FORTRAN 77
Chapter 5

Satish Chandra

satish0402@gmail.com

Subprograms

- When a programs is more than a few hundred lines long, it gets hard to follow.
- Fortran codes that solve real research problems often have tens of thousands of lines.
- The only way to handle such big codes, is to use a modular approach and split the program into many separate smaller units called subprograms.

Subprograms

- A subprogram is a (small) piece of code that solves a well defined sub-problem.
- In a large program, one often has to solve the same sub-problems with many different data.
- Instead of replicating code, these tasks should be solved by subprograms .
- The same subprogram can be invoked many times with different input data.

Subprograms

- Sub programs are of two types, they are

1. Function - a procedure that returns only a single value that is used in the evaluation of an expression.
2. Subroutine - a procedure that can return multiple results through calling arguments.

- Functions and Subroutines are subprograms in a sense complete programs that can be compiled independently. They are accessed with the help of a main program.

Subprograms

- They are used in the following contexts
  1. To reduce the length of a program, by avoiding repetition of the same set of instructions.
  2. To reduce the task of writing program of a complex algorithm, by dividing it into pieces

Function Subprograms

- A function is a procedure whose result is a single number, logical value, character string or array.
- This result can be used to form a FORTRAN expression.
- The expression may be on the right side of an assignment statement.
- There are two types of functions, built-in and user-defined.

Built-in Functions

- Built-in functions are those functions built into a FORTRAN language.
- Built in programs like sqrt, cos, abs etc which are available in the FORTRAN Library, can be accessed in any program by mentioning the name with necessary arguments.

Built-in Functions

There are many built-in functions in Fortran 77.

abs	absolute value
min	minimum value
max	maximum value
mod	remainder
sqrt	square root
exp	exponential (natural)
log	logarithm (natural)
log10	logarithm base 10

Built-in Functions

sin	sine
cos	cosine
tan	tangent
asin	arcsine
acos	arccosine
atan	arctangent
sinh	hyperbolic sine
cos h	hyperbolic cosine
tanh	hyperbolic tangent

User-defined Functions

- User-Defined functions are functions defined by programmers (not really users) to meet a specific need not addressed by the standard built-in functions.
- We write user defined functions in two ways.
  1. Statement functions
  2. External functions

User-defined statement function

- If there is only one line of *arithmetic* or *logical* instruction in a function program it is written in the following way,

  f(x) = x**3 - sin(x) + 8

  sum(a,b,c) = a + b + c
- Here x or a,b,c are dummy variable name. Later in the program if the function is to be used one may change. Example

  grand_total = last_total + sum (x, y, z)

User-defined statement function

- A statement function definition must appear in the same program which uses it.
- It must be placed at the beginning of the program before appearance of any executable statement.
- The statement function definition is similar to a declaration statement.
- The number, order and mode of the actual arguments used in the function and the dummy arguments match.

User-defined statement function

```
c     program to use statement function
      integer no_yrs, term
      real interest, amount
      interest (rate, bal, years)=bal*rate*years/100.
      read(*,*) amount, term, r
      yrs = 1
      do 100 no_yrs = 1, term
              amount=amount + interest(r, amount, yrs)
      print(*,50) no_yrs, amount
50    format (1x, 'balance after', i3, 'years = Rs.', f8.2)
100 continue
      stop
      end
```


User-defined external functions

- In case a function program is a sequence of instructions, then the general syntax of user-defined external functions is;

  type function name(*list of arguments*)

  declaration statements

  executable statements

  name = *the value to be returned*

  return

  end

User-defined external functions

- We see that the structure of a user-defined external functions closely resembles that of the main program. The main differences are:
  - ✓ Functions have a type. This type must also be declared in the calling program.
  - ✓ The functions are terminated by the return statement instead of stop.
  - ✓ The arguments in the subroutine are to be declared
  - ✓ The return value should be stored in a argument with the same name as the function.

Example 1

```
c     program using user-defined external function
      real a, b, c, d
      real cube_root
      read(*,*) b, c, d
            a = b*cube_root (c) +d
      write(*,*) a
      stop
      end
```


Example 1 continued

```
c     subprogram to calculate cube root
      real function cube_root (x)
      real x, log_x
      log_x = log(x)
      cube_root = exp (log_x/3.0)
      return
      end
```


Example 2

```
c     main program to find the value of e
      real sum
      write(*,*)' the number of terms'
      read(*,*) n
            sum=0
      do 10 i = 1, n
            sum=sum+1.0/fact(i)
10    continue
      write(*,20) sum
20    format(1x, 'the value of e is :', f16.10)
      stop
      end
```

Example 2 continued

```
c    subprogram to find the value of fact
     real function fact (k)
     real k, prod
          prod=1
     do 30 i =1, k
          prod=prod * i
30   continue
          fact=prod
     return
     end
```


Example 3

```
c    program to compute the annual rainfall in inches
     real r, t, sum
     integer m
     read (*,*) t
          sum = 0.0
     do 10 m = 1, 12
          sum = sum + r(m, t)
10   continue
     write (*,*) 'Annual rainfall is ', sum, 'inches'
     stop
     end
```


Example 3 continued

```
c    subprogram to compute the amount of rain r (m, t)
c    m is the month, and
c    t is a scalar parameter that depends on the location
     real function r(m, t)
     integer m
     real t
          r = 0.1*t * (m**2 + 14*m + 46)
          if (r .LT. 0) r = 0.0
     return
     end
```


Problems

1. Find the average and standard deviation of a set of numbers.
2. Evaluate a polynomial of degree n
3. Find the maximum of a set of numbers

Subroutines

- A Fortran function can essentially only return one value. Often we want to return two or more values (or sometimes none!). For this purpose we use the subroutine construct.
- Subroutines have no type and consequently should not (cannot) be declared in the calling program unit.
- The arguments in the subroutine are to be declared.

Subroutines

- The syntax of subroutine is as follows:

  subroutine name (*list of arguments*)

  declarations statement

  executable statements

  return

  end

- The subroutine is accessed in the main program by the statement

  call name (*list of arguments*)

Subroutines

- Difference between FUNTION and SUBROUTINE
  1. Function is accessed in the main program by mentioning its name, but for subroutine by the call statement
  2. Function can return only one value to the main program, but a subroutine can return more than one value.
  3. Function returns via its name but a subroutine returns through its arguments.

Example 1

```
c   the subroutine is to swap two integers.
    subroutine iswap (a, b)
    integer a, b
    integer tmp
        tmp = a
        a = b
        b = tmp
    return
    end
```


Example 1 continued

```
c   program to call the subroutine into main
    integer m, n
        m = 1
        n = 2
    call iswap (m, n)
    write(*,*) m, n
    stop
    end
```


Example 2

```
c   subroutine to transpose square matrix (nxn)
    subroutine trpose (mata, a)
    integer mata (20, 20)
    nmins1 = n – 1
    do 20 i = 1, nmins1
        iplus1 = i + 1
        do 10 j = iplus1, n
            temp = mata (i, j)
            mata (i, j) = mata (j, i)
            mata (j, i) = temp
10      continue
20  continue
    return
    end
```


Example 2 continued

```
c   program to call transpose matrix
C   the dimension of matsam and mata should be same
    integer matsam (20, 20)
    read(*,30) ((matsam (i, j), j=1, 5), i=1, 5)
30  format(5i2)
    call trpose (matsam, 5)
    write(*, 40) ((matsam (i, j), j=1, 5), i=1, 5)
40  Format (10x, 'Transpose matrix'/(1x, 5i6))
    stop
    end
```


Example 3

```
c subprogram to solve quadratic equation
  subroutine roots (a,b,c,xreal1,xreal2,ximag1,ximag2)
  if (a.EQ.0) then
    xreal1 = - c/b
    xreal2 = 0
    ximag1= 0
    ximag2=0
  return
  end if
```

Example 3 continued …

```
disc = b*b – 4*a*c
    if (disc.GE.0) then
    d = sqrt(disc)
        xreal1 = (- b + d)/(2.0*a)
        xreal2 = (- b + d)/(2.0*a)
        ximag1= 0
        ximag2=0
```


Example 3 continued …

```
    else
    d = sqrt(- disc)
        xreal1 = - b /(2.0*a)
        xreal2 = xreal2
        ximag1= d/(2.0*a)
        ximag2= - ximag1
    end if
  return
  end
```


Problems:

1. Find the product of two matrices.
2. Arrange a set of numbers in ascending order.

Chapter 5

ENDS