Debugging Fortran g77 programs

By Gilberto E. Urroz, September 2002

Debugging is a generic term used to indicate the process of finding and eradicating errors in a computer program. In Fortran programs you may find three types of errors:

- **Syntax errors**: these errors will be detected right away by the compiler and pointed out to the programmer in the computer screen. To fix these errors simply correct the offending line of code ensuring that it follows proper Fortran syntax.

- **Runtime errors**: these errors occur after the program has been compiled and are typically associated with illegal operations of the data processed by the program, e.g., division by zero, logarithm of a negative number, etc. Runtime errors typically result in the suspension of the program execution. Errors of this nature can only be fixed by ensuring the quality of the data or by inserting traps in the code to avoid such data processing pitfalls.

- **Logical errors**: these errors are difficult to detect except by comparing the program results with known results calculated either by hand or by a different program. Logical errors constitute problems in the algorithm or algorithms contained in the program. Many a times, the only way to pinpoint a logical error in a program is to trace the operation of the program step by step. Many Fortran compilers are provided with a *debugger* program that provides for ways tracing program operation. Fortran g77 does not provide a debugger, therefore, we must seek ways to look into data values during program execution.

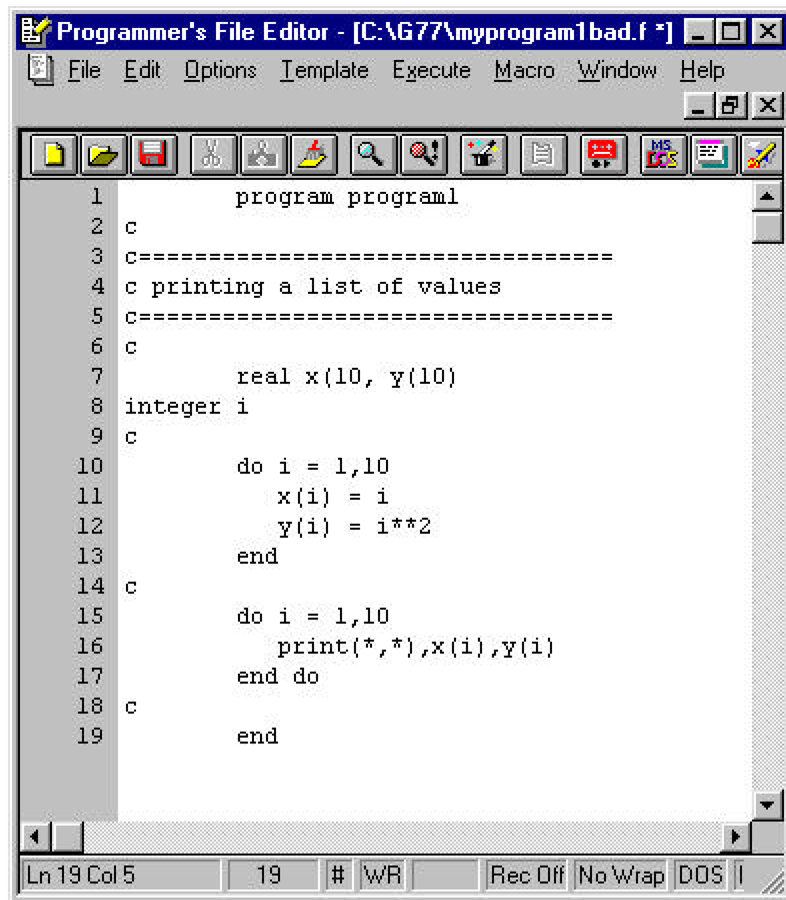
Fixing syntax errors

Whenever a Fortran source code that contains syntax errors is compiled, Fortran provides with a list of the location and type of the syntax errors. In Fortran g77 the report of syntax errors indicates the line number in the file where the error occur, prints a copy of the offending line pointing out the location of the error, and provides some information on the nature of the error.

Consider the following example in which the source code is given by the following file

```
      program program1
c
c==================================
c printing a list of values
c==================================
c
      real x(10, y(10)
      integer i
c
       do i = 1,10
          x(i) = i
          y(i) = i**2
        end
c
        do i = 1,10
           print(*,*),x(i),y(i)
        end do
c
        end
```

When viewed using the PFE editor with the line-number option activated, the file will look like this:



Here is a list of the syntax errors that I deliberately placed in this file:

- In line 7 there is a missing parenthesis. The line should read: `real x(10), y(10)`
- In line 8 the statement `integer i` occupies the region reserved for labels
- In line 13 the proper statement to end the *do* loop is `end do`
- In line 16 the *print* statement should read `print*,x(i),y(i)`

The file is saved under the name `c:\g77\myprogram1bad.f`. Compilation of the program, by using the command `g77 myprogram1bad.f`, produces the output screen shown in the figure below. Obviously, the screen shows only those lines reporting syntax errors that fit in the limited space of the MS DOS window. To facilitate capturing the output while compiling with Fortran g77, it is recommended that you download the program *etime*, available from:

http://kkourakis.tripod.com/etime.zip

The zip file thus downloaded contains a DOS program named *etime.exe* and an info file called *etime.txt*. Extract the file *etime.exe* to the directory *c:\g77*. When used in conjunction with *g77*, the application *etime.exe* sends the syntax error report to a file that can then be viewed with the text editor PFE. Here is the command used to produce a syntax error file for the example under consideration: `etime -2errfile g77 myprogram1bad.f`

The resulting file will be named *errfile* (no suffix) and will contain the following information:

```
myprogram1bad.f: In program `program1':
myprogram1bad.f:7:
           real x(10, y(10)
                      ^
Invalid declaration of or reference to symbol `y' at (^) [initially seen at
(^)]
myprogram1bad.f:8:
   integer i
   ^
Invalid first character at (^) [info -f g77 M LEX]
myprogram1bad.f:7:
           real x(10, y(10)
                           ^
Invalid form for type-declaration statement at (^)
myprogram1bad.f:11:
              x(i) = i
              ^
Invalid declaration of or reference to symbol `x' at (^) [initially seen at
(^)]
myprogram1bad.f:10:
           do i = 1,10
           1
myprogram1bad.f:13: (continued):
           end
           2
Statement at (2) invalid in context established by statement at (1)
myprogram1bad.f:10:
           do i = 1,10
              1
myprogram1bad.f:15: (continued):
           do i = 1,10
              2
Attempt to modify variable `i' at (2) while it serves as DO-loop iterator at
(1)
myprogram1bad.f:16:
              print(*,*),x(i),y(i)
                    ^
Missing first operand for binary operator at (^)
myprogram1bad.f:16:
              print(*,*),x(i),y(i)
                    ^
Expression at (^) has incorrect data type or rank for its context
myprogram1bad.f:16:
              print(*,*),x(i),y(i)
                      ^
Missing first operand for binary operator at (^)
myprogram1bad.f:16:
              print(*,*),x(i),y(i)
                      ^
Expression at (^) has incorrect data type or rank for its context
myprogram1bad.f:10:
           do i = 1,10
           1
myprogram1bad.f:19: (continued):
           end
           2
Statement at (2) invalid in context established by statement at (1)
myprogram1bad.f:10:
           do i = 1,10
           ^
End of source file before end of block started at (^)
5.16:\G77\BIN/g77.exe myprogram1bad.f (return code 1)
```

This listing suggests the existence of about 11 errors. In reality, the reason for all those syntax error messages is that an error in one line propagates to the following line. Thus, the missing parenthesis in line 7 is responsible for the following 4 errors:

```
myprogram1bad.f:7:
           real x(10, y(10)
                      ^
Invalid declaration of or reference to symbol `y' at (^) [initially seen at
(^)]
myprogram1bad.f:8:
   integer i
   ^
Invalid first character at (^) [info -f g77 M LEX]
myprogram1bad.f:7:
           real x(10, y(10)
                           ^
Invalid form for type-declaration statement at (^)
myprogram1bad.f:11:
              x(i) = i
              ^
Invalid declaration of or reference to symbol `x' at (^) [initially seen at
(^)]
```

Notice that each error report shows the source filename followed by a number, e.g., `myprogram1bad.f:7:`. This number is the line where the possible syntax error was detected. Also a caret (^) is included that points to the possition of the error in the line.

Correcting the syntax errors in a program is performed by looking at the syntax error report and trying to correct the errors at the location pinpointed. For example, the corrections necessary in this file are the following:

- Add closing parenthesis in line 7
- Move the statement in line 8 to the right by 7 spaces
- Replace `end` with `end do` in line 13
- Replace `print(*,*)` with `print *` in line 16

The resulting file should read:

```
       program program1
c
c===================================
c printing a list of values
c===================================
c
       real x(10), y(10)
        integer i
c
       do i = 1,10
           x(i) = i
           y(i) = i**2
        end do
c
        do i = 1,10
           print*,x(i),y(i)
        end do
c
        end
```

Compiling in the MS DOS windows with the command: `g77 myprogram1bad.f -o myprogram1` should produce no errors. The resulting executable program is named `myprogram1.exe`. Typing its name (`myprogram1`) in the MS DOS window will produce the following output:

```
C:\G77>myprogram1
  1.  1.
  2.  4.
  3.  9.
  4.  16.
  5.  25.
  6.  36.
  7.  49.
  8.  64.
  9.  81.
  10. 100.
```

To avoid syntax errors make sure to understand the rules of Fortran 77 and apply them properly while typing your source code. Review the source code throughly before attempting compilation.

Fixing runtime errors

A program without syntax errors compiles smoothly producing an executable file. However, when running the resulting application it is possible to find errors that will stop the program execution cold or that would produce extrange results. Here is a program that produces a division by zero:

```
      program myprogram2
c
      do j = -2,2
           y = 1/float(j)
            print *, j, y
       end do
c
      end
```

The compilation goes smoothly:

```
c:\G77>g77 myprogram2.f -o myprogram2.exe
```

However, when running the program we get the result 1.#INF (infinite) in the output:

```
c:\G77>myprogram2.exe
 -2 -0.5
 -1 -1.
 0  1.#INF
 1  1.
 2  0.5
```

Although execution did not stop, there is obviously an error involved and it should be avoided, by using, for example:

```
            program myprogram2
      c
            do j = -2,2
                 if (j.ne.0) then
                    y = 1/float(j)
                    print *, j, y
                 end if
             end do
      c
             end
```

This modification eliminates the infinite result, producing the following output:

```
c:\G77>g77 myprogram2.f -o myprogram2.exe

c:\G77>myprogram2
 -2 -0.5
 -1 -1.
 1  1.
 2  0.5
```

The "division by zero" runtime error detected in the previous example is pretty obvious. Other situations to watch out for runtime errors are:

- Square root of negative numbers (unless COMPLEX data is used)
- Logarithms of zero or negative numbers (unless COMPLEX data is used in the latter)
- Arcsine and arccosine functions with arguments less than -1 or larger than 1
- Large number of multiplications that may produce overflow (a number larger than the maximum absolute value allowed) or underflow (a number smaller than the smallest absolute value allowed)
- Opening a new file with a name that already exists.
- Opening a file that does not exist for input into a program.
- Programmer forgot to put an output statement for the program.
- Program does not end suggesting the process may be caught in an infinite loop.

Many runtime errors are relatively easy to detect since there will be either an abrupt end to the program or an unexpected output item. Others, such as the presence of an infinite loop, are not so obvious.

Fixing logical errors

Logical errors are errors in the algorithm itself. These may be difficult to detect because the program would have compiled and run without a hitch. However, the results may be nonsensical (e.g., a negative volume, or extremely large results, etc.). In that case, it will be necessary to first check the algorithm and see if there is an obvious mistake in the equations or in the algorithmic steps. Next, it may be necessary to check things like the following:

- Are you combining an integer data with a floating-point data in a manner that produces the wrong type of result?
- Is the index in a loop referred properly in the operations involved?
- Is the loop performing the required number of repetitions?
- Are the constants used properly typed, or is a decimal point missing?
- Are the values used appropriate for the units implied in an operation?
- Are the logical statements properly set up to check a condition in an IF statement?

- Is the order and type of arguments used in a subroutine or function call the same as in the subroutine or function definition?

Many of these situations can be resolved by looking carefully at the code and identifying the offending character(s) or statement(s).

When processing arrays of numbers, or when calculating complicated expressions, intermediate operations that may have been coded improperly may produce the wrong result. To trace the location where the problem is located you may have to place temporary *print* statements in your code. Use these statements to print the values of indices or specific variables as the program runs. After identifying the problem, remove the statements or simply comment them out (i.e., place a *c* in the first column of the lines corresponding to the temporary *print* statements).

You may also consider the use of the *pause* statement. The *pause* statement will temporarily stop the processing of a program until you type the command *go* (for Fortran g77). Thus, if you want to pinpoint the location of an error, you may want to place a *pause* statement near the line where you suspect the problem may be occurring. Before the *pause* statement you may want to place one or more temporary *print* statements to report on the values of the variables of interest at that point.

Modern debugging programs (simply referred to as a *debugger*) provide a number of features to step into a program as it runs, to stop the program temporarily, to look at the values of the variables at a stop point, etc. The aim of such programs is to facilitate the detection of logical errors, or, simply, typos, that may be producing the wrong results. Regretfully, Fortran g77 does not provide us with a debbuger (Can't ask much from a free compiler). Thus, debugging with Fortran g77 may require the use of lots of temporary *print* statements and *pause* statements in your program until you find the logical error(s).

Debugging for logical errors is an art similar to detective work. You have a few clues about the logical error and you need to tie them up to find where the problem is located. There are no rules-of-thumb in debugging for logical errors, you learn the art with practice.