Editor: Paul Oman
Computer Science Dept.
University of Idaho
Moscow, ID 83843
Compmail+: p.oman, CSnet: oman@oregon-state
Bitnet: paul@idui1

Serious Fortran for micros

Michael P. Johnson,
Oregon State University

In the early days of microcomputing I examined Fortran compilers, hoping my Fortran productivity might be improved as much as my word-processing productivity had been by that revolution.

I was disappointed to find that those early compilers did not support important features such as double-precision floating point, 32-bit integers, and large arrays. I concluded then that serious Fortran programming on microcomputers was not practical — but the new generation of microcomputer Fortran compilers is worth another look.

This review examines two Microsoft compilers, Fortran for the Macintosh (Version 2.2) and Fortran Optimizing Compiler for MS-DOS PCs (Version 4.0). These compilers appear to be developing into industry standards for the two major microcomputer processors, the Motorola 68000 series and the Intel 80000 series.

My review has convinced me that real Fortran is now available for microcomputers. Both perform well, are full implementations of the ANSI Fortran-77 standard, can handle industrial-strength program development, and are powerful enough for most current applications. These compilers are not toys — they are serious, professional tools.

Macintosh version. Developed by Absoft and marketed by Microsoft, this version includes a one-volume manual and two double-sided disks containing the compiler, debugger, library manager, linker, editor, resource compiler, include files, libraries, and demonstration programs.

The compiler requires 512K bytes of memory. While it is possible to use a single 800K-byte drive, two drives make an adequate system, and a hard disk reduces compile time by about 25 percent. I used a Macintosh SE with a hard disk and 1M-byte memory. The manual contains a user's guide, language reference section, utilities section, and toolbox section.

I began by trying to write a program heavy on toolbox calls. I coded the program in C, two brands of Pascal, and Fortran. It took less than two hours to code, key, and debug the program in each language — except Fortran. After four days I gave up on the Fortran version. The manual section on toolbox calls is brief, disorganized, and unclear, as were the programs that illustrated the use of the toolbox. I have not had similar problems with the toolbox using other compilers.

The editor, supplied by Apple Computer, is fast, simple, and quite adequate for coding programs. The user may have as many as four files open at once and may cut, copy, and paste among them using the mouse. A file of 11,000 lines loaded instantly; movement in the file was also very fast.

Two weaknesses with the editor are that it does not let you move by line number and it is not integrated with the compiler. An integrated compiler and editor — so compilation can be executed from the editor and errors detected by the compiler cause automatic placement back into the editor — is found in many other Macintosh compilers and greatly reduces development time.

With this compiler we have returned to the old mainframe edit/compile cycle. I should have expected that when I saw that the compiler icon was a deck of punched cards and the word "card" was used instead of "line" in much of the documentation. However, there is some integration in the form of a pull-down menu that lets you transfer to the editor, the linker, or the library manager from the compiler.

The library manager builds and manages libraries used by the linker. It builds sequentially searched libraries and lets you add, replace, and delete modules, including assembly-produced object modules. This utility and the linker were written in Microsoft Fortran and do not use the Macintosh interface. Library commands are typed interactively into a list and are not executed until you exit the library manager, which means that you can work on only one library each time you select the manager. The library manager also lists library directories and load maps.

The linker links object modules in object and library files to form an executable program. Linking is accomplished three ways: (1) If the source file includes all the user routines, linking is handled by the compiler. (2) With user subroutine libraries, the preferred method is to compile them separately and use the linker to produce the executable program. (3) Dynamic linking can be done at runtime. If a routine is called that is not linked, an object module of that name is loaded into memory and executed. If there is not enough room in memory the module

IN BRIEF

Microsoft Fortran Version 2.2 compiles Fortran source code for the Macintosh. It requires 512K bytes of memory, at least a single 800K-byte drive (a hard disk reduces compile time by about 25 percent). It costs $295.
RS 11

Microsoft Fortran Optimizing Compiler Version 4.0 compiles Fortran source code for MS-DOS PCs. It requires at least 320K bytes of memory (but 512K bytes is recommended), MS-DOS 2.0 or higher, and two double-sided disk drives. It is not practical to use this software without a hard disk. It costs $450.
RS 12

becomes an overlay. The runtime library is linked dynamically unless it is explicitly linked using the linker. Dynamic linking does not degrade performance if reasonable design care is taken to prevent disk thrashing. Dynamic linking is the default, so unresolved externals are not reported since they are assumed to be dynamically loaded segments.

The linker searches the libraries sequentially, making only one pass. If they satfisy an external symbol, the modules are conditionally linked. This means that when subroutine calls are more than one deep you must invoke the linker several times, use several link-library commands with the same library, or order the modules in the library very carefully. The linker can take commands from a command file.

The symbolic debugger has some very nice features: It lets the user step through a program, set and clear breakpoints with a click of the mouse, list breakpoints, examine the type and contents of variables, change the values of variables, search for lines or labels, check file contents and status, and execute segments of code. Moving around in the source code is easy; the debugger is as good or better as any I have seen with other compilers for the Macintosh.

Table 1.
Compile times for Fortran source code with more than 10,000 lines.

Computer	Compiler	Time (min:sec)
Macintosh SE	Microsoft Fortran (Mac Ver. 2.2)	2:53
Sperry PC IT (with 80286)	Microsoft Fortran (PC Ver. 4.0)	11:18*
HP 9000 (68010 processor)	HP/UX Fortran-77	11:15†
VAX 11/750	Unix Fortran-77	8:49

*With optimizing enabled.
† Symbol table overflow at this time.

Table 2.
Benchmarks using five translators on the Macintosh.

Compiler	Exec. time (sec) benchA	Exec. time (sec) benchB
Macintosh Pascal	1310.8	2098.2
Lightspeed Pascal	46.6	325.1
Turbo Pascal	22.1	108.0
Lightspeed C	34.3	48.7
Microsoft Fortran	45.8	38.5

Table 3.
Benchmarks using different Fortran compilers.

Machine	Compiler	benchB time (sec)	Sieve time (sec)
Macintosh SE	Microsoft Fortran (Mac Ver. 2.2)	38.5	5.9
Sperry PC IT (with 80286)	Microsoft Fortran (PC Ver. 4.0)	20.0	2.7
HP 9000 (68010 processor)	HP/UX Fortran-77	26.9	4.4
VAX 11/750	Unix Fortran-77	18.0	5.0

Figure 1 shows a screen from a debugging session.

The compiler includes many useful options, including Fortran-66 compilation. The language implementation is the full ANSI Fortran-77. There are many extensions, including Select Case/Case/End Case, Do While, While, End Do, Repeat, Cycle, Exit, Execute (for chaining), < > = (for relational operators), Type, and Accept. However, while all Fortran-77 features are implemented, this compiler is not as thorough as others in checking that the code adheres to all Fortran-77 restrictions.

This implementation does have some restrictions. Some are stated (a maximum of 1024 bytes per I/O record) and some are not (the number of I/O units that can be active). The floating-point representation conforms to the IEEE standards.

The toolbox is Pascal-based, which causes problems for Fortran subroutine calls. All toolbox calls must go through a Toolbx conversion subroutine (Call Toolbx(Moveto,10,10)). An include file is required to map parameters to hex addresses. This adds about 2100 lines of code, slowing compilation and execution of programs that use the toolbox extensively. This file may be condensed to save compile time at a high cost in execution time.

Benchmarks. To evaluate the compiler's performance, I timed the compilation by four Fortran compilers of a file with more than 10,000 lines of code. Table 1 lists the results. The times are for compilation only, except for the Macintosh version, which also includes linking. As the table shows, the Microsoft Fortran performs very well.

To compare execution times, I used two sets of benchmarks. The first compares different Macintosh programming systems; the second compares Fortran compilers on different machines.

In the first set, the benchA program checks the uniformity of random-number generation from the toolbox routine in ROM. It emphasizes array-index computation and toolbox calls. The benchB program is the same, except it uses a software random-number generator so the key factors are integer arithmetic and parameter passing. Table 2 lists the results.

The second set of benchmarks uses benchB and a sieve program that came as a demo with the compiler. Table 3 gives the results of these benchmarks. The Macintosh version compares favorably in execution time with those compilers that spend a considerable amount of time optimizing code.

Summary. As an instructional package, this compiler has several strong features: minimal hardware requirements, a quick and easy editor, a good debugger, and a fast compiler. This is an excellent environment for teaching Fortran. The drawbacks are the difficult toolbox interface and the cost ($295).

As a development system, I recommend it only if the project relies heavily on existing Fortran code and the toolbox is not used. For the development of Fortran programs targeted for other systems, the speed of compilation, the editor, and the debugger make this a good environment. The performance of the compiler at compilation is excellent and the code that is produced executes at adequate speeds. In fact, the execution times are surprisingly fast considering the high speed of the compiler.

MS-DOS version. Version 4.0 of the Microsoft Fortran Optimizing Compiler comes with a three-volume manual: user's guide, language reference, and debugger reference. The seven floppy disks contain the compiler, libraries, debugger, utilities, demonstration programs, a set-up program, and a tutorial for the debugger. Extras include a quick-reference guide, keyboard templates, and a reference to software and hardware available to support Fortran program development using this compiler (like the IMSL Library). There is no editor.

The compiler requires 320K bytes of memory (but Microsoft recommends 512K bytes), MS-DOS 2.0 or higher, and two double-sided disk drives. In my judgment it is not practical to use this software without a hard disk. The compiler will run on the IBM PC, XT, AT, and compatibles, and it supports the 8087 and 80287 coprocessors. I used a Sperry PC IT (an 80286 processor) with a 44M-byte hard disk and a 1024K-byte memory.

The manual is well-written, covers the material thoroughly, and is rich in examples. The indexes are complete and subjects are cross-referenced. Included are sections on compatibility with older versions and with other languages, the format of files, and data representation. The language reference clearly marks all non-ANSI extensions in blue print. But the manual is weak in giving limitations of the system (as in the Macintosh documentation, there is no information on how many I/O units can be active).

It is difficult and time-consuming to set up the system, so an extensive installation program is provided. This program presents a dialogue of choices with help options at every step. There are numerous choices in library type, memory model type, and mouse support — the installation is more complex than any I have done on minis or mainframes. The dialogue makes it easy but tedious. The installation program may also be used to build alternate support libraries for programs running in different floating-point environments (coprocessor, coprocessor emulator, or alternate math) and with different memory sizes.

The debugger is Microsoft's Codeview — the best debugger I have used. The display includes a command window, code window, and menu bar. Optional features include color and mouse control. The code may be at source or assembly level. You may also display register contents, status flags, the call stack, and variables in other windows. Memory contents can be displayed in a variety of formats. Program execution within the debugger may be done in several modes; trace, step, execute, restart, and go.

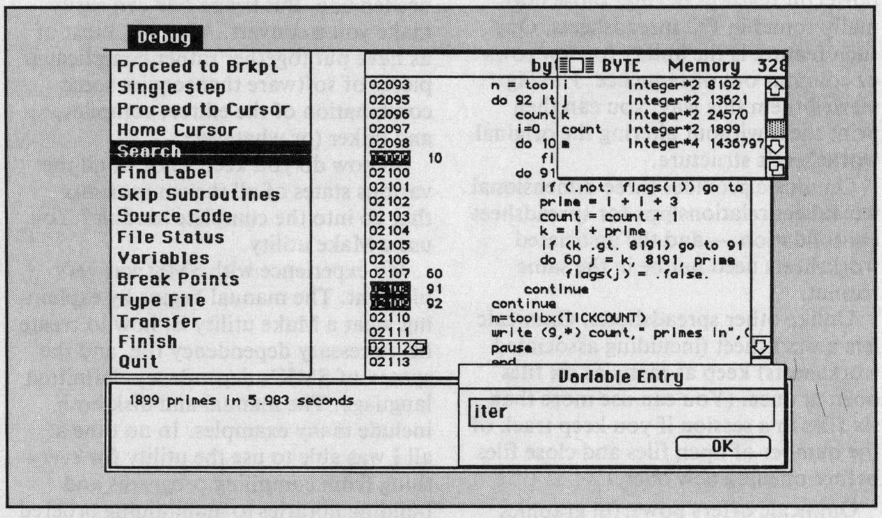
You can execute linker and library-manager commands with a command string, in response to a prompt, or with a response file. The library manager has the same options as the Macintosh version. The linker searches all object modules until all external references are satisfied. The linker will produce overlay segments in the traditional root-segment style. Linking may also be accomplished with the FL command, which compiles and links with one command. There is a Make facility that helps automate program maintenance. This facility may be used to help build complex libraries and link large, complex programs. The Make facility here is like the one on Unix systems, but the the manual with this one is much better.

The language implementation is a full ANSI Fortran-77. The IEEE standard for floating-point arithmetic is implemented, and the compiler meets the full GSA certification. About 134 confirmed bugs existed in Version 4.0, but a maintenance release (Version 4.01) has since corrected most of them. There are many extensions, including additional data types conforming to VAX and IBM extensions and extensive bit-twiddling functions. Missing are the additional control structures found in many other Fortran compilers (like While).

Benchmarks. I ran the same benchmarks for this compiler. The results in Table 1 shows that it performs slowly because it is an optimizing compiler. The payoff for this is that where optimization is possible there is a significant improvement in execution time, as Table 3 shows. The optimizing feature can be disabled, which results in a 30- to 35-percent improvement in compiler speed.

Summary. I would not recommend this compiler as an instructional environment. It costs $450 without an editor and the hardware requirements are considerable. The compiler is slow and the debugger difficult to learn. The environment is a complex one not easily handled by students.

However, for program development this system is as good as I have seen. There are real problems such as how large arrays and code must be handled, the lack of editor-compiler integration, and the compiler speed, but for the developer the support tools — a good manual, an outstanding debugger, and a Make facility — make this an attractive package.



```
i        Integer*2 8192
j        Integer*2 13621
k        Integer*2 24570
count    Integer*2 1899
m        Integer*4 1436797
```

```
02093       n = tool
02095       do 92 it
02096          count
02097          i=0
02098          do 10
02099   10        fl
02100          do 91
02101             if (.not. flags(i)) go to
02102             prime = i + i + 3
02103             count = count + 1
02104             k = i + prime
02105             if (k .gt. 8191) go to 91
02106             do 60 j = k, 8191, prime
02107   60           flags(j) = .false.
02108   91        continue
02109   92     continue
02110       m=toolbx(TICKCOUNT)
02111       write (9,*) count," primes in",(
02112       pause
02113       end
```

```
1899 primes in 5.983 seconds
```

```
iter
```

Figure 1. Macintosh screen of a debugging session.