CSCI 1300 CS1: Starting Computing
Ashraf, Cox, Spring 2020
Homework 4 Background
Homework Due: Saturday, February 15, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm February 14th)

Looooooooop

While Loops

Loops allow us to run a section of code multiple times. They will repeat execution of a single statement or group of statements as long as a specified condition continues to be satisfied. If the condition is not true, then the statement will not be executed.

Syntax and Form:

```
while (condition)
{
 //statement(s) to do something;
}
```

Here, *while* is a C++ reserved word, *condition* should be a Boolean expression that will evaluate to either **true** or **false**, and *statement to do something* is a set of instructions enclosed by curly brackets. If the condition is true, then the specified statement(s) within the loop are executed. After running once, the Boolean expression is then re-evaluated. If the condition is true, then the specified statement(s) are executed again. This process of evaluation and execution is repeated until the condition becomes false.

Example 1:

```
int userChoice = 1;
while (userChoice != 0)
{
   cout << “Do you want to see this question again? “ << endl;
   cout << “Press 0 for no, any other number for yes.” << endl;
   cin >> userChoice;
}
```

Entering ‘0’ will terminate the loop, but any other number will cause the loop to run again. Note how we must initialize the condition before the loop starts. Setting `userChoice` equal to 1 ensures that the while loop will run at least once.

Example 2:

```
int i = 0;
while (i < 5)
{
     cout << i << endl;
     i = i + 2;
}
```

Notice how you must manually initialize `i` to equal 0 and then manually increment `i` by 2. Inserting `cout` statements into your loops is a quick way to debug your code if something isn't working, to make sure the loop is iterating over the values you want to be using. A common error is to forget to update `i` within the loop, causing it to run forever.

For loop

Sometimes you know the exact number of iterations that a loop performs. A *for* loop possess three elements:

- Initialization. It must initialize a counter variable to a starting value.
- Condition. If it is true, then the body of the loop is executed. If it is false, the body of the loop does not execute and jumps to the next statements just after the loop.
- Update. It must update the counter variable during each iteration

Syntax form:

```
for (initialization; condition; update)
 {
  //statement(s) to do something;
 }
```

Example 1: for loop that prints “hello” five times

```
for (int count = 0; count < 5; count++)
{
      cout << “hello” << endl;
}
```

In this loop, `count` is initialized to `0`, the test expression is `count < 5`, and `count++` to increment the count value by one.

Example 2:

```
for (int i = 0; i < 5; i = i + 2)
{
      cout << i << endl;
}
```

Notice that this example behaves in the same way as the example 2 in the `while` loop section above.

Strings

In C++, `string` is a data type just like `int` or `float`. Strings, however, represent sequences of characters instead of a numeric value. A string literal can be defined using double quotes. So `“Hello, world!”`, `“3.1415”`, and `“int i”` are all strings. We can access the character at a particular location within a string by using square brackets, which enclose an ***index*** which you can think of as the ***address*** of the character within the string. Importantly, strings in C++ are indexed starting from zero. This means that the first character in a string is located at index 0, the second character has index 1, and so on. For example:

```
string s = “Hello, world!”;
cout << s[0] << endl;  //prints the character ‘H’ to the screen
cout << s[4] << endl;  //prints the character ‘o’ to the screen
cout << s[6] << endl;  //prints the character ‘ ’ to the screen
cout << s[12] << endl; //prints the character ‘!’ to the screen
```

There are many useful functions available in C++ to manipulate strings. One of the simplest is `length()`. We can use this function to determine the number of characters in a string. This allows us to loop over a string character by character (i.e. *traverse the string*):

```
string s = “Hello, world!”;
cout << s.length() << endl;                //This will print 13
for (int i = 0; i < s.length(); i++)
{
      cout << s[i] << endl;
}
```

This will print each character in the string “Hello, world!” to the screen one per line. Notice how the length function is called.
The correct way:

- `s.length()`

Common mistakes:

- `length(s)`
- `s.length`

This is a special kind of function associated with objects, usually called a *method*, which we will discuss later in the course.

What happens in the above code snippet if we try to print characters beyond the length of the string? In particular, what happens when we replace `s.length()` with `s.length()+3` in the above `for` loop?