* end if */
        } /* end for */
        }   /* end while */
```

```
    exit: return ierr;
} /* end load_mem */
void process_new_segn (void)

/*
c... A segment specification has been encountered. Actions at this point
c... are as follows:
c
c     1. Find the segment attributes in the segment tables. Incomplete
c        data in the tables causes the load to be aborted.
c
c     2. Validate the memory field for storing the values. This field
c        must lie in the range between RAMBAS and MAXFLD. If this is
c        not the case, a warning message is printed, and the segment
c        is not loaded.
c
c     3. Set up the memory pointers for the EPROM field and the EPROM
c        base address for storing the subsequent data values.
c
*/
{

    if (idata)
    {

            if (memact ) Mem[memfld][ipc] = iwdx;
     /*     fprintf(fp, "iwd1 %o iwd2 %o memfld %d ipc %d Mem[memfld][ipc] %o \n",
                        iwd1,iwd2,memfld, ipc, Mem[memfld][ipc]); */
            ipc = (++ipc) & 07777;
            isum = (iwd1+ iwd2 +isum) & 07777;
    }
    isegmt = ichar & 077;       /* isolate field ID & set to segment*/
    if ( (Sgebas[isegmt] < 0) ||
     (Sgefld[isegmt] < 0) ||
     (Sgrbas[isegmt] < 0) ||
     (Sgefld[isegmt] < 0) ||
     (Sgleng[isegmt] < 0))
    {
            printf ( "  \n incomplete specification %d \n  ", isegmt);
            segact = FALSE;
            memact = FALSE;
    }
    else
    {
            memfld = Sgefld[isegmt] ;
            memact = TRUE;
            segact = TRUE;
    }
}

void  segcpy(int iseg)
{
    int  i;
```

```
    if (Sgleng[iseg] > 0 &&
    Sgefld[iseg] >= ROMBAS &&
    Sgefld[iseg] <= MAXFLD &&
    Sgrfld[iseg] >= 0 &&
    Sgrfld[iseg] < ROMBAS)
    {
      for (i = 0; i<= (Sgleng[iseg]-1); i++)
      Mem[Sgrfld[iseg]][Sgrbas[iseg]+i] =
                        Mem[Sgefld[iseg]][Sgebas[iseg]+i];

    }  /* end if */

}
```

C.1.2 Function Assign

```
#include <stdio.h>
#include <asynch_1.h>
#include "com_port.h"
char out_buf[iq_size+oq_size+20];

int  assign(int port_no,int baud_rate)
{
int ercode;

/**
      Assign - This subroutine lets user select the COM port to download
                the binary file . It then pens the selected port and assigbn
                stop bits, parity, data bit ,flow control ... to the port.
                The valid choice of the COM port is 1 or 2 presently.


*/

/* open the port */
    ercode = open_a1(port_no, iq_size,oq_size,0,0,out_buf);
    if (ercode != A_OK)
    {
            printf(" **** can not open port  %d. Error = %d \n", port_no, ercode);
            return(ercode);
    }

    ercode=setop_a1(port_no, 1, baud_rate);      /* 9600 baud rate */
    if (ercode != A_OK)
    {
            printf(" **** can not set the baud rate %d of port %d . Error = %d \n",
                                                    baud_rate,port_no, ercode);
            return(ercode);
    }



    ercode=setop_a1(port_no, 2 , 0);     /* no parity */
```

```
if (ercode != A_OK) {
        printf(" **** can not set no parity of port %d. Error= %d \n",
                port_no, ercode);
        return(ercode);
}



ercode=setop_a1(port_no, 3,3);      /* 8 data bits */
if (ercode != A_OK) {
        printf(" **** can not set 8 data bits of port %d. Error= %d \n",
                port_no, ercode);
        return(ercode);
}


ercode=setop_a1(port_no, 4 ,0);      /* one stop bit */
if (ercode != A_OK) {
        printf(" **** can not set one stop bit of port %d. Error= %d \n",
                        port_no,  ercode);
        return(ercode);
}

ercode=setop_a1(port_no, 5, 0);      /* no remote flow conftol */
if (ercode != A_OK) {
        printf(" **** remote flow control error %d \n", ercode);
        return(ercode);
    }


    ercode=setop_a1(port_no, 6, 0);      /* disable local flow control */
    if (ercode != A_OK) {
            printf(" **** can not disable local flow control of port %d. Error= %d \n",
                    port_no, ercode);
            return(ercode);
    }

    ercode=setop_a1(port_no, 9, 0);      /* require clear to send */
    if (ercode != A_OK) {
            printf(" **** can nor clear CTS of port %d. Error= %d \n",
                    port_no, ercode);
            return(ercode);
    }

    return(ercode);

}
```

C.1.3 Function Proc_Cmd

```
#include <stdio.h>
int process_port (char **argv)
{
   char c;
   int port_no;
   c=*++argv[0];
   port_no = c-48;
   switch(port_no){
      case(1):              /* com1 */
      case(2):              /* com2 */
      case(3):              /* com3 */
      case(4):              /* com4 */
            break;
      default: printf("invalid port no %s \n", c);
      }
  return(port_no);

}
int process_rate (char **argv)
{
   char c;
   int baud_rate;
   c=*++argv[0];
   baud_rate = c-48;
   switch(baud_rate){
      case(1):          /* 150*/
      case(2):          /*300 */
      case(3):          /* 600 */
      case(4):          /* 1200 */
      case(5):          /* 2400 */
      case(6):          /* 4800 */
      case(7):          /* 9600 */
            break;
      default: printf("invalid baud rate %d \n", baud_rate);
      }
  return(baud_rate);

}
int process_field (char **argv)
{
   char c;
   int highest_field;
   c=*++argv[0];
   highest_field = c-48;
   switch(highest_field){
      case(2):
      case(3):
      case(4):
      case(5):
      case(6):
            break;
      default: printf("invalid field %d \n", highest_field);
      }
```

```
  return(highest_field);

}
```

C.1.4 Function Send

```
#include <memory.h>
#include <stdio.h>
#include <string.h>
#include <asynch_1.h>
#include "com_port.h"
#include "downcm.h"
int icksum, iptr=0 ;
int snd_byts(int,int);
void punch(int,int) ;
void punch1(int,int) ;
void leader(int);
extern int Mem[7][010000];
unsigned char buf[81];
void  send(int port_no,int Hifld)
{
/*
     Subroutine Send...Send memory down the RS232 port
*/

      const int       MASK12=07777;
      const int       TOPMEM=07777;
      const int       ORGBIT=010000;
      const int       FLDBIT =0300;


      char inbuf[20];  /* user input buffer */
      int i ,j ;       /* index variable  or temp variable */

      char *result;

 start:
      printf("start GEOS boot, hit return to start loading ...\n");
      gets(inbuf);

/* ... Start converting memory and send it down!


... Prepare and then decide how much to send
*/
     icksum = 0 ;            /*  Zero checksum */
     wait_a1(182);           /* Wait for 10 seconds */


     leader(port_no) ;               /* Punch initial leader */

      for (i=0; i<= Hifld; i++)
      {
              j= FLDBIT + i;       /* Field code  */
```

```
            punch1(j,port_no);          /* Output field */
            icksum += j ;
            punch(ORGBIT,port_no);      /* Output address 0 */

            for (j = 0; j<=TOPMEM ; j++)
            {
                    punch(Mem[i][j]&MASK12,port_no) ;

            }
    }


    i = icksum;
    punch(i,port_no);               /* Punch the checksum  */
    leader(port_no) ;               /* Punch trailing leader */

    printf(".....loading completed! check sum sent %o....\n",i);

    memset(inbuf,' ',10);
    inbuf[10]='\0' ;            /* append null character */
    printf( "repeat loading of this file [y/n]? \n");
    gets(inbuf);
    result=strpbrk(inbuf,"Yy");
    if (result != NULL)  goto start;

}
void  leader(int port_no)
/*
     Punch a leader on the binary output
*/
{
    snd_byts(iptr,port_no);         /* Flush output buffer */
    buf[0] = 0200;          /*   Leader value */
    iptr = 1 ;
    snd_byts(iptr,port_no);         /* Flush buffer again */

    iptr=0;             /* reset the pointer */


}

void  punch (int ival,int port_no)
/*
    Punch the integer value in binary format to the GEOS
*/
{
    int left,iright,scratch;

    scratch =ival >> 6 ;            /* shift left six position */
    left = scratch &  0177 ;
    iright = ival & 077  ;
    icksum = icksum + left + iright;
    icksum = icksum & 07777;
    punch1 (left,port_no) ;
```

```
    punch1 (iright,port_no);
}

void punch1 (int ival,int port_no)
/*
     Punch a singe nibble to the GEOS
*/
{

    if (iptr >= 80)
    {
            snd_byts(80,port_no);    /* Flush the buffer*/
            iptr = 0;
    }
    buf[iptr] = ival;
    iptr += 1;


}

int snd_byts(int out_len,int port_no)
{
    int written;      /* number of bytes successfully written to the queue */
    int ercode ;      /* error return code */
    int current ;

/*  int i, j;     */

 /*  for (i=0; i< out_len; i++)
    {
            fprintf(fp, " %o ", buf[i]);
    }

    fprintf(fp, "\n");
 */

try:
    ercode = oqsiz_a1(port_no, &current);
    if (ercode != A_OK)
            printf(" *** oqsiz_a1 error %d \n", ercode);
    else
    {  if ((oq_size - current)> (out_len+2) )
            {  ercode = wrtst_a1(port_no,out_len,buf,&written);
                    if (ercode != A_OK)
                    printf(" *** wrtst_a1 error %d %d %d %d %d \n",
                                    ercode,out_len,oq_size,current,written);
            }
            else
            {
                    wait_a1(18);
                    goto try;
            }
    }
```

```
        return (ercode);
    }
```

C.2 Program GDT

C.2.1 Function GDT

```
#include <stdio.h>
#include <string.h>
#include "gdtcom.h"

void readmemory(int *,char *);
void readsymf(char *);
void intact(void );
void print_error(void);
int memory[2][4096];
void main(int argc , char *argv[])
{
   int *ptr;
   char c;
   char dmpfinm[20], symfinm[20];
   /* set up the default file names */
   strcpy(dmpfinm,"mem.dmp");
   strcpy(symfinm,"record.stb");

/* process command options */
while( (-- argc > 0) && ((*++argv)[0] == '-'))
{
   c = *++argv[0];
   switch(c){
   case ('d'):
   case ('D'):
       if (( c= *++argv[0]) == '=')
       {
          strcpy(dmpfinm,++argv[0]);

       }
       else print_error();
       break;
   case ('S'):
   case ('s'):
       if (( c = *++argv[0]) == '=')
       {
         strcpy(symfinm,++argv[0]);
       }
       else print_error();
       break;
  default:
    print_error();
    break;
```

```
    }  /* end switch */
  }    /* end while */
  printf("Interactive Debugger of GEOS memory imagine file \n");
  printf("Memory Imagine File: %s \nSymbol Table File : %s \n",
       dmpfinm,symfinm);
  ptr = &memory[0][0];
  readmemory(ptr,dmpfinm );      /* read into memory from dumped disk file */
  readsymf(symfinm);             /* read symbol table file */
  intact();                 /*  Interact with user until exit */

} /* end of main */

int hash(char *symbol)

/* compute the hash value of input string symbol */
{
  int hashval;
  int n,i;
  for (n=0, i=0; i< 6;i++)
  {
   /* left shift the accumulator -n and add the netx character */

     n= (n<<1) + *(symbol+i) ;
  }
  hashval = n % hashtbllen;
  return(hashval);
}

void print_error(void)
{
  printf("invalid format \n");
  printf("gdt [-s=symfinm]  [-d=dmpfinm] \n");
}
```

C.2.2 Function Intact

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "gdtcom.h"
/* This file contains the definition of function lookup, convertnum,intact and printloc*/

int hash(char*);
extern struct st_entry *hashtbl[];
extern int memory[2][4096];
struct st_entry *lookup(char *symname)

 /*{ Lookup searches for the symbol in Buf array from the hash table.}
 { If the symbol is found, its hash entry pointer is returned as   }
 { the function result. If not found, then a NULL pointer is       }
 { returned. */
```

```
{
    struct st_entry *iptr;
    int hashindx;
    hashindx = hash(symname);


    iptr =  hashtbl[hashindx];
    while (iptr != NULL)
    {
            if (strncmp(iptr->name, symname,symnamlen) == 0)
                    break;

            else
                    iptr = iptr->next;
     }
     return(iptr);
}



int convertnum(char *buf)
{

 /*   { ConvertNum converts the octal number in the input buffer (Buf)  }
      { from ASCII to binary, and verifies that the number represented  }
      { is a twelve bit number. This number is returned as the function  }
      { value. If a bad conversion occurs, the return value is -1.       }
  */
      int val,index;

      for (index=0; buf[index] != '\0'; index++)
      {
              if ((buf[index] < '0') || (buf[index] >'7'))
               {
                       break ;
               }
      }
      if( buf[index] == '\0')
      {  sscanf(buf,"%o", &val);  /* convert the string to a decimal # */
              if (val >= 4096) val = -1 ;
      }
      else
              val = -1;
      return (val);
}

void printloc(int curloc, int curfield, int *ptr)

      /* { Printloc prints the current location and the location's value }
      */

       {
              printf("current field: %o location: %o content: %o \n",
                                      curfield, curloc, *ptr);
```

```
        }


void intact(void)
{   int curloc;
     int curfield;
     int found = 0;
     int prompt, len,i,val  ;
     char inbuf[10],buf[10];
     int *ptr_memory;
     struct st_entry *iptr;


     curloc = 0;              /* Initial location */
     curfield = 0;            /* Initial field */
     prompt = TRUE ;          /*  Force an error prompt */
     while (TRUE)
     {
               if (prompt)
                 printf("? ");
               gets(inbuf);
               len = strlen(inbuf);

               for (i=0; i< len ; i++)          /* convert to upper case */
                 inbuf[i] = toupper(inbuf[i]);
sscanf(inbuf,"%s", buf);       /* left justified the buffer  */
switch(buf[0]){
case ('\0'):            /* next location */
  curloc = (curloc +1 ) & 4095;
  ptr_memory = &memory[curfield][curloc];
  printloc(curloc,curfield ,ptr_memory);
  break;
case ('0'):case('1'):case('2'):case('3'):case('4'):
case ('5'):case('6'):case('7'):case('8'):case('9'):
  i=convertnum(&buf[0]);
  if (i < 0)
  {
            printf("enter numerical # in octal \n");
            prompt = TRUE;
          }
  else                    /* print the content of current location */
  {   curloc = i;
                    ptr_memory = &memory[curfield][curloc];
                    printloc(curloc,curfield ,ptr_memory);


  }
                    break;
case ('@'):             /* indirect */
  curloc= memory[curfield][curloc];
  ptr_memory = &memory[curfield][curloc];
  printloc(curloc,curfield ,ptr_memory);
```

```
        break;

case ('$'):
        if (buf[1] == 'S')
        {                       /*search for word */

                printf("octal number to search for \n");
                gets(inbuf);
                len = strlen(inbuf);
                if (len == 0) break ;       /* exit */
                else
                {
                        val = convertnum(&inbuf[0]);
                        if (val < 0)
                        {
                                printf("invalid value %s \n", inbuf);
                        }
                        else
                        {
                                for (i=0; i<=4095; i++)
                                if (val == memory[curfield][i])
                                {
                                        ptr_memory = &memory[curfield][i];
                                        printloc(i,curfield ,ptr_memory);
                                        found++;
                                 /*  break;  */
                                }
                                if (found == 0)
                                {
                                        printf("word not found %s \n", inbuf);
                                }
                                else
                                        curloc = i;
                        }
                }
 }
 break;

case ('/'):
  buf[0] = '0';
  i= convertnum(buf);
  if ((i < 0) || (i > memoryfields))  /* field change */
  {
        printf("** Invalid field specified ** \n");
  }
  else
         curfield = i;

  break;
 default :
        if ((buf[0] >='A')  && (buf[0] <= 'Z'))
        {
                iptr = lookup(&buf[0]);
```

```
                        if (iptr == NULL)
                        {
                                printf("symbol is undefined %s \n", buf);
                        }
                        else
                        {
                                curloc = iptr->value;
                                ptr_memory = &memory[curfield][curloc];
                                printf("symbol %s at ", buf);
                                printloc(curloc,curfield ,ptr_memory);
                         }
                }
                else
                        printf("invalid command %s \n", inbuf);
                break;
        }  /* end case */
    } /* end while */
    printf("end of while len %d\n", len);
} /* end intact */
```

C2.3 Function Memdisk

```
#include <stdio.h>
#include <stdlib.h>
#include "gdtcom.h"

void dumpmemory (int mem[][4096])
{
/* This routine prompts for the name of a new file and writes the
  memory image array to the file for later analysis by this program. */

  FILE *fp;
  char answer[40];
  char dumpfiname[10];
  int flag = 1 ;
  int fieldindex ;          /* index variable */

  while (flag == 1)
  {  /* prompt user for file name */
    printf("name of memory dump file ? \n");
    gets(answer);
    if (sscanf(answer, "%s", dumpfiname) != 1)
    {
      printf ("incorrect file name, reenter please \n");
      continue;
    }
    /* open memory dump file */

    fp = fopen(dumpfiname, "rb");
    if (fp == NULL)
    {
```

```
        printf("can not open dump file , enter a new name \n");
        continue;
      }
      flag = 0;
    }

     for (fieldindex = 0; fieldindex<= memoryfields; fieldindex++)
     {
       fwrite (&mem[fieldindex][0], 2,membuffersize ,fp);

     }

     fclose(fp);
}
void readmemory (int *mem, char *dumpfiname)
{
/* This routine prompts reads into the memory image array
   from the dumped disk file for later analysis
   by this program. */
/* mem is the name of two-dimensional array */

    FILE *fp;
    int membuffer[membuffersize];
    int fieldindex,i ;          /* index variable */

    fp = fopen(dumpfiname, "rb");
    if (fp == NULL)
    {
       printf("can not open dump file , try a new name \n");
       exit(1);

    }

   for (fieldindex = 0; fieldindex<= memoryfields; fieldindex++)
    {
       fread(membuffer, 2,membuffersize,fp);
       /*
       printf("%x %x %x %x \n",        membuffer[0],membuffer[1],membuffer[2],membuffer[4095]);
       */
       for (i = 0 ; i < membuffersize  ; i ++)
       *(mem+ (fieldindex *membuffersize ) + i) = membuffer[i] ;
    }

    fclose(fp);
}
```

C.2.4 Function Readsym

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "gdtcom.h"
```

```
int hash(char *);
struct st_entry *hashtbl[hashtbllen] ;


void readsymf(char *symfile)
{



/*

Procedure to initialize the hash symbol table, and read the symbols
into the hash table
 */

  int value,
     i  ;                        /* index variable */
  FILE *fp;
  char inbuf[80],
      symname[symnamlen];
  struct st_entry *symtab_entry;

   fp =fopen(symfile, "r");
   if (fp == NULL)
   {
      printf("can not open symbol file %s  \n",symfile);
      exit(1);
   }



   for (i=0; i< hashtbllen ; i++)
      hashtbl[i] = NULL ;

   while (fgets(inbuf,80,fp) != NULL)
   {
      sscanf(inbuf,"%s %*x %*x %x", symname, &value);
      symtab_entry = (struct st_entry *) (malloc(sizeof (struct st_entry)));
      if (symtab_entry == NULL)
      {
       printf("error in malloc %s \n", symname);
      }
      strcpy(symtab_entry->name,symname);
      symtab_entry->value = value;
      symtab_entry->next = hashtbl[hash(symname)];
      hashtbl[hash(symname)] = symtab_entry;
    }
   fclose(fp);


  }
```

C.3 Program Dataio

C.3.1 Function Dataio

```
/*
C+
C  DATAIO -- Program to send binary code to the DATA I/O EPROM Programmer
C
C  Written by:  G. L. Maxwell
C       24-Jan-85
C
C  Converted from the original EPROM program.
C
C  Modifications:
C   rewrote it in c and port to PC by Meei-You Lee
       5/10/94
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "dataio.h"


void prompt(char *);
int esim8(FILE *,FILE *);
int assign(int,int);
int sendit(int,int,int*);
void close_port(int);
void dmp_mem(void);
int process_port(char **);
int process_rate(char **);
void print_error(void);
void process_filenm(char *,char *, FILE **, FILE **);
unsigned int mem[ROWSIZE][COLSIZE];


void main(int argc, char *argv[])
{



      int    ierr;
      int    flag = 1 ;
      int    first = 1;
      int    port_no= 2;
      int    baud_rate=7 ;
      int    c;
      char   symtab[20], inbuf[20];
      FILE   *binfp, *symfp;
/*
C... MAIN CODE
C
*/
```

```
        printf(" DATAIO -- Data I/O Programmer Communicator %s  %s \n",
                                LASMOD,VRSION);



strcpy(symtab,"\0\0\0");

while (--argc > 0 && (*++argv)[0] == '-')
{
    c = *++argv[0];
    switch(c){

            case 'C':
            case 'c':
                                port_no= process_port(argv);
                                break;
            case 'B':
            case 'b':
                                baud_rate=process_rate(argv);
                                break;

            case 'S':
            case 's':
                               strcpy(symtab,++argv[0]);
                                break;
            default:
                                break;
       }  /* end of switch */
}

    if ((argc >1 ) || (argc < 1 ))
    {
            print_error();
            exit(1);
    }
    else
    {
            strcpy(inbuf,argv[0]);

            process_filenm(inbuf,symtab,&binfp,&symfp);
    }
/*
C... Assign  EPROM port
*/

    /* ans[0] ='N';    */

    ierr= assign(port_no,baud_rate);

/* load a binary file , send it to Eprom programmer depending on user choice */

    ierr=esim8(binfp,symfp);
    if (ierr< 0)
```

```
    {
      printf("error in program esim8 \n");
    }
    else
    {
      ierr=sendit(COLSIZE,port_no, &mem[0][0]);
      if (ierr < 0) printf("error in program sendit \n");

    }

dmp_mem();
close_port(port_no);

printf("end of program \n");

}

void prompt(char *ans)
{
  int flag = 1 ;
  char inbuf[20];

  /*  give user some options.   */
  while (flag == 1)
  {

      printf( "(Main) Enter : \n");
      printf("      N - To input a new binary file \n");
      printf("      C - To return to command mode \n");
      printf("      Q - To exit \n");
      gets(inbuf);
      if (sscanf(inbuf, "%s",ans) != 1)
      {
              printf("invalid input,enter N, C or Q to try again \n");
            }
      else
              flag = 0;

    }               /* end of while */
} /* end of prompt */
void dmp_mem(void)
{
int i,j,k;
FILE *fp_mem;


fp_mem = fopen("mem_out","w+");
for (i=2; i<=6; i++)
{
    fprintf(fp_mem, "00000    " );
    for (j=0,k=0; j<=07777; j++)
      {
              fprintf(fp_mem, "%5o ", (mem[i][j]& 07777));
              k++;
```

```
                if ((k% 8)  == 0)
                {
                        fprintf(fp_mem,"\n");
                        fprintf(fp_mem,"%5o    ", k);

                }
        }
}
}
void print_error(void)
{
     printf("invalid command format , try again \n");
     printf("dataio [-c] [-b] [-s=symname] filenam\n");
}

void process_filenm(char *inbuf,char *symtab,FILE **fp1,FILE **fp2)
{
 char *p;
 unsigned char binfil[81];
 FILE *binfp,*symfp;

 p = strchr (inbuf,'.');

 /* a period is found */
 if (p != NULL)
 {
      printf ("*** no extension allowed in filename **** \n");
      exit(1);
}
else
{
 /*  open the binary file */
      strcpy(binfil,inbuf);
      strcat(binfil,".bin");
      if ((binfp = fopen(binfil,"rb")) == NULL)
      {
                  printf(" \n Binary file not there \n ");
                  exit(1);
      }
      else
      {   /*... Open symbol table file */
                if (strlen(symtab) == 0)
                {

                     strcpy(symtab, inbuf);
                     strcat (symtab, ".stb");
                }
                if ((symfp =fopen(symtab,"r")) == NULL)
                {
                     printf( "Symbol table not there...\n");
                     exit(1);
                    }
      }         /* end if */
}          /* end if */
```

```
  *fp1 = binfp;
  *fp2 = symfp;
}
```

C.3.2 Function Assign

```
#include <stdio.h>
#include <asynch_1.h>
#include <string.h>
#include "com_port.h"

int  assign(int port_no,int baud_rate)
{

     int ercode ;       /* erro return code */
     int try =0;

/**
      Assign - This subroutine lets user select the COM port to download
                        the binary file . It then pens the selected port and assigbn
                        stop bits, parity, data bit ,flow control ... to the port.
                        The valid choice of the COM port is 1 or 2 presently.


*/

/* open the port */
     ercode = open_a1(port_no, iq_size,oq_size,0,0,out_buf);
if (ercode != A_OK)
{
          printf(" **** can not open port  %d. Error = %d \n", port_no, ercode);
          return(ercode);
}

ercode=setop_a1(port_no, 1, baud_rate);       /* 9600 baud rate */
if (ercode != A_OK)
{
          printf(" **** can not set the baud rate %d of port %d . Error = %d \n",
                                                  baud_rate,port_no, ercode);
          return(ercode);
}



ercode=setop_a1(port_no, 2 , 0);      /* no parity */
if (ercode != A_OK) {
          printf(" **** can not set no parity of port %d. Error= %d \n",
                    port_no, ercode);
          return(ercode);
}



ercode=setop_a1(port_no, 3,3);        /* 8 data bits */
```

```
if (ercode != A_OK) {
        printf(" **** can not set 8 data bits of port %d. Error= %d \n",
                port_no, ercode);
        return(ercode);
}


ercode=setop_a1(port_no, 4 ,0);      /* one stop bit */
if (ercode != A_OK) {
        printf(" **** can not set one stop bit of port %d. Error= %d \n",
                        port_no,  ercode);
        return(ercode);
}

ercode=setop_a1(port_no, 5, 0);      /* no remote flow conftol */
if (ercode != A_OK) {
        printf(" **** remote flow control error %d \n", ercode);
        return(ercode);
}


ercode=setop_a1(port_no, 6, 0);      /* disable local flow control */
if (ercode != A_OK) {
        printf(" **** can not disable local flow control of port %d. Error= %d \n",
                port_no, ercode);
        return(ercode);
}
    ercode=setop_a1(port_no, 9, 0);      /* require clear to send */
    if (ercode != A_OK) {
            printf(" **** can nor clear CTS of port %d. Error= %d \n",
                    port_no, ercode);
            return(ercode);
    }

    return(ercode);

}
void close_port(int port_no)
{
  close_a1(port_no);
}


int snd_byts(int out_len, int port_no, char *buf)
{
    int written;      /* number of bytes successfully written to the queue */
    int ercode ;      /* error return code */
    int current ;
    int try = 1;



    while (try == 1)
```

```
    {   try = 0;
            ercode = oqsiz_a1(port_no, &current);
            if (ercode != A_OK)
                    printf(" can not call function oqsize successfully. Error = %d \n", ercode);
            else
            {   if ((oq_size - current)> (out_len+2) )
                    {   ercode = wrtst_a1(port_no,out_len,buf,&written);
                            if (ercode != A_OK)
                            printf(" *** can not call wrtst_a1 successfully. Error= %d %d %d %d %d \n",
                                    ercode,out_len,oq_size,current,written);
                     }
                    else
                    {
                            wait_a1(30);
                            try = 1;
                     }   /* end if */
            } /* end if */
    }   /* end while */
    return (ercode);
}
```

C.3.3 Function Esim8

```
/*    ] -- Load and transfer a PAL-8 binary file
c
c    This is an upgraded version of several previous programs.
c    Support is included in this version for:
c
c      1. Up to six memory fields which can be written to magtape.
c      2. Support for FIELD definitions and segment loading support.
c      3. Support for PAL-8 symbol table files.
c      4. Support for filling uninitialized locations with "ERROR"
c         symbol values (which should trap to a crash location).
c
c
c    20-July-92      convert to C and port on PC          (V03.00)
*/



#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <math.h>
#include "dataio.h"

void process_new_segn(void);
int rd_sym_tbl (FILE *);
int load_mem(FILE *);
```

```
unsigned char   binbuf[81];


int    Sgrbas[HISEG]; /* RAM base address */
int    Sgrfld[HISEG]; /* RAM field ID (not fully supported)*/
int    Sgebas[HISEG]; /* EPROM base address */
int    Sgefld[HISEG]; /* EPROM field ID */
int    Sgleng[HISEG]; /* Length of segment */
extern unsigned int    mem[ROWSIZE][COLSIZE];

int    error;
/*
c    ESIM8 -- Read binary loader format file and symbol table file.
c
c    This subroutine performs the following functions:
c
c      1. User is prompted for binary/symbol table file (no extension)
c      2. The symbol table file is read, with all segment definitions
c         stored in the segment arrays.
c      3. The binary file is read, with all segments loaded into their
c         respective areas in the EPROM space.
c      4. Segments 00, 01, and 77 are loaded into its proper place in
c         RAM space. Note that these segments must contain all code
c         to bring the rest of the GEOS system up (such as loading segments).
c         (If no code has been defined for segment 00 or 77, then no
c         RAM space loading occurs for that particular segment.)
c
c
c
c    Notes:
c
c      For RS232 dumps, only the first RAM field (0) will be
c      transferred. Therefore, only segments 00 and 77 are ever loaded
c      and transferred for RS232 dumps.
c
c      For compatibility with previous versions, binary files with no
c      segment information will still be loaded into field zero.
c
c
c    Output:    ierr:  0 => User requested termination
c                     -1 => Checksum, read, or symbol table error
c                      1 => Successful load
c-
*/
int esim8(FILE *binfp, FILE *symfp)

{  int        ierr=0;        /* error return code */
     unsigned char       ans[2];
     int         i;
     size_t      nlen=0;
     int memfld ;          /* memory field */
/*
c... Initialize segment tables and other things
*/
```

```
    for (i=0; i< HISEG; i++)    /* Initialize segment tables*/
    {
            Sgrbas[i] = -1;
            Sgrfld[i] =  0;        /* Set this to zero since not used yet*/
            Sgebas[i] = -1;
            Sgefld[i] = -1;
            Sgleng[i] = -1;
    }

/*

*... *** Finished reading symbol table file ***
*/
    ierr = rd_sym_tbl (symfp);
    if (ierr == 0)
    {
/*.. Determine fill values in null locations
*/
            if (error == 0)
            {
                    printf("\n Do you want HLT opcodes in unused locations? \n" );
                    scanf("%s", ans);
                    if ((ans[0] == 'Y')|| (ans[0] == 'y')) error = 07402;   /* HLT opcode */
            }
/*... Fill in all memory buffers with the fill value
*/
            for (memfld = ROMBAS; memfld <= ROMEND; memfld++)
            {
                    for( i = 0; i<= 07777; i++)
                    mem[memfld][i] = error;
            }
/*
*... Start the loading of memory
*/

            load_mem(binfp);

    }
    return (ierr);

}

int rd_sym_tbl (FILE *symfp)
/*
*... This program reads the symbol table file. We are looking for segment
*.... definition.
*....
*... symbols of with the following values:
*...   SnnRBS      Base of segment in RAM
*...   SnnRFL      RAM field ID
*...   SnnPBS      Base of segment in EPROM
*...   SnnFLD      EPROM field ID
*...   SnnLEN      Length of segment in words
```

```
*...
*... In addition, we search for a symbol with the name "ERROR". This
*... symbol is assumed to be assigned a value which when executed by
*... the GEOS will generate a fatal error (crash). It is used to
*... fill in null words so that software errors are more likely to be
*... caught.
*/
{
      char symnam[7];
      int  symval ;
      int  num ;
      int jseg ;

      while((num=fscanf(symfp,"%s %*x %*x %x",symnam,&symval))!= EOF )
      {

            if (strcmpi(symnam,"error") ==0 )
            {
            error =symval;              /* error opcode */
            }
            else
            {
            if ((symnam[0] == 'S')   &&
                  (symnam[1] >= '0') &&   ( symnam[1]<= '7' ) &&
                  (symnam[2] >= '0')  &&   ( symnam[2]<= '7' ))
             {   /* the segment # is in octal, convert to integer */
                  jseg = (symnam[1]-'0')*8 + (symnam[2]-'0');

                  if (strcmpi(symnam+3, "RBS") == 0)
                  Sgrbas[jseg] = symval ;   /* RAM base address */

                  if (strcmpi(symnam+3, "RFL") == 0)
                  Sgrfld[jseg] = symval ;   /* RAM field ID */

                  if (strcmpi(symnam+3, "PBS") == 0)
                  Sgebas[jseg] = symval ;   /* EPROM base address */

                  if (strcmpi(symnam+3, "FLD") == 0)

                  Sgefld[jseg] = symval ;   /* EPROM field ID */

                  if (strcmpi(symnam+3, "LEN") == 0)
                  Sgleng[jseg] = symval ;   /* length of segment */
             }  /* end if */
            } /*end else */

      } /* end while */

      return (0);

} /* end rd_sym_tbl */

int memact = TRUE;    /* Memory loading enabled*/
int memfld = 0 ;        /* memory field */
```

```
int segact = FALSE;   /* No active valid segment yet*/
int ipc = 0;          /* Set default initial program counter*/
int isegmt = 0;       /* Set default segment number*/
int iwd1 ;            /* right 6  of 12 bits */
int iwd2 ;            /* left 6 of 12 bits */
int isum = 0;         /* Checksum of input file*/
unsigned char ichar;          /* byte storatge */
int idata = FALSE;    /* Data not pending now */
int iwdx;

int load_mem(FILE *binfp)
{
     int i ;              /* index variable */
     int ibeg = TRUE;     /* Start of file flag*/
     int nxtbyt = FALSE;  /* Byte polarity flag (0 => first byte)*/
     int num ;            /* number of fields read */
     int ierr=0;          /* error return code */

/*
*... Read next record from input file
*/

     while ((num=fread(binbuf,1,80,binfp)) != EOF && (num != 0))
     {

               for (i=0; i < num; i++)
               {
/*
*... Get next character from current buffer
*/
               ichar = binbuf[i] ;
               if (nxtbyt)     /* Process second half of word */
               {
                         iwd2 = ichar & 0377;
                         iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;       /*form full word */
                         if ((iwdx & 010000) != 0)          /* origin specification */
                         {
                                   ipc = iwdx & 07777 ;
                                   if (segact)                /* relocate to EPROM */
                                   {

                                             ipc =ipc +Sgebas[isegmt] -Sgrbas[isegmt];
                                   }
                                   isum = (iwd1+iwd2+isum ) & 07777 ;       /*compute check sum */
                         }
                         else
                                   idata = TRUE ;
                         nxtbyt = FALSE ;
               }
               else
                         {
                         switch ( (ichar & 0300)){
                         case (0200):
```

```
if (! ibeg)  /* end of input file, check checksum */
{
        iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;
        if (isum != iwdx)
        {
                printf ("\n  checksum error: internal=  %o actual = %o \n",
                isum , iwdx);
                ierr = -1 ;   /* return to the caller */
        }
else
        {
        /*
         *   successful program load. To save time programming
         *   the EPROMs we fill the upper four unused bits of
         *    the memory with 1s.
         */
                for (memfld = ROMBAS; memfld <= ROMEND; memfld ++)
                {
                /************need to dump the memory */
                /***********************************/
                        for (i=0; i<= 07777; i++)
                        {
                                mem[memfld][i] = mem[memfld][i]|
0170000;
                         };
                 }
                ierr = 0;
                        } /* end if */
                goto exit ;
                } /* end if */
        break;
                case (0300):
                        process_new_segn();
                        idata = FALSE ;
                        ibeg = FALSE ;
                        nxtbyt = FALSE ;
                        break;
          default:
                        if (idata )
                        { /* pending data, flush it out  */
                                idata = FALSE ;
                                if (memact) mem[memfld][ ipc]= iwdx ;
                                ipc = (++ipc) & 07777;
                                isum = (iwd1+iwd2+isum) & 07777 ;
                        }
                        iwd1 = ichar & 0377;          /* Get first half data */
                        ibeg = FALSE ;          /* Make sure leader switch off */
                        nxtbyt = TRUE;          /* Set for second half */
                        break;
                } /*end switch */
  }/* end if */
  } /* end for */
  }   /* end while */
exit: return ierr;
```

```
} /* end load_mem */
void process_new_segn (void)

/*
c... A segment specification has been encountered. Actions at this point
c... are as follows:
c
c     1. Find the segment attributes in the segment tables. Incomplete
c        data in the tables causes the load to be aborted.
c
c     2. Validate the memory field for storing the values. This field
c        must lie in the range between RAMBAS and MAXFLD. If this is
c        not the case, a warning message is printed, and the segment
c        is not loaded.
c
c     3. Set up the memory pointers for the EPROM field and the EPROM
c        base address for storing the subsequent data values.
c
*/
{

    if (idata)
    {

            if (memact ) mem[memfld][ipc] = iwdx;
    /*    fprintf(fp, "iwd1 %o iwd2 %o memfld %d ipc %d mem[memfld][ipc] %o \n",
                        iwd1,iwd2,memfld, ipc, mem[memfld][ipc]); */
        ipc = (++ipc) & 07777;
        isum = (iwd1+ iwd2 +isum) & 07777;
}
isegmt = ichar & 077;       /* isolate field ID & set to segment*/
if ( (Sgebas[isegmt] < 0) ||
 (Sgefld[isegmt] < 0) ||
 (Sgrbas[isegmt] < 0) ||
 (Sgefld[isegmt] < 0) ||
 (Sgleng[isegmt] < 0))
{
        printf ( "  \n incomplete specification of segment : %d \n  ", isegmt);
        segact = FALSE;
        memact = FALSE;
}
else
{
        if ((Sgefld[isegmt] < ROMBAS) || (Sgefld[isegmt] >= ROMEND))
        {
                memact = FALSE ;
                segact  = FALSE;
                printf( "segment %d cant not map to a valid EPROM \n", isegmt);
        }
        else
        {
                memfld = Sgefld[isegmt] ;
                memact = TRUE;
                segact = TRUE;
```

```
                }
        }
}
```

C3.4 Function Proc_cmd

```
#include <stdio.h>
int process_port (char **argv)
{
   char c;
   int port_no;
   c=*++argv[0];
   port_no = c-48;
   switch(port_no){
      case(1):              /* com1 */
      case(2):              /* com2 */
      case(3):              /* com3 */
      case(4):              /* com4 */
            break;
      default: printf("invalid port no %s \n", c);
      }
  return(port_no);

}
int process_rate (char **argv)
{
   char c;
   int baud_rate;
   c=*++argv[0];
   baud_rate = c-48;
   switch(baud_rate){
      case(1):              /* 150*/
      case(2):              /*300 */
      case(3):              /* 600 */
      case(4):              /* 1200 */
      case(5):              /* 2400 */
      case(6):              /* 4800 */
      case(7):              /* 9600 */
            break;
      default: printf("invalid baud rate %d \n", baud_rate);
      }
  return(baud_rate);

}
int process_field (char **argv)
{
   char c;
   int highest_field;
   c=*++argv[0];
   highest_field = c-48;
   switch(highest_field){
      case(2):
      case(3):
```

```
      case(4):
      case(5):
      case(6):
            break;
      default: printf("invalid field %d \n", highest_field);
      }
  return(highest_field);

}
```

C3.5 Function Sendit

```
/*
C  SENDIT -- Interactively select and transfer EPROM images
C       to DATA I/O Eprom Programmer
C
C  Inputs:   Memory array has been initialized with GEOS image
C
C  Outputs:  Ierr:  If <= 0, then user requested termination or fatal error
C          If >  0, then normal return
C
C  This routine transfers data to the programmer using the following
C  format:
C      Intel Intellec 8/MDS Format (Select code 83 on System 19)
C      Record size: 16 data bytes (43 actual characters)
C
C  Since two GEOS fields are accomodated by a 64K Eprom, this routine
C  assumes that the valid starting fields for loading EPROMS are
C  ROMBAS, ROMBAS+2, etc. up to ROMEND. Therefore, the user selects
C  the starting field number for the EPROM, and whether the low
C  or high byte should be transfered to System 19.
C
*/



#include "dataio.h"
#include <stdio.h>
#include <stdlib.h>

#define BYTCNT  16      /* ! No. data bytes per record */
#define BUFLEN  80      /* ! Size of data buffer */
#define BYTSIZ  8192    /*  No. bytes per EPROM  */
#define COLON       '\072'
#define CR          '\r'
#define LF          '\n'

static FILE *fp_debug;

void hexcnv(int,char*);
void close_port(int);
```

```
int snd_byts(int,int, char*);


int sendit (int cols, int port_no, unsigned int *mem)

{
#define MEM(i,j)  (*(mem + ((i*cols) + j)))

int   flag ,flag1,address,i,j,low,ierr ;
int   jfield, ifield,iadd,icksum ;
char  outbuff[BUFLEN]  ;
unsigned int  ival;
char line[81],inbuf[BUFLEN];

/*
  Begin SENDIT
*/
        fp_debug = fopen("dbg_out","w+");
        if (fp_debug == NULL) printf("can not open dbg_out \n");

        ierr = 1  ;
        outbuff[0] = LF  ;          /*  Set line feed as first character */
        flag = TRUE;

        while (flag)
        {
                flag1 = TRUE;
                printf(" (sendit) enter :\n");
                printf("       L - To load System 19 memory \n");
                printf("       Q - To exit                  \n");
                gets(line);
                sscanf(line ,"%s", inbuf);

                switch(inbuf[0]){
                case('L'):
                case ('l'):
                        /* select starting field and byte to treanfser */
                        while (flag1)
                        {
                                flag1 = FALSE ;
                                printf("Enter starting field and high (H) or low (L) byte, (e.g. '2H'): ");
                                gets(line);
                                if (sscanf(line, "%s", inbuf) == 1)
                                {  /* check alignment of base field */
                                        for(j= ROMBAS; j<=ROMEND; j += 2)
                                                if ( j == (inbuf[0]-'0'))  break;      /* exit the for loop */
                                        ifield=inbuf[0] -'0';

                                        if (j > ROMEND)
                                        {
                                                printf(" Bad field alignment. Starting field must be in one of
following: %o %o \n", ROMBAS, ROMEND);
                                                flag1 = TRUE;
```

```
                        }
                        else
                        {   /* check the low or high byte */
                                low = -1 ;
                                if ((inbuf[1] == 'L') ||  (inbuf[1] == 'l'))  low =1;
                                if ((inbuf[1] == 'H') || (inbuf[1] == 'h')) low = 0;
                                if (low < 0)
                                {
                                        printf(" Enter 'L' or 'H' for byte selection \n");
                                        flag1 = TRUE ;
                                }
                        } /* end if */
                } /* end if*/
        } /* end while */
        break;
case('Q'):
case ('q'):
        ierr = 0;               /* set  return flag */
        goto exit;
        break;
default:
        printf("invalid input , enter L or Q to try again \n");
        break;
} /* end switch */
```

```
/*
... Transfer the data down
*/

    printf(" Select format 83 and select D1 to start System 19\n");
    printf("   Hit RETURN to start transfer...\n");
    gets(line);
    printf("' Transferring...\n");



    address = 0;        /*  "Address" for System 19  */

    for (jfield = ifield; jfield <= (ifield+1) ; jfield ++)
            for (iadd = 0; iadd < 07777 ; iadd += BYTCNT)
            {
                    outbuff[1] =  COLON ;       /*  Start character  */
                    hexcnv(BYTCNT,(outbuff+2));    /*  Byte count     */
                    icksum = BYTCNT;            /* initialize checksum */
                    i = (address/256) & 0377;     /* Get high byte of address */
                    hexcnv(i,(outbuff+ 4));       /*  Convert to hex   */
                    icksum += i ;               /*  Add checksum    */
                    i = address % 256;            /*  Get low byte of address */
                    hexcnv(i,(outbuff+6));        /*  Convert to hex  */
                    icksum +=  i ;              /* Add checksum   */
```

```
                    hexcnv(0,(outbuff+8));        /*  Record type = '00'   */

                    for (j=10, i=iadd; j < 10 + BYTCNT*2; j+=2, i++)
                    {
                      if (low > 0)
                              ival =  (MEM(jfield,i) & 0377) ;   /* Low byte */
                      else
                              ival = ((MEM(jfield,i) & ~0377) >> 8); /* High byte*/

                      hexcnv(ival, (outbuff+j));          /*  Convert it  */
                      icksum += ival;                /*   Compute checksum  */

                     fprintf(fp_debug, "mem[jfield][i] %o %d %d ival %o \n",
                                        MEM(jfield,i),jfield,i,ival);

                     }

/*      Store checksum and output to System 19       */
                    j = 10 + BYTCNT*2 ;
                    icksum = -(icksum % 256);

                    hexcnv(icksum,(outbuff+j));          /* Store checksum     */
                    j +=  2 ;
                    outbuff[j] = CR ;                 /* Put in carriage return */
                /*debug **********************/
                fprintf(fp_debug,"outbuff %s \n", outbuff);
                /* SEND OUT THE BUFFER ********************/
                    ierr= snd_byts((j+1),port_no,outbuff);
                    address = address + BYTCNT ;          /*  Bump System 19 address  */
      }

/*.../*  Transfer end of file record  */

                outbuff[1] = COLON   ;
                hexcnv(0,(outbuff+2));        /* Byte count   */
                hexcnv(0,(outbuff+4));        /* Dummy address     */
                hexcnv(0,(outbuff+6));
                hexcnv(1,(outbuff+8));        /* Record type    */
                hexcnv(0377,(outbuff+10));     /*  Store checksum  */


/*
... Transfer complete

*/

     fprintf(fp_debug,"outbuf , %s \n", outbuff);
     ierr = snd_byts(12,port_no, outbuff);
     printf("transfer complete \n");
     }       /* end while */
/*...  End of routine  */
exit:
      close_port(port_no);
      return(ierr);
```

```
}



void hexcnv(int bval, char *buff)

/*
  hexcnv  - Convert value in "bval" to a two character Hex format
*/
{
      div_t  quot_rem;

     bval = bval & 0377;              /* mask off high order 8 bits */

     quot_rem = div(bval,16);        /* get the quotient and remainer */

     if (quot_rem.quot > 9)
               buff[0]  = (char) (quot_rem.quot -10 + 'A') ;
     else
               buff[0] = (char) (quot_rem.quot + '0');
     if (quot_rem.rem  >  9)
               buff[1]  = (char) (quot_rem.rem  -10 + 'A');
     else
               buff[1] = (char) (quot_rem.rem  + '0');

}
```

C.4 Program Getfld

C.4.1 Function Gtefld

```
/* This program generates EPROM memory map of GEOS application
   program.
   converted to C and ported to PC MS DOS by meei-you lee

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "getfld.h"

int initfiles(int, char *[]);
void readsymf(FILE *);
struct st_entry *lookup(char *);
void printerror(char *,int);
void printfield(int);
void createsegment(int,int,int,int,int);
void printmap(void);
void printbounds(int,int);
void printhole(int,int);

FILE *fp_map;
FILE *fp_sym;
```

```
FILE *fp_dsc;
SegmentRecordPtr FieldHead[7];
char *descriptor[64];

main(int argc, char *argv[])
{
   char lenstr[7] ;        /* length */
   char pbsstr[7] ;        /* EPROM base address */
   char rbsstr[7];         /* RAM base address */
   char fldstr[7];         /* EPROM field */
   char rflstr[7];         /* RAM field */

   int i,iema;
   int ch1,ch2,numdef;
   int fldval,pbsval,rbsval,lenval;
   struct st_entry *ptr;

   strcpy(lenstr, "S  LEN");
   strcpy(pbsstr, "S  PBS");
   strcpy(rbsstr, "S  RBS");
   strcpy(fldstr, "S  FLD");
   strcpy(rflstr,  "S  RFL");

   initfiles(argc,argv);
   readsymf(fp_sym);          /* build a symbol table */

   for (iema = BASEEMA ; iema<=TOPEMA ; iema++)
   {
    FieldHead[iema] =NULL;
   }

/* get the segment definition for each field used in the source assembly
   program. The field # starts from 0 and stops at 63. Not every number
   in between these two numbers is used . It gets info about segment length, eprom field,
   eprom base address, ram field, ram base address from the symbol table
*/
for (i=0; i<=63 ; i++)
{    /*   begin { Get map }   */
 ch1 = ((i /8) + 48);
 ch2 = ((i % 8) + 48);          /* field number is ascii */
 numdef = 0;

 /* Get the length   */

 lenstr[1] = (char)ch1;
 lenstr[2] = (char)ch2;
 ptr = lookup(lenstr);          /*lookup the symbol table */
 if (ptr != NULL)

    if ((ptr->value >= 0) && (ptr->value <= 07777))
    {

       lenval = ptr->value;
       numdef = 1 + numdef;
```

```
    }

   else
     printerror(lenstr, ptr->value);

/* Get EPROM base */

pbsstr[1] = (char)ch1;
pbsstr[2] = (char)ch2;
ptr = lookup(pbsstr);

if ( ptr != NULL)

   if ((ptr->value >= 0) && (ptr->value <= 07777))
   {
     /* begin { Valid }    */
     pbsval = ptr->value;
     numdef = numdef + 1 ;
   }    /* end { Valid }    */

   else
     printerror(pbsstr, ptr->value);

/* Get RAM base */

rbsstr[1] = (char)ch1;
rbsstr[2] = (char)ch2;
ptr = lookup(rbsstr);
if (ptr != NULL)

   if ((ptr->value >= 0) && (ptr->value <= 07777))
   {
     rbsval = ptr->value;
     numdef = 1 + numdef;
   }   /* end { Valid }    */

else
     printerror(rbsstr, ptr->value);

/* { Get EPROM EMA ID }    */

fldstr[1] = (char)ch1;
fldstr[2] = (char) ch2;
ptr = lookup(fldstr);
if (ptr != NULL)

   if ((ptr->value >= BASEEMA) && (ptr->value <= TOPEMA))
   {
    /*    begin { Valid }        */
      fldval = ptr->value;
      numdef = numdef + 1;
   }   /*    end { Valid }       */

else
```

```
      printerror(fldstr, ptr->value);

/* { Get RAM EMA ID } */

rflstr[1] = (char)ch1;
rflstr[2] = (char)ch2;
ptr = lookup(rflstr);
if (ptr != NULL)

  if ((ptr->value >= BASEEMA) && (ptr->value <= TOPEMA))
     numdef++;

  else
     printerror(rflstr, ptr->value);

switch (numdef) {

   case(1):
   case(2):
   case(3):
   case(4):
      /*  { Partial definition }  */
     fprintf(fp_map,"*** Incomplete specification of following field:\n");
     printfield(i);
     fprintf(fp_map,"\n");
     fprintf(fp_map,"     *** Definition ignored. ***\n");
     break ;  /* end { Partial definition } */

        case(5):
           createsegment(i, fldval, pbsval, rbsval, lenval);
           break;
        default: break;
       }
    }  /* end { Get map } */

   printmap();
   fclose(fp_map);

}


void createsegment(int SegID, int fld, int pbs,int rbs,int len)
{
  SegmentRecordPtr  prevptr, nextptr, segptr;
  boolean done;


 /*  begin { CreateSegment }     */
  segptr = (SegmentRecordPtr)(malloc(sizeof(struct SegmentRecordType)));
  segptr->ebase = pbs;
  segptr->rbase = rbs;
  segptr->id    = SegID;
  segptr->length = len ;
```

```
prevptr = NULL;
nextptr = NULL;

if (FieldHead[fld] != NULL)
{   /*  begin { Find spot in memory }  */
  nextptr = FieldHead[fld];
  done = FALSE;
  while (! done)
  {   /* begin { Compare bases }    */
    if (segptr->ebase < nextptr->ebase)
    {
      done = TRUE;
    }
    else
    {   /* begin { Chain }    */
      prevptr = nextptr;
      nextptr = prevptr->nextsegment;
      done = (nextptr ==NULL);
    }   /*  end { Chain }     */
  }   /* end { Compare bases }  */
 }   /* end { Find spot in memory } */

if (prevptr != NULL)
   prevptr->nextsegment = segptr ;
else
   FieldHead[fld] = segptr;
   segptr->nextsegment = nextptr ;

}   /* end { CreateSegment } ; */


void printmap(void)
{
  SegmentRecordPtr scratchsegment, prevsegment, nextsegment;
  int iema,offset,endaddr;

  /*  begin { Printmap }    */
  scratchsegment = (SegmentRecordPtr)malloc(sizeof(struct SegmentRecordType));
  scratchsegment->ebase = 0;
  scratchsegment->rbase = 0;
  scratchsegment->length = 0;

  fprintf(fp_map," \n   EPROM Memory Map: \n");

  for (iema = BASEEMA; iema <= TOPEMA ; iema++)
  {   /* begin { Map for each field }  */
    if (FieldHead[iema] == NULL)
    {
      fprintf(fp_map,"EMA Field%5o :  No space allocated \n", iema);
    }
    else
    {   /* begin { Get field map }  */
      fprintf(fp_map, "\nEMA Field%5o\n", iema);
```

```
prevsegment = scratchsegment;
nextsegment = FieldHead[iema];

do {
   endaddr = prevsegment->ebase + prevsegment->length; /*{ End of last segment } */
   offset = nextsegment->ebase - endaddr;
   if (offset > 0)
   {    /* begin { Memory hole }   */
      printhole(endaddr, offset);
      offset = 0;
   }  /* end { Memory hole } */

   if (offset < 0)
      fprintf(fp_map,"*  ");    /* { Segment overlap } */

    /* begin { Print segment }    */
    printfield(nextsegment->id);
    printbounds(nextsegment->ebase, nextsegment->length);
   /*  end { Print segment } */

    prevsegment = nextsegment;
    nextsegment = nextsegment->nextsegment;
}    while ( nextsegment != NULL) ;

endaddr = prevsegment->ebase + prevsegment->length;
offset = 010000 - endaddr;
if (offset > 0)
        printhole(endaddr, offset);
       fprintf(fp_map , " \n");

    }   /* end { Get field map }   */
  }    /* end { Map for each field } ;  */

}   /* end { PrintMap } ;   */

void printbounds(int base, int length)
{
  int endaddress;


  if (length == 0)
     endaddress = 0 ;
  else
     endaddress = ((base + length - 1) & 07777);

  fprintf(fp_map, "%10o : %6o %6o\n",base,endaddress,length);


}   /*   end { PrintBounds } ;   */


void printhole(int base, int length)
{
    fprintf(fp_map,"    <hole>                          ");
```

```
    printbounds(base , length);
}
void printerror(char *buf, int val)
{

  fprintf(fp_map, "** Illegal symbol value: %s val: %d   ** \n", buf,val);
}

void printfield(int field )
{
   fprintf(fp_map, "%3o ",field);
   if (descriptor[field] != NULL )
     fprintf(fp_map,"%-38s", descriptor[field]);
   else
     fprintf(fp_map,"                                    ");
}
```

C.4.2 Function Initfi

```
/* The subroutine forms map file name , symbol table file name and description
  file name from the command line argument. It then opens each one of these
  files and report error message if it can not open any one of these files.

*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "getfld.h"

extern FILE *fp_map;
extern FILE *fp_sym;
extern FILE *fp_dsc;

extern char *descriptor[64];
void print_format(void);


int initfiles(int argc, char *argv[])
{
     char mapfinm[FINMLEN];          /* filename of map */
     char symfinm[FINMLEN];          /* filename of symbol table */
     char dscfinm[FINMLEN];          /* filename of description file */
     char inbuf[BUFLEN],wka[BUFLEN];
     char filenm[FINMLEN];
     int flag =TRUE;
     int index;
     char *ptr;

     /*  set the default file names */
     if (argc > 1)
     {

               strcpy(filenm,argv[argc-1]);
               if (strpbrk(filenm,".")!= NULL)
               {   /* no dot in the file name */
```

```
                ptr = strstr(filenm,".");
                *(ptr) = '\0';   /* do not inclue dot */
            }
            /* form file names */
            strcpy(mapfinm,filenm);
            strcpy(symfinm,filenm);
            strcpy(dscfinm,filenm);
            strcat(mapfinm,".map");
            strcat(symfinm,".stb");
            strcat (dscfinm,".fld");
}

switch (argc) {
        case (1): /* incorrect format , print the correct format */
                        print_format();
                        break;
        case (2): /*  user selects the default file name */
                        break;

        default :  /* user enters the map file name &/! descriptor file name */
                        while (--argc > 1)
                                if ((*++argv)[0] == '-')
                                {
                                        flag = FALSE;
                                        switch(*++argv[0]){
                                                case ('m'):
                                                case ('M'): strcpy(mapfinm,argv[0]+2);
                                                                break;
                                                case ('d'):
                                                case ('D'): strcpy(dscfinm,argv[0]+2);
                                                                break;
                                                case ('s'):
                                                case ('S'): strcpy(symfinm,argv[0]+2);
                                                                break;
                                                default   : print_format();
                                                                flag = TRUE;
                                                                break;
                                         }
                                }
                                else  print_format();
        }    /* end switch */
/* open all files */
fp_sym=fopen(symfinm, "r");
fp_dsc= fopen(dscfinm,"r");
fp_map=fopen(mapfinm,"w+");
if (fp_sym == NULL)
        printf("can not open symbol table file %s \n", symfinm);
if (fp_map == NULL)
        printf("can not open map file %s \n",mapfinm);

if (fp_dsc == NULL)
        printf("can not open descriptor file %s \n", dscfinm);
```

```
    /* read the descriptor file and store the descriptor into
      array descriptor */

    while ( (fgets(inbuf,80,fp_dsc) != NULL))
    {
            sscanf(inbuf,"%o %[^\n]",&index, wka);
            descriptor[index]=(char *)(malloc(sizeof(inbuf)));
            strcpy(descriptor[index],wka);

     }

    return(1);

}
void print_format(void)
{
    printf(" command format : getfld [-D=xxx -M=yyy]  finm \n");
    printf (" D-descriptor file name M-map file name \n");
    exit(1);
}
```

C.4.3 Function Readsym

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "getfld.h"

int hash(char *);
struct st_entry *hashtbl[hashtbllen] ;


void readsymf(FILE *fp)
{



/*
Procedure initializes
the hash table, and reads the symbols into the hash table
 */
    int value,
            i ;                     /* index variable */
    char inbuf[80],
            symname[symnamlen];
    struct st_entry *symtab_entry;


     for (i=0; i< hashtbllen ; i++)
            hashtbl[i] = NULL ;

     while (fgets(inbuf,80,fp) != NULL)
     {
            sscanf(inbuf,"%s %*x %*x %x", symname, &value);
            symtab_entry = (struct st_entry *) (malloc(sizeof (struct st_entry)));
```

```
                if (symtab_entry == NULL)
                {
                  printf("error in function call malloc  \n");
                }
                strcpy(symtab_entry->name,symname);
                symtab_entry->value = value;
                symtab_entry->next = hashtbl[hash(symname)];
                hashtbl[hash(symname)] = symtab_entry;
        }
       fclose(fp);


}


int hash(char *symbol)
/* compute the hash value of the input string symbol */
{
      int hashval;
      int n,i;
      for (n=0, i=0; i< 6;i++)
      {
       /* left shift the accumulator -n and add the next character */
                n= (n<<1) + *(symbol+i) ;
      }
      hashval = n % hashtbllen;
      return(hashval);
}


struct st_entry *lookup(char *symname)

      /*{ Lookup searches for the symbol in Buf array from the hash table.}
      { If the symbol is found, its hash entry pointer is returned as   }
      { the function result. If not found, then a NULL pointer is       }
      { returned. */
{
      struct st_entry *iptr;
      int hashindx;

      hashindx = hash(symname);
      iptr =  hashtbl[hashindx];
      while (iptr != NULL)
      {

                if (strncmp(iptr->name, symname,symnamlen) == 0)
                        break;

                else
                        iptr = iptr->next;
       }
       return(iptr);
}
```

C.5 Program Getmap

C.5.1 Function Getmap

```
/* This program generates RAM memory map of GEOS application
   program.
   converted to C and ported to PC MS DOS by meei-you lee  5/1/94

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "getfld.h"

int initfiles(int, char *[]);
void readsymf(FILE *);
struct st_entry *lookup(char *);
void printerror(char *,int);
void printfield(int);
void createsegment(int,int,int,int,int);
void printmap(void);
void printbounds(int,int);
void printhole(int,int);

FILE *fp_map;
FILE *fp_sym;
FILE *fp_dsc;
SegmentRecordPtr FieldHead[7];
char *descriptor[64];

main(int argc, char *argv[])
{
   char lenstr[7] ;        /* length */
   char pbsstr[7] ;        /* EPROM base address */
   char rbsstr[7];         /* RAM base address */
   char fldstr[7];         /* EPROM field */
   char rflstr[7];         /* RAM field */

   int i,iema;
   int ch1,ch2,numdef;
   int rflval,             /* RAM field id */
      fldval,              /* EPROM field id */
      pbsval,              /* EPROM base value */
      rbsval,              /* RAM base value */
      lenval;              /* length of the field */
   struct st_entry *ptr;

   strcpy(lenstr, "S  LEN");
   strcpy(pbsstr, "S  PBS");
   strcpy(rbsstr, "S  RBS");
   strcpy(fldstr, "S  FLD");
   strcpy(rflstr,  "S  RFL");
```

```
initfiles(argc,argv);
readsymf(fp_sym);          /* build a symbol table */

for (iema = BASEEMA ; iema<=TOPEMA ; iema++)
{
 FieldHead[iema] =NULL;
}

/* get the segment definition for each field used in the source assembly
   program. The field # starts from 0 and stops at 63. Not every number
   in between is used . It gets info about segment length, eprom field,
   eprom base address, ram field, ram base address from the symbol table
*/
for (i=0; i<=63 ; i++)
{    /*  begin { Get map }  */
 ch1 = ((i /8) + 48);
 ch2 = ((i % 8) + 48);          /* field number is ascii */
 numdef = 0;

 /* Get the length  */

 lenstr[1] = (char)ch1;
 lenstr[2] = (char)ch2;
 ptr = lookup(lenstr);          /*lookup the symbol table */
 if (ptr != NULL)

    if ((ptr->value >= 0) && (ptr->value <= 07777))
    {

      lenval = ptr->value;
      numdef++;
    }

  else
    printerror(lenstr, ptr->value);

/* Get EPROM base */

pbsstr[1] = (char)ch1;
pbsstr[2] = (char)ch2;
ptr = lookup(pbsstr);

if ( ptr != NULL)

  if ((ptr->value >= 0) && (ptr->value <= 07777))
  {
    /* begin { Valid }   */
    pbsval = ptr->value;
    numdef = numdef + 1 ;
  }   /* end { Valid }  */

  else
    printerror(pbsstr, ptr->value);
```

```
/* Get RAM base */

rbsstr[1] = (char)ch1;
rbsstr[2] = (char)ch2;
ptr = lookup(rbsstr);
if (ptr != NULL)

  if ((ptr->value >= 0) && (ptr->value <= 07777))
  {
    rbsval = ptr->value;
    numdef = 1 + numdef;
  }  /* end { Valid }  */

else
    printerror(rbsstr, ptr->value);

/* { Get EPROM EMA ID }   */

fldstr[1] = (char)ch1;
fldstr[2] = (char) ch2;
ptr = lookup(fldstr);
if (ptr != NULL)

  if ((ptr->value >= BASEEMA) && (ptr->value <= TOPEMA))
  {
   /*   begin { Valid }      */
     fldval = ptr->value;
     numdef = numdef + 1;
  }  /*   end { Valid }      */

else
    printerror(fldstr, ptr->value);

/* { Get RAM EMA ID } */

rflstr[1] = (char)ch1;
rflstr[2] = (char)ch2;
ptr = lookup(rflstr);
if (ptr != NULL)

  if ((ptr->value >= BASEEMA) && (ptr->value <= TOPEMA))
  {
    rflval = ptr->value;
    numdef++;
  }
  else
    printerror(rflstr, ptr->value);

switch (numdef) {

   case(1):
   case(2):
   case(3):
   case(4):
```

```
            /*  { Partial definition }  */
           fprintf(fp_map,"*** Incomplete specification of following field:\n");
           printfield(i);
           fprintf(fp_map,"\n");
           fprintf(fp_map,"     *** Definition ignored. ***\n");
           break ;  /* end { Partial definition } */

         case(5):
            createsegment(i, rflval, pbsval, rbsval, lenval);
            break;
         default: break;
        }
     }  /* end { Get map } */

    printmap();
    fclose(fp_map);

}


void createsegment(int SegID, int fld, int pbs,int rbs,int len)
{
  SegmentRecordPtr  prevptr, nextptr, segptr;
  boolean done;


 /*  begin { CreateSegment }      */
segptr = (SegmentRecordPtr)(malloc(sizeof(struct SegmentRecordType)));
segptr->ebase = pbs;
segptr->rbase = rbs;
segptr->id    = SegID;
segptr->length = len ;

prevptr = NULL;
nextptr = NULL;

if (FieldHead[fld] != NULL)
{   /*  begin { Find spot in memory }  */
  nextptr = FieldHead[fld];
  done = FALSE;
  while (! done)
  {   /* begin { Compare bases }    */
    if (segptr->ebase < nextptr->ebase)
    {
      done = TRUE;
    }
    else
    {  /* begin { Chain }    */
      prevptr = nextptr;
      nextptr = prevptr->nextsegment;
      done = (nextptr ==NULL);
    }  /*  end { Chain }     */
  }   /* end { Compare bases } */
 }   /* end { Find spot in memory } */
```

```
  if (prevptr != NULL)
     prevptr->nextsegment = segptr ;
  else
     FieldHead[fld] = segptr;

  segptr->nextsegment = nextptr ;

}   /* end { CreateSegment } ; */


void printmap(void)
{
  SegmentRecordPtr scratchsegment, prevsegment, nextsegment;
  int iema,offset,endaddr;

  /*  begin { Printmap }    */
  scratchsegment = (SegmentRecordPtr)malloc(sizeof(struct SegmentRecordType));
  scratchsegment->ebase = 0;
  scratchsegment->rbase = 0;
  scratchsegment->length = 0;

  fprintf(fp_map," \n   RAM Memory Map: \n");

  for (iema = BASEEMA; iema <= TOPEMA ; iema++)
  {     /* begin { Map for each field }  */
    if (FieldHead[iema] == NULL)
{
   fprintf(fp_map,"RAM Field%5o :  No space allocated \n", iema);
}
else
{    /* begin { Get field map }  */
   fprintf(fp_map, "\nEMA Field%5o\n", iema);
   prevsegment = scratchsegment;
   nextsegment = FieldHead[iema];

   do {
      endaddr = prevsegment->ebase + prevsegment->length; /*{ End of last segment } */
      offset = nextsegment->ebase - endaddr;
      if (offset > 0)
      {    /* begin { Memory hole }    */
         printhole(endaddr, offset);
         offset = 0;
      }  /* end { Memory hole } */

      if (offset < 0)
         fprintf(fp_map,"*  ");    /* { Segment overlap } */

       /* begin { Print segment }    */
       printfield(nextsegment->id);
       printbounds(nextsegment->ebase, nextsegment->length);
      /*  end { Print segment } */

       prevsegment = nextsegment;
```

```
            nextsegment = nextsegment->nextsegment;
        }    while ( nextsegment != NULL) ;

        endaddr = prevsegment->ebase + prevsegment->length;
        offset = 010000 - endaddr;
        if (offset > 0)
          printhole(endaddr, offset);
        fprintf(fp_map , " \n");

    }   /* end { Get field map }   */
  }    /* end { Map for each field } ;  */

}   /* end { PrintMap } ;   */

void printbounds(int base, int length)
{
  int endaddress;


  if (length == 0)
     endaddress = 0 ;
  else
     endaddress = ((base + length - 1) & 07777);

  fprintf(fp_map, "%10o : %6o %6o\n",base,endaddress,length);

}   /*   end { PrintBounds } ;   */


void printhole(int base, int length)
{
    fprintf(fp_map,"    <hole>                          ");
    printbounds(base , length);
}
void printerror(char *buf, int val)
{

  fprintf(fp_map, "** Illegal symbol value: %s val: %d   ** \n", buf,val);
}

void printfield(int field )
{
   fprintf(fp_map, "%3o ",field);
   if (descriptor[field] != NULL )
     fprintf(fp_map,"%-38s", descriptor[field]);
   else
     fprintf(fp_map,"                                    ");
}
```

C.6 Bincmp32

C.6.1 function BINCMP32

```
/*
c+
c    BINCMP32 -- Compare two PAL 8 binary files for differences
c       for use with new 32 K program memory board
c
c    Written 3-Sep-81 by G.L.M.
c
c    Modifications:
c
c   1-Feb-82    Initialize memory with HLT opcodes
c
c   21-Sep-87   Upgrade to 32K version
c
c   07-Sep-88   Print base EPROM fields only (2, 4, or 6)
c
c    04-JUNE-94 convert to c and  port to PC
c
C    command syntax:
c
c    Bincmp32  -l=listfile  infile1 infile2
c
c    Where:
c
c    listfile  is the output log file indicating file differences. The
c           default output device is TI:.
c    infile1
c    infile2  are the PAL 8 binary files which are compared, chuck by
c           byte, for all defined EMA fields. Bytes that differ are
c           indicated in the output file.
c
c
*/
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include "bincmp32.h"

void process_input_file(char *,FILE **,FILE **);
void process_list_file(char *,FILE **);
void print_error(void);
void convert_to_uppercase(char *);
int esim8(FILE *,FILE *,int (**)[010000]);
void display_difference( FILE *,int,int,int (*)[010000], int (*)[010000]);
unsigned int     mem1[ROMTOP+1][010000], mem2[ROMTOP+1][010000] ;
char binbuf[81];
int num ;
main(int argc, char *argv[])
{
  FILE  *fp_bin1, *fp_bin2, *fp_sym1,*fp_sym2,*fp_lst;
  int field1,field2;
     int (**ptr)[010000];
     switch(argc){
     case (3):
```

```
                process_list_file("stdout",&fp_lst);
                process_input_file(argv[1],&fp_bin1,&fp_sym1);
                process_input_file(argv[2],&fp_bin2,&fp_sym2);
                break;
    case (4):
                process_list_file(argv[1], &fp_lst);
                process_input_file(argv[2],&fp_bin1,&fp_sym1);
                process_input_file(argv[3],&fp_bin2,&fp_sym2);
                break;
    default :
                print_error();
                break;
 }
 /* load the first binary file to memory */

 *ptr =mem1;
 field1= esim8(fp_sym1,fp_bin1,ptr  );

    /* load the second binary file to memory */
 *ptr = mem2 ;
 field2 = esim8(fp_sym2,fp_bin2,ptr );

 /* diplay the difference to th listing file */
 display_difference(fp_lst,field1,field2,mem1,mem2);
 exit(0);

}
void process_input_file(char *filenm, FILE **fp_1, FILE **fp_2)

/* This subroutine uses the input argument filenm to form the name of
   binary file and the name of symbol table. It opens both files and
   passes the file pointers to both files back to the caller */


{
     char bin_fi_nm[40],stb_fi_nm[40],*ptr;
     int i;
     FILE *fp , *fp_sym;
     convert_to_uppercase(filenm);
     /* search for dot in the file name */

     if (strpbrk(filenm,".")== NULL)
     {   /* no dot in the file name */
                /* form file names */
                strcpy(bin_fi_nm,filenm);
                strcat(bin_fi_nm,".BIN");
                strcpy(stb_fi_nm,filenm);
                strcat(stb_fi_nm,".STB");
      }
     else
     {   /* dot in the file name */
                /* copy the binary file name */
                strcpy(bin_fi_nm,filenm);
                ptr = strstr(bin_fi_nm,".BIN");
```

```
                strncpy(stb_fi_nm,bin_fi_nm,ptr-bin_fi_nm);
                i = (int) (ptr-bin_fi_nm);
                *(stb_fi_nm +i) = '\0';
                /* form the symbol table file name */
                strcat(stb_fi_nm,".STB");

        }

        fp=fopen(bin_fi_nm,"rb");
        if (fp == NULL)
        {
                printf("cant not open binary file %s \n", bin_fi_nm);
                exit(1);
         }
         else
         {
                 fp_sym=fopen(stb_fi_nm,"r+");

                 if (fp_sym == NULL)
                 {
                         printf("can not open symbol table file %s \n", stb_fi_nm);
                         exit(1);
                  }
          }
/* pass back the file pointers */

*fp_1 = fp;
*fp_2 = fp_sym;

}
void process_list_file(char *filenm,FILE **fp)
/* opens the list file and passes the file pointer back to the
  caller. The default list file is the standard output
*/

{
  char lstfil[20];
  char *ptr;

  if ( strcmp(filenm, "stdout") == 0)
  {
       *fp = stdout;

  }
  else
  {
       /* the list file format -l=filenm */
                ptr = strchr(filenm,'=');
       if (ptr != NULL)
       {    strcpy(lstfil,ptr+1);

                        *fp = fopen(lstfil,"w+");
       }
       else
```

```
        {
                print_error();
        }
    }
}
void display_difference(FILE *fp, int field1,int field2,int (* mem1)[010000],int (*mem2)[010000])
{
    int i;
    int j;
    int jbasl=0 ,jbash= 0;
    int flag = 0;
    if (field1 != field2)
                fprintf(fp," files have conflicting EMA's definitions \n");

    for (i=ROMBAS; i<=ROMTOP; i++)
    {
            if ((i % 2) == 0)
            {
                jbasl = 0;
                jbash = 0;
            }
            for (j=0; j<=07777;j++)
            {
                if (jbasl != 0) break;
                if ((mem1[i][j] & 0377) != (mem2[i][j] & 0377))
                {   flag = 1;
                        jbasl = 1;
                        break;
                }
            }
            for (j=0; j<=07777;j++)
            {
                if (jbash != 0) break;
                if ((mem1[i][j] >> 8) != (mem2[i][j] >> 8))
                {   flag =1;
                        jbash = 1;
                        break;
                }
            }
            if ((flag == 1) && ((jbasl ==1 ) || (jbash == 1)))
                fprintf(fp,"differing chip groups and high/low indications  ");

            if (jbasl == 1)
            {
                 fprintf(fp, "field %o low \n\n", i-(i%2));
                 jbasl = 2;
            }
            if (jbash == 1)
            {
                fprintf(fp, "field %o high \n\n", i-(i%2));
                jbash = 2;
            }
}
  for (i=ROMBAS; i<=ROMTOP;i++)
```

```
                for (j=0; j<=07777; j++)
                        if (mem1[i][j] != mem2[i][j])
                                fprintf(fp, "Field %o Loc %o  Old: %o  New: %o\n",
                                                                i, j, mem1[i][j],mem2[i][j]);
}
void convert_to_uppercase(char *string)
{
     int len,i;
     len =strlen(string);
     for (i=0; i<len;i++)
     {
                if (islower(string[i]))
                        string[i] = toupper(string[i]);
     }
}
void print_error(void)
{
      printf("illegal format \n");
      printf("bincmp32 [-l=listfile] infile1 infile2 \n");
      exit(1);
 }
```

C.6.2. Function Esim8

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "bincmp32.h"

void process_new_segn(int (**)[010000]);
int rd_sym_tbl (FILE *);
int load_mem(FILE *, int (**)[010000]);
unsigned char   binbuf[81];
/*
c... Declare arrays which will hold field definitions as they
c... are encountered. ESIM8 needs this information to copy the
c... EPROM image into the RAM image.
*/

int      Sgrbas[HISEG]; /* RAM base address */
int      Sgrfld[HISEG]; /* RAM field ID (not fully supported)*/
int      Sgebas[HISEG]; /* EPROM base address */
int      Sgefld[HISEG]; /* EPROM field ID */
int      Sgleng[HISEG]; /* Length of segment */
int      error;      /* value of symbol error in symbol table */



/*
c    ESIM8 -- Read binary loader format file and symbol table file.
c
```

```
c    This subroutine performs the following functions:
c
c      The symbol table file is read, with all segment definitions
c      stored in the segment arrays.
c      The binary file is read, with all segments loaded into their
c      respective areas in the EPROM space.
c
c
c
c
c    Output:    ifield:  0 => User requested termination
c                        -1 => Checksum, read, or symbol table error
c                        >0 => memory fields that have loaded value.
c-
*/
int esim8(FILE *symfp, FILE *binfp,int (**Mem)[010000])

{
     int ierr  ;          /* error return code */
     int        i;
     int memfld ;         /* memory field */
     /*
c... Initialize segment tables and other things
*/

     for (i=0; i< HISEG; i++)     /* Initialize segment tables*/
     {
               Sgrbas[i] = -1;
               Sgrfld[i] =  0;        /* Set this to zero since not used yet*/
               Sgebas[i] = -1;
               Sgefld[i] = -1;
               Sgleng[i] = -1;
     }

/*

*... *** Finished reading symbol table file ***
*/
     ierr = rd_sym_tbl(symfp);
     if (ierr == 0)
     {
/*.. Determine fill values in null locations
*/
               if (error == 0)
               {
                         error = 07402;   /* HLT opcode */
               }
/*... Fill in all memory buffers with the fill value
*/


               for (memfld = 0; memfld <= MAXFLD; memfld++)
               {
                         for( i = 0; i<= 07777; i++)
```

```
                    {
                            (*Mem)[memfld][i] = error;
                     }
              }



/*
*... Start the loading of memory
*/

              ierr = load_mem(binfp,Mem);

     }

 return (ierr);

}

int rd_sym_tbl (FILE *symfp)
/*
*... This program reads the symbol table file. We are looking for segment
*.... definition.
*....
*... symbols of with the following values:
*...   SnnRBS        Base of segment in RAM
*...   SnnRFL        RAM field ID
*...   SnnPBS        Base of segment in EPROM
*...   SnnFLD        EPROM field ID
*...   SnnLEN        Length of segment in words
*...
*... In addition, we search for a symbol with the name "ERROR". This
*... symbol is assumed to be assigned a value which when executed by
*... the GEOS will generate a fatal error (crash). It is used to
*... fill in null words so that software errors are more likely to be
*... caught.
0*/
{
      char symnam[7];
      int  symval ;
      int  num ;
      int jseg ;



      while((num=fscanf(symfp,"%s %*x %*x %x",symnam,&symval))!= EOF )
      {
            /* printf("input %s %d \n", symnam,symval);  */
            if (strcmpi(symnam,"error") ==0 )
            {
            error =symval;              /* error opcode */
            }
```

```
        else
        {
        if ((symnam[0] == 'S')    &&
                (symnam[1] >= '0') &&    ( symnam[1]<= '7' ) &&
                (symnam[2] >= '0')  &&    ( symnam[2]<= '7' ))
         {    /* the segment # is in octal, convert to integer */
                jseg = (symnam[1]-'0')*8 + (symnam[2]-'0');

                if (strcmpi(symnam+3, "RBS") == 0)
                Sgrbas[jseg] = symval ;   /* RAM base address */

                if (strcmpi(symnam+3, "RFL") == 0)
                Sgrfld[jseg] = symval ;   /* RAM field ID */

                if (strcmpi(symnam+3, "PBS") == 0)
                Sgebas[jseg] = symval ;   /* EPROM base address */

                if (strcmpi(symnam+3, "FLD") == 0)

                Sgefld[jseg] = symval ;   /* EPROM field ID */

                if (strcmpi(symnam+3, "LEN") == 0)
                Sgleng[jseg] = symval ;   /* length of segment */
          }   /* end if */
        } /*end else */

} /* end while */
```

```
      return (0);

} /* end rd_sym_tbl */

int memact ;     /* Memory loading enabled*/
int memfld  ;          /* memory field */
int segact ;    /* No active valid segment yet*/
int ipc ;          /* Set default initial program counter*/
int isegmt ;       /* Set default segment number*/
int iwd1 ;           /* right 6  of 12 bits */
int iwd2 ;           /* left 6 of 12 bits */
int isum ;         /* Checksum of input file*/
unsigned char ichar;           /* byte storatge */
int idata ;     /* Data not pending now */
int iwdx;
int ifield;        /* flag to indicate the EMA fields */
int load_mem(FILE *binfp, int (**Mem)[010000])
{
     int i ;               /* index variable */
     int ibeg = TRUE;       /* Start of file flag*/
     int nxtbyt = FALSE;    /* Byte polarity flag (0 => first byte)*/
     int num ;             /* number of fields read */
     int ierr=0;             /* error return code */
/*
*... Read next record from input file
```

```
*/
        ifield =0 ;
        isum =0;
        memact = TRUE;
        memfld = 0 ;
        segact = FALSE;
        ipc = 0;
        isegmt = 0;
        idata = FALSE;



    while ((num=fread(binbuf,1,80,binfp)) != EOF && (num != 0))
    {
            for (i=0; i < num; i++)
            {

/*
*... Get next character from current buffer
*/
            ichar = binbuf[i] ;
            if (nxtbyt)     /* Process second half of word */
            {
                    iwd2 = ichar & 0377;
                    iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;       /*form full word */
                    if ((iwdx & 010000) != 0)           /* origin specification */
                    {
                    ipc = iwdx & 07777 ;
                    if (segact)                 /* relocate to EPROM */
                    {

                            ipc =ipc +Sgebas[isegmt] -Sgrbas[isegmt];
                    }
                    isum = (iwd1+iwd2+isum ) & 07777 ;      /*compute check sum */
            }
            else
                    idata = TRUE ;
            nxtbyt = FALSE ;
}
else
            {
            switch ( (ichar & 0300)){
            case (0200):
                    if (! ibeg)  /* end of input file, check checksum */
                    {
                            iwdx = ((iwd1 & 0377) << 6 ) + iwd2 ;
                            if (isum != iwdx)
                            {
                                    printf ("\n  checksum error: internal=  %o actual = %o \n",
                                    isum , iwdx);

                            }
                     } /* end if */
            break;
```

```
                        case (0300):
                                process_new_segn(Mem);
                                idata = FALSE ;
                                ibeg = FALSE ;
                                nxtbyt = FALSE ;
                                break;
                default:
                                if (idata )
                                { /* pending data, flush it out  */
                                        idata = FALSE ;
                                        if (memact) (*Mem)[memfld][ ipc]= iwdx ;
                                        /* (fp, "iwd1 %o iwd2 %o memfld %d ipc %d Mem[memfld][ipc] %o
\n",
                                        iwd1,iwd2,memfld, ipc, Mem[memfld][ipc]); */
                                        ipc = (++ipc) & 07777;
                                        isum = (iwd1+iwd2+isum) & 07777 ;
                                }
                                iwd1 = ichar & 0377;          /* Get first half data */
                                ibeg = FALSE ;          /* Make sure leader switch off */
                                nxtbyt = TRUE;          /* Set for second half */
                                break;
                        } /*end switch */
        }/* end if */
        } /* end for */
        }   /* end while */
        return (ifield);
} /* end load_mem */
void process_new_segn (int (**Mem)[010000])

/*
c... A segment specification has been encountered. Actions at this point
c... are as follows:
c
c     1. Find the segment attributes in the segment tables. Incomplete
c        data in the tables causes the load to be aborted.
c
c     2. Validate the memory field for storing the values. This field
c        must lie in the range between RAMBAS and MAXFLD. If this is
c        not the case, a warning message is printed, and the segment
c        is not loaded.
c
c     3. Set up the memory pointers for the EPROM field and the EPROM
c        base address for storing the subsequent data values.
c
*/
{   int temp;

     if (idata)
     {

             if (memact ) (*Mem)[memfld][ipc] = iwdx;
      /*     fprintf(fp, "iwd1 %o iwd2 %o memfld %d ipc %d Mem[memfld][ipc] %o \n",
                        iwd1,iwd2,memfld, ipc, Mem[memfld][ipc]); */
             ipc = (++ipc) & 07777;
```

```
            isum = (iwd1+ iwd2 +isum) & 07777;
        }
        isegmt = ichar & 077;       /* isolate field ID & set to segment*/
        if ( (Sgebas[isegmt] < 0) ||
         (Sgefld[isegmt] < 0) ||
         (Sgrbas[isegmt] < 0) ||
         (Sgefld[isegmt] < 0) ||
         (Sgleng[isegmt] < 0))
        {
            printf ( "  \n incomplete specification of segment %d \n  ", isegmt);
            segact = FALSE;
            memact = FALSE;
        }
        else
        {
            memfld = Sgefld[isegmt] ;
            memact = TRUE;
            segact = TRUE;
            temp = (int) pow ((double)2, (double)memfld);
            ifield = ifield | temp;
        }

}
```