Chapter 1

Loops

1.1 Types of `do` loop

If a block of code is to be performed repeatedly it is out inside a `do` loop. The basic structure of which is,

```
do
    repeated section
end do
```

Indentation helps to clarify the logical structure of the code. It is easy to see which section is being repeated.

There are two basic types of `do` loops.

(a) Deterministic `do` loops: The number of times the section is repeated is stated explicitly. eg,

```
do i=1,10
    repeated section
end do
```

This will perform the repeated section once for each value of the counter i=1,2,...,10.
The value of i itself may or may not actually be used in the repeated section.

(b) Non-deterministic `do` loops: The number of repetitions is not stated in advance. The enclosed section is repeated until some condition is or is not met. This may be done in two alternative ways. The first requires a logical reason for stopping, whist the second requires a logical reason for continuing looping.

```
do
    repeated section
    if (logical expression) exit
end do
```

or,

```
do while (logical expression)
    repeated section
end do
```

1.2 Deterministic `Do` Loops

The general form of the `do` statement in this case is

```
do variable=value1, value2, value3
    repeated section
end do
```

Note that

- The loop will execute for each value of the variable from value1 to value2 in steps of value3.
- value3 is the stride, it may be negative or positive; if omitted it is assumed to be 1.
- The counter variable must be `integer` type (there could be round off errors if using `real` variables).
- value1, value2 and value3 may be constant (eg, 100) or expressions evaluating to `integer` (eg, 6*(2+j))

1.2.1 Sample program

```fortran
program lines
    implicit none
    integer n
    do n=1,100
        print*, "I must not talk in class."
    end do
end program lines
```

Listing 1.1: A sample program of Deterministic Do loop

Alternatively, the counter (i in the program bellow) may actually be used in the repeated section.

```fortran
program doloops
    implicit none
    integer i
    do i = 1, 20
        print *, i, i*i
    end do
end program doloops
```

Listing 1.2: A sample program of do loop with counter used in repeated section

1.3 Non-deterministic `Do` Loops

The `if (...) exit` form continues until some logical expression evaluates as `.true.`. Then it jumps out of the loop and continues with the code after the loop. In this form a `.true.` result tells you when to stop looping.

This can actually be used to exit from any form of loop.

The `do while(...)` form continues until some logical expression evaluates as `.false.`. Then it stops looping and continues with code after the loop. In this a `.true.` result tells you when to continue looping.

Non-deterministic `do` loops are particularly good for

- summing power series (looping stops when the absolute value of a term is less than some given tolerance)
- Single-point iteration (looping stops when the change is less than a given tolerance)

As an example of the latter consider the following code for solving the Colebrook-White equation for the friction factor λ in flow through a pipe

$$\frac{1}{\sqrt{\lambda}} = -2.0\log_{10}\left(\frac{k_s}{3.7D} + \frac{2.51}{Re\sqrt{\lambda}}\right)$$

The user inputs values of the relative roughness $\frac{k_s}{D}$ and Reynolds number Re. For simplicity the program actually iterates for

$$\begin{aligned} x &= \frac{1}{\sqrt{\lambda}} : \\ x &= -2.0\log_{10}\left(\frac{k_s}{3.7D} + \frac{2.51}{Re}x\right) \end{aligned}$$

```fortran
program friction
    implicit none
    real ksd
    real re
    real x ! 1/sqrt(lambda)
    real xold
    real, parameter :: tolerence = 1.0e-5
    print *, "Input Ks/D and Re"
    read *, ksd, re
    x = 1.0 ! initial guess
    xold = x + 1.0 ! anything different from x
    do while ( abs(x - xold) > tolerence)
        xold = x
        x = -2.0*log10(ksd/3.7+2.51*x/re) ! new value
    end do
    print *, "Friction factor = ", 1.0/(x*x)
end program friction
```

Listing 1.3: Sample program of simple-point iteration

1.4 Cycle and Exit statement

The `cycle` statement interrupts the current execution cycle of the innermost `do` construct and a new iteration cycle of the `do` construct can begin.

```fortran
    do i=1, 10
        A(i)=c+D(i)
        if (D(i)<0) cycle ! if true then the next statement is omitted
        A(i)=1
    end do
```

The `exit` statement terminates execution of the innermost (or named) `do` construct.

```fortran
    i=0
    do
        i=i+1
        if (i>100) exit
        print*, "i is ", i
    end do
    print*, "Loop finished"
```

1.5 Nested do loops

`Do` loops can be nested. Indention is highly recommended here to clarify the loop structure.

```fortran
program nestedLoop
    implicit none
    integer i,j
    do i=10, 100, 10
        do j=1,3
            print*, "i, j = ", i, j
        end do
        print*, "i begins again"
    end do
end program nestedLoop
```

Listing 1.4: A sample program for nested loops

1.6 Implied `Do` loops

This highly-compact syntax is often use to initialize arrays or for I/O or sequential data. The general form is,
`(expression, index = start, end, [stride])`
and like any other `do` loops it may be nested. For example,

```
do i=1, nx
    x=xo + (i - 1) * dx
    print*, x
end do
```

can be condensed to single line,

```
print*, (xo + (i - 1) * dx, i=1,nx)
```