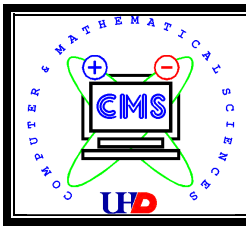

Department of Computer and Mathematical Sciences

CS 1410 Intro to Computer Science with C++

9

Lab 9: Alternate Controls

Objectives:
The objective of this lab is to demonstrate some of the ALTERNATIVE CONTROL STATEMENTS available in C++ and many other high-level programming languages.

The basic control statements we have studied so far are the IF-ELSE statement and the WHILE statement (or loop). In this lab we will examine the SWITCH statement, along with the FOR and DO-WHILE statements. The SWITCH statement is closely related to the IF-ELSE statement, and the FOR and DO-WHILE statements are types of loops and therefore similar to the WHILE-LOOP.

Task 1: DO-WHILE loop statement
The purpose of this task is to demonstrate the DO-WHILE loop.

Recall that in a standard WHILE-LOOP, the loop condition comes before the loop body and is evaluated each time BEFORE the loop body executes. Thus a standard WHILE-LOOP is called a PRE-TEST loop. In some instances it is convenient to delay the evaluation of the loop condition until after the loop body is executed. This is called a POST-TEST loop. In C++, a DO-WHILE statement is used to implement a post-test loop. The general syntax for a DO-WHILE loop is shown below:

```
do
{
   list of statements [loop body]
}
while ( loop condition )
```

The token "do" is a reserved word, as is "while" of course. A DO-WHLE statement executes very much as one would expect. First, the loop body executes. Then the loop condition is evaluated. If it is TRUE, the loop body executes again, the loop condition is evaluated again, and so forth. If the loop condition is FALSE, the program proceeds to the next statement. Therefore, the loop body must execute at least one time, which is not the case for a standard WHILE-LOOP.

Another new feature introduced in this program is the use of the data type " bool ", which is a reserved word. This data type has only two data values associated with it, " true " and " false ", both of which are also reserved words. The data value true is stored in memory as the integer value 1 while false is stored as 0.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
//Program name: Decoder
const int Inverse = 19, Intercept = 21, SizeAlpha = 26;
int main(void)
{
   fstream InFile;
   int Shift;
   bool StopFlag;
   char C;
   InFile.open("g:\\InLab91.txt", ios::in);
   do
   {
      InFile>>C;
      StopFlag = (C == '#');
      if (!StopFlag)
      {
         Shift = C - 'A';
         C = char(Inverse * (Shift+Intercept) % SizeAlpha + 'A');
         cout<<C;
      }
   }
   while ( !StopFlag );
   InFile.close();
   cin.get(); cin.get();
   return 0;
}
```

Activity 1.1: Study the following program carefully.

Activity 1.2: Predict the output from the program **Decoder** if the input file **InLab91.txt** contains the data shown in the Input box. NOTE: You need the ASCII table from page A17 in your text to determine the ASCII value for each character.

Input:	*Predicted Output:*
XYBXWWXSG#	

Activity 1.3: Download **Lab9Tsk1.cpp** and **Inlab91.txt** files from my webpage. Start MS Visual Studio and create a C++ project called **Lab9**. Then add the file **Lab9Tsk1.cpp** to the project. Then run the program **Decoder** and observe the output. Explain any discrepancies with your predicted output.

Obeserved Output:

Activity 1.4: Modify the **Decoder** program contained in this file by replacing the DO-WHILE loop with a WHILE-LOOP. The modified program should give the same output as with the DO-WHILE loop. Test your program for correctness.

Task 2: For-Loop

The purpose of this task is to demonstrate the FOR-loop.

Because count-controlled loops are used so frequently in programming, a type of statement has been included in the C++ language that makes it easy to set up count-controlled loops. This is a FOR statement. The loop condition initialization and loop condition update are included as part of the syntax of the FOR-loop statement. The outline of the basic syntax for a standard FOR-loop is shown below:

```
for (loop counter = inital value; loop counter <= terminal value; loop counter++)
{
list of statements [loop body]
}
```

There are lots of variations permitted on this basic outline. The token "for" is a reserved word. Since a FOR-loop is a count-controlled loop, a loop counter variable of type int is used to "count" the number of times the loop body executes. As the first part of the FOR-loop syntax, this counter is initialized to some initial value (often 1 or 0). This initialization happens only once. Following a semicolon is the loop condition, which checks to see if the counter is no more than some fixed terminal value. If this condition is true, the loop body will execute, otherwise the loop terminates and execution proceeds to the next statement in the program. Next is the update of the loop condition, which for a count-controlled loop usually consists of adding 1 to the loop counter. This is often accomplished by using the INCREMENT operator ++, which adds 1 to an integer variable. The loop condition is not updated until the loop body is executed. Here is a specific example of a FOR-loop:

```
int i;
for ( i = 1; i <= 10; i++ )
cout<<i<<endl;
```

This loop prints the integers 1 to 10 on the screen, one per line. The loop counter is the variable i. The initial value is 1 and the terminal value is 10. The loop body consists of one output statement and executes 10 times altogether. It is possible to write a FOR-loop so that the loop counter is DECREMENTED after each execution of the loop body, rather than incremented. For instance, this FOR loop also prints the integers 1 to 10 on the screen, but in reverse order:

```
int i;
for ( i = 10; i >= 1; i-- )
cout<<i<<endl;
```

Activity 2.1: The following program is the **Extrema** program from Lab8 Task3. Make sure you understand what the program does.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
//Program name: Extrema
int main(void)
{
   int Size, I;
   double Extremum1, Extremum2, Number;
   fstream InLab83;
   InLab83.open("g:\\inlab83.txt", ios::in);
   InLab83>>Size;
   InLab83>>Number;
   Extremum1 = Number;
   Extremum2 = Number;
   I = 1;
```

```
   while (I <= Size-1)
   {
      InLab83>>Number;
      if (Number < Extremum1)
         Extremum1 = Number;
      if (Number > Extremum2)
         Extremum2 = Number;
      I++;
   } // while I
   cout<<"The extrema are "
   <<fixed<<showpoint<<setprecision(2)
   <<Extremum1<<" and "<<Extremum2;
   InLab83.close();
   cin.get(); cin.get();
   return 0;
} // main
```

Activity 2.2: If you don't have **Lab8Tsk3.cpp** file from Lab8, download **Lab8Tsk3.cpp** file along with **Inlab83.txt** from my webpage. Then run the program and observe the output.

Input:	*Observed Output:*
7 89.25 89 -1.75 150 -1.8 10 59.3	

Activity 2.3: Modify the program **Extrema** by replacing the WHILE-LOOP with a FOR-LOOP. The modified program should give the same output as with WHILE-LOOP. Test your program for correctness.

Task 3: For-Loop

The purpose of this task is to give experience with writing FOR loops.

As you probably have learned by now, one of the trickiest aspects of writing or tracing a program with loops is when one looping statement is NESTED inside the loop body of another looping statement. Nevertheless, nested loops are a necessary and quite useful tool in programming. In this task, you will complete a program that requires nested FOR-loops. A general syntax outline for nested FOR-loops is shown below:

```
for (counter1 = initial value1; counter <= terminal value1; counter1++)
{
  for (counter2 = initial value2; counter2 <= terminal value2; counter2++)
  {
     inner loop body
  }
}
```

The first of these FOR-loops (controlled by the variable counter1) is usually called the OUTER loop. As part of its loop body it contains the FOR loop controlled by the variable counter2. This is usually called the INNER loop. With nested loops, the body of the inner loop may execute many times. Here is a specific example:

```
int M, N;
for ( M=1; M <= 5; M++ )
for ( N=1; N <= 10; N++ )
cout<<"This is a test"<<endl;
```

In this example, the outer loop is controlled by the variable M and the inner loop is controlled by the variable N. The inner loop body is the output statement. How many times does this statement execute? In other words, how many lines of output does this program fragment produce? Each time the inner loop executes, the output statement executes 10 times. But the inner loop executes 5 different times since it contained within the outer loop. Thus this code fragment produces 5*10 = 50 lines of output. That is quite a lot for such few statements, which is evidence of the power of nested loops. Here is a syntax question: Why are no braces required in this code fragment?

Activity 3.1: The following program is the skeleton of a program which is supposed to output to a file called **MultFile** the multiplication table for integers from 0 to 9. This program will require two FOR-loops, one nested inside the other. The control variables for these loops should be **`I`** and **`J`**, which have already been declared. You will be asked to complete this program in the next activity.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
//Program name: MultiplicationTable
int main(void)
{
   int I, J;
   fstream MultFile;
   MultFile.open("g:\\multfile.txt", ios::out);
   // begin outer for-loop (for rows)
   {
      // begin inner for-loop (for columns)
      {
         MultFile<<"Insert proper output here";
      }
      MultFile<<endl;
   }
   cin.get(); cin.get();
   return 0;
} // MultiplicationTable
```

Activity 3.2: Download **Lab9Tsk3.cpp** file from my webpage. Then complete the program **MultiplicationTable** so that it gives the output as shown below. The partial program can be found in the file **Lab9Tsk3.cpp**

MultiFile:

```
0*0=0 0*1=0 0*2=0 0*3=0 0*4=0 0*5=0 0*6=0 0*7=0 0*8=0 0*9=0
1*0=0 1*1=1 1*2=0 1*3=3 1*4=4 1*5=5 1*6=6 1*7=7 1*8=8 1*9=9
                                  *
                                  *
                                  *
9*0=0 9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=64 9*8=72 9*9=81
```

Task 4: Switch Statement

The purpose of this task is to demonstrate the SWITCH statement.

Recall that an IF-ELSE statement is used in programming to select between alternative lists of statments to execute. Because IF-ELSE statements are built around Boolean expressions, which can be either true or false, then an IF-ELSE statement can only be used to select between TWO alternative lists of statements. If a programming problem requires multiple alternatives (more than two), then NESTED IF-ELSE statements must be used. This can sometimes lead to confused or "messy" program structure. Therefore in C++, another type of selection statement can be used to select between any numbers of alternative lists of statements to execute. This is called a SWITCH statement. The general syntax outline for a SWITCH statement is given below:

```
switch ( expression )
{
   case expression value: list of statements; break;
   case expression value: list of statements; break;
     .
     .
     .
   case expression value: list of statements; break;
   default: list of statements;
}
```

The reserved words in this statement are "switch", "case", "break", and "default". The expression within the parentheses can evaluate as any type, not just Boolean. When a SWITCH statement executes, first the expression is evaluated. Then from top to bottom, the list of cases is scanned to find the value that matches the actual value of the expression within the parentheses. If such a case is found, then the list of statements for that case is executed. Afterwards, the program proceeds to the next statement after the SWITCH statement. If no matching case is found, then the default list of statements is executed. The SWITCH statement is not a loop, and at most one of the cases executes when a SWITCH statement executes. The default case is optional. For example, any IF-ELSE statement can be rewritten as SWITCH statement in the following way:

```
switch (Boolean expression)

{
   case true: list of statements; break;
   case false: list of statements; break;
}
```

Note that the list of statements for each case does not need to be a block, that is, it doesn't require braces. But the reserved word "break" is essential.

```
#include <iostream>
#include <iomanip>
using namespace std;
//Program name: TaskFour
int main(void)
{
   char CapLetter;
   for (CapLetter = 'V'; CapLetter >= 'Q'; CapLetter--)
      switch (CapLetter)
      {
         case 'R': cout << "Chuck is a ranger" << endl; break;
         case 'C': cout << "Carl is a carpenter" << endl; break;
         case 'V': cout << "Buffy is not a vampire" << endl; break;
         case 'S':
         case 'U': cout << "Sam is a student" << endl; break;
         case 'T': cout << "Tom is a teacher" << endl; break;
         default : cout<< "Out of range" << endl;
      }
   cin.get(); cin.get();
   return 0;
} // Task4
```

Activity 4.1: Study the following program carefully.

Activity 4.2: Predict the output from the program **TaskFour.**

Predicted Output:

Activity 4.3: Download **Lab9Tsk4.cpp** file from my webpage. Now run this program and observe the output. Explain any discrepancies between your predicted and observed outputs.

Obebserved Output:

Activity 4.4: Modify the program so that the For loop initial value is 'R' and the terminal value is also 'R'. Again predict the output from this program, and then run and observe the output. Explain any discrepancies between your predicted and observed outputs.

Predicted Output: ***Observed Output:***

Activity 4.5: Modify the program in **Activity 4.4** by removing all " break; " statements. Again predict the output from this program, and then run and observe the output. Explain any discrepancies between your predicted and observed outputs.

Predicted Output: ***Observed Output:***