PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

- This week we are going to cover some basic coding constructs.
- Constructing a program is a bit like trying to generate a sophisticated play from a limited dictionary of words.
- In this lecture we will cover loops.
- Loops allow many operations to be performed using a few lines of code.

Loops

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- There may be a situation, when you need to execute a block of code many times.
- There are two types of loop:
  1. for loop,
  2. while loop.
- **for** loops are designed to execute the same set of instructions a fixed number of times.
- **while** loops repeat until a condition is meet. The danger here is that the loop never terminates.

For Loop

```
for(init; condition; update)
{
  code to be executed;
}
```

```
do init, condition, update
  code to be executed
end do
```

- Below is a simple loop which outputs the value of the loop counter each time around the loop.

```
int i;
for (i=0; i<10; i++) {
  printf("%2d,",i);
}
printf("\n");
```

```
integer (kind=4) :: i
do i = 0,9,1
   write(6,'(i2,a)', &
&    advance='NO') i,','
end do
write(6,*)
```

- The output of the loops are the same for both FORTRAN and C and is below.

```
 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Loop Example

- Notice that an integer variable is used for the loop counter.
- This is important because we want to know for sure how many times the loop is executed and integer arithmetic is exact.
- The loop counter i starts at zero and 1 is added each time around the loop. The final time around the loop is when $i = 9$. Thus the loop is executed 10 times.
- How then can we use floating point numbers in a loop?

```
int i;
float sum;
sum=0.0;
for (i=0; i<10; i++) {
  sum = sum + i*0.01;
}
```

```
integer (kind=4) :: i
real (kind=4) :: sum
sum=0.0
do i = 0,9,1
   sum = sum + i*0.01
end do
```

- The loop counter can be used in expressions to generate more complex behaviour.

While Loop

```
while(condition)
{
  code to be executed;
}
```

```
do while(condition)
  code to be executed
end do
```

- The loop terminates when the condition is false.
- Using a while loop in the above examples:

```
int i;
float sum;
sum=0.0;
i = 0;
while (i<10) {
  sum = sum + i*0.01;
  i++;
}
```

```
integer (kind=4) :: i
real (kind=4) :: sum
sum=0.0
i = 0
do while (i .lt. 10)
  sum = sum + i*0.01
  i = i + 1
end do
```

Which to choose?

- Under normal circumstances either type of loop can be employed.
- However if the maximum number of loop iterations is know then a for loop is best.
- If the number of iterations is unknown then a while loop is required.
- This brings up the possibility that the while condition is never meet.
- This is an infinite loop which you can terminate an infinite loop by pressing Ctrl + C keys.

Loop Control Statements

- There are additional controls for loops.
  1. break (exit): Terminates the loop at that point.
  2. continue (cycle): causes the loop to skip the remaining lines and begin the next iteration.
- These additional options allow more control over how many of the loop statements are executed above all or none.

Equivalent Loops

- Below we use a while loop and stop it when it is accurate enough.

```
sum=0.0; i=0;
while (fabs(sum-2.0) >
                    0.0001) {
  sum = sum + pow(0.5,i);
  i = i + 1;
}
```

```
sum=0.0; i=0;
do while (abs(sum-2.0) > &
&                   0.0001)
  sum = sum + 0.5**i
  i = i + 1
end do
```

- They can be changed into for loops but the maximum number of iterations = 20.

```
sum=0.0;
for (i=0; i<20; i++) {
  sum = sum + pow(0.5,i);
  if (fabs(sum-2.0) >
                 0.0001) {
    i = i + 1;
  } else {
    break;
  }
}
```

```
sum=0.0
do i=0,19,1
  sum = sum + 0.5**i
  if (abs(sum-2.0) > &
&          0.0001) then
    i = i + 1
  else
    exit
  endif
end do
```