General Information About Iteration (Loops)

There are many ways of setting up a program to iterate/loop (re-execute the same statements many times):

- Define a fixed number of iterations.
- Have the number of iterations determined by the program at run time.
- Iterate indefinitely, stopping only when some condition changes at run time.

Everyday loops:

Example 1:
1. Lather
2. Rinse
3. Repeat

Example 2:
While there is still at least one dirty dish in the pile:
1. Get a dirty dish from the pile and place it in soapy water.
2. Wash the dish clean
3. Place the dish in the rinse water.
4. Rinse the dish.
5. Remove the dish from the rinse water and place it in the drying rack.
6. Check the pile for dirty dishes.

Loops in General:

- A loop is a control statement – it controls the execution of other statements
- A loop executes its controlled statements many times, over and over again
- There are two types of loops:
  - An iterative loop executes its statement(s) a predetermined number of times
  - A conditional loop executes its statement(s) as long as some condition is true, evaluating a conditional (Boolean) expression.
- When your program reaches a loop statement, it stops sequential execution and loops through the controlled statements.
- When the looping finishes, your program picks back up with sequential execution, starting at the next statement after the loop.
WHILE LOOP

- Conditional loops keep repeating while a condition is true. A conditional loop has only one expression – typically a Boolean expression – to control the operation of the loop.
- The loop evaluates its test expression each time it executes:
  - The loop continues looping if the result is true.
  - The loop stops looping if the result is false.
- In a while loop, the expression is evaluated BEFORE the loop is first executed, and it’s possible that the body of the loop many never execute at all.

Format:

```c++
while (expression) {
    statement;
    statement;
    statement;
}
```

Example 1:

```c++
int num=0;
cout << "Enter a number";
cin >> num;
while (num > 0) {
    cout << num << endl;
    num--;
}
```

Example 1 Output:
Example 2:

```cpp
int num=0, sum=0;
cout<<"Enter a number";
cin>>num;

while(num!=0)
{
    sum=sum+num;
    cout<<"Enter another number (zero to quit): ";
    cin>>num;
}

cout<<"The total is: "<<sum<<endl;
```

Example 2 Output:

```
Enter a number?
Enter another number (zero to quit): 2
Enter another number (zero to quit): 1
Enter another number (zero to quit): 0
The total is: 6
```
Do-While Loops

- Conditional loops keep repeating while a condition is true. A conditional loop has only one expression – typically a Boolean expression – to control the operation of the loop.
- The loop evaluates its test expression each time it executes:
  - The loop continues looping if the result is true.
  - The loop stops looping if the result is false.
- In a do-while loop, the expression is evaluated AFTER the loop is executed, so that the body of the loop will ALWAYS execute AT LEAST ONCE.

**Format:**

```
   do
      { statement;
         statement1;
      }while(expression);
```

**Example 1:**

```cpp
   int num=0;
   do
      { cout<<"Enter a number: ";
         cin>>num;
      }while(num!=0);
   cout<<"Loop exited";
```

**Example 1 Output:**

```
Enter a number: 10
Enter a number: 1
Enter a number: 0
Loop exited
```

A sentinel watches out for something in particular. In your program, that would be a particular value for a variable.
Example 2:

```cpp
int num=0, iterations=0;

do {
    iterations++;
    cout<<"Enter a number between 1 and 5: ";
    cin>>num;
} while(num>0 && num<6);

cout<<"Loop iterated "<<iterations<<" times."
```

Example 2 Output:

```
Enter a number between 1 and 5: 1
Enter a number between 1 and 5: 3
Enter a number between 1 and 5: 5
Enter a number between 1 and 5: 4
Enter a number between 1 and 5: 10
Loop iterated 5 times.
```
Special Considerations - Loops

Boolean Expressions Using == or !=

Avoid using these with decimal values. Because of roundoff error, doubles cannot be exactly represented in binary notation by the computer. When you compare two values using ==, sometimes you may get unexpected results. It is safe to use this type of comparison with integer numbers. If doubles are required, your best bet is to use < or >.

Sentinels

When you are waiting for a specific variable to reach a certain value to trigger the end of your loop, that variable is called a SENTINEL. For example, if you give the user an option to “Enter Y to continue, or N to finish,” then the variable into which the user’s value is entered is a sentinel “watching” for the value ‘N’.

Example:

```c
char con=0;
do {
    cout<<"Your loop has iterated. Enter Y to go again and N to stop. ";
    cin>>con;
}while(con!='N' && con!='n');

cout<<"Finished!";
```

Example Output:

```
Your loop has iterated. Enter Y to go again and N to stop. y
Your loop has iterated. Enter Y to go again and N to stop. y
Your loop has iterated. Enter Y to go again and N to stop. y
Finished!
```
Counters

Many times, you may want to count the number of iterations of your loop. This would be performed by using a counter. A counter will be an integer variable. It must be initialized (given a starting value) prior to your loop. Usually this will mean setting its value to zero, since the loop will not have executed yet. Inside the body of the loop, the counter will be incremented. There are several ways to increment.

**Format 1:**
```c
int counter=0; //declare and initialize my counter
...
//loop starts
...
  counter++;  //increment counter
...
//loop end
```

**Format 2:**
```c
int counter=0; //declare and initialize my counter
...
//loop starts
...
  counter=counter+1;  //increment counter
...
//loop end
```

**Format 3:**
```c
int counter=0; //declare and initialize my counter
...
//loop starts
...
  counter+=1;  //increment counter
...
//loop end
```

Accumulators

Sometimes in your loops, you will want to accumulate value in a variable to keep track of some amassed quantity. For example, if you want to sum the values a user is entering, you might add each of them to a variable on each iteration of the loop. The variable “accumulates” value like a growing snowball rolling down a hill. The accumulator must be initialized before the loop, typically to the value 0. There are two ways to write accumulation.
Format 1:
```cpp
int sum=0;
int num=0;

//start of loop
cout<<"enter number: ";
cin>>num;
    sum=sum+num;
//end of loop

cout<<"Sum is "+<sum;
```

Format 2:
```cpp
int sum=0;
int num=0;

//start of loop
cout<<"enter number: ";
cin>>num;
    sum+=num;
//end of loop

cout<<"Sum is "+<sum;
```

Infinite Loops

An infinite loop occurs when the programmer has constructed an expression that, when tested, will NEVER evaluate to false. This means the loop will NEVER stop executing. It will NEVER reach its exit condition.