Loops

A loop is a block of statements that is repeated. C has three types of loops: **`while`**, **`for`** and **`do`**. A loop has a Boolean expression to test. If the expression is true the statements in the block are executed and then the test is performed again. When the condition is false, the next statement after the loop is executed. A while loop and a for loop test a Boolean expression at the beginning , a do loop tests the Boolean expression at the end. While and for loops are executed *zero or more* times. Do loops are always executed *at least once*. The Boolean expression is a check point: The decision is made to execute the entire loop or not. If the value of the Boolean expression changes during execution of the loop, the loop continues until the test is made again. The loop does not end in the middle if the variable changes.

Endless Loops

Occasionally, a programmer writes a loop that never ends. In fact, it is really easy to write an endless loop. Before you start writing loops, it is a good idea to find out how to stop the program if it is in an endless loop.

- DOS: Ctrl+C should work. If not, try Ctrl+Alt+Delete to reboot the computer.
- Windows: Ctrl+Alt+Delgives you a chance to end a task that is not responding.
- UNIX: You need to kill the process. Ask the local UNIX guru for details before writing a program with a loop.

While Loops

The format for a while loop is:

`while (` _____ **`)`** _____ **`;`**

Boolean expression — statement or block of statements

Example: the program below prints the numbers 0, 1, 2, 3, 4, 5.

```
 0 // Print 0 1 2 3 4 5
 1 #include <iostream.h>
 2 int main()
 3 { int num=0;
 4   while (num<=5)
 5   {  cout<<num<<" ";
 6      num = num+1;
 7   } // while
 8   cout<<"\n";
 9   return 0;
10 } // main
```

Explanation:

3: The variable num is given a value of 0 when it is declared.

4: The Boolean expression is tested, if true, the statement or block after the expression is executed, then the test is repeated.

5: If the braces are left off, the only statement in the loop is the cout statement: since num never changes inside the loop this would create an endless loop!
6: This statement makes it possible for the Boolean expression to become false, without this statement there would be an endless loop!
7: At the end of the loop, control goes back to the Boolean expression and the test is performed again.
8: When the Boolean expression is false in line 4, this statement is executed. Notice that the cout in line 5 does not do a line feed. Line 5 prints the number and a space. This statement does the line feed after printing the entir line: 0 1 2 3 4 5.

Using ++

A variable can be incremented at the same time it is printed or used in another statement by including a ++ before or after the variable. If the ++ is before the variable, the variable is incremented before it is used. If the ++ is after the variable, the variable is used and then incremented.

Example:

```
x = 5;   // x is assigned a value of 5
y = x++; // y is assigned the current value 5, then x becomes 6

x = 5;   // x is assigned a value of 5
y = ++x; // x becomes 6, then y is assigned the new value, 6
```

The example below writes the program above using ++:

```
0 // Print 0 1 2 3 4 5
1 #include <iostream.h>
2 int main()
3 { int num=0;
4   while (num<=5)
5     cout<<num++ <<" ";
6   cout<<"\n";
7   return 0;
8 } // main
```

Explanation: In line 5 the variable is increment *after* printing. The separate line to increment is not needed now. Since there is only one statement in the loop, no braces are needed. The statement is indented to show human readers that the statement is the loop. It is probably better to always use the braces for the body of a loop. **Question**: What is the output if you use **++num** in the cout statement? Think it through first, then try it.

The minus sign can be used in the same way to decrement:

```
0 // Print 10 9 8 7 6 5 4 3 2 1 0 Blast Off!
1 #include <iostream.h>
2 int main()
3 { int num=10;
4   while (num>=0)
5     cout<<num-- <<" ";
6   cout<<"Blast Off!\n";
```

```
 7   return 0;
 8 } // main
```

The next example reads in numbers until a negative number is entered, then prints the total, the count, and the average. (No average is printed if count is zero.)

```
 0 // Find total, count, print average
 1 #include <iostream.h>
 2 int main()
 3 { int count=0, total=0, num;
 4   cout<<"Enter number: ";
 5   cin>>num;  // read first number
 6   while (num>=0)
 7   { total+=num;
 8     count++;
 9     cout<<"Enter number (-1 to end): ";
10     cin>>num; // read next number
11   } // while
12   cout<<"Total = "<<total<<"\n";
13   cout<<"Count = "<<count<<"\n";
14   if (count > 0)  // avoid dividing by zero!
15     cout<<"Average = "<<1.0*total/count<<"\n";
16   return 0;
17 } // main
```

Explanation:

3: Integer variables count and total are given initial values of 0. The variable num is not given an initial value, a value will be read in before num is used.

5: The first number is read in before the loop, this value will make it possible to enter the loop the first time.

6: If this statement is true, all of the statements in the loop will be executed, then return to this statement to test again. If this statement is false, execution will proceed with line 12.

7: num is added to the total. This statement is shorthand for `total = total+ num;`

8: count is incremented. This statement is shorthand for `count = count + 1;`

9: Print prompt to read next number. The prompt must include information about ending the loop.

10: Read in a number. Notice that the next statement after reading a number is to test to see if it is valid.

14: The average is printed *only* if count is greater than 0. There is no else, if count is zero the average cannot be computed.

Sample output:

```
Enter number: 8
Enter number: 12
Enter number: 3
Enter number: 5
Enter number: 0
Enter number: -6
Total = 28
Count = 5
```

```
Average =  5.60
```

```
-----------RUN-------------
   Enter number: 6
   Enter number: 5
   Enter number: -2
   Total = 11
   Count = 2
   Average =  5.50
   Enter number: -3
   Total = 0
   Count = 0
-----------RUN-------------
   Enter number: 0
   Enter number: -2
   Total = 0
   Count = 1
   Average =  0.00
```

The next program finds and prints the smallest and largest:

```
 0 // Find total, count, print average
 1 #include <iostream.h>
 2 int main()
 3 { int count=0, total=0, num, small, large;
 4   cout<<"Enter number: ";
 5   cin>>num;  // read first number
 6   small=large=num; //first num is smallest & largest so far.
 7   while (num>=0)
 8   { total+=num;
 9     count++;
10     if (num < small) small = num; // store new smallest
11     if (num > large) large = num; // store new largest
12     cout<<"Enter number (-1 to end): ";
13     cin>>num; // read next number
14   } // while
15   cout<<"Total = "<<total;
16   cout<<" Count = "<<count<<"\n";
17   if (count > 0)  // avoid dividing by zero!
18   {  cout<<"Average = "<<1.0*total/count<<"\n";
19      cout<<"The numbers range from "<<small<<" to "<<large<<"\n";
20   } //print statistics if any numbers
21   return 0;
22 } // main
```

Explanation:

6: The first number is both the smallest and the largest so far. This statement assigns num to large first. An assignment statement returns the value that was assigned, so the statement **`large=num`** returns the value that was assigned to large, that value is then assigned to small. Remember, everything is a function in C, even assignment statements.

7. Each number is tested to make sure it is >= zero. If num is negative the loop ends and execution picks up at line 15.

10: Each number is tested to see if it is smaller than the smallest number so far. If it is, then small is changed to store num.
11: Same as #10, but checks for new largest.
13: Read in another number as possible escape from loop. Without this statement it would be an endless loop!
14: Braces to enclose the body of the loop: without the braces it is an endless loop!
15: When the loop ends, line 15 is executed. If the user entered a negative number in line 5, count and total will both be zero at this point.
17: This statement makes sure that we do not print any statistics if there were no values. This is especially important to avoid trying to divide by zero.
21: If the program got to this point, it worked: return 0 to indicate that program executed with 0 errors.

In the two examples above, only positive numbers are valid data. The program could use a negative number to signal the end because it was not in the valid range of numbers. If all numbers are valid, then the program needs to continually ask the user if there are more numbers. This program does not find large and small.

```
 0 // Find total, count, print average
 1 #include <iostream.h>
 2 int main()
 3 { int count=0, total=0, num;
 4   char answer='Y';
 5   cout<<"Are there any numbers (Y or N):";
 6   cin>>answer;
 7   while (answer=='Y' || answer=='y') //accept either Y or y
 8   { cout<<"Enter number: ";
 9     cin>>num; // read next number
10     total+=num;
11     count++;
12     cin.ignore(100,'\n'); //flush the \n from the buffer
13     cout<<"Are there more numbers (Y or N):";
14     cin>>answer;
15   } // while
16   cout<<"Total = "<<total;
17   cout<<" Count = "<<count<<"\n";
18   if (count > 0)  // avoid dividing by zero!
19     cout<<"Average = "<<1.0*total/count<<"\n";
20   return 0;
21 } // main
```

Explanation:
3: Total is initialized to 0, but num will get its initial value from the user.
4: Answer will be used to determine when the loop ends.
5,6: Get answer to determine if loop executes at all.
7: Loop continues until user answers anything other than ‘Y’ or ‘y’.
8: Without the braces, it will print prompt over and over!
10: Add num to total. This is same as `total = total + num;`
11: Add 1 to count: same as `count = count + 1;`

12: After entering a number and pressing ENTER, the '\n' is left in the buffer. If you did not flush the buffer by using cin.ignore, the '\n' that is in the buffer would be assigned to answer. The statement cin.ignore gets rid of the line feed that is in the input buffer. This is necessary whenever characters are being read in.
13-14: Prompt and read in another answer to determine whether to end the loop or not.
15: End of the while loop, but of course you knew that because there is a COMMENT!!!
16-19: If no numbers were read in total and count will print as zero, which is OK. However, the average can only be calculated if count is greater than 0, divide by zero must be avoided.

Output:

```
-----------RUN-------------
Are there any numbers: Y or N:n
Total = 0 Count = 0
-----------RUN-------------
Are there any numbers: Y or N:y
Enter number: -5
Are there more numbers? Y or N:y
Enter number: 12
Are there more numbers? Y or N:y
Enter number: 0
Are there more numbers? Y or N:n
Total = 7 Count = 3
Average = 2.3333
```

do … while

The **do..while** loop does the test at the end of the loop. A **do…while** always executes at least once. After each execution, the Boolean expression is tested, if true the loop repeats, if false the next statement after the loop is executed. The program below is the same as the one above except that a do…while is used. Because there will be at least one number, the test to avoid dividing by zero is not necessary. Notice the indentation: all statement within { and } are indented the same amount.

```
 0 // Find total, count, print average
 1 #include <iostream.h>
 2 void main()
 3 { int count=0;
 4   double total=0.0, num;
 5   char answer;
 6   do
 7   { cout<<"Enter number: ";
 8     cin>>num; // read next number
 9     total+=num;
10     count++;
11     cin.ignore(100,'\n'); //flush the \n from the buffer
12     cout<<"Are there any more numbers? Y or N:";
13     cin>>answer;
14   } while (answer=='Y' || answer=='y'); //accept either Y or y
15   cout<<"Total = "<<total<<"\n";
16   cout<<"Count = "<<count<<"\n"; //count is always at least 1
17   cout<<"Average = "<<total/count<<"\n";
18 } // main
```

Explanation:

2: Function main will return nothing (void). There is no return at the end of this program.
6: There is no prompt to ask if there are any numbers: the do loop executes at least once.
14: Do loops have the Boolean expression at the end of the loop. Notice the semi-colon at the end of the line.
17: There will always be at least one number, no test to prevent divide by 0 is necessary.

Output:

```
Enter number: 5.6
Are there more numbers? Y or N:y
Enter number: 3.2
Are there more numbers? Y or N:y
Enter number: 0
Are there more numbers? Y or N:n
Total = 8.8
Count = 3
Average = 2.93333
```

Give careful thought to whether to use a do loop or a while loop. A while loop is used much more often than a do loop. A do loop is often used when the user would not have run the program if they did not have at least one set of information to process. Games often use a do loop because if you ran the program you usually want to play at least once. At the end of the game it asks if you want to play again.

The program below uses a do loop to convert temperatures from Fahrenheit to Celsius. Could the program have been written with a while loop?

```
 0 // convert Fahrenheit to Celcius
 1 #include <iostream.h>
 2 void main()
 3 { int fahr;
 4   double celcius;
 5   char answer='Y';
 6   do
 7   { cout<<"Enter Fahrenheit temperature: ";
 8     cin>>fahr; // read temperature
 9     celcius=(fahr-32)*5.0/9; //force to decimal
10     cout<<fahr<<" Fahrenheit = "<<celcius<<" Celcius\n";
11     cin.ignore(100,'\n'); //flush the \n from the buffer
12     cout<<"Are there any more temperatures Y or N:";
13     cin>>answer;
14   } while (answer=='Y' || answer=='y'); //accept either Y or y
15 } // main
```

Output:

```
Enter Fahrenheit temperature: 32
32 Fahrenheit = 0 Celcius
Are there any more temperatures Y or N:y
Enter Fahrenheit temperature: 0
0 Fahrenheit = -17.7778 Celcius
Are there any more temperatures Y or N:n
```