CITY UNIVERSITY
SCHOOL OF ENGINEERING AND MATHEMAICAL SCIENCES
COMMON PART I
FORTRAN LECTURE 4

ARRAYS, SUBPROGRAMS, DATA and COMMON

4.1 ARRAYS

4.1.1 DATA STORAGE

Write a program to calculate the average of a long list of numbers. We do not know how many numbers to expect, but we know if we type in an incorrect number type Fortran will give a run time error. We can trap this error ourselves and use it to terminate the input, also to count how many numbers we have types in.

```
      PROGRAM PROG14
!---------------------------------
! read in a long list of numbers and calculate the average
!---------------------------------
!
      WRITE (*,*) ' Please type in a list of real numbers. To end input type a letter'
      N=0
      SUM=0.0
      AVERAGE=0.0
101   READ (*,1000, ERR=100) X ! trap format errors ourselves
      N=N+1
      SUM=SUM+X
      GOTO 101                 ! loop until format error
!
!---------------------------------
! user ended input
!with non numeric character
!---------------------------------
!
100  IF ( N.GT.0 )AVERAGE=SUM/N
     WRITE (*,*) 'Sum of ', N ,' numbers is ',SUM, 'average is ',AVERAGE
     STOP
1000 FORMAT ( F12.3)
     END
```

Now if we want to calculate the number above average there is a problem, since the previous program did not save the input numbers. The computer has no record of what the user typed. We can use some memory to store the long list of numbers. This is called an *ARRAY*. We can store a matrix with a two dimensional array. Think of an array as a matrix with only one column. Just as a

matrix needs some subscripts to identify each number and array also needs a subscript. We can write in Fortran:-

```
      PROGRAM PROG15
      DIMENSION X(100)
!------------------
! read in a long list of numbers and calculate the average
!-----------------
       WRITE (*,*) ' Please type in a list of real numbers. To end input type a letter'
       N=0
       SUM=0.0
       AVERAGE=0.0
       I=1                    ! ARRAY SUBSCRIPT
101    READ (*,1000, ERR=100) X(I) ! trap format errors ourselves
       N=N+1
       SUM=SUM+X(I)
       I=I+1
       IF (I.GT.100) GO TO 102
       GOTO 101                    ! loop until format error
!------------------------
! ran out of array memory
!------------------------
    102 WRITE (*,*) 'ONLY ROOM FOR 100 NUMBERS'
!!
!------------------------
! user ended input
!with non numeric character
!------------------------
 100  IF ( N.GT.0 )AVERAGE=SUM/N
      WRITE (*,*) 'Sum of ', N ,' numbers is ',SUM, 'average is ',AVERAGE
 !----------------------------
 !       now we can go back and recheck the data
 !----------------------------
     NABOVE=0
     IF (N.LE.0) GO TO 106
     DO 105 II=1,N
     IF(X(II).GT.AVERAGE) NABOVE=NABOVE+1
  105 CONTINUE
  106 Write (*,*) 'number above average-=',NABOVE
       STOP
 1000 FORMAT ( F12.3)
       END
```

4.1.2 A simple sort program

Once the data is stored in memory we can write a sort program to put the data in numerical order. A simple method is called a BUBLE sort. We compare every number with every other number and change the position in the array if they are not in order. Eventually the biggest number will rise to the top. Then we look for the second largest and so on. This is a poor sorting algorithm, but good for teaching Fortran.

```
        PROGRAM PROG16
        DIMENSION X(100)
!---------------------------
! read in a long list of numbers and calculate the average
!-------------------------
         WRITE (*,*) ' Please type in a list of real numbers. To end input type a letter'
         N=0
         SUM=0.0
         AVERAGE=0.0
        I=1                    ! ARRAY SUBSCRIPT
101      READ (*,1000, ERR=100) X(I) ! trap format errors ourselves
         N=N+1
        SUM=SUM+X(I)
          I=I+1
        IF (I.GT.100) GO TO 102
        GOTO 101                    ! loop until format error
!-----------------------------
! ran out of array memory
!-----------------------------
    102 WRITE (*,*) 'ONLY ROOM FOR 100 NUMBERS'
!-----------------------------
! user ended input
!with non numeric charcter
!-----------------------------
 100  IF ( N.GT.0 )AVERAGE=SUM/N
      WRITE (*,*) 'Sum of ', N ,' numbers is ',SUM, 'average is ',AVERAGE
 !-------------------------------
 !        we can go back and recheck the data
 !-------------------------------
       NABOVE=0
      IF (N.LE.0) GO TO 106
      DO 105 II=1,N
      IF(X(II).GT.AVERAGE) NABOVE=NABOVE+1
  105 CONTINUE
  106 Write (*,*) 'number above average-=',NABOVE
 ! -------------------------------
 !sort into numerical order
 !-------------------------------
```

```
!--------------------
! nested do loops compare all the numbers in our list with each other
! outer loop starts with II =1, then inner loop starts with JJ=II
!---------
!
       IF (N.EQ.0) STOP
      DO 110 II=1,N
      DO 110 JJ=II,N
      IF (X(II).GE.X(JJ)) GO TO 110
      TEMP=X(II)
      X(II)=X(JJ)
      X(JJ)=TEMP
110    CONTINUE

!------------------
!print out answers
!------------------
     DO 111 JJ=1,N
     WRITE (*,1002) X(JJ)
111 CONTINUE
       STOP
1000 FORMAT ( F12.3)
 1002 FORMAT (1X,F12.3)
       END
```

4.2 SUBPROGRAMS

Engineering programs are often many thousands of lines long. The use of subroutines and user-defined functions make such programs easier to write and understand. Subprograms can also decrease the size of a program as the same mathematical expressions or operations frequently appear at several different locations in a program.

4.2.1 USER-DEFINED FUNCTIONS

User-defined functions are called by the (main) program in a manner similar to that used for intrinsic functions. The structure of the user-defined function is shown in the example below.

```
      Program prog17
      write(*,*)'Radius of cylinder?'
      read(*,*)radius
      write(*,*)'Height of cylinder?'
      read(*,*)height
!
      vol = volume(radius,height)
!
      write(*,100)vol
  100 format(' ','Volume = ',e10.3)
      stop
      end
!
!---------------
      function volume(r,h)
      pi = 4*atan(1.0)
      base = pi*r**2
      volume = base*h
      return
      end
```

Notes:

- The function is defined in a self-contained, separate, program unit (or sub-program) that starts with the word "function" followed by the name of the function (volume in the above), which is followed by an argument list.
- There must be a one-to-one correspondence between the number and type (i.e., real or integer) of variables in the calling argument list and the function argument list.
- The names of the variables in the function argument list are local to the function. That is, "r" and "h" are only known within the function. All other variables appearing in the function are also only known within the function. For example, in the above, the main program would not know the value of pi.
- The result of the calculation performed in the function is returned to the main program through the function name.
- The first letter of the function name defines the type of the function (i.e., real or integer).
- The function must contain a "return" and an "end" statement.

4.2.2 Single-line user-defined functions

A simplified user-defined function is possible if the required expression fits on one line. The structure of this type of user-defined function is shown below

```
!
      volume(r,h) = h*pi*r**2
!
      pi = 4*atan(1.0)
      write(*,*)'Radius of cylinder?'
      read(*,*)radius
      write(*,*)'Height of cylinder?'
      read(*,*)height
      vol = volume(radius,height)
      write(*,100)vol
  100 format(' ','Volume = ',e10.3)
      stop
      end
```

Notes:

- The definition of the function has to be placed above all executable or declaration statements.
- Not all parameters used in the function need to be passed through the function argument list (e.g., pi in the above).

4.2.3 SUBROUTINES

Subroutines are in many ways similar to user-defined functions, but subroutines allow more than one variable to be returned to the main program. For example, in the program below, three variables, d, v, and a are returned.

```
      PROGRAM PROG18
       s0 = 0.0
       v0 = 98.1
       a0 = -9.81
       write(*,*)'     time      dist      vel       accel'
       do 10 i = 1,11
       t = 2.0*(i-1)
       ha0 = 0.5*a0
!
!-------------
       call trak(s0,v0,ha0,t,d,v,a)
!--------------
!
       write(*,100)t,d,v,a
    10 continue
   100 format(' ',4f10.3)
       stop
       end
!------------
!-------------

       subroutine trak(a,b,c,x,y,dy,dy2)
       y = a+b*x+c*x*x
       dy = b+2*c*x
       dy2 = 2*c
       return
       end
```

Notes:

- Unlike user-defined functions, results of the calculation performed in the subroutine (y,dy,dy2 in the above) are passed back to the main program via the arguments list.
- Control is passed to the subroutine by a "call" to the subroutine name.
- Control is passed back to the main program by the subroutine's "return" statement.
- As with functions, variables in the subroutine are local to the subroutine. However, if the variables that are nominally thought of as inputs (e.g., a, b, c, x in the above) are changed in the subroutine, then the values of the corresponding variables (e.g., s0, v0, ha0, t in the above) are changed in the calling program. That is, a variable in a subroutine argument list can be both an input and an output parameter.
- There must be a one-to-one correspondence between the number and type (i.e., real or integer) of arguments in the calling argument list and the function argument list.
- Subroutines can be called by other subroutines.
- Subroutines may contain read and write statements.

4.2.4 Three more subroutines

Program prog16 is getting a bit long. We can make it easier to understand by splitting it into three different subroutines. Each subroutine can be in a separate source file, compiled separately and then linked together at build time.

Try editing PROG16 as below. Type in a new Prog16 and save as a file PROG16.f95.

```
 ! type and store as prog16.f95
 !Compile into a file prog16.obj
 PROGRAM PROG16
DIMENSION X(100)

  CALL INPUT (X,N)
 IF(N.EQ.0) STOP
 CALL AVERAGE(X,N,AVERAGE)
  CALL ABOVE (X,N,AVERAGE,NABOVE)
 CALL SORT(X,N)
 CALL OUTPUT (X,N,AVERAGE,NABOVE)
 STOP
 END PROG16
```

The file prog16.f5 cn be compiled without any errors, but it will not build into an executable file. The subroutines must be defined. These can be typed in and stored and compiled in separate files.

```
        SUBROUTINE INPUT(Y,M)
! read in a long list of numbers
!Save in a file input.f95. Compile into a file input.obj
!----------------
         DIMENSION Y(100)
          WRITE (*,*) ' Please type in a list of real numbers. To end input type a letter'
          M=0
101       READ (*,1000, ERR=100) Y(M) ! trap format errors ourselves
           IF (I.GT.100) GO TO 102
          GOTO 101                    ! loop until format error
! run out of array memory
!
  102     WRITE (*,*) 'ONLY ROOM FOR 100 NUMBERS'
!-----------------
! user ended input
!with non numeric charcter
!------------------
  100      RETURN
 1000   FORMAT ( F12.3)
          END
```

```
      SUBROUTINE AVERAGE (Z,N,AV)
      DIMENSION Z(100)
!
! CALCULATE AVERAGE OF OUR LIST OF NUMBERS
! STORE IN FILE AVERAGE.F95
!
      SUM=0.0
      AV=0.0
      DO 100 KK=1,N
      SUM=SUM+Z(KK)
100   CONTINUE
      AV=SUM/N
      RETURN
      END
```

```
      SUBROUTINE ABOVE (F,NM,AV1,ABV2)
      DIMENSION F(100)
      ABV2=0
      DO 100 KK=1,NM
      IF (F(KK).GT.AV1) ABV2=ABV2+1
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SORT(G,NUMBERS)
      DIMENSION G(100)
      DO 110 II=1,NUMBERS
      DO 110 JJ=II,NUMBERS
      IF (G(II).GE.G(JJ)) GO TO 110
      TEMP=G(II)
      G(II)=G(JJ)
      G(JJ)=TEMP
110   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE OUTPUT (X,N,AVERAGE,NABOVE)
      DO 111 JJ=1,N
      WRITE (*,1002) X(JJ)
111 CONTINUE
      WRITE (*,*) 'average of ' ,N,' is ',AVERAGE,'nmber above is',NABOVE
      STOP
1002 FORMAT (1X,F12.3)
      END
```

4.3 COMMON BLOCKS

If variables are to be used in several subroutines it is sometimes more convenient to place them in a special part of the computer memory, accessible by all routines, known as a common block. Several features of common blocks are illustrated in the example below.

```
      common i,a,b
      i = 2
      a = 3.0
      b = 6.0
      c = 1.0
      d = 0.0
      e = 0.0
      f = 0.0
      call sub01(c,d)
      write(*,100)i,a,b,c,d,e,f
      call sub02(d,e)
      write(*,100)i,a,b,c,d,e,f
      call sub03(i,a,b,e,f)
      write(*,100)i,a,b,c,d,e,f
   10 continue
  100 format(' ',i4,6f8.2)
      stop
      end
      subroutine sub01(a,sum)
      common jj,aa,bb
      sum = 0.0
      do 10 i = 1,jj
      sum = sum+a+aa+bb
   10 continue
      return
      end
      subroutine sub02(a,sum)
      common k,ak,bk
      sum = 0.0
      do 10 i = 1,k
      sum = sum+a+2*ak+bk
      ak = 0.5*sum
   10 continue
      return
      end
      subroutine sub03(m,a,b,c,sum)
      sum = 0.0
      do 10 i = 1,m
      sum = sum+a+3*b+c
   10 continue
      return
      end
```

Output from above program

```
   2    3.00    6.00    1.00   20.00    0.00    0.00
   2   45.00    6.00    1.00   20.00   90.00    0.00
   2   45.00    6.00    1.00   20.00   90.00  306.00
```

Notes:

- Variables appearing in a common statement in a subroutine cannot also appear in that subroutine's argument list. However, variables in common in the calling routine can be passes to a subroutine through its argument list (e.g., sub03 in the above).
- Like argument lists, variable names used in common are local to the program unit. It is only the position in the common list, and type (real or integer) that must correspond. For instance, "a" in the main program is "aa" in sub01, and "ak" in sub02; and the "c" in the argument list of sub03 is the variable "e" in the main program.

- The value of variables in common can be changed at any point; i.e., in a subroutine. (e.g., ak is redefined in sub02, which means that a in the main program is changed).
- Common statements appear before all other statements in the main program and in subroutines.

4.4 DATA STATEMENTS

DATA statements can be used to set the numerical value of constants stored in common.

```
  COMMON X,Y,Z
DATA X,Y,Z /1.0,2.0,3.0/
```