Chapter Fifteen

INTRODUCTION TO COMPAQ ALPHA FORTRAN

Introduction

Fortran (Formula Translation) is the oldest, but one of the most widely used computer programming languages in the sciences and engineering. Its continued use is due to its efficiency and rapid program execution as well as its power and versatility in dealing with computationally intensive problems . This chapter introduces the history and features of this programming language and examines the basic elements of the Compaq Alpha implementation of Fortran.

Objectives

To begin solving a problem using the Fortran high-level language, a programmer should be able to:

- Understand the Compaq Alpha Fortran standards
- Know the elements that make up a Fortran source program
- Know the character set supported by Compaq Alpha Fortran
- Understand the general rules for coding in Compaq Alpha Fortran

History and Features of Fortran

Overview

Fortran has undergone a number of revisions since its introduction by IBM in the late 1950's. Fortran IV was the first standardized version issued in 1966 by the American National Standards Institute (ANSI). The Compaq Alpha version of the Fortran language bears little resemblance to the initial standard.

This section will take a brief look at the language and its three major revisions:

- Fortran-IV (ANSI 1966)
- Fortran-77
- Fortran-90/95

Fortran Language Features

Fortran (*FORmula TRANslation*) was the result of a project begun by John Backus at IBM in 1954. The goal of this project was to provide a way for programmers to express mathematical formulas through a formal language that computers could translate into machine instructions. Fortran has evolved continuously over the years in response to the needs of users. Areas of evolution have addressed mathematical expressivity, program maintainability, hardware control (such as I/O), and of course, code optimization. In the meantime, other languages such as C and C++ have been designed to better meet the non-mathematical aspects of software design. By the 1980's concerns for the future of Fortran prompted language designers to propose extensions which incorporated the best features of these other high-level languages and, in addition, provided new levels of mathematical expressivity that had become popular on supercomputers such as the CYBER and CRAY systems. This language became standardized as Fortran 90.

Fortran was designed to support computer problem solving in the sciences and engineering, owing its success to its power and versatility in dealing with computationally intensive problems and the availability of a wide range of specialized mathematical and statistical library programs. It has always maintained an ability to compile efficiently and execute rapidly. The Compaq Alpha Fortran compiler has a reputation of being the most efficient of all Compaq Alpha compilers, and perhaps, among the most efficent of all compilers in the industry.

Elements of a Fortran Program

Overview

This section provides an overview of the make up of a Fortran source program. It describes the concepts of a program unit and the rules governing the use of statements and symbols within a program unit. It also describes the use of comments within programs and the Fortran character set.

This section discusses:

- Program Units
- Statements
- Symbolic Names
- Comments
- The Character Set

Program Units

A Fortran program consists of one or more program units. A ***program unit*** is a sequence of statements that defines the data environment and the steps necessary to perform a computing procedure. A program unit can be a main program, a subprogram, a code module, or a block of data. An executable program contains one main program and, optionally, one or more other types of program units. Program units can be compiled separately or in groups determined by the programmer.

A ***subprogram*** is a function or subroutine. Subprograms can be built in the same source code file as the main program or in separate files. Subprograms that are built separately are referred to as external subprograms. Subprograms define procedures to be performed and can be called or invoked from other program units in a Fortran program. Subroutines and functions will be discussed in Chapter Seventeen.

Two additional program units, the module and block data, are supported by the Fortran language, but not supported in the SuRPAS development environment. These two program units are discussed in the following paragraphs for your information.

A ***module*** contains definitions that can be made accessible to other program units. These modules can take the form of data and type definitions, definitions of procedures, and procedure interfaces. Module subprograms can be either functions or subroutines. They can be invoked by other subprograms in the module or by other program units that access the module. *Note that the module is not used in PFPC. It is not supported until Fortran-90.*

A ***block data*** program unit specifies initial values for data objects in named common blocks. The block data unit has been a key element of the Fortran language. With the implementation of new features in Fortran 90/95, the block data unit can be replaced by the module. *Block data program units are not used at PFPC.*

Program Statements

Program statements are grouped into two general classes: executable and nonexecutable. Executable statements describe an action to be performed by the program. Nonexecutable statements describe data characteristics and environment, data-conversion information, and editing capabilities.

Program statements are divided into lines. Only one statement can be placed on any line, and traditionally, a line does not extend beyond 72 characters (columns 73 through 80 were reserved for line sequence numbers). In modern Fortran, including Compaq Fortran, statement lines can extend to 132 characters. If the statement is too long to fit on a single line it can be continued on the next line by using a continuation indicator. A continuation line is identified by a continuation character in column 6 of the line.

Program statements can be easily identified or referred to by placing a statement label at the beginning of the statement line. The statement label must be an integer and must appear in the first 5 columns of the statement's first line. Any statement can have a label, but you can only point to labels attached to executable statements or FORMAT statements.

Statement Order in a Program Unit

The ordering of source code statements in a Fortran program has become quite vague as the language has evolved. In addition to the freedoms that have come with each succeeding standard, the Compaq version of the language has added additional freedoms.

The orderly manner in which the Cobol language categorized activities was evident in the early versions of Fortran and that consistency can still be seen today in the code of more traditional programmers. Cobol forced its programmers to describe the program's environment before describing the data. Only after all the data was defined could the programmer begin to execute code. PFPC maintains this this orderly structure through its SuRPAS Fortran coding standards and two coding templates used when creating Fortran code.

Fortran has attempted to impose the same order in its program units. Each program unit begins with a unit declaration (PROGRAM, SUBROUTINE, FUNCTION, BLOCK DATA, MODULE, etc.) and ends with a termination statement (RETURN, STOP, END). Traditionally, environment statements follow the unit statement. These statements provide general rules for data and code found in the program unit. It is in this area that we can limit the way variables are typed, arrays or data structures are referenced, and code and algorithms are executed.

Environment statements are followed by data declaration statements. These statements allow the programmer to declare variables to be used, define their type, and provide a beginning value. This is also where arrays and data structures are built and populated, where global data and structures are declared, and where memory overlays are built. Unlike languages such as *Cobol*, *Ada,* and *Pascal*, Fortran does not require that a variable be defined before it is used. Fortran follows the simple rule that unless a restriction is declared (using the IMPLICIT statement), all variables beginning with I, J, K, L, M, and N are of type integer and all other variables are of type character. Modern Fortran programmers begin all programs with an IMPLICIT NONE statement, which forces all variables to be typed. This is a PFPC standard; all variables must be pre-declared.

Only after all environment and data definitions have been provided does a programmer issue the first executable statement. PFPC maintains this orderly structure through the SuRPAS Fortran Coding Standards and two coding templates used when creating Fortran code.

Symbolic Names

In Fortran a symbolic name is an entity in a program unit. A symbolic name is a string of letters, numbers, and certain special characters (the dollar sign and the underscore). The first character in a symbolic name must be a letter and there can be a maximum of 31 characters (Fortran-90), although Compaq Alpha Fortran allows up to 63 characters.

Be careful when using the dollar sign ($) in a symbolic name.
OpenVMS naming conventions reserve names containing dollar signs for use by the operating system.

Unique Symbolic Names

Symbolic names must be unique throughout a Fortran program unit with two exceptions - the name of a structure and the name of a common block. The name of a structure can also be used as the name of fields of records. Common block names can also be used as variable or array names. This duplication of names should be kept to a minimum to avoid confusion.

When building an executable program containing two or more program units, there are certain entities whose symbolic names must be unique throughout the entire program. These entities are:

- Functions
- Subroutines
- Common blocks
- Main programs
- Block data modules
- Function entry points
- Subroutine entry points

Entities Identified by Symbolic Names

There are many entities within Fortran, some more obvious than others. The table below lists the Fortran entities that have symbolic names and indicates whether or not each entity can be assigned a data type (e.g., integer, character, real, string, etc.).

Fortran Entity	Typing Allowed
Variables	Yes
Arrays	**Yes**
Structures	No
Records	**No**
Record Elements	Yes
Statement functions	**Yes**
Intrinsic functions	Yes
Function subprograms	**Yes**
Subroutine subprograms	No
Common Blocks	**No**
Namelist data groups	No
Main programs	**No**
Block Data subprograms	No
Function entry points	**Yes**
Subroutine entry points	No
Parameter constants	**Yes**

Table 15-1 - Fortran Entity Data Typing

Comments

A comment is a line, or part of a line, which is included purely as information or description for the programmer or anyone reading the program; it is ignored by the compiler. Traditional Fortran requires that a comment character (PFPC standards allow the capital letter "C") be placed in column one to indicate that what follows is a comment.

The exclamation point signals that a comment follows, thus allowing comments to coexist on the same line as code; any characters that follow the exclamation point are ignored by the Fortran compiler. All text from the exclamation point to the end of the line will be ignored, so comments may appear on the same line as an executable statement.

In the example that follows, comments are placed at the beginning of a line and also on the same line as a line of executable code (as in the PRINT statement). Stand-alone comments are surrounded by a comment box (also called a "flower box").

```
       PROGRAM hello

       IMPLICIT NONE

C     /**********************************/
C     /*      This program displays     */
C     /*  HELLO WORLD on the terminal   */
C      /*********************************/

C     Issue the Print Statement

      PRINT *,' HELLO WORLD'  ! Display this character string

      END                     ! End of Program hello
```

Code Example 15-2 - Commenting Code

The AMS editor supports macros that create **(CBOX)** and remove **(DCBOX)** comment boxes.

The Fortran Character Set

Compaq Alpha Fortran supports the full Fortran 90 character set. This includes:

- All upper and lower case letters (A through Z and a through z)
- All numbers (0 through 9)
- The underscore (_)
- A set of special characters which include:
    - Blank (space or tab)
    - Equal sign (=)
    - Plus sign (+)
    - Minus sign (-)
    - Asterisk (*)
    - Slash (/)
    - Left Parenthesis (()
    - Right Parenthesis ())
    - Comma (,)
    - Period, Decimal Point (.)
    - Apostrophe - also referred to as Single Quote (‘)
    - Colon (:)
    - Exclamation Point (!)
    - Quotation Mark (“)
    - Percent sign (%)
    - Ampersand (&)
    - Semicolon (;)
    - Less than (<)
    - Greater than (>)
    - Question mark (?)
    - Dollar sign ($)
- Other printable characters
  Printable characters include the tab character (ASCII 09 hex) and characters that can be expressed as ASCII characters with codes ranging from 20 hex through 7E hex.
- Use of the space in source code
  You can use the space character to improve the legibility of a Fortran statement. The compiler ignores all spaces in a statement field except those within a character.   GO TO and GOTO are equivalent.

Except in character constants, the Fortran compiler makes no distinction between upper and lower case letters.

Fortran Coding Rules

Overview

Each line of Fortran source code can have up to four different fields. This section discusses line formats as well as these four statement fields and the rules that govern their use.

The four types of fields are:

- Statement label field
- Continuation field
- Statement field
- Sequence number field

A Line of Fortran Code

There are three ways to code a Fortran line: fixed format, free form, or tab format. The fixed format method was used when punching cards or using a coding form. The free form and tab format methods are used when entering lines at a workstation with a text editor such as EVE, EDT, or AMS.

Fixed Format Lines

The fixed format method for building a line of Fortran code represents the original manner in which code was produced. In this format all comments are designated by a letter “C” or an asterisk (*) in the first column of the line of code. The compiler ignores all characters in the line that follow a comment designator.

Line labels are placed in columns 1 through 5. Column 6 is used as a line extender. When a character is placed in column 6 (other than a blank or a zero), the remaining characters are designated as a continuation of the preceding line of code. The Fortran statement itself is placed in lines 7 through 72. Except within a character constant or string, blank columns have no meaning in a Fortran line of code and may be used freely to improve program appearance and readability. The compiler ignores all characters after column 72. These columns can be used for line sequencing or other program identification purposes, or they may be left blank.

On the following page is an example (Example 15-3) of a Fortran program that is coded using the fixed format method. Note that all code lines begin in column 7 and that all comments have a ”C” in column 1.

```
      PROGRAM circle

      IMPLICIT NONE

c This program calculates the equation of a circle passing
c through three points.

c Variable declarations
      REAL*4  x1,y1,x2,y2,x3,y3
      REAL*4  a,b,r

c Step 1
      PRINT *,” Please type the coordinates of three points”
      PRINT *,” in the order x1, y1, x2, y2, x3, y3”
      READ *,x1,y1,x2,y2,x3,y3

c Step 2
      CALL calculate_coeffs(x1,y1,x2,y2,x3,y3,a,b,r)

c Step 3
      PRINT *,” The center of the circle through three points”,
     *        “ is (“,a,”,”,b,”)”
      PRINT *,” Its radius is “,r

      END circle
```

Code Example 15-3 - Fixed Format Method

Some Basic Fortran Coding Concepts

It has been claimed both that programming is an art (Knuth, 1969) and a science (Gries, 1991). While programming does contain elements of both art and science, it is actually an engineering discipline (Ellis, 1994), governed by rules of procedure - rules determined by you and by the people for whom you design and write programs (your customers and end users).

The process of writing a program is composed of many steps - from clearly specifying the problem, to breaking the problem down into its most fundamental elements, to analyzing each element and forming success paths, to coding the solution according to a plan based on your analysis, (and finally) to exhaustively testing the solution you call your program.

In the following chapters we will concentrate on the use of Fortran as a tool for coding solutions to the problems addressed in class. Some of the techniques will be based on standards that have been set down by PFPC over the years, and others will be based on accepted Fortran standard coding practice. The PFPC standards can be found on the company intranet and in the standards section of each of the six Fortran chapters in this manual (beginning with this one). Some of the basic Fortran coding concepts are discussed in the paragraphs that follow.

Case Code

Fortran imposes no case restrictions on its programmers. Fortran keywords (those reserved command words used by the language and recognized by the compiler), constants, and variables can be typed in either upper or lower case. This manual follows the PFPC standard. All keywords, variables, and constants are typed in uppercase; comments are typed in upper and lower case.

The PROGRAM Statement

Every main Fortran program unit begins with a PROGRAM statement . This statement consists of the keyword PROGRAM followed by the name of the program (and usually the name of the source code file). As with all Fortran symbolic names, the program name must begin with a letter, and can only contain the letters A-Z and a-z, the numbers 0-9, and the underscore (_). It can have a maximum of 31 characters.

The END Statement

The final statement of the program must be an END statement. The END statement terminates execution of the program and returns control back to the OpenVMS operating system. The END statement should always be followed by a comment describing the construct that is ending. In the case of the program end, this comment should include the program name, as shown in the examples below:

- END ! of right_triangle
- END ! of PROGRAM right_triangle

The commented END statement makes the source code more readable and provides better documentation.

When using the full form of the **END** statement, be sure that the program name used is identical to the one found in the **PROGRAM** statement.

The IMPLICIT NONE Statement

Fortran supports the use of variables that have not been predeclared. In Chapter Sixteen we will discuss the use of implicit and explicit data typing, a feature that allows the definition of integer and character variable defaults. This feature dates back to the first Fortran compiler, but allows for variables to be created without any explicit definition or in-line documentation. At PFPC and in this manual, all variables must be explicitly defined. In order to force this rule on the compiler, all programs declare explicit variable definitions by placing the Fortran statement `IMPLICIT NONE` at the beginning of the program.

SuRPAS Coding Standards for Fortran

Overview

The SuRPAS Fortran coding standards cover many topics and issues that are addressed in the following chapters. Standards that are specific to topics covered in those chapters will be addressed in their standards sections.

The standards discussed in this section address the following Fortran coding topics:

- Indentation
- Comments
- Labels
- Continuation
- Variable Names

General SuRPAS Coding Standards

A piece of code is rarely written and maintained by a single programmer. A programmer may be asked to understand and modify code written by someone else - perhaps by a person who no longer works in the department, or even in the company. It is true that everyone has his own style of programming; however, it is necessary to write code in such a way that another individual will be able to pick it up and understand it.

Write code as efficiently as possible and make sure that it will be easily understood if another programmer picks it up.

Use TEMPLATE.FOR
There are small routines on FAL$MAIN and FAL$SUB named TEMPLATE.FOR and on FAL$COM named TEMPLATE.COM. These are a good basis for a routine (subroutine or main program) if you have nothing else to copy for your new routine. Reference the “Best Practice” page (on the PFPC intranet) for examples of programs.

The Prologue
The description section of any code module should provide a detailed explanation of what the process is doing. The inputs to the program should be defined as well as the outputs.

Indentation
The simplest way to indent is to use the tab key. However, when there are several indentations within the same block of code, you can end up working on less than half of the screen. Sometimes it makes more sense to just use 3 spaces to indent. Whatever you do, be consistent throughout the routine.

Comments
The more comments you can provide in the code, the easier the code will be to support. Be sure to explain areas of code that another programmer may not understand. Describe the situation rather than simply echoing back what the code is doing. Do not repeat actual code in your comments; changes in code may be missed in the comments.

Do NOT put a "C" at the start of a blank line used as a white space separator (except when the blank line is embedded in a larger comment block). The extra characters create "clutter" that makes it more difficult to read and manipulate the program.

Comments beginning in column 1 should be indicated by a capital C. Comments anywhere else (called in-line comments) are indicated by an exclamation point.

Comments should be entered in mixed case. In most cases, standalone comments (as opposed to in-line comments) should be located in a flower box that is lined up with the code that follows it. The first column should contain a "C" followed by multiple tabs to allow the comments to be lined up with the code and then the slash-asterisk (/*). An example of a preferred comment is:

```
C               /****************************************/
C               /*        This is a sample comment   */
C               /*****************************************/
```

Use the AMS Editor commands CBOX and DCBOX to create and remove comment boxes. Select the lines to be commented or uncommented, press "DO" to open the command window, and type the desired command.

Comments that begin with an exclamation point and are located on the right side of a line of code are an acceptable comment format.

If an in-line comment is not preceded by functional code, such as a second explanatory line in the data declaration section of a program, simply place the '!" in the appropriate column. A "C" in column 1 would be redundant and is not needed. For example, a leading "C" is not needed on the second of the following two lines because no code precedes the comment text:

```
      INTEGER*4    BLIP     ! Input  from  the
                            ! BLOP  subroutine
```

If you feel it necessary to use comments to disable code in a routine, make sure that there are at least 5 "C's" at the beginning of the line of code so that it isn't confused with the functioning code. The preference is to delete the unnecessary code.

Labels
With the exception of the standard 1000, 8000, and 9000 labels, no extraneous labels should be used unless they designate a main section of code or are connected to a GOTO or DO loop. All labels should be in numerical order within the program for readability. If there is a break in the code or something that needs to be noted, set it off with comments; a label isn't necessary.

When creating a label (with the exception of a FORMAT statement), it must appear on a CONTINUE statement before the target executable statement. Labels should never appear on executable statements themselves. By using a CONTINUE statement, additional execution statements can be added prior to the target of a GOTO statement without the need to move or reassign the label.

Continuations
A number or special character (* or +) in column 6 is an acceptable continuation, as is a tab followed by a number or special character.

Variable Names
Within a CFD, all variables should start with the 3 letter name of the CFD. For example, MASFAL.CFD has variable names of MASxxx. Usually the name is limited to 6 characters. The API's are the exception to this, but it is very clear when an API variable is used. Do not declare local variables with the same name as a SuRPAS file variable. This way, if someone looking at your code sees a variable named MASACT, he will know it is from MASFAL.CFD. If a local account number is needed, name it something like ACCOUNT instead of MAS_ACCOUNT.

If you are using the internal Read or Write command, the name of the variable that you are converting to should contain one of the following notations at the end:

- _I (if it is integer)
- _R (if it is real)
- _J (if it is Julian date)
- _C (if it results in a character)

APPENDIX: The History & Background of Fortran

The Fortran programming language was introduced by IBM in the late 1950's. It represents a milestone in computer history because it was the first high level language to be built with a working, optimizing compiler. The Fortran I optimizing compiler built for the IBM 704 computer held the record for code optimization for 20 years.

The Fortran I compiler became an instant success. Programs computing nuclear power reactor parameters took hours (instead of weeks) to write, and required much less programming skill. Another great advantage of the new compiler was that programs became portable. Fortran won the battle against Assembly language, and was adopted by the scientific and military communities; it was used extensively in the Space Program and military projects.

Fortran II, introduced in 1958, was a significant improvement; it added the capability for separate compilation of program modules, and assembly language modules could also be "linked loaded" with Fortran modules. Although Fortran III was never released to the public, it made possible the use of assembly language code right in the middle of Fortran code. Such "inline" assembly code can be more efficient, but the advantages of portability are lost. Fortran IV was introduced in 1961; it was a "clean up" of Fortran II, improving things like the implementation of the COMMON and EQUIVALENCE statements and eliminating some machine-dependent language irregularities.

In May of 1962 another milestone was reached as the ASA committee began developing a standard for the Fortran language. This very important step made it worthwhile for vendors to produce Fortran systems for every new computer and made Fortran an even more popular high level language. The new ASA standard was known as FORTRAN-66 since it was published in 1966. It was the first high level language standard in the world.

The FORTRAN 77 standard is recognized as the Fortran standard by many today, and compilers still conform to it. FORTRAN 77 added the following:

- DO loops with a decreasing control variable (index)
- Block if statements - IF ... THEN ... ELSE ... ENDIF (before FORTRAN-77 there was only the IF ... GOTO)
- Pretest of DO loops (before FORTRAN-77 DO loops were always executed at least once, so you had to add an IF...GOTO before the loop)
- The CHARACTER data type (before FORTRAN-77 characters were always stored inside INTEGER variables)
- Apostrophe delimited character string constants
- Main program termination without a STOP statement

Fortran 90 was published many years after FORTRAN-77, allowing other programming languages to evolve and compete. The most popular of these, C and its evolved variant C++, have become more popular in the traditional strongholds of Fortran (the scientific and engineering worlds) in spite of the fact that they are non-computationally oriented. A new standard, unofficially called Fortran-90, has been designed and widely implemented. It adds many powerful extensions to FORTRAN 77, making the language competitive with computer languages created later (e.g., C). Fortran 90 adds:

- Free format source code form (column independent)
- Modern control structures (CASE & DO WHILE)
- Records (structures)
- Array notation (array sections, array operators, etc.)
- Dynamic memory allocation
- Derived types and operator overloading keyword argument passing, INTENT (in, out, inout)
- Numeric precision and range control modules

Summary

Concepts

Elements of a Fortran Program

- The program unit is a sequence of statements that defines the data environment and the steps necessary to perform a computing procedure.
- Subprograms define procedures to be performed and can be called or invoked from other program units in a Fortran program.
- A module contains definitions that can be made accessible to other program units.
- Executable statements describe an action to be performed by the program.
- Nonexecutable statements describe data characteristics and environment, data-conversion information, and editing capabilities.
- Symbolic names must be unique throughout a Fortran program unit with two exceptions: the name of a structure and the name of a common block.
- A comment is a line, or part of a line, which is included purely as information or description for the programmer or anyone reading the program; it is ignored by the compiler.

Fortran Coding Rules

- The fixed format method for building a line of Fortran code represents the original manner in which code was produced.
- The free form and tab format methods are used when entering lines at a workstation with a text editor such as EVE, EDT, or AMS. Fortran code written in free form allows code statements to be written anywhere on the line, enabling the programmer to arrange the layout of a program to meet specific style preferences.
- All coding styles allow lower case characters to be used interchangeably with upper case characters. The PFPC coding standards require that all variable names be written in uppercase characters.

SuRPAS Coding Standards

- When indenting, tabs are easy; however, you may prefer to indent by hitting the space bar 3 times in order to conserve space. Whatever you do, be consistent throughout the routine.
- The more comments you can provide in the code, the easier the code will be to support.
- A number or special character in column 6 is an acceptable continuation, as is a tab followed by a number or special character. Note that SuRPAS code has traditionally used the plus sign (“+”) as the continuation character of preference.