

Remember "Reviewing C++" has many more examples and topics.

Tutorial 4 **Looping**

In most programs, a loop can be used to do something an x number of times. For the purpose of code saving it certainly helps and it can also prove quite useful.

A loop is used for repetition and contains some decision making. The decision you need to make is what condition(s) will continue to make this loop run.

There are a couple of types of loops. The first is a while loop (see tutorial 3). The other is a **for loop**. This means that **for** a certain number of times, do something:

```
for( loop counter; condition(s); counter changes ){  
    //code for the for  
}
```

Let's break down what each section of the for loop mean. Note that there are 3 sections, each of which (except the last) is separated by a semicolon.

First, the loop counter. The loop counter is used to keep track of how many times you want the loop to run. It is always a numeric value such as an int, double or float. What you do with the counter will come later.

Second, the condition(s) of the loop. This is where the logic comes into play. This is logic generally done on the loop counter. Say you wanted the loop to run 30 times. The counter would have logic that kept track of that (counter < 30).

Finally, the counter changes. Without this, you would have AN INFINITE LOOP!!! Those are quite a pain in the neck as the loop WILL NEVER END!!! Generally, you will either ++ or -- a loop but not always. Sometimes, a program calls for a loop to have a change by 2, 3, 5 etc. Just use the correct operations.

Example 1: **Simple looping (for)**

Below is a program that will print out the current value of the loop counter. It will ask you a user to enter a number (less than 50 but

Remember "Reviewing C++" has many more examples and topics.

positive) which will be how many times you want to let the loop run. This is a good way to view how a loop will run.

```
// for loop to print out index
#include <iostream>
using namespace std;

int main(){
    int times;
    cout << "How many times? ";
    cin >> times;

    if(times < 0 || times >= 50) exit(1);

    for(int i = 1; i <= times; i++){
        cout << i << endl;
    }
    return 0;
}
```

As you will observe, the program prints out each number on a separate line.

When dealing with loops, it is possible to have loops nested inside one another. The rule about this is that the inner most loop will be used first, then the one above that and the one above that one etc...

An example of this is when we are perhaps trying to print out a table of some kind. Here, let's print out a simple square of stars using two for loops:

Example 2: Simple Square

This program will print a 5 x 5 square of stars. It requires no input. The program uses a constant integer in the event you wish to change the sizes.

```
// for loops for square
#include <iostream>
using namespace std;

int main(){

    const int SIZE = 5;

    for (int r = 1; r <= SIZE; r++) {
        for (int c = 1; c <= SIZE; c++) {
            cout << "*";
        }
    }
}
```

Remember "Reviewing C++" has many more examples and topics.

```
        cout << endl;
    }

    return 0;
}
```

Here is what this square would look like:

```
*****
*****
*****
*****
*****
```

Example 3: Forward Triangle

The objective of this program is to get used to nested looping. This program will print out a triangle of stars using two for loops, one nested inside the other. It will ask a user for an positive input less than or equal to 10.

```
// for loop for forward triangle
#include <iostream>
using namespace std;

int main(){
    int size;
    cout << "How large a triangle do you want? ";
    cin >> size;

    if(size < 0 || size > 10) exit(1);

    for (int r = 1; r <= size; r++) {
        for (int c = 1; c <= r; c++)
            cout << "*";
        cout << endl;
    }

    return 0;
}
```

When the user enters a correct number, a triangle is printed that will go from the top left to the bottom right. Let's see what this program does with the nested loop.

The "row" loop is first followed by the "column" loop. The rule is, the inside loop(s) will exercise fully, followed by the one above it, then above that etc... So here, the row loop will have a value of 1 to begin and the column loop will also have a value of 1. But each time the

Remember "Reviewing C++" has many more examples and topics.

column loop changes, the value of the row DOES NOT change, until the column loop is completely finished.

Say that the user enters the number 6. Here is what this triangle would look like:

```
*
**
***
****
*****
*****
```

Example 4: Downward Triangle

Just like the previous example, the program will involve nested loops to print out a downward facing triangle. The triangle will go from the top right to the bottom left.

```
// for loop for downward triangle
#include <iostream>
using namespace std;

int main(){
    int size;
    cout << "How large a triangle do you want? ";
    cin >> size;

    if(size < 0 || size > 10) exit(1);

    for (int r = 0; r < size; r++) {
        for (int c = 1; c <= size; c++) {
            if (c <= r) cout << " ";
            else cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

Most of this program is like the previous example however the logic is a bit different. Inside the column loop here, the if statement checks that if the column is less than the row, print out a white space, otherwise print out a star. Why is this? It is so because in order for the triangle to be printed out correctly, there must be white spaces BEFORE the stars to keep the order correct.

Remember "Reviewing C++" has many more examples and topics.

As an example, say the user enters the number 6. Here is what this triangle would look like:

```
* * * * *  
 * * * *  
  * * *  
   * *  
    *  
     *  
      *
```