COMPUTING TEACHING WITH FORTRAN 90

IAN FURZER
University of Sydney
Sydney, New South Wales, Australia 2006

Fortran 77 is taught in chemical engineering departments throughout the world and is the center of all scientific and engineering computing. Fortran was developed by John Backus of IBM and has passed through a number of stages including Fortran 66 and in 1978 to the then-new standard Fortran 77. ANSI and the International Standards Organisation (ISO) began work on a new standard for completion in 1982 and in 1991 introduced ISO/IEC 1539: 1991. This new standard is Fortran 90.

What teaching changes will be involved for computing with Fortran 90? This question will undoubtedly be of primary concern to chemical engineering departments during the next few years. The first element to be considered is that everything written in Fortran 77 will be fully compatible with Fortran 90. It will be possible to make no teaching changes and to simply call on a new compiler, f90. But a department that follows this policy will miss out on all of the advantages developed over a decade of work by experts in Fortran compilers. The advantages make use of the best features of other languages so that a robust and reliable software code can be written.

The intent of this paper is not to list all of the Fortran 90 features but instead to simply introduce it and give the reader an idea of the nature of the new programs. Most of these new programs will not look like Fortran 77 programs. The selection of the programs presented in this paper will demonstrate what Fortran 90 is all about.

One of the introductory topics discussed in the teaching of Fortran 77 is the requirement that Fortran statements lie between columns 7 and 72. Comments start with a C in column 1 and continuation lines with a character such as + in column 6, with statement numbers in columns 1 to 5. This is called fixed format Fortran and can be compiled with one option of the Fortran 90 compiler. But this fixed format is now considered obsolete and instead, programs can now be written in free format. This makes teaching much easier.

With free format, Fortran 90 programs can start in column 1 or any column through 132. There is no need for indentation at the start of a Fortran line, although this is often done for readability. Comments begin with an exclamation mark (`!`) which can be in any column. Comments can be added after a Fortran statement by `!`, followed by the comments. Continuation of a long Fortran 90 statement is performed by adding an ampersand (`&`) to start the next continuation line, and then an `&` to start the next continuation line. This change to a free form will be the first teaching change for Fortran 90.

Obviously, a free format form of Fortran will not compile on a f77 compiler, so this begins the use of added Fortran 90 features that require the f90 compiler. It should be noted that the word obsolete was used above—a number of Fortran 77 statements are not recommended as they are considered obsolete. This recommendation leads to robust and reliable code.

Teaching Fortran 77 led to difficult statements like COMMON and BLOCK COMMON which were useful in transferring information from a main program to subroutines, or from subroutine to subroutine. But neither of the COMMON forms are recommended in Fortran 90. How can a Fortran 90 pro-

***Ian Furzer** has been a faculty member in the Department of Chemical Engineering at the University of Sydney for over twenty-five years. He has extensive teaching and research interests that include computing, process simulation, and chemical engineering plant design. He is the author of over eighty research publications and the textbook,* Distillation for University Students.

gram operate without a COMMON statement? It has a new feature, called the module. It removes any doubts on program reliability that originated in the COMMON statement.

Teaching changes may be progressive from giving the full COMMON details in a Fortran 77 course to giving no details at all in a Fortran 90 course. The object of this teaching change concerns thinking about robust and reliable code while writing the code.

Other Fortran 77 statements that have become obsolete include DOUBLE PRECISION, computed GO TO, and arithmetic IF statements, to mention a few. Teachers of computing in chemical engineering who are aware of these statements will need to know the new replacements statements in Fortran 90.

A better approach to teaching Fortran 90 is to present it as a new language with a wide range of new features and statements. It covers a wider range of features than Fortran 77, introducing structures, pointers, arrays, and procedures, and can process character strings and bits of information. Its vocabulary is well defined and includes what at first glance looks like unusual expressions, such as "structure constructor." (Further details on Fortran 90 can be found in Metcalf and Reid.[1]) The following are a few simple free format Fortran 90 programs, presented to provide a flavor of Fortran 90 to the reader.

FORTRAN 90 EXAMPLES

Example 1

Example 1 is shown below and includes the program and end program statements. Note how the program name, example 1, assists in locating the full extent of the main program.

```
program example_1
print *, 'This is the output from example_1'
stop
end program example_1
```

The output from this program is given by

```
This is the output from example_1
```

Example 2

Example 2 is a Fortran 90 program that no longer looks like a Fortran 77 program. Its function is to demonstrate some features, including high precision calculations and derived data types. Real variables can be calculated with at least 10 decimal places by the definition of an integer parameter r10, shown on line 2. Real variables such as sum, c, and d in the program carry the "r10" notation, giving

What teaching changes will be involved for computing with Fortran 90? . . . The first element to be considered is that everything written in Fortran 77 will be fully compatible with Fortran 90.

these variables 10-figure precision. It is possible to define other data types that have components. These structures can be complex, but a simple example is given of a type called university. It has three components with data types: character, real, and integer. It is bounded by the end type university statement. The variables nsw and queens are defined to be of type university and each will have three components.

```
program example_2
integer, parameter    :: r10=selected_real_kind(10)
real (kind=r10) sum, total, c, d, e
integer difference
  type university
      character (len=30) name
      real engineering_depts
      integer academic_numbers
  end type university
      type (university) nsw
      type (university) queens
! simple examples follow
   sum= 1.23456789_r10
   total=sum**2
!
   c=1.0_r10
   d=3.0_r10
   e=c/d
!
write (*,*)'Kind=R10 Variables SUM=', sum
write (*,*)'TOTAL=', total
write (*,*)'    E=', e
!
  nsw = university ('University of NSW', 8, 150)
  queens = university ('University of Queensland', 6, 120)
  difference = nsw%academic_numbers-queens%academic_numbers
!
write (*,*)'Difference in Academic Numbers=', difference
!
  stop
end program example_2
```

The output from Example 2 is given by

```
Kind=R10 Variables  SUM=  1.2345678899999999
TOTAL=  1.5241578750190519
    E=  0.3333333333333333
Difference in Academic Numbers= 30
```

The precision of the output should be carefully noted. Example 2 continues with the structure constructors for the type university variables nsw and queens. This input of information includes the "name" of the university, the number of "engineering_depts," and the "academic_numbers" in all departments. The difference between the "academic_numbers" at nsw and queens is given by the variable difference. The output shown above is of course only as accurate as the information in the structure constructors.

Example 3

Example 3 is similar to Example 2 but demonstrates the use of procedures. The program consists of three parts: the main program "example_3," a module called type maker, and a subroutine called calculation. Each part may be separately compiled and the object code linked for execution. The main program contains the statements

```
use type_maker
external calculation
call calculation
call write_result
```

The main program listing is

```
program example_3
  use type_maker
  external calculation
  integer, parameter :: r10=selected_real_kind(10)
  real (kind=r10) sum, total, c, d, e
  integer difference
    type (university) nsw
    type (university) queens
!
!simple examples follow
!
  sum=1.23456789_r10
  c=1.0_r10
  d=3.0_r10
    call calculation ( sum, total, c,d,e)
!
  write(*,*)'Kind=R10 Variables SUM=', sum
  write(*,*)'TOTAL=', total
  write(*,*)'     E=', e
    nsw= university('University of NSW', 8, 150)
    queens=university('University of Queensland', 6, 120)
    difference=nsw%academic_numbers-queens%acadedmic_numbers
    call write_result (difference)
  stop
end program example_3
```

Modules are a very important feature of Fortran 90. They can be used by the main program and subroutines to access information (such as the definition of the type university) that more than one of them needs.

```
MODULE TYPE_MAKER
  type university
    character (len=30) name
    real engineering_depts
    integer academic_numbers
  end type university
contains
  subroutine write_result (number)
  integer, intent (in) :: number
  write (*,*)'Difference in Academic Numbers=', number
  end subroutine write_result
END MODULE TYPE_MAKER
```

Modules are procedures that can also contain subprograms such as the subroutine write result. Note that the module is named type maker and ends with end module type maker. An example of an external subroutine is given by the subroutine calculation.

```
subroutine calculation (a, b, c, d, e)
integer , parameter :: r10=selected_real_kind(10)
real ( kind=r10), intent (in) :: a,c,d
real ( kind=r10), intent (out):: b,e
     b=a**2
     e=c/d
end subroutine calculation
```

The subroutine arguments a, b, c, d, and e are of kind, r10, that is of at least 10-figure precision. Fortran 90 also uses the attributes, intent (in) and intent (out), to be used to specify the input and output arguments to the subroutine. The output from Example 3 is identical with the output from Example 2.

Example 4

Example 4 shows some of the powerful features of Fortran 90: pointers, targets, automatic arrays, do loops, and matrix multiplication.

```
program example_4
  real, pointer  :: finger
  real, target   :: a,b
  real, dimension ( : , : ), allocatable :: matrix_a, matrix_b
  real, dimension ( : , : ), allocatable :: matrix_c
  integer n
  a=1.0 ; b=2.0 ; n=5 !n could be entered by read (*,*)n
  allocate ( matrix_a(n , n), matrix_b(n , n) , matrix_c(n , n) )
j_loop :      do j=1, n
k_loop :      do k=1, n
     matrix_a(j,k)=real(j)*real(k)
     matrix_b(j,k)=matrix a(j,k)
       if(matrix_b(j,k) <= 10.0 ) cycle
     matrix_b(j,k)=matrix_b(j,k) + sqrt(0.5)
    end do k_loop
    end do j_loop
  matrix_c=matmul(matrix_a, matrix_b)
    finger=>b
      if(n==1) finger => a
  write(*,*) 'Example_4   Finger=', finger
  stop
end program example_4
```

A real variable, "finger," is given pointer attributes by its definition. A pointer points to a target and real variables, "a,b," are given target attributes. They are a new and powerful feature of Fortran 90, particularly useful in operating on linked lists such as an adapted refined grid. Their use in engineering may not immediately appear to be obvious, but a simple example is shown as part of Example 4.

Arrays in Fortran 90 can have fixed bounds such as a(10), but only a section of the array can be used with the colon (:) notation, such as "a(5 : 10)." A two-dimensional array with initially unspecified lower and upper bounds is given by matrix a (: , :). These bounds can be allocated during execution, making for greater flexibility. Example 4 shows three arrays: matrix_a, matrix_b, and matrix_c, with the attribute allocatable. An integer n that is given the value 5 in Example 4 could have been read in and could take on a wide range of integer values. The allocate statement then allocates the required amount of memory for these arrays.

Do loops are considerably different in Fortran 90 and may include no labels. They start with a do statement and end with an end do statement. Each loop may be given a name, such as "j_loop," which assists in locating the limits of a particular loop. There are no statement numbers in the recommended form of the do loop, the final statement being the end do statement. The cycle instruction permits a direct jump to the end do statement and the exit statement permits a direct exit from the loop.

Fortran 90 permits direct matrix multiplication through the "matmul" statement. Other matrix operations are standard in Fortran 90.

The statement

```
finger => b
```

shows the pointer finger is pointing at the target b. The next statement contains the logical equal comparator, and if it is satisfied

```
finger => a
```

The output from Example 4 is given by

```
Example_4        Finger=     2.0000000
```

SUPERCOMPUTERS

Fortran 90 could well be the new world standard in computing until the year 2000. Fortran 90 compilers can exploit the advanced architecture of parallel processors or supercomputers. It might be expected that desktop supercomputers will lead to considerable advances in engineering, particularly in finite element methods and computational fluid mechanics. It might also be expected that the obsolete and archaic features of some parts of Fortran 77 (such as the arithemetic IF statements, some DO statements, and the H edit descriptor) will be removed in later versions of Fortran 90. Fortran 90 statements can begin in column 1, thus removing the obsolete card image concept in Fortran 77. One of the important advances of Fortran 90 is the availability of instructions that give a good methodology in program design. This can lead to both robustness and an error-free code, which can be fully exploited on supercomputers.

CONCLUSIONS

Engineers will need to spend some time learning the new features of Fortran 90 if they wish to undergo the conversion from Fortran 77. A program written in Fortran 90 may not look like a Fortran 77 program because of the many new features of Fortran 90, and many obsolete features of Fortran 77 should no longer be used. The only Fortran 90 compiler available from NAG provides reasonable error messages during compiling, which is an improvement over Fortran 77. In some cases the error messages even identify a line number and print a part of the statement that contains the error. Also, the compiler lists undefined variables. Error messages during execution are good and, for example, provide the dimensions of matrices if they do not compute.

One of the most important advantages of Fortran 90 will be the portability of code. A large number of software products such as mathematical subroutines and graphical packages will be rewritten in Fortran 90 to provide good interfaces.

This article makes no attempt to list all the statements and features of Fortran 90, as it is an extensive and powerful new language. There can be no doubt that it will have an important impact on the engineering profession for a number of years to come.

ACKNOWLEDGMENT

Comments by John Reid of the Rutherford Lab, England, were most welcome.

REFERENCES

1. Metcalf, M., and J. Reid, *"Fortran 90 Explained,"* Oxford University Press (1990) ❐

REVIEW: *Plastics Recycling*

Continued from page 199.

rapidly changing field, however, some of the information in this book is already dated. Advances in recycling have produced better processes which allow recycled plastics with specifications (much like virgin plastics) to be produced, particularly by companies like Union Carbide, Dow, Mobil, Quantum, and Waste Alternatives. These improvements in processes and products have inevitably led to the demise of several small companies—especially in the plastic lumber area. Many large companies, however, (such as Mobil and Amoco) have entered the field with more efficient processes and better quality control.

A more recent book, published by the American Chemical Society, ACS Symposium Series 513, *Emerging Technologies in Plastics Recycling,* (1992), is also becoming dated, but has significantly more scientific data. Clearly, plastics recycling is a dynamic area of research and business, and continued developments are in progress. This book provides an excellent starting point for those who are interested in plastics recycling. ❐