Lab 4: Iteration Statements in C++

Iteration statements (or loops) in C++ allow a program to execute a single statement multiple times. The execution of this single statement is controlled by a predicate that is evaluated at run-time. Sometimes this predicate is evaluated before executing the single statement (pre-evaluation) and sometimes it is evaluated after executing the single statement (post-evaluation). The predicate of an iteration statement is very important.

The three iteration statements in C/C++ are for, while, and do-while. Please see Chapter 3 in your textbook for a complete discussion of these constructs. As an example, the basic structure of the for statement is:

```cpp
for ( command 1 ; command 2 ; command 3 )
    // statement to be repeatedly executed
```

If the single statement to be repeatedly executed is a compound statement, the sequence of statements appearing within the compound statement are executed in order as long as the predicate controlling the iteration is true. As an example, the following for statement

```cpp
for ( command 1 ; command 2 ; command 3 )
{
    statement-1;
    statement-2;
}
// rest of the program.
// Not part of the loop.
```

contains two statements that will be executed if the predicate controlling the iteration is true. Each of these statements will be executed in order (from top to bottom) as long as the predicate is true.

command 1 is executed once at the beginning of the loop, before the predicate controlling the iteration statement is evaluated, and before any statement or statements in the iterator are executed. command 1 is typically used to initialize variables used within the loop.

command 2 is the predicate that controls the execution of the statement within the for loop. This predicate is evaluated before executing the statement within the loop. If the predicate resolves to be true, the statement controlled by the iterator will be executed. If the predicate does not resolve to be true, the single statement controlled by the predicate will not be executed, and execution will resume at the first statement following the for loop.
command 3 is executed at the end of every iteration of the loop. command 3 is not executed if the predicate controlling the iteration is not true. For example, the following loop repeats exactly 100 times.

```c
for ( int j = 0 ; j < 100 ; j++ ) // loop 100 times
{
   // commands to be
   // executed
   // during the loop.
}
```

Note that the execution of statements within the iterator is predicated upon the value of the variable j being less than 100. Also note that the value of the variable j is modified at the end of every iteration of the loop, after the last command within the compound statement is executed. This behavior ensures that the statements within the compound statement will be executed a fixed number of times.

**Exercise 4.1**

Your task in this exercise is to build a program that prints the ASCII representation of characters numbered from 40 to 119 inclusive.

Nested iteration statements are iteration statements that contain an iteration statement. Nested iteration statements are often used for printing tables of data. The following is a nested iteration statement:

```c
for (i=0; i<100; i++)
{
   j = 0;
   while (j < 100)
   {
      cout << "i * j: " << i * j << endl;
      j++;
   }
}
```

In this example, the while loop is “nested inside” the for loop. When using nested iteration statements, don’t forget to indent properly to make it clear which sections of code are controlled by a given iteration statement. It is relatively easy to become confused by your own code if you do not use formatting to clearly indicate your intentions. This becomes particularly important when you are building a program over an extended period of time.
1. Log in to wormulon, and in your HOME directory create a new directory called Lab_4.

2. Step into the Lab_4 directory and create a file named ex1.cpp. In this file you are to build a C++ program that prints a table of a portion of the ASCII values for different characters. The table should look roughly like this:

```
  0  1  2  3  4  5  6  7  8  9
40 (   )   ...  
50    ...    
60    
70    
80    
90    
100   
110   
```

This table shows, for example, that the ASCII value 40 (40+0) represents the character `(` and the ASCII value 41 (40+1) represents the character `)`. We will not worry about the values below 40 or above 119 because many of these are non-printable characters. To create this table you will need to use nested iteration statements. For the rows you may want to count from 40 to 110 in steps of 10. The other iteration statement, for the columns, will need to count from 0 to 9. The value for each entry in the table will be the character corresponding to the ASCII value of the sum of the two loop variables. For example, if the loop variables have values 100 and 5 respectively then the table entry will be the character corresponding to the ASCII value 105. To print this character use the following statement:

```
cout << char(x+y);
```

where x and y are the two loop variables.

3. Compile and test your program to ensure that it works as specified. In order to make your table look nice, you will need to be a little careful about where you print spaces and new lines.

4. When you have finished the program use cscheckin to submit your program in ex1.cpp to your lab instructor. You do not need to create an archive of your Lab_4 directory for this assignment; just submit the source code in ex1.cpp using cscheckin.