Lab 2: Branching

Relational Operators

Use comparison and relational operators to determine whether a condition is true or false.

- **==** is the "equal to" operator
  ex. 1 == 1 returns true; 1 == 2 returns false; 6+2 == 8 returns true

- **!=** is the "not equal to" operator
  ex. 1 != 0 returns true; 1 != 1 returns false; 3-2 != 2 returns true;

- **>** is the "greater than" operator
  ex. 1 > 0 returns true; 0 > 1 returns false; 1+1 > 0 returns true;

- **<** is the "less than" operator
  ex. 0 < 1 returns true; 100 < 50 returns false; 'a' < 'b' returns true;

- **>=** is the "greater than or equal to" operator
  ex. 0 >= -1 returns true; 0 >= 0 returns true; 6+2-1 >= 9 returns false;

- **<=** is the "less than or equal to" operator
  ex. 0 <= -1 returns false; 0 <= 0 returns true; 6+2-1 <= 9 returns true;

If Statements

```
#include<iostream>

using namespace std;

int main()
{
        int num1 = 10;
        int num2 = 5;
        int num3 = 20;
        int num4 = 30;

      // Here is a simple if statement using curly brackets.
      // Good style dictates that we should indent everything
      // inside of the if statement (PLEASE DO THIS).

        if (num1 < num4)  // (10 < 30)
        {
                cout << num1 << " is less than " << num4 << endl;
                num4 += 5;
                cout << "num 4 is now " << num4 << endl << endl;
        }

        // if you only have one statement inside the IF statement you
        // do not need curly braces

        if (num1 > num3)   // (10 > 20)
                cout << num1 << " is greater than " << num3 << endl;


        // you can also fine tune your if statements by nesting them

        if (num1 != num3)  // (10 != 20)
        {
                cout << num1 << " is not equal to " << num3;

                if(num1 < num2)  // (10 < 5)
                {
                        cout << " and " << num1 << " is less than " << num2
                            << endl << endl;
                }

                if(num1 > num2)  // (10 > 5)
                {
                        cout << " and " << num1 << " is greater than " << num2
                            << endl << endl;
                }
        }

        return 0;
}
```

IF statements help to control the flow of a program. An if statement will execute its code only if the condition is true.

Syntax

```
if (condition)
{
// statements inside braces run only if the condition is true
}
```

```
#include<iostream>

using namespace std;

int main()
{
	int num1 = 10;
	int num3 = 20;

	// Do NOT put a semicolon after your IF statement.
	// This mistake will cause your IF statement to not work.
	if (num1 == num3);
		cout << num1 << " is equal to " << num3 << endl << endl;

	// Another common mistake.
	// = is the assignment operator.
	// == is the comparison operator.
	if (num3 = num1)
	{
		cout << num3 << " is equal to " << num1 << endl << endl;
	}

	// Another tip is to be very careful with your braces. It is very
	// easy to forget one, or add an extra brace on accident. This is
	// where using good style is crucial to helping you avoid wasting
	// time debugging syntax errors.

	return 0;
}
```

Do not fall prey to these common mistakes. You should never have a semicolon after an if statement, and make sure to never use the assignment operator in a condition.

Else If Statements

```
#include <iostream>
using namespace std;

int main()
{
	// Example, all are true but only the first will execute.

	if (1 == 1)
	{
		cout << "First" << endl << endl;
	}
	else if (2 == 2)
	{
		cout << "Second" << endl << endl;
	}
	else if (3 == 3)
	{
		cout << "Third" << endl << endl;
	}

	return 0;
}
```

The ELSE IF statement are pretty straight forward. If your IF statement is false, then it will check the following ELSE IF statement's condition. This will repeat until one of the statement's condition is true, or if they are all false none of the statements will be executed. However, if several conditions are true, it will only execute the first true condition it comes to, skipping the rest.

Else Statements

```
#include <iostream>
using namespace std;

int main()
{
	// ELSE Statements

	// the "else" statement will only execute if
	// all of the previous IF and ELSE IF statements
	// were false.

	if (100 <= 4)
	{
		cout << "will never execute" << endl << endl;
	}
	else
	{
		cout << "Inside Else" << endl << endl;
	}

	// Here is an example of IF-ELSE IF statemnts

	int grade = 95;
	char lettergrade;

	if (grade < 60)
	{
		lettergrade = 'f';
	}
	else if (grade < 70)
	{
		lettergrade = 'd';
	}
	else if (grade < 80)
	{
		lettergrade = 'c';
	}
	else if (grade < 90)
	{
		lettergrade = 'b';
	}
	else
	{
		lettergrade = 'a';
	}

	cout << "Your final grade is: " << lettergrade << endl << endl;
	return 0;
}
```

The "else" statement will always (and only) execute if all of the previous IF and ELSE IF statements were false.

Logical Operators

```
#include <iostream>
using namespace std;

int main()
{

	// && and || and ! are logical operators and can be used to evaluate
	// logical expressions.


	// && - Logical AND operator will return true only when both sides are true

	// true  && true  = true
	// true  && false = false
	// false && true  = false
	// false && false = false


	if ( 1 < 3 && 5 == 5)
		cout << "both expressions are true" << endl << endl;


	// || - Logical OR operator will return true if either of the sides is true

	// true  || true  = true
	// true  || false = true
	// false || true  = true
	// false || false = false

	if ( 10 == 5 || 1 != 0)
		cout << "one or both of the expressions are true" << endl << endl;

	// ! - NOT operator will flip the logical value.

	// !true  = false
	// !false = true

	if ( !(1 == 5) )
		cout << "NOT operator flipped a false operation to true." << endl << endl;

	return 0;
}
```

Checking for Invalid User Input

This is a common use for logical operators, however many students find this concept confusing. In this section, we will walk through the process of creating an appropriate conditional statement to check for invalid user input.

For example, say we only want to take the letters A, B, C, and D from the user. How would we go about doing this?

A common mistake students make is constructing their if statements in the following fashion:

```
if( user_input != ‘A’ || ‘B’ || ‘C’ )

{

cout << “Invalid input”;

}
```

There are a few things wrong with this approach. First of all, although this compiles, it will not give you the appropriate output. For each part of the conditional statement, you must compare each “option” to user_input. Once we correct this, we end up with the following:

```
if( user_input != ‘A’ || user_input != ‘B’ || user_input != ‘C’ )
```

This statement seems all well and good, but we still end up with the incorrect output when the user enters a valid option. This is because there is a logical error in the above statement.

Well, let’s think about what the OR (||) logical operator means. If one condition is true, then the entire conditional is true. So if the user chose A, the section user_input != ‘A’ is false, but the others are true therefore the overall conditional is true. Then, what we really want is to use AND (&&) because if one of these section is false, the overall conditional is false. Fixing these problems, we end up with the following correct statement:

```
if( user_input != ‘A’ && user_input != ‘B’ && user_input != ‘C’ )
```

File Error Detecting

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
	ifstream fin;
	fin.open("results.txt");

	if (!fin)
	{
		cout << "ERROR - Failed to open file. Terminating program.\n";
		return -1;  // terminate program
	    //NOTE: This is the ONLY time it is valid to terminate the program with a premature
return statement.
	}

	fin.close();

	return 0;
}
```

On a final note. Always use an if statement to check if you successfully opened a file or not. If the file successfully opens it will return 'true', otherwise 'false'. Therefore, we can use the NOT operator to detect "if NOT successful" and terminate the program.