3 Loops

3.1 Aims

By the end of this worksheet, you will be able to:

- Understand more about the use of **real** and **integer** variables and how to use a mixture of data types in expressions
- Understand how to re-use code by **looping**
- Know how to control the number of times a section of code is executed by using a **do loop**

3.2 Mixing variable types

Exercise 3.1

Copy **divide.f95**

Make sure you understand this thoroughly!

```
program divide
implicit none
integer :: x
real :: y
x = 1
y = x/3
print *, y
end program divide
```

And run it. This program produces the following output:

```
0.00000
```

Something odd is happening. The problem is the line:

```
y=x/3
```

FORTRAN evaluates the **right hand side** of the assignment **first** using **integer** arithmetic, because both x and 3 are integer. 1 divided by 3 cannot be stored as an integer, and so the value 0 is returned. The result, 0, is then converted to a real number and the assigned to y.

Replace the line in program **divide**

```
x = 1        by
x = 10
```

Your output should now be:

```
3.00000
```

Can you see what is happening? FORTRAN is keeping the integer part of the answer and throwing the rest away.

To get over this problem, we have to signal to FORTRAN that we want it to calculate the right hand side of the expression using **real** arithmetic. If we want to keep x as **integer** data type, we could re-write our expression as follows:

```
y=x/3.0
```

The presence of a **real** number on the right hand side causes the right hand side of the expression to be evaluated using floating point arithmetic.

Actually, the problem is even more complicated! Where we have an expression like

```
y=x * ((2**i)/3)
```

where **x** and **y** are real and **i** is integer, FORTRAN computes the result in stages:

First it calculates (2**i)/3 and evaluates it as an integer number, then multiplies the result by x and evaluates it as real.

Exercise 3.2

Copy check.f95 to your computer.

```
    program check
!Integer and real arithmetic
    implicit none
    real :: x,y
    integer i
    x=2.0
    i=2
    y=x*((2**i)/3)
    print *,y
    y=x*((2.0**i)/3)
    print *,y
    end program check
```

… and examine its output. Make sure you understand why this is happening.

3.3 The do loop

Unless we are able to re-execute code, we might as well use a calculator… Now we start to take advantage of the power of the computer.

Exercise 3.3

Copy program loop.f95

```
program loop
implicit none
integer :: i
do i=0,20
      print *,i
end do
```

```
end program loop
```

Run the program. It prints out the numbers from 0 to 20 in steps of 1.

Note:

- **i** is called a **loop counter**. In this example, it has a start value of zero.
- All the statements within the **do** and **end do** are executed. In this example there is just the one statement, ie **print**.
- Each time the statements are executed, the **loop counter, i,** is **incremented** by 1.
- When the value of **i** is 20, the loop terminates, and the program resumes after the **end do**.

Change the **do** statement in program **loop** as follows:

```
do i = 50,70,2
```

Run the program. What happens?

The third argument in the **do statement**, is the **increment step**. If omitted, the value is taken as 1.

Loops can also decrement: try this

```
do i = 5,-5,-2
```

Exercise 3.4

Using a **do loop** to generate integer values of x between –10 and 10 in steps of 1, write a program that constructs a table of values of

y=1.0/x

What happened when x had the value zero? Use an **if, end if** to test for the condition that gives the incorrect value, and print out an appropriate message. Compare your result with divbyzero.f95.

Division by zero is one of the commonest reasons for a program to fail.

3.4 Nested Do Loops

We want to construct a table of values for z where

$$z = x^y$$

for values of x in the range 1 to 2 in steps of 0.5 and
y in the range 1 to 2 in steps of 0.5

Work through the next exercise which illustrates this:

Exercise 3.5

Copy program **xytab.f95** to your filespace.

```
      program xytab
      implicit none
!constructs a table of z=x/y for values of x from 1 to 2 and
!y from 1 to 4 in steps of .5
      real        ::  x, y, z
      print *, '    x     y     z'
      do x = 1,2
            do y = 1,4,0.5
                  z = x/y
                  print *, x,y,z
            end do
      end do
      end program xytab
```

Examine its output. Notice the use of the first **print** to give a heading to the table.

3.5 Using loops to do summation

Earlier on, we discussed the idea of assignments.

```
x = 1.0
```

means store the value 1.0 in the memory location called x.

If we had the following code:

```
x = 1.0
x = x + 1.0
print *, x
```

Can you guess what value would be printed out for x?

The answer would be 2.0.

Bearing in mind the definition of an assignment, the statement

Really important!

```
x = x +1.0
```

means "**add 1.0 to the value currently stored in memory location x and then store the result in memory location x"**.

Exercise 3.6

Copy file **increment.f95** to your file space and examine its output.

```
program increment
implicit none
integer :: i
 real :: x
 x=1.0
 do i=1,10
       x=x+1.0
       print *, i,x
 end do
 end program increment
```

- **Note carefully** that we have set the initial value of x *outside* of the **do** loop. Why have we done this? If you aren't sure – change the code to put the line x = 1.0 *inside* the loop – then examine the output.
- It is **important** to understand that if we use constructions such as **x = x + 1.0**, then it is vital to initialise x to some value. If we don't, it is possible that the value might be set to **any** random number. Run the program, make a note of the final value of x then put an exclamation mark in front of the **x = 1.0** statement and run the program again.

Exercise 3.7

Edit the line x = x + 1.0 in program **increment.f95**, and change it to x = x * i. Re-run the program and examine the output. What is significant mathematically about the sequence of numbers that has been generated?