Chapter 3: The Elements of a Program

We start this chapter with a couple of laundry lists (sometimes with mini-examples) that outline the data types available in fortran, the common control sequences, and file I/O. After we walk through the different program elements we will finish the chapter with an example problem from numerical integration.

3.1 Data Types

Some of the most common data types in fortran are:

- INTEGER – Exact whole number
- CHARACTER – Standard text ASCII character with one byte per character
- CHARACTER(LEN=4) – String of a specified length
- LOGICAL – Set to either .TRUE. or .FALSE.
- REAL – Floating point number with 7 significant figits (usually)
- REAL(KIND=RP) – Floating point number with significant digits specified by RP = SELECTED_REAL_KIND(15), where 15 yields double precision (16 significant digits)
- COMPLEX(KIND=RP) – Two floating point numbers with significant digits specified by RP

Keep in mind, we can make any of the preceding data types a PARAMETER.

Let's examine a few situations of how fortran handles casting one variable type to another:

typesExample.f90

```
PROGRAM typesExample
   IMPLICIT NONE
   INTEGER,PARAMETER :: RP = SELECTED_REAL_KIND(15)
   INTEGER           :: i,j,k
   REAL(KIND=RP)     :: x,y,z
!
   i = 7
   j = 4
   x = 3.7_RP
   y = 4.0_RP
   k = j**i
   PRINT*,'4**7= ',k
   k = i/j
   PRINT*,'7/4= ',k ! division with integers always truncates down
   k = x
   PRINT*,'automatic conversion 3.7 to INT= ',k ! always truncated towards zero
   z = i
   PRINT*,'automatic conversion 7 to REAL= ',z   ! converts integer to real
   z = i/j
   PRINT*,'automatic conversion 7/4 to REAL= ',z ! truncates then converts to real
   z = i/4.0_RP
   PRINT*,'automatic conversion 7/4.0_RP to REAL= ',z ! retains accuracy of a real
   z = 7.0_RP/j
   PRINT*,'automatic conversion 7.0_RP/4 to REAL= ',z ! retains accuracy of a real
   z = REAL(i,RP)/REAL(j,RP) ! compiler to treat the integers as reals
   PRINT*,'automatic conversion 7_RP/4_RP= ',z
END PROGRAM typesExample
```

We compile and run the types example to illustrate the meaning of the comments in the program

typesExample.f90 - Commands and Output

```
gfortran typesExample.f90 -o typesExample
./typesExample
 4**7=        16384
 7/4=            1
 automatic conversion 3.7 to INT=            3
 automatic conversion 7 to REAL=    7.0000000000000000
 automatic conversion 7/4 to REAL=    1.0000000000000000
 automatic conversion 7/4.0_RP to REAL=    1.7500000000000000
 automatic conversion 7.0_RP/4 to REAL=    1.7500000000000000
 automatic conversion 7_RP/4_RP to REAL=    1.7500000000000000
```

3.2 Control Sequences

This is by no means a comprehensive list of the control sequences available in fortran; however, these are ones that come up the most often:

- IF/THEN/ELSE – Execute certain pieces of code based on a logical condition(s). The main logical operators are:
  - ⊛ Less than: < or .LT.
  - ⊛ Less than or equal to: <= or .LE.
  - ⊛ Greater than: > or .GT.
  - ⊛ Greater than or equal to: >= or .GE.
  - ⊛ Equal to: == or .EQ.
  - ⊛ Not Equal: /= or .NE.
  - ⊛ Logical and: .AND.
  - ⊛ Logical or: .OR.
  - ⊛ Logical not: .NOT.

IF/THEN/ELSE Statement

```
IF (x.EQ.7) THEN
   do stuff
ELSE IF ((x.LT.5).AND.(x.GT.2)) THEN
   do other stuff
ELSE
   do other stuff
END IF
```

- DO loops – Perform a piece of code in the loop structure a specified number of times. The bounds of the loop are INTEGERS.

DO Loop

```
DO i = 1,13
   PRINT*,i
END DO
!  Loops can also run backwards
DO j = 9,0,-1 ! This means step backwards from 9 to 0, decrement by 1 each iteration
   PRINT*,j
END DO
!  In general, we have the bounds on the loop
DO j = start_value,end_value,increment ! the default increment is 1
   CODE
END DO
```

- DO WHILE loops – Perform a piece of code in the loop structure until a logical condition is met.

DO WHILE Loop

```
b = 2
DO WHILE (b/=128) ! Make sure this termination condition will be met!
   b = b*b
   PRINT*,b
END DO
```

- EXIT – Exit out of a DO loop based on a logical argument.

EXIT Example

```
DO i = 1,35
   PRINT*,i
   IF (i**2.GT.55) THEN ! or: IF(i**2.GT.55) EXIT
      EXIT
   END IF
END DO
```

- CYCLE – Increment to the next iteration in a DO loop based on a logical argument.

CYCLE Example

```
DO i = 1,6
   IF (i.EQ.4) THEN! or: IF(i==4). Be careful you don't write IF(i=4),
      CYCLE
   END IF
   PRINT*,i
END DO
```

3.3 Input/Output Constructs

Next, we examine the different options fortran offers for the input of data into a program to be operated on and output of data to the screen or to a file. For now we will only cover unformatted (i.e., list-directed) read/write, but know that it is possible to format the data to suit one's needs. We cover formatted reads/writes in Chap. 7 Additional Topics, as they can be somewhat complicated.

3.3.1 I/O to the Screen

- Input – We can ask the user to provide information from the terminal. We use a READ(*,*) statement to read in data from the screen inputted by the user. Again, the first * means 'the screen' and the second * means 'unformatted'.

readExample.f90

```
PROGRAM readExample
   IMPLICIT NONE
   INTEGER,PARAMETER :: RP = SELECTED_REAL_KIND(15)
   INTEGER           :: j
   REAL(KIND=RP)     :: x
   CHARACTER(LEN=20) :: name

   READ(*,*)'Enter an integer',j ! if you don't input an integer, an error is thrown
   READ(*,*)'Enter a real number',x
   READ(*,*)'Enter your name',name

   WRITE(*,*)j,x,name
END PROGRAM readExample
```

- Output – We've used this a couple times already, often we print to the screen only for debugging purposes. So we will only worry about unformatted output to the screen.

Unformatted Printing to Screen

```
! Both commands will produce the same output
PRINT*,'The number is',a
! or
WRITE(*,*)'The number is',a
```

3.3.2 I/O to a File

When we input or output data to a file we first have to tell the program to open a file. To do so we use the OPEN command, which assigns an integer (sometimes called the fileUnit) to reference a file's name. In general, don't set the fileUnit to be 0, 5 or 6 as most compilers reserve those values in the following way:

- ***Standard Error*** is 0 – Used by programs to output error messages or diagnostics.
- ***Standard In*** is 5 – Used by programs to input data from the terminal, similar to READ(*,*)
- ***Standard Out*** is 6 – Used by programs to output data to the terminal, similar to WRITE(*,*)

Always remember, anytime you OPEN a file you should CLOSE it once you are done reading/writing.

- Input – To read in from a file we replace the first * in the READ statement with the Unit that represents the file name.

readFileExample.f90

```
PROGRAM readFileExample
   IMPLICIT NONE
   INTEGER,PARAMETER :: RP = SELECTED_REAL_KIND(15)
   INTEGER           :: j
   REAL(KIND=RP)     :: x

   OPEN(UNIT=15,FILE='testInput.dat')
   READ(15,*)j,x
   CLOSE(15)

   PRINT*,j,x
END PROGRAM readFileExample
```

- Output – Similar to the input example we replace the first * in the WRITE statement with the fileUnit.

writeFileExample.f90

```
PROGRAM writeFileExample
   IMPLICIT NONE
   INTEGER,PARAMETER :: RP = SELECTED_REAL_KIND(15)
   REAL(KIND=RP)     :: x,y,z

   x = 6.25_RP
   y = 1.15_RP
   z = x + y
!
   OPEN(UNIT=15,FILE='testOutput.dat')
   WRITE(15,*)z
   CLOSE(15)
END PROGRAM writeFileExample
```

3.4 Example: Quadrature

Use the Riemann sum with left endpoints to approximate the integral

$$I = \int_a^b \frac{x^2}{3}\,dx \approx \sum_{i=0}^{N-1} \frac{x_i^2}{3}\Delta x.$$

In order to compute this integral approximation, we need to decide the discretization rule for the x_i's, the simplest is a uniform spacing. So, $x_i = a + i\Delta x$, with

$$\Delta x = \frac{b-a}{N}.$$

We will store the answer in the real variable 'sum' and display the result to the screen. The pseudocode of the integral approximation, for a given value of a, b, and N, is

Algorithm 1: *Left Riemann Sum*: Approximate an integral with the left Riemann sum.

Procedure Left Riemann Sum
Input: a, b, N

$\Delta x \leftarrow \frac{b-a}{N}$
$sum \leftarrow 0$
for $i = 0$ **to** $N-1$ **do**
  $x_i \leftarrow a + i\Delta x$
  $sum \leftarrow sum + (x_i^2/3) \cdot \Delta x$

Output: sum

End Procedure Left Riemann Sum

We next provide a fortran implementation that asks the user for a, b, and N values and prints the result to the screen.

leftRiemann.f90

```
PROGRAM leftRiemann
   IMPLICIT NONE
   INTEGER,PARAMETER :: RP =SELECTED_REAL_KIND(15)
   INTEGER           :: i,N
   REAL(KIND=RP)     :: a,b,x_i,dx,sum

   WRITE(*,*)' Enter a value for a, b, and number of sub-rectangles N'
   READ(*,*)a,b,N ! You don't include the RP when inputting real numbers
                  ! For example, just type -2.0,4.0,25
   dx  = (b-a)/N
   sum = 0.0_RP
   DO i = 0,N-1
      x_i = a + i*dx
      sum = sum + dx*(x_i**2/3.0_RP)
   END DO

   WRITE(*,*)' The integral is approximately',sum
END PROGRAM leftRiemann
```

Note, that we hard-coded the function that we wished to integrate. Later, we will learn how to pass an arbitrary function $f(x)$ into the program. We save our program as leftRiemann.f90 and run the program to find

leftRiemann.f90 - Commands and Output

```
gfortran leftRiemann.f90 -o leftRiemann
./leftRiemann
  Enter a value for a, b, and number of sub-rectangles N
-2.0,4.0,25
  The integral is approximately   7.5391999999999992
```

We can compare the output to the known answer for the integral

$$I = \int_{-2}^{4} \frac{x^2}{3}\, dx = \left.\frac{x^3}{9}\right|_{x=-2}^{4} = 8.$$

If we take N larger, the approximation will become better (i.e. closer to 8).