DATA ENCAPSULATION USING FORTRAN-77 MODULES–A FIRST STEP TOWARD OBJECT-ORIENTED PROGRAMMING IN WATER RESOURCES

by David B. Thompson, Lewis L. DeLong, and Janice M. Fulford

U.S. GEOLOGICAL SURVEY

Water-Resources Investigations Report 92-4123

Bay St. Louis, Mississippi
1992

Department of the Interior

Manuel Lujan, Jr., Secretary

U.S. GEOLOGICAL SURVEY

DALLAS L. PECK, Director

For additional information write to:
Project Chief, Deterministic Models Project
U.S. Geological Survey
Building 2101
Stennis Space Center, MS 39426

Copies of this report can be purchased from:
U.S. Geological Survey
Books and Open-File Reports Section
P.O. Box 25425
Denver, CO 80225

CONTENTS

Page

Abstract 1

Introduction 1

Purpose and Scope 2

Example Programs 2

The Procedural Approach 3

Data Encapsulation and the Module Approach 6

Advantages of the Module Approach 12

Programming Considerations 13

References 14

Appendix A–Source Code for the Procedural Program 15

Appendix B–Source Code for the Module Program 30

Appendix C–Source Code for the Rectangular Channel Properties Module 36

Appendix D–Source Code for the Trapezoidal Channel Properties Module 50

Appendix E–Control Data for Example Programs 65

Appendix F–Rectangular Channel Geometry Data 66

Appendix G–Trapezoidal Channel Geometry Data 67

ILLUSTRATIONS

Figure 1.–Outline structure of procedural program (Norml) 4

Figure 2.–Flow of control and data through the procedural program (Norml) 7

Figure 3.–Flow of control and data through the module program (Norm2). 9

Figure 4.–Outline of the module program (Norm2). 11

FACTORS FOR CONVERTING INCH-POUND UNITS TO INTERNATIONAL SYSTEM (SI) UNITS

Multiply inch-pound units	By	To obtain SI units
foot (ft)	0.3048	meter (m)
mile (mi)	1.609	kilometer (km)
cubic foot per second (ft^3/s)	0.02832	cubic meter per second (m^3/s)

DATA ENCAPSULATION USING FORTRAN-77 MODULES–A FIRST STEP TOWARD OBJECT-ORIENTED PROGRAMMING IN WATER RESOURCES

by David B. Thompson, Lewis L. DeLong, and Janice M. Fulford

Abstract

Programming is a costly aspect of numerical modeling. Recognizing the need to increase reusability and maintainability of source codes, programs developed using FORTRAN-77 often conform to a method of program design called top-down structured programming, or procedural programming. This method emphasizes the relation between procedures necessary to solve a particular programming problem. An alternative approach is to consider the relation between data and manipulation of data fundamental to solution of a programming problem. The programming construct arising from this perspective is termed a module. In this report, the module programming construct is defined. Through figures and code examples, it is shown how FORTRAN-77 modules are used to encapsulate fundamental data from other parts of the program. FORTRAN-77 modules provide a level of program structure in addition to that available using procedural practices alone. Construction and use of FORTRAN-77 modules improves program clarity, reduces argument passing, and encapsulates fundamental data at the level where they are manipulated.

Introduction

FORTRAN-77 (American National Standards Institute, 1978) is a procedural programming language commonly used for numerical solution of mathematical equations describing physical phenomena. The increase in number and sophistication of computer programs written to simulate physical phenomena (numerical models) has progressed to the point where programming concerns, in addition to concerns related to development and solution of governing equations, are of significant importance.

Software development, support, and maintenance are costly aspects of numerical modeling. Therefore, programs developed using FORTRAN-77 often conform to a design method called top-down structured programming. This method (Yourdon and Constantine, 1979) is applied by reducing each major task of a programming problem into successively smaller sub-tasks until each sub-task can be solved by a relatively short piece of code, usually one procedure. Because this approach to programming is task-oriented, the top-down structured method is often referred to as the procedural approach, the procedural paradigm, or simply procedural programming. The general rule governing program development using procedural programming may be simply stated: determine which procedures are required and implement the procedures as efficiently as you can (Stroustrup, 1989). In the FORTRAN-77 language, a procedure is a called a subprogram, which can be either a subroutine or a function.

Procedural programming results from a particular view of the structure of a computer program–that of the interaction of algorithms required to solve a given problem, usually with little regard for data used by the program. However, programming problems can be viewed from other perspectives that may require a shift of paradigm. One such paradigm is based on the data used by a program and manipulation of data fundamental to solution of a programming problem. This paradigm will be referred to as the data-encapsulation paradigm, or the module paradigm. Divergent meanings are attributed to the term module. In this report, module refers to the composition of data and procedures to manipulate these data (Stroustrup, 1989).

Purpose and Scope

The purpose of this report is to present a comparison of programming paradigms through the use of two example programs. The first example program is developed using procedural programming principles and the second is developed using data encapsulation in FORTRAN-77 modules. Guidelines for developing modules in the FORTRAN-77 language are presented. FORTRAN-77 modules are shown to provide advantages, including an additional level of program structure, not obtainable using procedural practices alone.

Example Programs

Analysis of two programs which perform the same task will demonstrate differences between procedural and modular programs. These programs are fully operational and FORTRAN-77 source code for each example is presented in an appendix. Diagrams of flow of program control and flow of data between procedures are presented to illustrate differences between the methods.

The example problem used to analyze the two programs is to determine normal depth of flow at specific locations within a network of interconnected channels. Normal depth is defined to be the depth of flow that occurs when the water surface is parallel to both the total energy line and the bottom of the channel for a specific volumetric discharge (Chow 1959). The Manning equation is used to relate normal depth to discharge. The Manning equation is:

$$Q = K(H)S^{1/2} \tag{1}$$

where:

$$\begin{aligned}
Q &= \text{volumetric discharge }, (L^3/T) \\
H &= \text{depth of flow }, (L) \\
S &= \text{bottom slope}, (\textit{dimensionless}) \\
K(H) &= \text{channel conveyance}, (L^3/T) \\
&= \frac{C}{n} \cdot \frac{A(H)^{5/3}}{P(H)^{2/3}} \\
C &= 1.486 (\text{for English units}) \\
&= 1.0 (\text{for metric units}) \\
A &= \text{cross-sectional area of flow}, (L^2) \\
P &= \text{wetted perimeter}, (L) \\
n &= \text{Manning flow resistance coefficient}, (L^{1/6})
\end{aligned}$$

Input data required to solve the problem include the locations and volumetric discharges for which normal depths are requested and a description of geometric and hydraulic properties of each channel in the network. For both programs, all channels are assumed to be rectangular in cross section, and channel width is assumed to vary linearly with channel distance. Therefore, each channel in the network is completely described by eight data: a channel number, flow-resistance coefficient, and for each end of the channel a downstream-distance coordinate, channel width, and elevation of the channel bottom. A number is assigned to each channel (channel number) for use as an index in storing channel properties data.

The Manning equation is non-linear in depth of flow. Both programs solve it using a Newton-Raphson iterative scheme. Given discharge, bottom slope, and cross-sectional properties of the channel, an iterative solution of the Manning equations is:

$$H^{+} = H^{*} + \frac{K - K(H^{*})}{\frac{dK(H^{*})}{dH}} \tag{2}$$

where:

$$\begin{aligned} H^{+} &= \text{new estimate of depth,} \\ H^{*} &= \text{current estimate of depth,} \\ K(H) &= \text{conveyance of the channel section,} \\ &= \left|\frac{Q}{S^{1/2}}\right|, \\ K(H^{*}) &= \text{conveyance for current estimate of depth,} \\ \frac{dK(H^{*})}{dH} &= \text{derivative of conveyance with respect to depth for the current estimate of depth.} \end{aligned}$$

Iterations terminate when the absolute change in depth between iterations is less than some fixed tolerance or when a fixed number of iterations are completed.

The Procedural Approach

From the procedural perspective, the problem comprises five sub-tasks (associated function names, Appendix A, are contained in parenthesis):

1. Read locations and discharges for which normal depths are desired (ReadControlData).
2. Read data describing the geometry of the channel network (ReadHydraulicProperties).
3. Compute the bottom slope required for normal-depth computations from channel properties (BtmSlope).
4. Compute normal depth (Norm), by applying the iterative solution scheme developed above.
5. Report the results (Report).

```
Program: Norm1
 +-P.Norm1
    |-S.ReadControlData (ControlFileUnit, ControlFileName, MaxLocations,
    |  |                  NumberOfChannels, NumberOfLocations, Branch,
    |  |                  XLocation, Q)
    |  +-F.OpenOldFlatFile (NUA, CFileA)
    |-S.ReadHydraulicProperties (ControlFileUnit, NumberOfChannels,
    |  |                         ControlFileName, MaxChannels, X1, T1,
    |  |                         Btm1, X2, T2, Btn2, ManningN)
    |  +-F.OpenOldFlatFile (NUA, CFileA)
    |-F.BtmSlope (X1, Btm1, X2, Btm2)
    |-F.Norm (X, H, Q, S, X1, T1, X2, T2, ManningN)
    |  |-F.Conveyance (X, H, X1, T1, X2, T2, ManningN)
    |  |  |-F.CXArea (X, H, X1, T1, X2, T2)
    |  |  |  +-F.ChannelWidth (X, H, X1, T1, X2, T2)
    |  |  +-F.WetPerimeter (X, H, X1, T1, X2, T2)
    |  |     +-F.ChannelWidth (X, H, X1, T1, X2, T2)
    |  +-F.DConveyance (X, H, X1, T1, X2, T2, ManningN)
    |     |-F.ChannelWidth (X, H, X1, T1, X2, T2)
    |     |-F.CXArea (X, H, X1, T1, X2, T2)
    |     |  +-F.ChannelWidth (X, H, X1, T1, X2, T2)
    |     +-F.DWetPerimeter ()
    +-S.Report (Branch, XLocation, Q, Slope, H)
```

Figure 1: Outline structure of procedural program (Norm1).

The structure of the resulting program and the argument lists for each procedure is shown in outline form in figure 1. A complete source listing, including definitions of variables and subprograms, is given as Appendix A.

The sub-tasks of the solution algorithm are executed by the main program through calls to the subprograms listed above. The problem is solved by executing a call to BtmSlope and a call to Norm for each branch number,

```
      ...
      DO 100 I = 1, NumberOfLocations
         ...
         Brn   = Branch(I)
         Slope = ABS(BtmSlope(X1(Brn),Btm1(Brn),X2(Brn),Btm2(Brn)))
         H     = Norm (
     #                XLocation(I), H, Q(I), Slope,
     #                X1(Brn), T1(Brn),
     #                X2(Brn), T2(Brn),
     #                ManningN(Brn)
     #               )
  100 CONTINUE
      ...
```

XLocation(I) is the downstream distance for the current branch number, H is the depth to be computed, Q(I) is the discharge at the current location, the pairs X1(Brn), T1(Brn) and X2(Brn), T2(Brn) are the respective downstream distances and bottom widths associated with upstream and downstream channel ends, and ManningN is the flow-resistance coefficient. These data are not used in the main program, but are passed to subprograms.

The Newton-Raphson solution algorithm is implemented in subprogram Norm. The sub-tasks executed by procedure Norm are:

1. Compute actual conveyance of the channel as discharge divided by the square root of the bottom slope.
2. Make an initial estimate of the depth at the current location.
3. Compute conveyance from channel properties and depth (Conveyance).
4. Compute the partial derivative of conveyance with respect to depth (dConveyance).
5. Compute a new estimate of depth of flow.
6. Check for closure and iterate again if not closed, subject to a maximum of ten iterations.

Procedures Conveyance and dConveyance require functions to compute cross-sectional area of flow (CxArea), channel width (ChannelWidth), wetted perimeter (WetPerimeter), and the partial derivative of wetted perimeter with respect to depth (dWetPerimeter). In each of these procedures, fundamental data are passed down from the level above. The structure of the program, as outlined on figure 1, is such that data are passed through each call to lower levels of the program, regardless of which routine actually uses those data. This is in accord with good procedural programming practice.

Connections between the set of procedures form a tree structure, as shown on figure 2. Flow of program control is indicated by dashed lines with arrows pointing to called functions. Flow of channel data is indicated by solid bold lines and parallels the flow of program control.

The parallel flow of control and channel data occurs because channel data are defined at the highest levels of the program. Channel data are passed from the input procedure (ReadHydraulicProperties) up the call tree to the main program, then down the call tree to procedures where channel data are actually used to compute specific hydraulic properties. Therefore, channel data are viewed as having global scope, even though they are used only in a few procedures.

A FORTRAN-77 COMMON statement could eliminate much argument passing. This commonly applied solution only exacerbates the problem by hiding the fact that channel data are global in scope. COMMON blocks tend to contain unrelated data and their use often results in code which is program specific and not easily used in other programs or by other programmers. Furthermore, COMMON blocks frequently conceal the flow of data between procedures. Therefore, top-down structured programming methods fundamentally encourage passing data through procedure calls. As a result, use of COMMON blocks is discouraged in procedural programming.

Data Encapsulation and the Module Approach

There are two ways to access a datum. One is to retrieve or modify its value directly through an assignment statement. Another method is to surround the datum with a function or subroutine to prevent direct access to the datum and protect it from inadvertent modification. The restriction of access to a datum is called encapsulation. Modifications to such data are accomplished through function calls designed explicitly to modify encapsulated data. Retrieval is also accomplished through invocation of a function designed to deliver encapsulated data to a requesting program. FORTRAN-77 modules are based on the concept of data encapsulation.

An alternative to procedural programming, referred to as FORTRAN-77 modules, comprises specific data and procedures necessary to manipulate those data contained in one program unit. It is an extension to procedural programming. In the FORTRAN-77 module approach, it is recognized that fundamental data are actually manipulated only by specific procedures, and that the composition of data and code form a complete entity. Data are separated into groups based on common characteristics and use. Module design is based on encapsulating fundamental data with code to interact with the rest of the program, and hiding fundamental data by restricting their scope to a few procedures and excluding direct access to such data by external procedures. This philosophy results in storage and manipulation of data at the lowest level. Results are passed up program branches to procedures which actually use the results. The general rule governing program development using the module paradigm may be simply stated: determine which modules are required and partition the program so that all data reside in modules (Stroustrup, 1989).

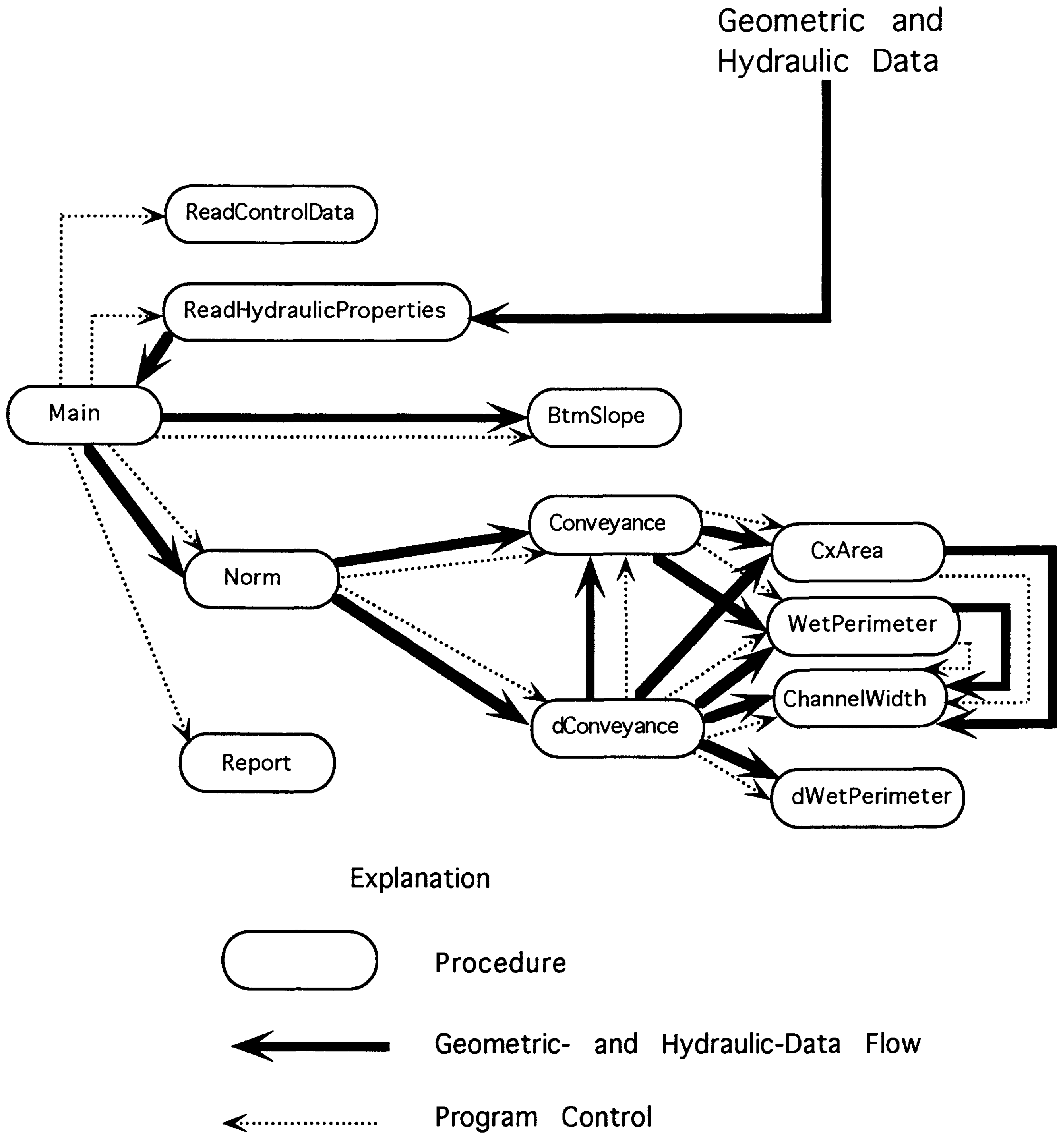


Figure 2: Flow of control and data through the procedural program (Norm1).

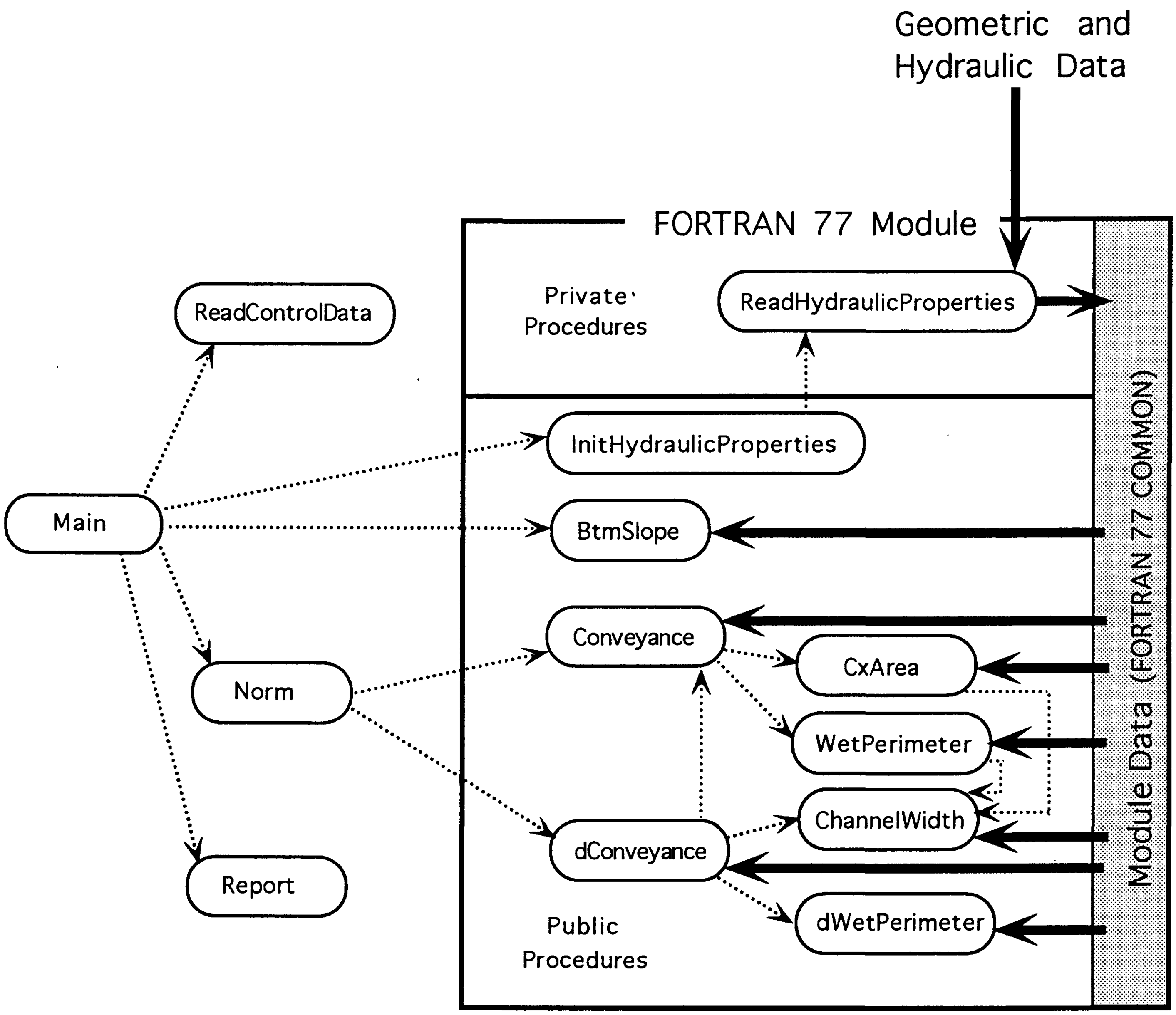
The module paradigm is applied by examining the fundamental data to be manipulated. Data related either by association with a physical process or computational need are segregated into groups. Procedures that directly manipulate data in the group are identified and combined with related data in respective modules. Subprograms are assigned names reflecting the data they return or the service they provide. For convenience, each FORTRAN-77 module is stored in a separate file, a parallel to use of modules in the programming language C. In C, data can be given module scope by defining them at the top of the file (or in a header). No such provision exists in FORTRAN-77. However, this feature can be emulated by constructing one or more COMMON blocks. Module data are associated in these COMMON blocks and only shared among module procedures. These COMMON blocks are restricted to module procedures and require use of a SAVE statement (under the FORTRAN-77 standard) to preserve variable contents between calls. The FORTRAN-77 compiler does not enforce this restriction; it is based only on programmer discipline.

In the example problem only one data group is used, that comprised of data physically describing the channels. These channel data provide the basis for a channel-properties module in this report. The module concept applies to other types of data, including those describing time-varying boundary conditions, initial conditions, channel network schematics, and simulation results. Modules may then be developed, tested, and stored in libraries independent of calling programs, and linked with calling programs as required.

The structure resulting from the module paradigm applied to the example problem is shown on figure 3. (See Appendix B for main program source code and Appendix C for module source code.) In the example, procedures in the channel properties module manipulate fundamental channel data. Output from these procedures provide channel data and derived quantities to extramodule procedures (as do BtmSlope, Conveyance, and dConveyance), or to intramodule subprograms (as do CxArea, ChannelWidth, WetPerimeter, and dWetPerimeter). Prior to invocation of module subprograms, module data must be initialized. This is completed by invoking a procedure (InitHydraulicProperties) which opens a table of channel properties and calls another procedure (ReadHydraulicPropertries) to read fundamental channel data from a file.

Module procedures appear in either a public or private partition, as shown on figure 3. Public procedures are so designated because they must be available for both intramodule and extramodule calls. In fact, public procedures provide both a precise functional definition of the module and a calling interface through which all access to module data is restricted. Therefore, every complete replacement module must supply the same functionality, that is, the same list of public functions and sub-procedures coupled with arguments passed to these procedures.

In contrast, private procedures are so designated because they are used only within the context of the module in which they are defined. In the example, only the procedure for reading channel data (ReadHydraulicProperties) is found in this partition. A more complete implementation of a channel-properties module would include additional private procedures for interpolation, error checking, input and output, and other utilities specific to the module. Restricting use of private procedures to intramodule calls prevents dependence of calling programs on module-specific implementations.



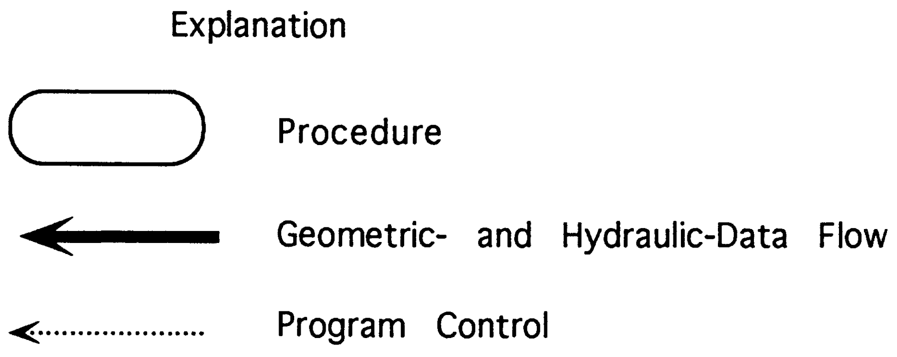


Figure 3: Flow of control and data through the module program (Norm2).

The calling program is constructed in a procedural fashion, with the exception that the scope of program variables is now strictly addressed. The example problem is broken into tasks; initializing the channel-properties module, reading program-control data, computing normal depth, and reporting results. The program obtains channel information by invoking public module procedures, passing to those procedures only data specifically required to identify the location for which information is requested, and then continues to compute the solution of the problem.

Returning to the example problem, an outline of the module version of the program is shown in figure 4. This version of the program reads control data through a call to ReadControlData, then initializes the channel-properties module through a call to the procedure InitHydraulicProperties. If the initialization of the channel-properties module is successful, then a loop similar to that of the procedural program is entered,

```
      ...
      DO 100 I = 1, NumberOfLocations
        H = Norm (Branch(I), XLocation(I), H, Q(I), Slope)
  100 CONTINUE
      ...
```

This is distinctly different from the procedural version, which passed additional data into subprogram Norm so that lower-level procedures could compute channel properties necessary to solve the problem.

It is apparent that only those data within the scope of the calling program are passed as arguments. This is in strict contrast to the procedural approach (compare fig. 4 with fig. 1) which requires that calling programs supply all necessary channel data. The procedural approach only separates computational procedures and ignores the scope of data manipulated by the program. In contrast, the module paradigm isolates fundamental data in the procedures in which they are used. This eliminates:

1. passing channel data through many subprogram levels (shortens data paths),
2. simplifies the invocation of lower-level procedures by reducing the number of arguments passed,
3. moves procedures dependent on channel data into modules with independent data structures, and
4. provides a well-defined interface for communication of channel data to the program.

The module paradigm results in calling programs not directly dependent on the form of fundamental data, and allows access to fundamental data only through subprogram invocation. This practice is referred to as data encapsulation. Modifications required to effect a new implementation, such as allowing channel sections of general geometry, would be restricted to the channel-properties module and would not affect the interface between the module and calling programs. This effectively

```
Program Norm2
 +-P.Norm2
    |-S.ReadControlData (ControlFileUnit, ControlFileName, MaxLocations,
    |  |                 NumberOfChannels, NumberOfLocations, Branch,
    |  |                 XLocation, Q)
    |  +-F.OpenOldFlatFile (FUnit, FileName)
    |-F.InitHydraulicProperties (NumberOfChannels)
    |  | C/PrpDat/
    |  | C/PropUnit/
    |  |-F.OpenOldFlatFile (FUnit, FileName)
    |  +-F.ReadHydraulicProperties (ChannelNumber)
    |     | C/PrpDat/
    |     + C/PropUnit/
    |-F.BtmSlope (Branch, X)
    |  + C/PrpDat/
    |-F.Norm (Branch, X, H, Q, S)
    |  |-F.Conveyance (Branch, X, H)
    |  |  |-F.CXArea (Branch, X, H)
    |  |  |  +-F.ChannelWidth (Branch, X, H)
    |  |  |     + C/PrpDat/
    |  |  |-F.WetPerimeter (Branch, X, H)
    |  |  |  +-F.ChannelWidth (Branch, X, H)
    |  |  |     + C/PrpDat/
    |  |  +-F.ManningN (Branch, X, H)
    |  |     + C/PrpDat/
    |  +-F.dConveyance (Branch, X, H)
    |     |-F.ChannelWidth (Branch, X, H)
    |     |  + C/PrpDat/
    |     |-F.CXArea (Branch, X, H)
    |     |  +-F.ChannelWidth (Branch, X, H)
    |     |     + C/PrpDat/
    |     |-F.dWetPerimeter (Branch, X, H)
    |     +-F.dManningN (Branch, X, H)
    +-S.Report (Branch, XLocation, Q, Slope, H)
```

Figure 4: Outline of the module program (Norm2).

decouples the fundamental solution algorithm in the calling program from the form of underlying channel data.

In fact, a second channel-properties module was developed for this report. The source code for channel-properties for channels with a trapezoidal cross section is given in Appendix D. This code can be linked to the client program (Norm2) without changes to any source code. The data file for the module differs slightly from that for the rectangular section module (a side slope is required for a trapezoidal section), but all public procedures are invoked with the same names and with the same data passed through the invocation.

Advantages of the Module Approach

The procedural approach is a significant improvement over earlier methods used to develop FORTRAN programs. The impact of procedural programming on the development of quality programs is unquestioned. FORTRAN-77 modules are a replacement for top-down structured approach (procedural programming), but not for structured programming.

Computer models concerned with simulation of physical phenomena share many distinct types of data related to the fundamental physics of the prototype. Modules can be constructed according to the form of fundamental data and can be linked without modification to many different calling programs. This capability results in reduced programming effort and maintenance. Alternate modules can be constructed and linked with a calling program to solve problems with different forms of fundamental data. For example, data describing geometric and hydraulic properties of channels used in the example programs are used in programs concerned with computation of steady-state water-surface profiles, computation of unsteady flow fields, and transport of suspended and dissolved substances. All of these models could use the same channel-properties module (or one from a suite of channel-properties modules), resulting in less model-specific code to be maintained by the model developer. With development of modules for other groups of data, including those describing time-varying boundary conditions, initial conditions, network schematics, and results of simulations, modules could be shared by many programs.

Fundamental data rarely are used directly by models. Models most frequently use quantities derived from manipulation of fundamental data. In the example, three quantities were required by the calling program, bottom slope, conveyance, and the partial derivative of conveyance with respect to depth of flow. More complex models require additional quantities, such as channel-bottom elevation, and at specific depths of flow, cross-sectional area, top width, or distance from the centroid of the wetted cross section to the water surface. A module similar to the example channel-properties module could supply information to any program requiring such quantities.

As demonstrated by example, programs based on FORTRAN-77 modules are less dependent on fundamental data than corresponding procedural programs. Such programs are less likely to require modification as a result of changes in the form of fundamental data. Programs developed using FORTRAN-77 modules are more likely to perform with a larger variety of fundamental data than their procedural counterparts, limited only by availability of appropriate modules. Use of modules results in sharing of source code resources and encourages development of codes independent of the form of fundamental data.

In the procedural example, passing channel data through the program results in long data paths. That is, channel data are passed from the highest-level procedures (where the data are not used) to the lowest-level procedures (where actual computations are executed). Therefore, all procedures in any branch of the call tree which pass channel data are dependent on the form of the data and require modification each time the form of the channel data changes, regardless of any other need for modification. Consequently, all procedures passing or using any part of the channel data would require modification. This creates unnecessary work and is a source of error during software development and maintenance. Furthermore, because changes to codes programmed this way are inherently difficult, procedural programming can result in programmer resistance to enhancement. Use of FORTRAN-77 modules circumvents these problems.

In the module example, a COMMON block is used to share data in the module. Its use is restricted to module routines and the data it contains are not accessed by any procedures outside the module. This use of COMMON blocks may evoke dismay in procedural method programmers. However, the restriction of COMMON usage to module procedures and using data strictly related by module function differs substantially from the unstructured aggregation of unrelated data in COMMON prevalent in old FORTRAN programs.

The additional structure provided through use of FORTRAN-77 modules is seen by comparison of figure 2 and figure 3. The channel-properties module forms a larger entity than a single subprogram. The calling code can be viewed as a separate box, served by a channel-properties module and a utility library (which contains the procedure OpenOldFlatFile). Advantages of this additional structure are more apparent when used in applications comprised of tens or hundreds of procedures. The additional structure allows programs to be viewed at a simpler level in terms of a few modules, instead of a labyrinth of interconnected individual procedures. Separation of code and data into modules based on the data manipulated provides a natural isolation of errors and division of programming tasks, including designing, coding, and testing. The cost for implementing this strategy is not great. The procedural code has 13 procedures, whereas the modular code has 16 procedures. The benefit of increased structure, isolation of errors, and ability to reuse code resources offsets this small increase in number of procedures present in a program.

Programming Considerations

The rules for creating and using modules in FORTRAN-77 as presented in this report are entirely voluntary. Neither direct support nor enforcement of these rules for modules are provided by FORTRAN-77. For example, programmers may call private functions or access private module data directly without warning from the compiler. Therefore, successful application of FORTRAN-77 modules is based entirely on programmer discipline.

However, the FORTRAN-90 standard (Metcalf and Reid, 1990) contains direct support and enforcement of FORTRAN modules through direct implementation of a MODULE construct and addition of PRIVATE and PUBLIC attributes for both data and procedures. Unfortunately, no vendor has yet supplied a FORTRAN-90 compliant compiler, and no complete implementation is expected for about two years. Use of FORTRAN-77 modules represents a transition from strict (or not-so-strict) procedural programming to modular programming (modular in the sense of this report) with language and compiler support for modules.

Other computer languages, such as C++ (Stroustrup, 1987), have the capability to support programming paradigms similar to that presented, as well as other potentially powerful concepts neither allowed nor supported by the FORTRAN-77 language. In particular, C++ is a language for object-oriented programming, a paradigm which may prove as useful in solution of water resources problems as in programs concerned with computer graphics. Because data encapsulation is elemental to object-oriented programming, it is a first step toward object-oriented programming.

References

American National Standards Institute, 1978, *American National Standard Programming Language FORTRAN.* American National Standards Institute, New York, NY.

DeLong, L.L., Thompson, D.B., and Fulford, J.M., 1992, "Data Encapsulation Using Fortran 77 Modules," *Fortran Forum* Volume 11, Number 3, ACM, New York.

Metcalf, M. and Reid, J., 1990. *FORTRAN 90 Explained.* Oxford Science Publications, New York.

Stroustrup, B., 1987. *The C++ Programming Language.* Addison-Wesley, Reading MA.

Stroustrup, B., 1989. Chapter 4: Object-Oriented Programming, in *UNIX System V AT&T C++ Language System Release 2.0, Selected Readings*, AT&T.

Appendix A–Source Code for the Procedural Program

```
*=========================================================================
      PROGRAM Norm1
*-------------------------------------------------------------------------

*     Purpose:  Compute normal depth at locations within a network
*               of open channels -- procedural approach.

        IMPLICIT NONE

*     Local variables:
        INTEGER       NumberOfLocations,MaxLocations,NumberOfChannels,I
        PARAMETER  ( MaxLocations = 20 )
        INTEGER       ControlFileUnit,PropertiesFileUnit
        INTEGER       Branch(MaxLocations)
        REAL          XLocation(MaxLocations),Q(MaxLocations),Slope,H
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        INTEGER Brn
        REAL X1(MaxChannels),T1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),T2(MaxChannels),Btm2(MaxChannels)
        REAL ManningN(MaxChannels)
        CHARACTER*12 ControlFileName,PropertiesFileName

*     Definitions:
*       MaxChannels    - maximum number of channels.
*       NumberOfLocations    - number of locations for normal depth computations.
*       MaxLocations - maximum allowable number of locations.
*       NumberOfChannels   - number of channels in network.
*       ControlFileUnit  - FORTRAN unit number for file containing locations.
*       PropertiesFileUnit - FORTRAN unit number for file containing properties.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.
*       ManningN     - effective Manning's n for the branch.

*     Functions:
        REAL     Norm,BtmSlope
        EXTERNAL Norm,BtmSlope
```

```
*     Subroutines:

        EXTERNAL ReadControlData, ReadHydraulicProperties, Report

*     Intrinsics:
        REAL   ABS

      DATA  ControlFileUnit  / 12 /
      DATA  ControlFileName  / 'cntrl.dat   ' /
      DATA  PropertiesFileUnit / 11 /
      DATA  PropertiesFileName / 'cx_tbl.dat  ' /

*-----------------------------------------------------------------------

*---- Read locations and flows for which normal depths are required.
      CALL ReadControlData
     I        (ControlFileUnit,ControlFileName,MaxLocations,
     O         NumberOfChannels,NumberOfLocations,Branch,XLocation,Q)

*---- Read cross-sectional properties data.
      CALL ReadHydraulicProperties
     I           (PropertiesFileUnit,NumberOfChannels,
     I            PropertiesFileName,MaxChannels,
     O            X1,T1,Btm1,X2,T2,Btm2,ManningN)

*---- Begin normal-depth computations.
      WRITE(*,*) ' '
      WRITE(*,*)
     # 'Branch  Location Discharge   Slope   Normal depth'

      DO 100 I=1,NumberOfLocations

        H = 1.0
        Brn = Branch(I)
        Slope = ABS( BtmSlope( X1(Brn),Btm1(Brn),X2(Brn),Btm2(Brn) ) )
        H     = Norm( XLocation(I),H,Q(I),Slope,
     #                X1(Brn),T1(Brn),
     #                X2(Brn),T2(Brn),ManningN(Brn)        )

        CALL Report(Branch(I), XLocation(I), Q(I), Slope, H)

 100  CONTINUE

      STOP
      END


*==========================================================================
        SUBROUTINE Report
     I                  (Branch, XLocation, Q, Slope, H)
*--------------------------------------------------------------------------

*     Purpose:   Output results of computation.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL XLocation, Q, Slope, H

*     Definitions:
*       Branch    - branch number.
*       XLocation - downstream distance.
*       H         - depth of flow.
*       Q         - volumetric discharge.
*       Slope     - slope.

*--------------------------------------------------------------------------

        WRITE(*,'(I4,2X,2F10.2,F8.4,F10.2)')
     #         Branch, XLocation, Q, Slope, H

      RETURN
      END


*==========================================================================
        REAL FUNCTION Norm (X,H,Q,S,
     #                      X1,T1,X2,T2,ManningN)
*--------------------------------------------------------------------------

*     Purpose:   Estimate normal depth in the Branch branch,
*                at X downstream distance, for discharge Q,
*                and slope S.

        IMPLICIT NONE

*     Arguments:
        REAL X,H,Q,S
        REAL X1,T1
        REAL X2,T2
        REAL ManningN

*     Definitions:
*       X       - downstream distance.
*       H       - depth of flow.
*       Q       - volumetric discharge.
*       S       - slope.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       ManningN     - effective Manning's n for the branch.

*     Local variables:
        INTEGER I
        REAL    NewK,NewH,dH

*     Functions:
        REAL     Conveyance,dConveyance
        EXTERNAL Conveyance,dConveyance

*     Intrinsics:
        REAL      ABS,SQRT
        INTRINSIC ABS,SQRT

*----------------------------------------------------------------------------------

      NewK = Q/SQRT(ABS(S))
      NewH   = H

      DO 100 I=1,10

        dH = (NewK-Conveyance(X,NewH,X1,T1,X2,T2,ManningN))
     #          / dConveyance(X,NewH,X1,T1,X2,T2,ManningN)

        NewH   = NewH+dH

        IF(ABS(dH/NewH).LT.0.001) GO TO 102

 100  CONTINUE
```

```
      WRITE(*,*) '***Warning(Norm) did not close in 10 iterations...'
      WRITE(*,*) ' X = ',X,' H = ',H
      WRITE(*,*) 'Q = ',Q,'  Slope = ',S

  102 CONTINUE

      Norm = NewH

      RETURN
      END


*================================================================================
        SUBROUTINE ReadControlData
     I                 (ControlFileUnit,ControlFileName,MaxLocations,
     O                  NumberOfChannels,NumberOfLocations,
     O                  Branch,XLocation,Q)
*--------------------------------------------------------------------------------

*     Purpose:  Read locations and discharges.

        IMPLICIT NONE

*     Arguments:
        INTEGER ControlFileUnit,MaxLocations,NumberOfChannels
        INTEGER NumberOfLocations,Branch(MaxLocations)
        REAL    XLocation(MaxLocations),Q(MaxLocations)
        CHARACTER*12 ControlFileName

*     Definitions:
*       ControlFileUnit  - FORTRAN unit number for input file.
*       ControlFileName  - corresponding file name.
*       MaxLocations - maximum number of locations.
*       NumberOfChannels   - number of channels.
*       NumberOfLocations    - current number of locations.
*       Branch  - array of channel numbers.
*       XLocation    - corresponding array of downstream locations.
*       Q       - corresponding array of discharges.

*     Local variables:
        INTEGER I

*     Functions:
      LOGICAL  OpenOldFlatFile
      EXTERNAL OpenOldFlatFile

*-----------------------------------------------------------------------

*------ Open file.
      IF(OpenOldFlatFile(ControlFileUnit,ControlFileName)) THEN
      ELSE
        WRITE(*,*) 'Attempt to open file failed...'
        WRITE(*,'(A12)')  ControlFileName

        STOP

      END IF

*------ Get number of channels in network.
      READ(ControlFileUnit,*) NumberOfChannels

*------ Get locations and discharges.
      DO 100 I=1,MaxLocations
        READ(ControlFileUnit,*,END=102) Branch(I),XLocation(I),Q(I)
        NumberOfLocations = I
        IF(NumberOfLocations.GE.MaxLocations) GO TO 102
 100  CONTINUE
 102  CONTINUE
      RETURN
      END

*=======================================================================
      LOGICAL FUNCTION OpenOldFlatFile(NUA,CFILEA)
*-----------------------------------------------------------------------

*    Purpose:  Open an existing flat file for input.

      IMPLICIT NONE

*    Arguments:
      INTEGER      NUA
      CHARACTER*(*) CFILEA

*    Definitions:
*      NUA    - FORTRAN unit number of file to be opened.
*      CFILEA - Character name of file to be opened.
*     Local variables:
        INTEGER IOS

*-----------------------------------------------------------------------

*---- Open the existing file.
      OPEN (UNIT=NUA, FILE=CFILEA, STATUS='OLD', IOSTAT=IOS)
      IF (IOS .NE. 0) THEN
        OpenOldFlatFile = .FALSE.
      ELSE
        OpenOldFlatFile = .TRUE.
      ENDIF
      RETURN
      END


*=======================================================================
        REAL FUNCTION BtmSlope(X1,Btm1,X2,Btm2)
*-----------------------------------------------------------------------

*     Purpose:   Compute bottom slope.

        IMPLICIT NONE

*     Arguments:
        REAL X1,Btm1
        REAL X2,Btm2

*     Definitions:
*       X1     - downstream reference distance for upstream extent of branch.
*       Btm1   - channel-bottom elevation corresponding to x1.
*       X2     - downstream reference distance for downstream extent of branch.
*       Btm2   - channel-bottom elevation corresponding to x2.
*-----------------------------------------------------------------------

      BtmSlope = ( Btm2 - Btm1 ) / (X2-X1)

      RETURN
      END


*=======================================================================
        REAL FUNCTION Conveyance(X,H,X1,T1,X2,T2,ManningN)
```

```
*-----------------------------------------------------------------------

*     Purpose:   Compute conveyance.

        IMPLICIT NONE

*     Arguments:
        REAL X,H
        REAL X1,T1
        REAL X2,T2,ManningN

*     Definitions:
*       X       - downstream distance.
*       H       - depth of flow.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.
*       ManningN    - effective flow resistance.

*     Local variables:
        REAL R53,R23
        PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)

*     Functions:
        REAL     CxArea,WetPerimeter
        EXTERNAL CxArea,WetPerimeter
*-----------------------------------------------------------------------

      Conveyance = 1.486*(
     #        CxArea(X,H,X1,T1,X2,T2)**R53
     #                  )
     #                   / (
     #     WetPerimeter(X,H,X1,T1,X2,T2)**R23
     #                      ) /ManningN

      RETURN
      END


*=======================================================================
        REAL FUNCTION dConveyance(X,H,X1,T1,X2,T2,ManningN)
```

```
*-----------------------------------------------------------------------

*     Purpose:   Compute d(Conveyance)/dH.

        IMPLICIT NONE

*     Arguments:
        REAL X,H
        REAL X1,T1
        REAL X2,T2
        REAL ManningN

*     Definitions:
*       X       - downstream distance.
*       H       - depth of flow.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.
*       ManningN     - effective flow resistance.

*     Local variables:
        REAL R53,R23
        PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)

*     Functions:
        REAL      ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        REAL      Conveyance
        EXTERNAL ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        EXTERNAL Conveyance
*-----------------------------------------------------------------------

      dConveyance = Conveyance(
     #        X,H,X1,T1,X2,T2,ManningN
     #                       ) * (
     #          R53*ChannelWidth(X,H,X1,T1,X2,T2)
     #             /CxArea(X,H,X1,T1,X2,T2)
     #           -R23*dWetPerimeter()/WetPerimeter(X,H,X1,T1,X2,T2)
     #                            )

      RETURN
      END
```

```
*===============================================================================
        REAL FUNCTION ChannelWidth(X,H,X1,T1,X2,T2)
*-------------------------------------------------------------------------------

*     Purpose:   Compute width of channel.

        IMPLICIT NONE

*     Arguments:
        REAL X,H
        REAL X1,T1
        REAL X2,T2

*     Definitions:
*       X      - downstream distance.
*       H      - depth of flow.
*       X1     - downstream reference distance for upstream extent of branch.
*       T1     - channel width corresponding to x1.
*       Btm1   - channel-bottom elevation corresponding to x1.
*       X2     - downstream reference distance for downstream extent of branch.
*       T2     - channel width corresponding to x2.
*       Btm2   - channel-bottom elevation corresponding to x2.
*-------------------------------------------------------------------------------

      ChannelWidth = ( (X2-X)*T1 + (X-X1)*T2 ) /  (X2-X1)

      RETURN
      END


*===============================================================================
        REAL FUNCTION CxArea(X,H,X1,T1,X2,T2)
*-------------------------------------------------------------------------------

*     Purpose:   Compute cross-sectional area.

        IMPLICIT NONE

*     Arguments:
        REAL X,H
        REAL X1,T1
        REAL X2,T2

*     Definitions:
*       X       - downstream distance.
*       H       - depth of flow.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.

*     Functions:
        REAL     ChannelWidth
        EXTERNAL ChannelWidth
*-----------------------------------------------------------------------------

      CxArea = H*ChannelWidth(X,H,X1,T1,X2,T2)

      RETURN
      END


*=============================================================================
        REAL FUNCTION WetPerimeter(X,H,X1,T1,X2,T2)
*-----------------------------------------------------------------------------

*     Purpose:  Compute wetted perimeter.

        IMPLICIT NONE

*     Arguments:
        REAL X,H
        REAL X1,T1
        REAL X2,T2

*     Definitions:
*       MaxChannels    - maximum number of channels.
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.

*     Functions:
        REAL     ChannelWidth
        EXTERNAL ChannelWidth
*-----------------------------------------------------------------------------

      WetPerimeter = 2.0*H+ChannelWidth(X,H,X1,T1,X2,T2)

      RETURN
      END


*=============================================================================
        REAL FUNCTION dWetPerimeter()
*-----------------------------------------------------------------------------

*     Purpose:   Compute d(wp)/dH.

        IMPLICIT NONE
*-----------------------------------------------------------------------------

      dWetPerimeter = 2.0

      RETURN
      END


*=============================================================================
        SUBROUTINE ReadHydraulicProperties
     I               (ControlFileUnit,NumberOfChannels,
     I                ControlFileName,MaxChannels,
     O                X1,T1,Btm1,X2,T2,Btm2,ManningN)
*-----------------------------------------------------------------------------

*     Purpose:  Read geometric and hydraulic properties.

*               Two data lines are input for each channel.
*               All input is free-field.

*       Line 0:
*         PrintOption (integer)

*       line 1:
*         Channel number (integer)

*       line 2:
*         X1  , T1, Btm1, X2, T2, Btm2, ManningN  (all real),

*       where,

*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.
*       ManningN    - effective Manning's n for the branch.

        IMPLICIT NONE

*     Arguments:
        INTEGER ControlFileUnit,NumberOfChannels
        INTEGER MaxChannels
        REAL X1(MaxChannels),T1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),T2(MaxChannels),Btm2(MaxChannels)
        REAL ManningN(MaxChannels)
        CHARACTER*(*) ControlFileName

*     Definitions:
*       ControlFileUnit  - FORTRAN unit number
*       NumberOfChannels   - number of channels
*       ControlFileName - file name
*       MaxChannels    - maximum number of channels
*       X1      - downstream reference distance for upstream extent of branch.
*       T1      - channel width corresponding to x1.
*       Btm1    - channel-bottom elevation corresponding to x1.
*       X2      - downstream reference distance for downstream extent of branch.
*       T2      - channel width corresponding to x2.
*       Btm2    - channel-bottom elevation corresponding to x2.
*       ManningN    - effective Manning's n for the branch.
*     Local variables:
        INTEGER I,Brn,PrintOption

*     Functions:
        LOGICAL  OpenOldFlatFile
        EXTERNAL OpenOldFlatFile

*     Output formats:
 2001 FORMAT(//'                 Cross-sectional properties...'//)
 2002 FORMAT('                 Distance     Width   Btm_elev'/
     #'Branch',I3/
     #'Upstream.....',3f10.2/
     #'Downstream...',3f10.2/
     #'Effective n..',F6.3//)

*-----------------------------------------------------------------------

      IF(OpenOldFlatFile(ControlFileUnit,ControlFileName)) THEN

        READ(ControlFileUnit,*) PrintOption

        IF(PrintOption.GT.0) THEN
          WRITE(*,*) 'Reading ',ControlFileName
          WRITE(*,*) NumberOfChannels,' channel(s) specified...'
          WRITE(*,2001)
        END IF

        DO 100 I=1,NumberOfChannels
          READ(ControlFileUnit,*,END=200) Brn

          IF(I .NE. Brn ) THEN
            WRITE(*,*) 'Channel sequence error...'
            WRITE(*,*) 'Read number', Brn
            WRITE(*,*) 'Changed to ',I
          END IF

          Brn = I
          READ(ControlFileUnit,*,END=200)
     #      X1(Brn),T1(Brn),Btm1(Brn),
     #      X2(Brn),T2(Brn),Btm2(Brn),ManningN(Brn)
          IF(PrintOption.GT.0) THEN
            WRITE(*,2002) Brn,
     #        X1(Brn),T1(Brn),Btm1(Brn),
     #        X2(Brn),T2(Brn),Btm2(Brn),ManningN(Brn)
```

```
        END IF
100   CONTINUE

      ELSE

        WRITE(*,*) 'Could not open Control file...'
        WRITE(*,'(A12)') ControlFileName

        STOP

      END IF

      RETURN

200   CONTINUE

      WRITE(*,*) 'End of file...reading branch',Brn
      WRITE(*,*) NumberOfChannels,' channels specified...'

      RETURN
      END
```

Appendix B–Source Code for Module Program

```
*=============================================================================
      PROGRAM Norm2
*=============================================================================

*     Purpose:  Compute normal depth at locations within a network
*               of open channels -- an approach using FORTRAN modules.

        IMPLICIT NONE

*     Local variables:
        INTEGER      NumberOfLocations,MaxLocations,NumberOfChannels,I
        PARAMETER  ( MaxLocations = 20 )
        INTEGER      ControlFileUnit
        INTEGER      Branch(MaxLocations)
        REAL         XLocation(MaxLocations),Q(MaxLocations),Slope,H
        CHARACTER*12 ControlFileName

*     Definitions:
*       NumberOfLocations - number of locations for normal depth computations.
*       MaxLocations - maximum allowable number of locations.
*       NumberOfChannels - number of channels in network.
*       ControlFileUnit - FORTRAN unit number for file containing locations.
*       PropertiesFileUnit - FORTRAN unit number for file containing properties.

*     Functions:
        LOGICAL  InitHydraulicProperties
        REAL     Norm,BtmSlope
        EXTERNAL InitHydraulicProperties,Norm,BtmSlope

*     Subroutines:
        EXTERNAL ReadControlData, Report

*     Intrinsics:
        REAL   ABS
        INTRINSIC ABS

      DATA  ControlFileUnit  / 11 /
      DATA  ControlFileName  / 'cntrl.dat    ' /

*-----------------------------------------------------------------------------

*---- Read locations and flows for which normal depths are required.
```

```
      CALL ReadControlData
     I           (ControlFileUnit,ControlFileName,MaxLocations,
     O            NumberOfChannels,NumberOfLocations,Branch,XLocation,Q)

*---- Initialize properties module.
      IF(InitHydraulicProperties( NumberOfChannels ) ) THEN

      ELSE

        WRITE(*,*) 'Initialization of properties module failed...'
        STOP

      END IF

*---- Begin normal-depth computations.

      WRITE(*,*) ' '
      WRITE(*,*)
     # 'Branch  Location Discharge   Slope   Normal depth'

      DO 100 I=1,NumberOfLocations

        H = 1.0
        Slope = ABS( BtmSlope( Branch(I),XLocation(I) ) )
        H     = Norm( Branch(I),XLocation(I),H,Q(I),Slope )
        CALL Report(Branch(I), XLocation(I), Q(I), Slope, H)

 100  CONTINUE

      STOP
      END


*==============================================================================
        SUBROUTINE Report
     I                    (Branch, XLocation, Q, Slope, H)
*------------------------------------------------------------------------------

*     Purpose:  Output results of computation.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL XLocation, Q, Slope, H

*     Definitions:
*       Branch    - branch number.
*       XLocation - downstream distance.
*       H         - depth of flow.
*       Q         - volumetric discharge.
*       Slope     - slope.

*----------------------------------------------------------------------------

        WRITE(*,'(I4,2X,2F10.2,F8.4,F10.2)')
     #           Branch, XLocation, Q, Slope, H

      RETURN
      END


*============================================================================
        REAL FUNCTION Norm(Branch,X,H,Q,S)
*============================================================================

*     Purpose:  Estimate normal depth in the Branch branch,
*               at X downstream distance, for discharge Q,
*               and slope S.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H,Q,S

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.
*       Q       - volumetric discharge.
*       S       - slope.

*     Local variables:
        INTEGER I
        REAL    NewK,NewH,dH
```

```
*     Arguments:
        INTEGER ControlFileUnit,MaxLocations,NumberOfChannels
        INTEGER NumberOfLocations,Branch(MaxLocations)
        REAL    XLocation(MaxLocations),Q(MaxLocations)
        CHARACTER*12 ControlFileName

*     Definitions:
*       ControlFileUnit  - FORTRAN unit number for input file.
*       ControlFileName  - corresponding file name.
*       MaxLocations - maximum number of locations.
*       NumberOfChannels   - number of channels.
*       NumberOfLocations    - current number of locations.
*       Branch  - array of channel numbers.
*       XLocation   - corresponding array of downstream locations.
*       Q      - corresponding array of discharges.

*     Local variables:
        INTEGER I

*     Functions:
        LOGICAL  OpenOldFlatFile
        EXTERNAL OpenOldFlatFile

*-----------------------------------------------------------------------

*------ Open file.
        IF(OpenOldFlatFile(ControlFileUnit,ControlFileName)) THEN
        ELSE
          WRITE(*,*) 'Attempt to open file failed...'
          WRITE(*,'(A12)')  ControlFileName

          STOP

        END IF

*------ Get number of channels in network.
        READ(ControlFileUnit,*) NumberOfChannels

*------ Get locations and discharges.
        DO 100 I=1,MaxLocations
          READ(ControlFileUnit,*,END=102) Branch(I),XLocation(I),Q(I)
          NumberOfLocations = I
          IF(NumberOfLocations.GE.MaxLocations) GO TO 102
```

```
 100    CONTINUE
 102    CONTINUE
        RETURN
        END


*==============================================================================
        LOGICAL FUNCTION OpenOldFlatFile(fUnit,FileName)
*==============================================================================

*     Purpose:  Open an existing flat file for input.

        IMPLICIT NONE

*     Arguments:
        INTEGER       fUnit
        CHARACTER*(*) FileName

*     Definitions:
*       fUnit    - FORTRAN unit number of file to be opened.
*       FileName - Character name of file to be opened.

*     Local variables:
        INTEGER IOS

*------------------------------------------------------------------------------

*---- Open the existing file.
      OPEN (UNIT=fUnit, FILE=FileName, STATUS='OLD', IOSTAT=IOS)

      IF (IOS .NE. 0) THEN
        OpenOldFlatFile = .FALSE.
      ELSE
        OpenOldFlatFile = .TRUE.
      ENDIF

      RETURN
      END
```

Appendix C–Source Code for Rectangular Channel Properties Module

```
*******************************************************************************
*
*     This file is a FORTRAN module for the computation of the geometric
*     and hydraulic properties of multiple, rectangular, non-prismatic
*     channels.   Manning's "n" is assumed constant with depth and downstream
*     distance within a branch.    A branch is represented by  two
*     rectangular cross sections at its extremities.  Width,
*     depth, and wetted perimeter are interpolated linearly.  Area and
*     conveyance are subsequently computed from these interpolated
*     values.  Two data lines are input for each branch.  All input
*     is free-field.
*
*       line 0:
*       PrintOption (integer)
*
*       line 1:
*       ChannelNumber (integer)
*
*       line 2:
*       X1 , Width1, Btm1, X2, Width2, Btm2, Eta  (all real),
*
*          (Repeat above lines 1 and 2 for additional branches.)
*
*       where,
*
*     PrintOption - index for writing to standard output device.
*          [0] input data will not be written to standard output.
*          [1] input data will be echoed to standard output.
*     ChannelNumber - sequence number of channel, monotonically increasing,
*          beginning with 1.
*     X1   - downstream reference distance for upstream extent of branch
*     Width1    - channel width corresponding to X1
*     Btm1 - channel-bottom elevation corresponding to X1
*     X2   - downstream reference distance for downstream extent of branch
*     Width2    - channel width corresponding to X2
*     Btm2 - channel-bottom elevation corresponding to X2
*     Eta  - effective Manning's n for the branch
*
*
*
*     Module note: Only functions or subroutines marked "public" should be
*                  used outside of this module as those marked "private"
*                  may not be supported by future revisions of this module
*                  or replacement modules.  Likewise, no data or common
*                  blocks contained within this module should be accessed
*                  by routines outside of this module accept through the
*                  use of "public" functions or subroutines contained
*                  within this module.
*
*     Non-standard usage: Symbolic names in this module may be represented by
*                         as many as 31 characters in order to provide better
*                         definition directly in the code.  Standard FORTRAN
*                         allows only 6 characters, but this restriction is
*                         generally extended to 32 characters by most compilers.
*
*
*     Public functions:
*
*     LOGICAL FUNCTION InitHydraulicProperties(NumberOfChannels)
*            - begin initialization of module.
*
*     REAL FUNCTION BtmSlope(Branch,X)
*            - returns channel-bottom slope.
*
*     REAL FUNCTION ChannelWidth(Branch,X,H)
*            - returns channel width.
*
*     REAL FUNCTION CxArea(Branch,X,H)
*            - returns cross-sectional area.
*
*     REAL FUNCTION Conveyance(Branch,X,H)
*            - returns sinuosity-weighted conveyance.
*
*     REAL FUNCTION dConveyance(Branch,X,H)
*            - returns d(Conveyance)/dH.
*
*     REAL FUNCTION ManningN(Branch,X,H)
*            - returns Mannings n.
*
*     REAL FUNCTION dManningN(Branch,X,H)
*            - returns d(Mannings n)/dH.
*
*     REAL FUNCTION WetPerimeter(Branch,X,H)
*            - returns wetted perimeter.
*
*     REAL FUNCTION dWetPerimeter(Branch,X,H)
*     Functions:
        REAL     Conveyance,dConveyance
        EXTERNAL Conveyance,dConveyance

*     Intrinsics:
        REAL      ABS,SQRT
        INTRINSIC ABS,SQRT

*----------------------------------------------------------------------------

      NewK = Q/SQRT(ABS(S))
      NewH   = H

      DO 100 I=1,10

        dH = (NewK-Conveyance(Branch,X,NewH))/dConveyance(Branch,X,NewH)
        NewH  = NewH+dH
        IF(ABS(dH/NewH).LT.0.001) GO TO 102

 100  CONTINUE

      WRITE(*,*) '***Warning(Norm) did not close in 10 iterations...'
      WRITE(*,*) 'Branch = ',Branch,' X = ',X,' H = ',H
      WRITE(*,*) 'Q = ',Q,'  Slope = ',S

 102  CONTINUE

      Norm = NewH

      RETURN
      END



*============================================================================
        SUBROUTINE ReadControlData
     I                  (ControlFileUnit,ControlFileName,MaxLocations,
     O                   NumberOfChannels,NumberOfLocations,
     O                   Branch,XLocation,Q)
*============================================================================

*     Purpose:   Read locations and discharges.

        IMPLICIT NONE
```

```
*            - returns d(WetPerimeter)/dH.
*
*     Arguments:
*       fUnit  - FORTRAN unit number for proerties input data file
*       FileName - properties input data file name
*       NumberOfChannels    - number of branches in the current network
*       Branch  - current branch number
*       X       - current downstream distance
*       H       - current depth of flow
*
*
*     Module data:
*       X1(i)      - downstream reference distance,
*                    at upstream extent of channel "i".
*       Width1(i) - channel width at X(i).
*       Btm1(i)    - channel-bottom elevation at X(i).
*       X21(i)     - downstream reference distance,
*                    at downstream extent of channel "i".
*       Width2(i) - channel width at X(i).
*       Btm2(i)    - channel-bottom elevation at X(i).
*       Eta(i)     - effective Manning's n for channel "i".
*       i          - channel sequence number.
*
*       FileName   - name of a file containing hydraulic properties
*                      appropriate to his module.
*       fUnit      - FORTRAN unit number for the FileName.
*
*******************************************************************************


*==============================================================================
*     Public:
        LOGICAL FUNCTION InitHydraulicProperties(NumberOfChannels)
*==============================================================================

*     Purpose:  Begin initialization of hydraulic properties module.

        IMPLICIT NONE

*     Arguments:
        INTEGER NumberOfChannels
        CHARACTER*12 FileName

*     Definitions:
```

```
*       fUnit  - FORTRAN unit number for properties data.
*       NumberOfChannels   - current number of channels in network.
*       FileName - file namae corresponding to fUnit.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,X2,Width2,Btm2,Eta
        SAVE /PRPDAT/
        INTEGER fUnit, PrintOption
        COMMON / PROPUNIT / fUnit, PrintOption
        SAVE / PROPUNIT /

*     Local variables:
        INTEGER ChannelsInitialized, ChannelNumber, I

*     Functions:
        LOGICAL  ReadHydraulicProperties, OpenOldFlatFile
        EXTERNAL ReadHydraulicProperties, OpenOldFlatFile


*-----------------------------------------------------------------------

      InitHydraulicProperties = .FALSE.

*---- Set properties file name and unit number.
      FileName = 'cx_tbl.dat  '
      fUnit = 12

      IF(NumberOfChannels.LE.MaxChannels) THEN

        IF(OpenOldFlatFile(fUnit,FileName)) THEN
          READ(fUnit,*) PrintOption
        ELSE
          WRITE(*,*) 'Could not open ',FileName
          RETURN
        END IF

        ChannelsInitialized = 0

        DO 100 I = 1,NumberOfChannels
```

```
          ChannelNumber = I

          IF( ReadHydraulicProperties(ChannelNumber) ) THEN
            ChannelsInitialized = ChannelsInitialized + 1
          END IF
 100    CONTINUE


      ELSE

        WRITE(*,*) 'Error(InitHydraulicProperties)...'
        WRITE(*,*) NumberOfChannels,
     #            ' specified is greater than ',MaxChannels
        RETURN

      END IF

      IF(ChannelsInitialized .EQ. NumberOfChannels) THEN
        InitHydraulicProperties = .TRUE.
      ELSE
        WRITE(*,*) 'Failed to initialize',
     #      NumberOfChannels - ChannelsInitialized,' channels...'
      END IF

      CLOSE(fUnit)

      RETURN
      END



*==========================================================================
*     Public:
        REAL FUNCTION BtmSlope(Branch,X)
*==========================================================================

*     Purpose:   Compute bottom slope.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X
```

```
*     Definitions:
*       Branch  - channel number.
*       X       - downstream distance within channel.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,X2,Width2,Btm2,Eta
        SAVE /PRPDAT/

*-----------------------------------------------------------------------------

      BtmSlope      = ( Btm2(Branch) - Btm1(Branch) )/
     #                 (X2(Branch)-X1(Branch))

      RETURN
      END


*==============================================================================
*     Public:
        REAL FUNCTION Conveyance(Branch,X,H)
*==============================================================================

*     Purpose:   Compute conveyance.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Local variables:
        REAL R53,R23
        PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)
```

```
*     Functions:
        REAL     CxArea,WetPerimeter,ManningN
        EXTERNAL CxArea,WetPerimeter,ManningN

*-----------------------------------------------------------------------

      Conveyance = 1.486*(CxArea(Branch,X,H)**R53)
     #               /(WetPerimeter(Branch,X,H)**R23)
     #                 /ManningN(Branch,X,H)

      RETURN
      END


*=======================================================================
*     Public:
        REAL FUNCTION dConveyance(Branch,X,H)
*=======================================================================

*     Purpose:   Compute d(Conveyance)/dH.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Local variables:
        REAL R53,R23
        PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)

*     Functions:
        REAL     ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        REAL     ManningN,dManningN,Conveyance
        EXTERNAL ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        EXTERNAL ManningN,dManningN,Conveyance
```

```
*-----------------------------------------------------------------------

      dConveyance = Conveyance(Branch,X,H) * (
     #       R53*ChannelWidth(Branch,X,H)/CxArea(Branch,X,H)
     #      -R23*dWetPerimeter(Branch,X,H)/WetPerimeter(Branch,X,H)
     #      -   dManningN(Branch,X,H)/ManningN(Branch,X,H)
     #                                       )

      RETURN
      END


*=======================================================================
*     Public:
        REAL FUNCTION ChannelWidth(Branch,X,H)
*=======================================================================

*     Purpose:   Compute width of channel.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,X2,Width2,Btm2,Eta
        SAVE /PRPDAT/

*-----------------------------------------------------------------------

      ChannelWidth = (
     #     (X2(Branch)-X)*Width1(Branch)+(X-X1(Branch))*Width2(Branch)
     #               ) / (X2(Branch)-X1(Branch))

      RETURN
      END


*=================================================================================
*     Public:
        REAL FUNCTION CxArea(Branch,X,H)
*=================================================================================

*     Purpose:   Compute cross-sectional area.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Functions:
        REAL ChannelWidth
        EXTERNAL ChannelWidth

      CxArea = H*ChannelWidth(Branch,X,H)

      RETURN
      END



*=================================================================================
*     Public:
        REAL FUNCTION WetPerimeter(Branch,X,H)
*=================================================================================

*     Purpose:   Compute wetted perimeter.

        IMPLICIT NONE
```

```
*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Functions:
        REAL ChannelWidth
        EXTERNAL ChannelWidth

*-----------------------------------------------------------------------------

      WetPerimeter = 2.0*H+ChannelWidth(Branch,X,H)

      RETURN
      END



*==============================================================================
*     Public:
        REAL FUNCTION dWetPerimeter(Branch,X,H)
*==============================================================================

*     Purpose:   Compute d(wp)/dH.
        IMPLICIT NONE
*     Arguments:
        INTEGER Branch
        REAL X,H

*-----------------------------------------------------------------------------

      dWetPerimeter = 2.0

      RETURN
      END



*==============================================================================
*     Public:
        REAL FUNCTION ManningN(Branch,X,H)
*=============================================================================

*     Purpose:   Compute effective flow-resistance coefficient.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,X2,Width2,Btm2,Eta
        SAVE /PRPDAT/

*----------------------------------------------------------------------------

      ManningN = Eta(Branch)

      RETURN
      END


*=============================================================================
*     Public:
        REAL FUNCTION dManningN(Branch,X,H)
*=============================================================================

*     Purpose:   Compute d(ManningN)/dH.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*-----------------------------------------------------------------------

      dManningN = 0.0

      RETURN
      END



*=======================================================================
*     Private:
        LOGICAL FUNCTION ReadHydraulicProperties(ChannelNumber)
*=======================================================================

*     Purpose:  Read geometric and hydraulic properties.

        IMPLICIT NONE

*     Arguments:
        INTEGER ChannelNumber

*     Definitions:
*       NumberOfChannels   - number of channels

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,X2,Width2,Btm2,Eta
        SAVE /PRPDAT/
        INTEGER fUnit, PrintOption
        COMMON / PROPUNIT / fUnit, PrintOption
        SAVE / PROPUNIT /
```

```
*     Local variables:
        INTEGER Brn, I

*     Functions:
        LOGICAL  OpenOldFlatFile
        EXTERNAL OpenOldFlatFile

*     Output formats:
 2001 FORMAT(//'                  Cross-sectional properties...'//)
 2002 FORMAT('                    Distance     Width   Btm_elev'/
     #'Branch',I3/
     #'Upstream.....',3F10.2/
     #'Downstream...',3F10.2/
     #'Effective n..',F6.3//)

*-----------------------------------------------------------------------

      ReadHydraulicProperties = .FALSE.

      IF(PrintOption.GT.0) THEN
        WRITE(*,*) ' '
        WRITE(*,*) 'Channel ',ChannelNumber,'...'
        WRITE(*,2001)
      END IF

      Brn = ChannelNumber
      READ(fUnit,*,END=200) I

      IF(I .NE. Brn ) THEN
        WRITE(*,*) 'Channel sequence error...'
        WRITE(*,*) 'Read number', I
        WRITE(*,*) 'Changed to ',Brn
      END IF

      READ(fUnit,*,END=200)
     #  X1(Brn),Width1(Brn),Btm1(Brn),
     #  X2(Brn),Width2(Brn),Btm2(Brn),Eta(Brn)

      IF(PrintOption.GT.0) THEN
        WRITE(*,2002) Brn,
     #  X1(Brn),Width1(Brn),Btm1(Brn),
     #  X2(Brn),Width2(Brn),Btm2(Brn),Eta(Brn)
      END IF
      ReadHydraulicProperties = .TRUE.

      RETURN

200   CONTINUE

      WRITE(*,*) 'End of file...reading Channel',Brn

      RETURN
      END
```

Appendix D–Source Code for Trapezoidal Channel Properties Module

```
***** BOF TRAPGEOM ****************************************************
*
*     This file is a FORTRAN module for the computation of the geometric
*     and hydraulic properties of multiple, trapezoidal, non-prismatic
*     channels.  Manning's "n" is assumed constant with depth and
*     downstream distance within a branch.   A branch is represented by
*     two rectangular cross sections at its extremities.  Width, depth,
*     and wetted perimeter are interpolated linearly.  Area and
*     conveyance are subsequently computed from these interpolated
*     values.  Two data lines are input for each branch.  All input is
*     free-field.
*
*     line 0:
*       PrintOption (integer)
*
*     line 1:
*       ChannelNumber (integer)
*
*     line 2:
*       X1 , Width1, Btm1, Z1, X2, Width2, Btm2, Z2, Eta  (all real),
*
*          (Repeat above lines 1 and 2 for additional branches.)
*
*     where,
*
*       PrintOption - index for writing to standard output device.
*          [0] input data will not be written to standard output.
*          [1] input data will be echoed to standard output.
*
*       ChannelNumber - sequence number of channel, monotonically
*          increasing, beginning with 1.
*
*       X1      - downstream reference distance for upstream extent of
*                 branch
*       Width1  - channel width corresponding to X1
*       Btm1    - channel-bottom elevation corresponding to X1
*       X2      - downstream reference distance for downstream extent of
*                 branch
*       Width2  - channel width corresponding to X2
*       Btm2    - channel-bottom elevation corresponding to X2
*       Eta     - effective Manning's n for the branch
*
```

```
*
*
*     Module note: Only functions or subroutines marked "public" should
*                  be used outside of this module as those marked
*                  "private" may not be supported by future revisions of
*                  this module or replacement modules.  Likewise, no data
*                  or common blocks contained within this module should
*                  be accessed by routines outside of this module accept
*                  through the use of "public" functions or subroutines
*                  contained within this module.
*
*     Non-standard usage: Symbolic names in this module may be
*                         represented by as many as 31 characters in
*                         order to provide better definition directly in
*                         the code.  Standard FORTRAN allows only 6
*                         characters, but this restriction is generally
*                         extended to 32 characters by most compilers.
*
*
*     Public functions:
*
*     LOGICAL FUNCTION InitHydraulicProperties(NumberOfChannels)
*            - begin initialization of module.
*
*     REAL FUNCTION BtmSlope(Branch,X)
*            - returns channel-bottom slope.
*
*     REAL FUNCTION ChannelWidth(Branch,X,H)
*            - returns channel width.
*
*     REAL FUNCTION dChannelWidth(Branch,X,H)
*            - returns channel side slope.
*
*     REAL FUNCTION CxArea(Branch,X,H)
*            - returns cross-sectional area.
*
*     REAL FUNCTION Conveyance(Branch,X,H)
*            - returns sinuosity-weighted conveyance.
*
*     REAL FUNCTION dConveyance(Branch,X,H)
*            - returns d(Conveyance)/dH.
*
*     REAL FUNCTION WetPerimeter(Branch,X,H)
*            - returns wetted perimeter.
```

```
*
*     REAL FUNCTION dWetPerimeter(Branch,X,H)
*            - returns d(WetPerimeter)/dH.
*
*     Arguments:
*       fUnit  - FORTRAN unit number for proerties input data file
*       FileName - properties input data file name
*       NumberOfChannels   - number of branches in the current network
*       Branch  - current branch number
*       X      - current downstream distance
*       H      - current depth of flow
*
*
*     Module data:
*       X1(i)      - downstream reference distance,
*                    at upstream extent of channel "i".
*       Width1(i) - channel width at X(i).
*       Btm1(i)    - channel-bottom elevation at X1(i).
*       Z1(i)      - channel side slope at X1(i).
*       X2(i)      - downstream reference distance,
*                    at downstream extent of channel "i".
*       Width2(i) - channel width at X(i).
*       Btm2(i)    - channel-bottom elevation at X2(i).
*       Z2(i)      - channel side slope at X2(i).
*       Eta(i)     - effective Manning's n for channel "i".
*       i          - channel sequence number.
*
*       FileName   - name of a file containing hydraulic properties
*                      appropriate to his module.
*       fUnit      - FORTRAN unit number for the FileName.
*
*     Private functions:
*
*     REAL FUNCTION ManningN(Branch,X,H)
*            - returns Mannings n.
*
*     REAL FUNCTION dManningN(Branch,X,H)
*            - returns d(Mannings n)/dH.
*
*     LOGICAL FUNCTION ReadHydraulicProperties(ChannelNumber)
*            - reads hydraulic properties from flat file.
*
********************************************************************************
```

```
*==============================================================================
*     Public:
        LOGICAL FUNCTION InitHydraulicProperties(NumberOfChannels)
*==============================================================================

*     Purpose:  Begin initialization of hydraulic properties module.

        IMPLICIT NONE

*     Arguments:
        INTEGER NumberOfChannels
        CHARACTER*12 FileName

*     Definitions:
*       fUnit  - FORTRAN unit number for properties data.
*       NumberOfChannels   - current number of channels in network.
*       FileName - file namae corresponding to fUnit.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
        SAVE /PRPDAT/
        INTEGER fUnit, PrintOption
        COMMON / PROPUNIT / fUnit, PrintOption
        SAVE / PROPUNIT /

*     Local variables:
        INTEGER ChannelsInitialized, ChannelNumber, I

*     Functions:
        LOGICAL  ReadHydraulicProperties, OpenOldFlatFile
        EXTERNAL ReadHydraulicProperties, OpenOldFlatFile


*-----------------------------------------------------------------------------

      InitHydraulicProperties = .FALSE.
```

```
*---- Set properties file name and unit number.
      FileName = 'cx_tbl.dat  '
      fUnit = 12

      IF(NumberOfChannels.LE.MaxChannels) THEN

        IF(OpenOldFlatFile(fUnit,FileName)) THEN
          READ(fUnit,*) PrintOption
        ELSE
          WRITE(*,*) 'Could not open ',FileName
          RETURN
        END IF

        ChannelsInitialized = 0

        DO 100 I = 1,NumberOfChannels
          ChannelNumber = I

          IF( ReadHydraulicProperties(ChannelNumber) ) THEN
            ChannelsInitialized = ChannelsInitialized + 1
          END IF
 100    CONTINUE


      ELSE

        WRITE(*,*) 'Error(InitHydraulicProperties)...'
        WRITE(*,*) NumberOfChannels,
     #            ' specified is greater than ',MaxChannels
        RETURN

      END IF

      IF(ChannelsInitialized .EQ. NumberOfChannels) THEN
        InitHydraulicProperties = .TRUE.
      ELSE
        WRITE(*,*) 'Failed to initialize',
     #      NumberOfChannels - ChannelsInitialized,' channels...'
      END IF

      CLOSE(fUnit)

      RETURN
      END


*==============================================================================
*     Public:
        REAL FUNCTION BtmSlope(Branch,X)
*==============================================================================

*     Purpose:   Compute bottom slope.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X

*     Definitions:
*       Branch  - channel number.
*       X       - downstream distance within channel.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
        SAVE /PRPDAT/

*-----------------------------------------------------------------------------

      BtmSlope  = (Btm2(Branch) - Btm1(Branch))
     #          / (X2(Branch)   - X1(Branch))

      RETURN
      END


*==============================================================================
*     Public:
       REAL FUNCTION Conveyance(Branch,X,H)
*==========================================================================

*     Purpose:   Compute conveyance.

       IMPLICIT NONE

*     Arguments:
       INTEGER Branch
       REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Local variables:
       REAL R53,R23
       PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)

*     Functions:
       REAL     CxArea,WetPerimeter,ManningN
       EXTERNAL CxArea,WetPerimeter,ManningN

*-------------------------------------------------------------------------

      Conveyance = 1.486 / ManningN(Branch,X,H)
     #           * (CxArea(Branch,X,H)**R53)
     #           / (WetPerimeter(Branch,X,H)**R23)

      RETURN
      END



*==========================================================================
*     Public:
       REAL FUNCTION dConveyance(Branch,X,H)
*==========================================================================

*     Purpose:   Compute d(Conveyance)/dH.

       IMPLICIT NONE
```

```
*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Local variables:
        REAL R53,R23
        PARAMETER (R53 = 5.0/3.0, R23 = 2.0/3.0)

*     Functions:
        REAL     ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        REAL     ManningN,dManningN,Conveyance
        EXTERNAL ChannelWidth,CxArea,WetPerimeter,dWetPerimeter
        EXTERNAL ManningN,dManningN,Conveyance

*-----------------------------------------------------------------------

      dConveyance = Conveyance(Branch,X,H)
     #            * (R53*ChannelWidth(Branch,X,H) / CxArea(Branch,X,H)
     #            -  R23*dWetPerimeter(Branch,X,H)
     #               / WetPerimeter(Branch,X,H)
     #            -  dManningN(Branch,X,H) / ManningN(Branch,X,H)
     #              )

      RETURN
      END


*=======================================================================
*     Public:
        REAL FUNCTION ChannelWidth(Branch,X,H)
*=======================================================================

*     Purpose:   Compute width of channel.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
C       SAVE /PRPDAT/

*-----------------------------------------------------------------------

      ChannelWidth = ((X2(Branch)-X) * Width1(Branch)
     #             +  (X-X1(Branch)) * Width2(Branch))
     #             / (X2(Branch)-X1(Branch))

      RETURN
      END


*=======================================================================
*     Public:
        REAL FUNCTION dChannelWidth(Branch,X,H)
*=======================================================================

*     Purpose:   Compute side slope of channel.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
C       SAVE /PRPDAT/

*-----------------------------------------------------------------------

      dChannelWidth = ((X2(Branch)-X) * Z1(Branch)
     #              +  (X-X1(Branch)) * Z2(Branch))
     #              / (X2(Branch)-X1(Branch))

      RETURN
      END


*=======================================================================
*     Public:
        REAL FUNCTION CxArea(Branch,X,H)
*=======================================================================

*     Purpose:   Compute cross-sectional area.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Functions:
        REAL     ChannelWidth, dChannelWidth
        EXTERNAL ChannelWidth, dChannelWidth
      CxArea = H * ChannelWidth(Branch,X,H)
     #       + H * H * dChannelWidth(Branch,X,H)

      RETURN
      END


*=============================================================================
*     Public:
        REAL FUNCTION WetPerimeter(Branch,X,H)
*=============================================================================

*     Purpose:   Compute wetted perimeter.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Functions:
        REAL     ChannelWidth, dChannelWidth
        EXTERNAL ChannelWidth, dChannelWidth

*----------------------------------------------------------------------------

      WetPerimeter = ChannelWidth(Branch,X,H)
     #             + 2.0 * H * (1.0 + dChannelWidth(Branch,X,H)**2)**0.5

      RETURN
      END


*=============================================================================
*     Public:
        REAL FUNCTION dWetPerimeter(Branch,X,H)
```

```
*=============================================================================

*     Purpose:   Compute d(wp)/dH.
        IMPLICIT NONE
*     Arguments:
        INTEGER Branch
        REAL X,H

*     Functions:
        REAL     dChannelWidth
        EXTERNAL dChannelWidth

*-----------------------------------------------------------------------------

      dWetPerimeter = 2.0 * (1.0 + dChannelWidth(Branch,X,H)**2)**0.5

      RETURN
      END



*=============================================================================
*     Private:
        REAL FUNCTION ManningN(Branch,X,H)
*=============================================================================

*     Purpose:   Compute effective flow-resistance coefficient.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
```

```
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
C       SAVE /PRPDAT/

*-----------------------------------------------------------------------

      ManningN = Eta(Branch)

      RETURN
      END



*=======================================================================
*     Private:
        REAL FUNCTION dManningN(Branch,X,H)
*=======================================================================

*     Purpose:   Compute d(ManningN)/dH.

        IMPLICIT NONE

*     Arguments:
        INTEGER Branch
        REAL X,H

*     Definitions:
*       Branch  - branch number.
*       X       - downstream distance.
*       H       - depth of flow.

*-----------------------------------------------------------------------

      dManningN = 0.0

      RETURN
      END



*=======================================================================
*     Private:
      LOGICAL FUNCTION ReadHydraulicProperties(ChannelNumber)
*==========================================================================

*     Purpose:  Read geometric and hydraulic properties.

      IMPLICIT NONE

*     Arguments:
        INTEGER ChannelNumber

*     Definitions:
*       NumberOfChannels   - number of channels

*     Module data:
        INTEGER MaxChannels
        PARAMETER (MaxChannels = 25)
        REAL X1(MaxChannels),Width1(MaxChannels),Btm1(MaxChannels),
     *    Z1(MaxChannels)
        REAL X2(MaxChannels),Width2(MaxChannels),Btm2(MaxChannels),
     *    Z2(MaxChannels)
        REAL Eta(MaxChannels)
        COMMON  /PRPDAT/ X1,Width1,Btm1,Z1,X2,Width2,Btm2,Z2,Eta
C       SAVE /PRPDAT/
        INTEGER fUnit, PrintOption
        COMMON / PROPUNIT / fUnit, PrintOption
        SAVE / PROPUNIT /

*     Local variables:
        INTEGER Brn, I

*     Functions:
        LOGICAL  OpenOldFlatFile
        EXTERNAL OpenOldFlatFile

*     Output formats:
 2001 FORMAT(//'            Cross-sectional properties...'//)
 2002 FORMAT('            Distance     Width   Btm_elev     Z'/
     #' Branch',I3/
     #' Upstream.....',4F10.2/
     #' Downstream...',4F10.2/
     #' Effective n..',F6.3//)

*--------------------------------------------------------------------------
```

```
      ReadHydraulicProperties = .TRUE.

      IF(PrintOption.GT.0) THEN
        WRITE(*,*) ' '
        WRITE(*,*) 'Channel ',ChannelNumber,'...'
        WRITE(*,2001)
      END IF

      Brn = ChannelNumber
      READ(fUnit,*,END=200) I

      IF(I .NE. Brn ) THEN
        WRITE(*,*) 'Channel sequence error...'
        WRITE(*,*) 'Read number', I
        ReadHydraulicProperties = .FALSE.
      END IF

      READ(fUnit,*,END=200)
     #  X1(Brn), Width1(Brn), Btm1(Brn), Z1(Brn),
     #  X2(Brn), Width2(Brn), Btm2(Brn), Z2(Brn),
     #  Eta(Brn)

      IF(PrintOption.GT.0) THEN
        WRITE(*,2002) Brn,
     #    X1(Brn), Width1(Brn), Btm1(Brn), Z1(Brn),
     #    X2(Brn), Width2(Brn), Btm2(Brn), Z2(Brn), Eta(Brn)
      END IF

      RETURN

 200  CONTINUE

      WRITE(*,*) 'Unexpected end of file...reading Channel',Brn
      ReadHydraulicProperties = .FALSE.

      RETURN
      END

***** EOF TRAPGEOM *****************************************************
```

Appendix E–Control Data for Example Programs

```
3
1, 3500.0,  250.0
1, 50000.0, 2000.0
2, 55000.0, 250.0
2, 3500.0,  250.0
1, 50000.0, 2000.0
1, 55000.0, 250.0
3, 3500.0,  250.0
1, 50000.0, 2000.0
1, 55000.0, 250.0
```

Appendix F–Rectangular Channel Geometry Data

```
1
1
0.0 100.0 70.0 70000.0 100.0 0.0 0.045
2
0.0 100.0 70.0 70000.0 100.0 0.0 0.045
3
0.0 100.0 70.0 70000.0 100.0 0.0 0.045
```

Appendix G–Trapezoidal Channel Geometry Data

```
1
1
0.0 100. 70. 1.5 70000. 100. 0.0 1.5 0.045
2
0.0 100. 70. 1.5 70000. 100. 0.0 1.5 0.045
3
0.0 100. 70. 1.5 70000. 100. 0.0 1.5 0.045
```