CHAPTER 4

RUNNING FORTRAN-77 PROGRAMS ON THE IBM/PC

4.1 FORTRAN - A History

FORTRAN (short for FORmula TRANslation) was the first symbolic language of the procedure-oriented type (see Section 1.3). The earliest version of the language, now called FORTRAN-I, was developed at one of IBM's research laboratories in the mid-1950s. An improved version, called FORTRAN-II, was introduced in 1958 for use on IBM's first large scientific computer, the IBM 704.

Virtually every other computer manufacturer produced translating programs called compilers (see Section 1.3) for versions of the language for use on their own machines, and by the time the IBM 360 family of computers came onto the market in the mid-1960s, FORTRAN was well entrenched as the major computing language for scientific and engineering applications (another language, called COBOL, played a similarly dominant role for business applications).

IBM released translators for a much-improved, but compatible, version of the language called FORTRAN-IV with the introduction of the 360 computers. There were (and still are) many variants or dialects of the FORTRAN-IV language in use on different computers or by different industrial and academic groups. In 1966, ANSI, a professional standards group, prepared definitions for two FORTRAN standard languages, called the full language (intended for implementation on large computers), and the subset language (for implementation on small computers). The subset language is "upward" compatible with the full language (i.e., programs written in the subset language also satisfy all the syntactical rules for the full language). These standard languages are called FORTRAN-66, but the labels "FORTRAN-IV" and "FORTRAN-66" are used more or less interchangeably.

4.2 FORTRAN-77

In 1977, the ANSI group formulated a new set of standards for a considerably improved structured version of FORTRAN. Here, "structured" refers to a programming concept first formulated by Prof. Niklaus Wirth of the Technical University of Zurich (the designer of the procedure oriented language PASCAL and of the newer Modula-II language).

Roughly speaking, the structured approach to programming consists of breaking the overall problem into individual tasks or "modules" based on notions of iteration, sequencing, and selection. Individual modules are arranged in a hierarchial structure, each module having just one entry point (starting place) and one exit point. Control is passed from module to module in a downward sequence, with few if any unconditional branches to higher levels of the structure. Structured programs have, for example, very few GOTO statements, and tend to be much easier to "read" (in the sense of reading text in a natural language from top to bottom of a page) than nonstructured ones. Flow diagrams are not much used for describing structured algorithms because of this natural forward sequencing in the logic.

The structured version of FORTRAN defined by the ANSI group is now called FORTRAN-77. Again the standard consists of two definitions, a full implementation language and an upwardly compatible subset language. This version of the language is now the most popular one.

Interestingly, the FORTRAN-77 language standards are defined in such as way that, with few exceptions, programs written in standard FORTRAN-66 (an unstructured language) are upward compatible with FORTRAN-77 (a structured language). Therefore, it is possible to write unstructured as well as structured programs using FORTRAN-77 (the syntax for languages such as PASCAL tends to enforce the writing of structured programs). For this reason, many computer scientists consider FORTRAN to be an archaic programming language. Nevertheless, FORTRAN is still the most-used computing language for numerical calculations in engineering and the sciences.

Most "good" programmers use structured approaches to program formulation insofar as possible, regardless of the programming language used. In this context a "good" program is by definition a program that not only "works" (i.e., produces correct answers), but one whose logic is clear and can be understood easily by others. This latter characteristic is critically important when one programmer (or group of programmers) writes programs but different people maintain and improve them later on (a common situation in almost all industrial, governmental, and academic organizations).

4.3 FORTRAN Compilers on MTS

There are several different compilers for various versions of FORTRAN on the University's mainframe IBM 3090 computer operating under control of MTS (described in Chapter 5). The most important of these are available to all users of MTS in the following public files:

1.	***FTN**	IBM "G-level" compiler for FORTRAN-IV programs.
2.	***FORTRANH**	IBM "H-level" compiler for FORTRAN-IV programs.
3.	***WATFIV**	University of Waterloo (Canada) complier for WATFIV (a dialect of FORTRAN-IV) programs.
4.	***FORTRANVS**	IBM compiler for programs written in FORTRAN-77.

The ***FTN** compiler is probably the most used of the four; it is a resident program in the virtual memory (see Section 5.2) of the IBM 3090 computer, meaning that it is available essentially instantaneously (no loading from system disk storage) and reentrant, meaning that many users can be accessing the same memory-resident copy of the compiler at essentially the same time (in timesharing operation mode, as described in Chapter 5).

The ***FORTRANH** compiler is an optimizing compiler, meaning that it produces object versions of FORTRAN-66 source programs that run considerably faster than those produced by ***FTN**. Its compilation costs are quite high in comparison with ***FTN**, so it is not normally used until the program has been thoroughly tested and debugged with a less expensive compiler, and then only for programs that will be used frequently.

The ***FORTRANVS** compiler, from IBM, is an implementation of the full FORTRAN-77 language standards (with some nonstandard additional features also). Like ***FTN**, this compiler is resident and reentrant; it is somewhat more expensive to use than ***FTN**, but produces excellent object code.

4.4 FORTRAN-77 Compilers for the IBM/PC

FORTRAN compilers for personal computers are not very common, BASIC being the most used programming language for microcomputers. However, quality compilers have come onto the market during the past few years. The most popular of these are: (1) Microsoft FORTRAN compiler, for the subset version of FORTRAN-77, (2) Professional FORTRAN compiler (by Ryan-McFarland), for a full implementation version of FORTRAN-77, and (3) Microsoft FORTRAN Optimizing compiler, another full-implementation version of FORTRAN-77.

We have adopted Microsoft's Optimizing FORTRAN 77 compiler for use in the introductory computer course for engineering students because: 1) it supports all of the structured features of ANSI standard FORTRAN 77, 2) it is compatible with previous versions of FORTRAN (still very popular with scientists and engineers, with an enormous base of well tested programs), 3) it is upwardly compatible with Microsoft's subset FORTRAN compiler used on the CAEN Laboratory IBM/PCs, and 4) it is fully compatible with the MTS version of the IBM mainframe FORTRAN-77 compiler (the ***FORTRANVS** compiler available on the IBM 3090).

We have authored a companion text, FORTRAN-77 (with MTS and the IBM PC) that covers algorithm development and programming using the ANSI Standard FORTRAN-77 language, with emphasis on the Microsoft implementation for the IBM PC, PC-XT and PC-AT. Henceforth, when we refer to Microsoft (MS) FORTRAN we will be referring to the optimizing full-implementation compiler.

The Microsoft (MS) (optimizing) FORTRAN compiler is now installed on all of the IBM PCs, PC-XTs and PC-ATs in the Freshman Engineering College laboratories. The intent of this chapter is to familiarize you with use of this compiler on these IBM/PCs. Later, in Section 4.8, we describe use of Microsoft's subset FORTRAN compiler installed on the the CAEN laboratory IBM/PCs.

Throughout this chapter, we assume that your FORTRAN-77 program has already been written, probably with the assistance of a visual-editing program, such as VEDIT, and that it has been stored on a diskette in a DOS file with the file-name extension **.FOR**. The particular extension **.FOR** is required by the MS compiler (it also helps to point out which files contain FORTRAN programs). Throughout this chapter, we will call the DOS diskette file containing the source FORTRAN-77 program **PROG.FOR**.

4.5 Compilation, Linking, and Execution

The translation of your FORTRAN source program and its eventual execution by the computer is a three-step process, as described briefly in Section 1.3 and illustrated in Figure 1.3.

Step 1: Compilation is the process that translates an algorithm represented in the symbolic FORTRAN language (the source program) into an equivalent algorithm in the machine's language called the object program. There may be several individual source programs (called subprograms in FORTRAN parlance) involved in a "package" or "suite" of programs required to solve a problem. Typically, the compiler treats each subprogram separately to produce individual object programs (the individual source programs can even be translated at different times).

Step 2: Linking is the process that gathers together the individual object programs, along with any library programs, and produces a single interconnected machine language program called the object module. Library programs perform commonly needed tasks like trigonometric function evaluation, and are already available in object form, usually on disk storage in a library with name extension .LIB.

Individual machine language programs are usually compiled into relocatable form, meaning that they can be moved (relocated) as blocks of instructions to any block of addresses in the main store, provided there is sufficient unused memory available. The process of linking the individual programs together to create the object module involves a significant amount of address recalculation by the linker, since any addresses associated with operands in the program must be modified to account for the relocation of the programs in the memory.

Step 3: Execution involves the loading of the object module produced by the linker into the computer's main store (the program that does the memory loading is called a loader) and then turning over control of the central processor to it. The executing program reads the user's data from input devices or files and writes results for display or printing on output devices or for storage in a file(s).

The three steps, viewed from the standpoint of the programs involved and their inputs and outputs are:

Compilation	Program:	compiler
	Input :	source language programs
	Output :	relocatable object programs
Linking	Program:	linker
	Input :	relocatable object programs from compilation and library
	Output :	object module
Execution	Program:	object module
	Input :	user's data (from keyboard, disk files)
	Output :	user's results (to monitor, printer, disk files)

The responsibility for actually loading each of the programs (compiler, linker, object module) into the main store is assumed by yet another program called the loader, one of the operating system (e.g., DOS) programs. In some cases (almost always, in large computing systems), the linking is done in conjunction with the loading, and the program that does the dual job is called a linking loader. For microcomputers, the linker tends to be tied more closely to the compiler than to the operating system, and the compiler manufacturer distributes a linker with the compiler as part of the software package. This is the case for Microsoft's FORTRAN-77 compilers.

4.6 The Microsoft Compiler/Linker for the IBM PC

In Microsoft's implementation of the compiler/linker, the compilation task involves either two or three passes, i.e., the compilation "step" is in fact more than one step involving processing by two or more programs, all of which constitute the compiler. The two (or three, depending on some user options, such as code optimization) passes are:

Pass 1 Translates the source program, stored in the file **PROG.FOR** for example, into an intermediate form, consisting of some temporary files that will be used by later passes of the compiler. It also produces a program-listing file, whose name has the extension **.LST** (**PROG.LST**, for example), which is important because it identifies and explains, as far as possible, any syntactical (language) errors in the source program.

<ins>Pass 2</ins> Takes the intermediate files produced in Pass 1 and generates the relocatable object code. The output from Pass 2 is an object program (or programs) that is (are) stored in a file whose name has the extension **.OBJ** (**PROG.OBJ**, for example). Also deletes the intermediate files from Pass 1, and generates more intermediate files that may be needed by Pass 3.

<ins>Pass 3</ins> Optimizes the object code, improving the efficiency of calculation of certain expressions, particularly if they have elements in common. Can also produce an optional listing of the object program (in hexadecimal code) in a file with the name extension **.COD**, or of an assembly language version of the program in a file with the name extension **.ASM**, and/or of a file containing detailed information about all of the programs in an object module in a file with name extension **.MAP**. Since very few programmers need such listings, we will ignore this latter feature.

The <ins>linker</ins> then takes the individual relocatable object programs produced by the compiler, stored in the file (or files) with extension **.OBJ**, links it (them) with the necessary library programs (present in one or more library files with the name extension **.LIB**), reassigns memory addresses, and generates a final object module that can subsequently be loaded into the fast memory of the IBM/PC and executed. The output from the linker is in a file whose name has the extension **.EXE** (**PROG.EXE**, for example).

As supplied by Microsoft, the compiler files are stored on six double-sided diskettes. Although it is possible use floppy disk versions of the compiler and linker on a dual-floppy drive IBM PC, it is a very tedious process. In essentially all cases, one wants to have the compiler/linker files stored on the hard disk of a PC-XT or PC-AT, or for networked machines to have the files stored on the hard disk of the network file server(s). The most important of these files are listed in Table 4.1.

Table 4.1 Important Files in the MS FORTRAN-77 Compiler

File Name	Size (bytes)	Purpose
F1.EXE	150,343	Pass 1 of the compiler.
F1.ERR	15,776	Error messages for pass 1.
F2.EXE	185,117	Pass 2 of the compiler.
F3.EXE	126,667	Pass 3 of the compiler.
F23.ERR	2,914	Error messages for pass 2 and 3.
FL.EXE	25,011	Compiler manager.
FL.HLP	1,584	Help messages for manager.
FL.ERR	1,817	Error messages for manager.
F3S.EXE	82,629	Alternate pass 3 when optimization is disabled.
LINK.EXE	50,531	Linker.
LLIBFORE.LIB	194,048	Library file that supports coprocessor (e.g., Intel 8087) if present; otherwise, floating-point calculations are done using software.

Depending on the options desired, there are several ways of calling the various programs that constitute the Microsoft compiler. The sequence of compilation activity is controlled by the "manager" program in the file **FL.EXE**, in response to input from the user.

Some of the more important files that are used or produced (some optionally) by the compiler are shown in Figure 4.2.

Table 4.2 Symbolic Names Used by Compiler

Name	Meaning
progfile.FOR	File containing FORTRAN source program.
progfile.LST	File that will contain the program listing and diagnostics generated by the compiler.
progfile.OBJ	File that will contain the relocatable object program generated by the compiler.
progfile.ASM	File that will contain the assembly language version of the program.
progfile.COD	File that will contain the machine language version of the program.
profile.MAP	File that will contain memory mapping information about the object programs generated.

Here **progfile.FOR** is your original FORTRAN source program file, e.g., **PROG.FOR**. The file **progfile.LST** will contain a listing of your FORTRAN program statements, any diagnostic error messages describing errors in FORTRAN syntax, and other information such as a listing of all FORTRAN variables, arrays, and functions and subroutines referenced in the source program and total memory space required. The other files are in general not of interest to ordinary mortals.

If your source program contains an error (usually caught by pass 1 of the compiler), then the compilation/linking process will be aborted, and you can examine the listing file to determine the nature of any errors found. VEDIT can then be used to make corrections in the file **progfile.FOR**, and the compilation/linking operations can be attempted again.

Table 4.3 lists the symbolic names of the most important files involved in the linking operation (when **LINK.EXE** is being executed).

Table 4.3 Symbolic Names Used in Linking

Name	Meaning
progfile.OBJ	File(s) that contains the object program(s) produced by the compiler.
library.LIB	File(s) containing object versions of the library programs. The major library file is LLIBFORE.LIB, but other library files may also be used under some circumstances, depending in particular on the availability of a coprocessor.
progfile.EXE	File that will contain the final executable object module. available.

The most important file from the user's standpoint is the final executable load module file **progfile.EXE**. The file **progfile.OBJ** may be of interest if other object program files, produced with a different execution of the compiler, are to be linked together.

4.7 FORTRAN-77 in the FEC Laboratories

In the Freshman Engineering Computing Laboratories, all of the compiler program files shown in Table 4.1 are stored permanently in a subdirectory named **\FORT77** on the hard disk (drive **D:**) of the Ethernet Network fileserver. [Note that the IBM PCs in the FEC laboratories are not equipped with Intel 8087 coprocessors, and the file **LLIBFORE.LIB** is the appropriate library file needed by the linker.]

To simplify student use of the compiler, there is a batch file (see Section 2.11.2) named **FORT77.BAT** on the Network fileserver. This batch file handles all of the details of the multiple-pass compilation/linking operations. During the processing, the compiler and linker files are downloaded from the Network disk into the main store of your IBM/PC. Intermediate files produced by the compiler are stored on your personal diskette. In most cases, the compiler programs erase these files, once they are no longer needed, so you will not see them in your diskette directory after processing. In addition, the relocatable object program file with the name extension **.OBJ** is erased after the linking process is complete, to save space on your diskette.

To compile and link a FORTRAN-77 program, do the following:

Step 1: Prepare your FORTRAN source program using VEDIT, and store it in a file named **progfile.FOR** on your diskette in Drive B: Here **progfile** is any legal DOS file name.

Step 2: Be sure that the master diskette is in Drive **A:** and your personal diskette is in Drive **B:**. Do not change the default drive from **A:**.

Step 3: Enter the DOS command:

```
A:\>FORT77 progfile<cr>
```

Note: Do NOT append the drive name **B:** to the name of your source program file, even though **A:** is the default drive; the batch file takes care of this apparent inconsistency.

Note: Do NOT include the filename extension **.FOR** in the DOS command line; doing so may cause your file **progfile.FOR** to be modified unexpectedly during the compilation process.

If there are no FORTRAN language errors, compilation and linking will be successful and when the DOS prompt **A>** shows again, the following files will be present on your personal diskette:

progfile.LST

progfile.EXE

If there is an error encountered during the compilation process, one or more intermediate files with name extension (**.TMP**) produced by the compiler may be present on your diskette. You can erase them, if you are short of space on your diskette.

The **.LST** file contains a listing of the program, some information about the size of the program, etc. You can print a copy of the program listing on the printer if you wish (using the DOS commands **COPY** or **PRINT**), or display the file on the screen using the **MORE** filter.

The file **progfile.EXE** contains the executable linked object module. To execute the object program, simply enter the DOS command:

```
A:\>B:progfile<cr>
```

If there are FORTRAN source language errors in your program, then the Microsoft compiler will abort execution of the later passes of the compiler and of the linker, and no **progfile.OBJ** or **progfile.EXE** files will be generated. However, the listing file **progfile.LST** will be present on your diskette. You should list the file on your printer or display it on the screen using the MORE filter, and note any diagnostics (error messages) produced by the compiler. Once you decide what the problem is, use VEDIT to correct the errors in the source program in **progfile.FOR** and attempt to compile the program again.

In what follows, we illustrate the above material by reproducing the output that appeared on the monitor during compilation of a short FORTRAN-77 program stored in the file **TRI1.FOR** (the program that appears in Example 1.1 in Chapter 1 of our companion text, FORTRAN-77 (with MTS and the IBM PC). For emphasis, we have put what we typed in **boldface**.

```
A:\>FORT77 TRI1<cr>

A:\>Echo Off
Microsoft (R) FORTRAN Optimizing Compiler Version 4.0
Copyright (C) Microsoft Corp 1987.  All rights reserved.

tril.FOR


Microsoft (R) Overlay Linker  Version 3.55
Copyright (C) Microsoft Corp 1984, 1985, 1986.  All rights
reserved.

Compilation/Linking OK ...
```

Compilation is now complete, and the final object module is in the executable file **TRI1.EXE** on the **B:** drive. It is executed simply by typing its name, with or without the extension **.EXE**:

```
A:\>B:TRI1<cr>


45.0, 10.0<cr>
      45.0000000      10.0000000       9.9999880 14.1421300
60.0, 10.0<cr>
      60.0000000      10.0000000      17.3204800 19.9999700
```

(At this stage, execution was terminated by entering **<Ctrl-Break>**, an attention interrupt. We could also have entered <Ctrl-Z), an end-of-data-file indicator.)

4.8 FORTRAN-77 in the CAEN Laboratories

The program files for the Microsoft subset FORTRAN compiler and linker are stored in a subdirectory named **FORT77** on the hard disk (Drive C:). The subdirectory is in the directory search path (see Section 2.21.3), so the compiler and linker can be accessed at any time, regardless of current directory and default drive assignments. The names of the files for the three passes of the compiler are **FOR1.EXE**, **PAS2.EXE** and **PAS3.EXE** in this case; however, the general sequencing of compilation/linking with the subset compiler is very similar to that used for the full-implementation compiler described in the previous sections.

There is a batch file named **FORT77.BAT** in the subdirectory **FORT77** that is very similar to the batch file named **FORT77.BAT** used to compile and execute FORTRAN-77 programs on the IBM/PCs in the FEC Laboratories described in Section 4.7. The procedure for using the batch file **FORT77.BAT** to compile and execute FORTRAN programs on the PC-XT and PC-AT computers in the CAEN Laboratories is described in the on-line documentation (present in a file in the **\HELP** subdirectory on the hard disk. The document is reproduced here verbatim. You can see a copy of this document on the screen by entering:

```
C:\USER>HELP MSFORT77<cr>
```

USING MICROSOFT FORTRAN 77 ON THE CAEN IBM PC-XTs and PC-ATs

General Information

MicroSoft FORTRAN for the IBM/PC running under PC/DOS is an implementation of the subset FORTRAN-77 language standard with many features from the full FORTRAN language standard included. Programs written in Microsoft FORTRAN are upward-compatible with the full standard, and will normally compile without change using the full language IBM FORTRAN-77 compiler ***FORTRANVS** on MTS (see Section 5.11.1).

Programs written in FORTRAN-66 that compile successfully using the ***FTN** and/or ***FORTRANH** compilers on MTS, will compile with little or no change using the Microsoft FORTRAN-77 compiler. If you have a FORTRAN-66 program stored in an MTS file and you would like to run it on the PC-XT or PC-AT, you can use the WINDOW program (catalogued under subdirectory C:\BIN) to telecopy the source program to a PC-XT or PC-AT file and then compile it using the Microsoft FORTRAN compiler.

Depending on the nature of the Input/Output statements in your program, it may be necessary to make a few minor changes in the source program when switching from MTS to the PC-XT or PC-AT and *vice-versa*.

A reference manual for Microsoft FORTRAN 77 is available from the CAEN Laboratory Monitor.

Compiling and Linking a MicroSoft Source Program

You should prepare your FORTRAN-77 source program using the VEDIT or EDIX visual editors for the PC-XT or PC-AT, catalogued under subdirectories C:\VED and C:\WPIX, respectively (or telecopied from MTS).

The program should be stored on your own floppy diskette present in Drive **A**: of the PC-XT or PC-AT, and the program file should have the file name extension **.FOR** . For example, the following would be acceptable file designations:

```
A:MYPROG.FOR
A:PROB1.FOR
A:SOURCE.FOR
```

The usual FORTRAN statement formats should be used for the source program file lines (columns 1-5 for a statement number, column 6 for continuation, columns 7-72 for the statement, a C in column 1 for a comment).

When you are ready to compile and execute your FORTRAN program, do the following:

1. Change the default drive to **A:** by entering

```
A:<cr>
```

where <cr> is the RETURN (ENTER) key.

2. Enter (boldface characters only)

```
A:\>FORT77 xxxxxxxx<cr>
```

where **xxxxxxxx** is the name of the source program file <u>without</u> the file name extension **.FOR**, e.g.:

```
A:\>FORT77 MYPROG<cr>
```

DOS will then process the commands in the batch file named **FORT77.BAT** that is catalogued under subdirectory

```
C:\FORT77
```

3. The first pass of the FORTRAN compiler, named **FOR1.EXE** will automatically be loaded from the C:\FORT77 subdirectory and begin to process your source program. Several files will be written onto your diskette in Drive **A:** including:

```
xxxxxxxx.LST
PASIBF.BIN
PASIBF.SYM
```

As before, **xxxxxxxx** is the name (without extension) of your original source program file (e.g., **MYPROG**). The first of these files contains a listing of the source program, plus diagnostic messages (if syntactical errors are detected). If an error message appears on the screen, then further compilation is aborted, and you can examine the file **xxxxxxxx.LST** for pertinent error messages, make necessary corrections in the source program file **xxxxxxxx.FOR**, and then return to step 2.

4. If no errors are encountered in step 3., then the second pass of the compiler from file C:\FORT77\PAS2.EXE is executed automatically. The files **PASIBF.BIN** and **PASIBF.SYM** will be erased, and three new files will be created.

```
xxxxxxxx.OBJ
PASIBF.TMP
```

PASIBF.OID

If errors are detected, compilation will stop. You can then examine the file **xxxxxxxx.LST** for diagnostic messages, make corrections in **xxxxxxxx.FOR**, and return to step 2.

If the compilation is successful, the object version of your program will be left in the file **xxxxxxxx.OBJ**. The files **PASIBF.TMP** and **PASIBF.OID** are temporary files used by the compiler for scratch space. They will be erased automatically if the compilation is successful.

5. If no errors are detected by the compiler, the MicroSoft linking program (from file C:\FORT77\\LINKER.EXE) will be loaded automatically and your object program in file **xxxxxxxx.OBJ** will be linked with any essential routines from the FORTRAN library (from file C:\FORT\\FORTRAN.L87). The linked object module will be stored in the file

xxxxxxxx.EXE

on your diskette in Drive **A**:.

Executing a Compiled and Linked Microsoft FORTRAN 77 Program

To run your linked object program in the file **xxxxxxxx.EXE**, simply enter:

```
A:\>xxxxxxxx<cr>
```

For example,

```
A:\>MYPROG<cr>
```

Depending on how you have specified the Input/Output files/ devices in the program, your program will then process your data and produce the output directly, or will prompt you for device assignments before proceeding with the calculations.

If you wish to practice with an existing demonstration program, simply copy the file C:\FORT77\\DEMO.FOR to your diskette in Drive **A**:, switch the default drive to **A**: and enter **DEMO** as the name of the source file in step 2.