The While and Do..While Loops

Each of these loops are examples of conditional loops, and are found in a similar form in many different programming languages (for example, as a **repeat . until** in Pascal). These types of repetition structures are important for use when it is not known how many times the loop will have to repeat before exiting.

Structure of a **while** Loop

```
i=0;
while( i<10)                    //Loop to display #'s from 0 to 9. No semicolon
{
        printf("The value of i is %i\n",i);
        i++;                    //The value must be incremented
}
```

Structure of a **do ..while** Loop

```
i=0;
do                              //Loop to display #'s from 0 to 9
{
        printf("The value of i is %i\n",i);
        i++;
}while(i<10);                   //semicolon on the end of this line
```

Important Features of these Loops

• Note that there is **not** a semicolon on the end of the while statement in the first type but that there **is** in the second type of loop.

• Each type of loop checks that a **condition** is true to 'decide' whether or not to repeat the loop.

• You may put as many operations as wanted between the {}

• You may have any number of conditions inside the () brackets.

• Any data types may be used within the () brackets.

• Each of these types of loops can be made to operate in exactly the same manner as the for loops used in the last exercises, but require more careful control over the logical condition to ensure that the loop both repeats when it is supposed to AND more importantly, exits when it is supposed to.

Exercises:

Do each of the following in a simple C program to experiment with the **while** loop.

1. Write a **while** or **do while** loop that :
   - displays all the multiples of 5 from 5 to 55.
   - prompts the user to enter a high and a low integer and will calculate and display the sum of the numbers between these two values. (Error trap input values.)

2. Write a C program which prompts the user to enter 2 positive integers and have the program determine both the LCM (lowest common multiple) and the GCF (greatest common factor) for these two numbers.

3. Write a C program which allows the user to play a simple number guessing game. The program should generate a random number (remember to use srand to seed the random number generator) between 1 and 10, and allow the user to keep guessing until the number is guessed.

4. Expand on the problem in question 3 to consider each of the following:
   a) to limit the number of guesses to 5.
   b) to allow the user to repeat to try again (with a new random number)
   c) to allow the user to enter a maximum value for the random number (ex. up to 100)
   d) to enter a user name for the low score. At this point it will only be valid until you exit the program.
   e) something of your own design??