Lab Session 07

Looping Constructs, do while Loop and Nested loop.

Objectives:

1. Illustration of Looping Constructs
2. Illustration of do while Loop and Nested loop.
3. Compile and execution of loop constructs program using C++.

do-loop:

Do-while loops are special-purpose and fairly rare. The main purpose of do-while loops is to make it easy to write a loop body that happens at least once. The structure is

```
do
{
// body...
} while (condition);
```

The condition is tested at the end of the loop body instead of the beginning; therefore, the body of the loop will be executed at least once before the condition is checked. If the condition is true, we jump back to the beginning of the block and execute it again. A do-while loop is basically a reversed while loop. A while loop says, "Loop while the condition is true, and execute this block of code", a do-while loop says, "Execute this block of code, and then loop back while the condition is true". Here's a simple example that lets a user enter the password until it is correct:

```
#include <string>
#include <iostream>
using namespace std;
int main ()

{
      string password;
      do
      {
            cout << "Please enter your password: ";
            cin >> password;
      } while (password != "foobar" );
      cout << "Welcome, you got the password right";
}
```

This loop will execute the body at least once, allowing the user to enter the password; if the password is incorrect, the loop will repeat, prompting the user for the password again until the user enters the correct password. Notice the trailing semi-colon after the while in the above example!

It's easy to forget to add the semicolon because the other loops do not require it; in fact, the other loops should not be terminated with a semicolon, adding to the confusion.
A do…while with no braces around the single statement body appears as

```
do
statement
while ( condition );
```

which can be confusing. You might misinterpret the last line—while(*condition*);—as a while statement containing as its body an empty statement. Thus, the do…while with one statement often is written as follows to avoid confusion:

```
do
{
statement
} while
```

Figure 5.7 uses a do…while statement to print the numbers 1–10. Upon entering the do…while statement, line 12 outputs counter's value and line 13 increments counter. Then the program evaluates the loop-continuation test at the bottom of the loop (line 14). If the condition is true, the loop continues from the first body statement in the do…while (line 12). If the condition is false, the loop terminates and the program continues with the next statement after the loop (line 16).

```
// do...while repetition statement.
#include <iostream>
using namespace std;
int main()
{
int counter = 1; // initialize counter
      do
      {
            cout << counter << " "; // display counter
            ++counter; // increment counter
      } while (counter <= 10 ); // end do...while
      cout << endl; // output a newline
} // end main

Output:    1 2 3 4 5 6 7 8 9 10
```

When to Use Which Loop

We've made some general statements about how loops are used. The for loop is appropriate when you know in advance how many times the loop will be executed. The while and do loops are used when you don't know in advance when the loop will terminate (the while loop when you may not want to execute the loop body even once, and the do loop when you're sure you want to execute the loop body at least once).

These criteria are somewhat arbitrary. Which loop type to use is more a matter of style than of hard-and-fast rules. You can actually make any of the loop types work in almost any situation. You should choose the type that makes your program the clearest and easiest to follow.