LABORATORY 3

Inquiring minds want to know about the if statement

Objective

As your knowledge of the C++ language grows, your capacity to write more useful and complex programs also increases. The `if` statement is a powerful tool for decision making. However, it can lead to considerable confusion if you do not fully understand its usage. This laboratory familiarizes you with the `if` statement, Boolean logic, and program decision making.

Key Concepts

- Debugging
- Stepping through a program
- Logical operations
- `if` statement

3.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab03.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

3.2 SYNTAX ERRORS AREN'T THE ONLY PROBLEM

There are two kinds of errors—syntax and logic. Code often compiles with no difficulty (i.e., there are no syntax errors). Does this mean that the code will work? Well, it will do *exactly* what you told it to do, but not necessarily what you want it to do. Careless programmers write programs that they think *look* correct and that compile just fine, but *do not* run properly (i.e., there are logic errors). This situation results in frustrated users.

Besides carelessness, another source of programmer logic errors is misunderstandings about the uses of particular language features and their effects on the control of the flow of execution in the program (what the program does and where it goes when you run it).

The `if` statement allows the programmer to make decisions. If some expression is true, one action can be taken. If the expression is false, another action can be executed. The `if` statement can definitely lead to problems. The following exercises will help you to better understand the `if` statement so that you can avoid some its more troublesome pitfalls.

- Start the compiler.
- Open the program `triangle.cpp` and create a default workspace when necessary. This program expects three numbers in nondecreasing order as input. It first determines whether the user correctly entered the numbers. If the user did not, the program terminates. If the instructions were followed, the program determines whether the numbers correspond to the sides of a triangle, and if they do, what kind of triangle.
- Examine the program to get an understanding of how it works.
- What output do you expect with the inputs `5 4 9` (in that order)? What output do you expect with the inputs `4 5 9` (in that order)? Record your answers. Build and run the program to verify your understanding.

Ordering	Expected Result	Actual Result
`5 4 9`		
`4 5 9`		

- As a post laboratory exercise provide additional test inputs that cause the other output messages to be displayed (i.e., inputs that correspond to the various kinds of triangles).

- Why do you think it is important to develop a set of test inputs so that every part of a program is executed at least once? ✓

- Close the workspace associated with `triangle.cpp`.

Next use the `if` statement to help compute the value of a standard Boolean operation.

- Open the file `nand.cpp` and create a default workspace when necessary. This program offers one solution to the following truth table for the Boolean nand operation, which is the negation of the and operation.

p	q	nand
false	false	true
false	true	true
true	false	true
true	true	false

- Run it on the four possible input combinations to verify its correctness.
- Observe that only the final combination—`p` and `q` both `true`—cause the nand result to be `false`. Restructure your code so that it uses a single `if` and a single `else`. (Test only whether the combination defined in the last row of the truth table is the one you have).
- Build and run the program. Once it is correct, show it to a laboratory instructor. If you cannot determine the proper test expression, ask for help. ✓

3.3 TRACKING DOWN PROBLEMS IN CODE

The debugger is a powerful utility to help you find software defects in your program—it helps you "get the bugs out."

- You are to continue to work with `nand.cpp`. Make sure it is the active window.
- The first feature you will explore is the Step feature. Stepping through a program enables you to watch the flow of control in your program.
- There are several methods of accessing the Step feature. One method is to select the Step Over button from the Debugger toolbar. (The Step Over button has an arrow point above a pair of curly braces). Another method is to press the `F10` key. Every time you do a Step Over, the IDE will execute one more line of code. The IDE will use a colored arrow to indicate which line is to be executed next.
- Step all the way to the curly brace that ends the body of function `main()` in `nand.cpp`. Note that before entering input you may have to first click in the console window. After entering the input, press `F10` to get to the next statement.
- Pressing `F10` once you reach the closing curly brace terminates the program and causes the console window to disappear.

- Another feature the debugger offers is the Watches option. By using Watches, you are able to observe the value of objects at different points in the program. Restart the program using the debugger. Observe there are two windows at the bottom of the IDE. The Variable window displays the values of variables that are currently being used. The Watches window can be used to track specific variables throughout the course of the program. By highlighting a variable and dragging it to the Watch window, you will start a watch on that variable.

- Watch variables `p` and `q`, and step through the program one line at a time. Observe that once program execution begins and even before the objects are initialized in an assignment statement, that the objects have a value. The values are the ones left from some previous use of the computer memory. Also, observe that the Watch box shows the values of both `p` and `q` as they change during the program's execution.

A third feature that the IDE provides is the Breakpoint option. By inserting a breakpoint, you can run your program in the usual fashion (e.g., by choosing Run). When the program reaches the statement where you have inserted a breakpoint, it pauses its execution and awaits your commands. Breakpoints are useful because they enable you to run large programs until they reach the place in the code that you want to observe. Let's insert a breakpoint into the `nand.cpp` code.

- Use your mouse to select the line in the program that contains the first `if` statement. Right click and select the Insert/Remove Breakpoint option from the context menu. A colored indicator appears on the line you selected. A breakpoint can also be toggled through the function key `F9`. If you repeat the action, the breakpoint will be toggled off.

- Restart the program by pressing `CTL-F5`. (Restarting is also available from the Debug menu). This action causes the program to run until it reaches a statement with a breakpoint. Observe that execution stops at the breakpoint you inserted. At this point you have several options.

  — Add or remove watches.

  — Press `F10` to execute the next statement.

  — Press `F5` to continue the normal execution.

  — Terminate the process to start over by entering `SHIFT-F5`. (This option is also available under the Debug menu).

  — Remove the breakpoint by toggling the statement again.

- You are to remove the Watch on object `q`. To do so, right click on `q` in the Watch Window and press `Delete`. Continue the program execution by pressing `F5`. ✓

- Close the workspace for `nand.cpp`.

3.4 A SIMPLE PREDICTION

- Consider the following program from the file `nested.cpp`. What do the inputs need to be for the strings `"1"`, `"2"`, `"3"`, `"4"`, or `"5"` to be displayed?

```
int main() {
    cout << "Enter 3 logical values (e.g., 0 1 0): ";
    bool p;
    bool q;
    bool r;
    cin >> p >> q >> r;
    if (p && q) {
        if (r) {
            cout << "1" << endl;
        }
        else {
            cout << "2" << endl;
        }
    }
    else if (q && r) {
        cout << "3" << endl;
    }
    else {
        if (p || !r) {
            cout << "4" << endl;
        }
        else {
            cout << "5" << endl;
        }
    }
    return 0;
}
```

- Record your answers in the following table. Show two ways for `"4"` to be displayed. Are there others?

p	q	r	output
			1
			2
			3
			4
			4
			5

- Open the file `nested.cpp` and create a default workspace. Try your inputs. If they do not work, modify them. Show your successful inputs to the laboratory instructor. ✓
- Close the workspace for n`ested.cpp`.

3.5 A COMMON MISTAKE

- Consider the following program. What is it output?

```
int main() {
    int x = 0;
    if (x = 0) {
        cout << "true" << endl;
    }
    else {
        cout << "false" << endl;
    }
    return 0;
}
```

- Record your answer in the following table.

Your Answer	Program Output

- Open the file `zero.cpp`. and create a default workspace. Step through the program. Record the output in the preceding table. What is happening in this program? Do you believe such code should successfully compile? ✓

3.6 MAKING THE RIGHT CHOICE

Close any open program files. Open the project `overlap.dsw`. This program is to perform the following actions.

— Prompt its user for the sizes and positions of two rectangles.

— Indicate to the user whether the upper left corner of the second rectangle lies within the first rectangle.

— Display the two rectangles.

The program uses objects

— `cx1` and `cy1`, and `cx2` and `cy2` to represent the two rectangle centers

— `l1` and `h1`, and `l2` and `h2` to represent the dimensions of the two rectangles

— `ux1` and `uy1`, and `ux2` and `uy2` to represent the upper left corners of the two rectangles

— `lx1` and `ly1` to represent the lower right corner of the first rectangle.

The upper left corner of the second rectangle lies within the first rectangle if the following conditions are true.

— `ux1` ≤ `ux2` ≤ `lx1`

— `uy1` ≤ `uy2` ≤ `ly1`

Take note that the preceding expressions require some care in being translated into C++.

- Complete the program by developing the following code.
  — Initialize the objects `lx1` and `ly1`.
  — Provide the test expression for the `if` statement.
- Test your program using a variety of inputs. Show your completed work to the laboratory instructor. ✓

3.7 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.