Loops

Sometimes we want our code to execute multiple times. It would be really unfortunate if we had to keep typing the same exact lines of code again and again, and what if we didn't know exactly how many times we wanted the code to repeat? Fortunately, we have loops to help us out. A *loop* is a section of code that the computer runs over and over again until it reaches some stopping criteria. There are two kinds of loops that we will work with: for-loops and while-loops.

A for-loop repeats code for a specific number of times. A good hint to know when you want to use this type of loop is if you know exactly how many times you want something to repeat. For example, let's say we want to print "Hello World" five times. We could write the following for loop:

```
for (int i = 0; i < 5; i++){
     cout << "Hello World\n";
     //more lines of code can go here
}
```

This will print:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Let's dissect this code. We begin with the word "for", which tells the computer that we have a for loop. Then, inside the parentheses, we define how many times we want the loop to run, or we "initialize" the loop. The first phrase (int i=0;), defines the loop counter, which is the variable that keeps track of how many times the loop executes. We tell the computer that the loop counter will be named i, and that it will begin by being equal to 0. 0 is the customary place to begin counting in computer science, instead of starting at 1 like we normally do. The second phrase (i < 5;) tells the computer to keep looping as long as i is less than 5; when i is greater than or equal to five, the loop will stop. The third phrase (i++) tells the computer how to count. The shorthand means that i will increase by 1 every time the loop runs. Then, inside curly braces, we put all the code that we want to execute each time the loop runs.

Now we will look at another example. Say we want to count from 0 to 9, and we want to print out the number each time. We can easily use a loop to do this:

```
for (int i = 0; i < 10; i++){
     cout << "The current number is " << i << ".\n";
}
```

This will print:

```
The current number is 0.
The current number is 1.
The current number is 2.
The current number is 3.
The current number is 4.
The current number is 5.
The current number is 6.
The current number is 7.
The current number is 8.
The current number is 9.
```

Notice how 10 doesn't print, because we told the loop to stop as soon when i was no longer less than 10. The first time the loop executes, i equals 0, so the loop prints out the message "The current number is 0." and then adds one to i, making i 1. Then the loop executes again, and we print "The current number is 1." Then i is increased by 1 again, making it 2, etc.

If we want to change how we are counting, we can change the loop initialization. Say we want to count backwards from 4 to 1. We can initialize the loop like this:

```
for (int i = 4; i >= 1; i--){
     cout << "The current number is " << i << ".\n";
}
```

Here, we begin with i being equal to 4. We loop until it is not true that i > = 1 (in other words, until i < 1). We tell the computer to subtract one from i each time the loop runs with "i-". This loop prints:

```
The current number is 4.
The current number is 3.
The current number is 2.
The current number is 1.
```

We can also count by numbers other than one and negative one. Let's say we only want to print out odd numbers. We could change the counter part of the loop initialization to count by 2s from 1 to 7 in the following way:

```
for (int i = 1; i < 9; i += 2){
     cout << "The current number is " << i << ".\n";
}
```

With this loop initialization, we get the following output:

```
The current number is 1.
The current number is 3.
The current number is 5.
The current number is 7.
```

The loop begins with i=1 and increases by 2 every time. Notice that it stops before reaching 9. To make more complicated loops, all you have to do is think carefully about the loop counter. Where do you want it to start? Where should it stop? How do you want to count?

Challenge 1:

Write a loop that prints every third number starting at 15 and going down to 6 (so 6 is included in the loop).
You can also nest loops inside one another. Let's say we want to do a simple summation program. Imagine that we have two lists each containing the numbers 1-4, and we want to sum together every number from the first list with each number from the second list. So, we essentially want to do this:

List 1	+	List 2	=	Sum
1	+	1	=	2
1	+	2	=	3
1	+	3	=	4

1 + 4 = 5
2 + 1 = 3
2 + 2 = 4
2 + 3 = 5
2 + 4 = 6
…
…

At the end, we will have 16 sums (4x4). We can easily do this in 3 C++ by using two for-loops, one nested inside the other. Let's think of List 1 as the i variables, and List 2 as the j variables. We know from the examples above how to generate a loop that generates the numbers from 1 to 4, so we will do this for List 1 with the loop variable i:

```
for (int i = 1; i < 5; i++){
     //put code here
}
```

We know that *for each i variable*, we want to sum it with all the j variables, which are the numbers 1 to 4. So we can put the j for-loop inside of the i for-loop, like this:

```
for (int i = 1; i < 5; i++){  // i for-loop
     for (int j = 1; j < 5; j++){  //j for-loop
          //prints out i+j
          cout << i << " + " << j << " = " << i + j << "\n";
     }
}
```

And this gives us the output:

```
1 + 1 = 2
1 + 2 = 3
1 + 3 = 4
1 + 4 = 5
2 + 1 = 3
2 + 2 = 4
2 + 3 = 5
2 + 4 = 6
3 + 1 = 4
3 + 2 = 5
3 + 3 = 6
3 + 4 = 7
4 + 1 = 5
4 + 2 = 6
4 + 3 = 7
4 + 4 = 8
```

Notice that the first number that prints out is the i variable, and the second number is the j variable, and the final number is the sum of i and j. Let's quickly talk through the loop. We begin with i being equal to 1 and j being equal to 1, and i+j equaling 2. Then the j loop advances by 1, because the inner loop has to finish before the outer loop can advance. So i is 1 and j is 2 and their sum is 3. Then j advances by 1 again, and i is 1 and j is 3 and their sum is 3. Finally, i is 1 and j is 4 and their sum is 5. Since j can't advance anymore (because j must be less than 5), the j loop finishes. Now the i loop advances by 1, so i is 2. Now the j loop starts all over again from the beginning, and j is 1 and i+j is 3, and so forth and so on. Nested loops can be a little tricky, so if they are confusing, ask for help!

Challenge 2:

Write a program that multiplies all the odd numbers between 1 and 9 by all the even numbers between 2 and 10.

Hint: follow the example above, but don't forget to change your loop initializations.

While-loops are another kind of loop that are used when you don't know how many times the loop should execute. For example, let's say you are trying to get input from a user that is a number between 1 and 10. *While* the user keeps typing in numbers that are too large or too small, you can ask him for new input until he enters a number between 1 and 10. Here's what this while loop would look like:

```
int main(){
     int userInput;

     //get input from user
     cout << "Enter a number between 1 and 10: ";
     cin >> userInput;

     //as long as userInput is less than 1 or greater than 10
     while ((userInput < 1) || (userInput > 10)){
          cout << "Read directions!  Enter a number between 1 and 10: ";
          cin >> userInput;  //get new input from the user
     }
     //when we reach this line, we know the user entered
     //a number between 1 and 10,
     //or else we would still be stuck in the while loop.
     cout << "You entered " << userInput << ".\n";
}
```

This while-loop executes as long as userInput is less than 1 or greater than 10. Let's walk through what is happening in this program. We first get some input from the user. We begin the while loop by checking the input to see if it is between 1 and 10. If it isn't, we yell at the user and ask for new input, and we **make sure to save the new input to *userInput***. Why is this so important? We'll see in a bit. Then the program loops back to the beginning of the while loop and checks to see if *userInput* is between 1 and 10. If it still isn't, we execute the loop again and ask for new input. We continue this as many times as necessary until the user cooperates. As soon as the user enters a number between 1 and 10, we break out of the while-loop, because we only execute the while loop if *userInput*is less than 1 or greater than 10. We then print out the number.

In the above program, we made sure to save the new user input to the same variable as the old user input. This is important because if we never change the old user input variable, the while-loop would never stop! To illustrate this, let's change the above while-loop to make it incorrect by introducing another variable called *newUserInput*:

```
while ((userInput < 1) || (userInput > 10)){
     cout << "Read directions!  Enter a number between 1 and 10: ";
     //get new input from the user and save it to new variable
     cin >> newUserInput;
}
```

This loop begins by checking to see if *userInput* is less than 1 or greater than 10. Let's say that *userInput* equals 100, so the loop will run. Inside the loop, the program prints out a nasty message to the user and gets new input from the user, the number 5. This time, however, 5 is saved to a different variable called *newUserInput*. Then the program checks the loop condition again. Since we never changed the value in *userInput*, it still equals 100, so the loop will run again even though the user just entered a valid number. In fact, this loop will run forever! This is called an infinite loop. To avoid this, always make sure that you update the loop variable somehow by changing its value so that eventually your loop will stop. (To break out of an infinite loop, type [Ctrl] + C.)

Challenge 3:

Create a program that asks the user to enter a number between 5 and 50. If the user enters invalid input, print out a message to the user asking them to try again and keep getting input from the user until they enter a valid number.

Bonus: Can you keep track of the number of attempts a user makes to enter a number, and then print it out when they finally enter a valid number?

Source