Review

Relational Operators

You have already seen that the statement `total=5` is an assignment statement; that is, the integer 5 is placed in the variable called `total`. Nothing relevant to our everyday understanding of equality is present here. So how do we deal with equality in a program? How about greater than or less than? C++ allows the programmer to compare numeric values using **relational operators**. They are the following:

>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to
= =	Equal to
! =	Not equal to

An expression of the form `num1 > num2` is called a **relational expression**. Note that it does *not* assert that `num1` is greater than `num2`. It actually tests to see if this is true. So relational expressions are boolean. Their value must be either *true* or *false*. The statement `cost!=9` is false if `cost` has value 9 and true otherwise. Consider the following code:

```
int years;
years = 6;        // assignment statement years is assigned the value of 6
years == 5;       // relational expression, not an assignment statement
years = years - 1;      // assignment statement
years == 5;             // relational expression
```

In this sequence the first occurrence of `years==5` is a false statement whereas the second occurrence is true. Can you see why?

The `if` Statement

Sometimes we may only want a portion of code executed under certain conditions. To do so, we use **conditional statements**. For example, if you are writing a payroll program to compute wages, then the program should only compute overtime pay *if* the employee worked more than 40 hours in a given week. Otherwise, when the program is executed the overtime portion of the code should be bypassed. An **`if` statement** is one kind of conditional statement.

Consider the following program:

Sample Program 4.1:

```
// This program prints "You Pass" if a student's average is 60 or higher and
// prints "You Fail" otherwise

#include <iostream>
using namespace std:

int main()
{
    float average;

    cout << "Input your average" << endl;
    cin >> average;
```

```
    if (average >= 60)       // note the use of a relational operator
        cout << "You Pass" << endl;

    if (average < 60)
        cout << "You Fail" << endl;

    return 0;
}
```

Note that it is not possible for this program to print out both "You Pass" and "You Fail". Only one of the `if` statements will be executed. Later we will see a way to write this program without using 2 `if` statements.

If you want to conditionally execute several statements using `if`, the following syntax is required:

```
if (expression)
{
    statement_1;
    statement_2;
      .
      .
    statement_n;
}
```

Note the curly braces surrounding the set of statements to be conditionally executed.

The `if/else` Statement

In Sample Program 4.1 we used two `if` statements. A more elegant approach would be to use the **`if/else` statement** as follows:

```
if (average >= 60)
    cout << "You Pass" << endl;
else
    cout << "You Fail" << endl;
```

In every `if/else` statement the program can take only one of two possible paths. Multiple statements can be handled using curly braces in the same way as the `if` statement.

The `if/else if` Statement

The `if/else` statement works well if there are only two possible paths to follow. However, what if there are more than two possibilities? For example, suppose we need to decide what kind of vacation to take based on a yearly work bonus:

```
if the bonus is less than $1,000, we set up a tent and eat hot dogs in the back yard
if the bonus is less than $10,000 and greater than or equal to $1,000, we go to Disney World
if the bonus is $10,000, we go to Hawaii
```

We could code this using the **`if/else if`** statement as follows:

```
float bonus;

cout << "Please input the amount of your yearly bonus" << endl;
cin >> bonus;
```

```
if (bonus < 1000)
    cout << "Another vacation eating hot dogs on the lawn" << endl;

else if (bonus < 10000)
    cout << "Off to Disney World!" << endl;

else if (bonus == 10000)
    cout << "Lets go to Hawaii!" << endl;
```

Can you explain why the first `else if` conditional statement does not require a greater than or equal to 1000 condition?

In general we can use as many `else if` expressions as needed to solve a given problem.

The Trailing `else`

What happens in the code above if the bonus entered is greater than $10,000? Actually, nothing will happen since none of the conditional expressions are true in this case. Sometimes it is advantageous to add a final or **trailing `else`** at the end of a chain of `if/else if` statements to handle "all other cases." For example, we could modify the code to read:

```
if (bonus < 1000)
   cout << "Another vacation on the lawn" << endl;
else if (bonus < 10000)
   cout << "Off to Disney World!" << endl;
else if (bonus == 10000)
   cout << "Lets go to Hawaii!" << endl;
else
{
   cout << bonus << " is not a valid bonus" << endl;
   cout << "Please run the program again with valid data" << endl;
} // Note the necessary use of the curly brackets here
```

Of course, few would complain about a bonus greater than $10,000 and the Hawaii trip could still be done on this budget. However, if the maximum possible bonus is $10,000, then the trailing `else` will let the user know that an illegal value has been entered.

Nested `if` Statements

Often programmers use an `if` statement within another `if` statement. For example, suppose a software engineering company wants to screen applicants first for experienced programmers and second for C++ programmers specifically. One possible program is the following:

Sample Program 4.2:

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

int main()
{
    char programmer, cPlusPlus;

    cout << "Before we consider your application, answer the following"
         << endl;
    cout << " yes ( enter Y ) or no ( enter N )" << endl;
    cout << "Are you a computer programmer?" << endl;

    cin >> programmer;
```

```
    if (programmer == 'Y')
    {
        cout << "Do you program in C++?" << endl;
        cin >> cPlusPlus;

        if (cPlusPlus == 'Y')
            cout << " You look like a promising candidate for employment"
                 << endl;
        else if (cPlusPlus == 'N')
            cout << " You need to learn C++ before further consideration"
                 << endl;
        else
            cout << " You must enter Y or N" << endl;
    }

    else if (programmer == 'N')
        cout << " You are not currently qualified for employment" << endl;

    else
        cout << " You must enter Y or N" << endl;

    return 0;

}
```

Note how C++ programmers are identified using a nested `if` statement. Also note how the trailing `else` is used to detect invalid input.

Logical Operators

By using relational operators C++ programmers can create relational expressions. Programmers can also combine truth values into a single expression by using **logical operators**. For example, instead of a statement such as "if it is sunny, then we will go outside," one may use a statement such as "if it is sunny and it is warm, then we will go outside." Note that this statement has two smaller statements "it is sunny" and "it is warm" joined by the **AND** logical operator. To evaluate to `true`, both the sunny and warm requirements must be met.

The **NOT** operator negates a single statement. For example, "it is sunny" can be negated by "it is not sunny."

The **OR** operator is similar to the AND in that it connects two statements. However, there is an ambiguity about the meaning of the word *or* in English. In the statement "tonight at 8:00 I will go to the concert in the park or I will go to the stadium to see the ball game," the word **or** is *exclusive*. That is, I can go to the concert or to the game, but not both. However, in the statement "I need to draw an ace or a king to have a good poker hand," the word **or** is *inclusive*. In other words, I can draw a king, an ace, or even both, and I will have a good hand. So we have a choice to make. Let A and B be two statements. A OR B could mean A or B but not both. It could also mean A or B or both. In computer science we use the second meaning of the word *or*. For example, in the statement "if it is sunny or it is warm, then I will go outside," there are three scenarios where I will go outside: if it is sunny but not warm, if it is warm but not sunny, or if it is sunny and warm.

The syntax used by C++ for logical operators is the following:

AND	&&
OR	||
NOT	!

Consider the following:

```
if (dollars <= 0 || !(accountActive) )
    cout << " You may not withdraw money from the bank";
```

It is good programming practice to enclose the operand after the (!) operator in parentheses. Unexpected things can happen in complicated expressions if you do not. When will this code execute the `cout` statement? What type of variable do you think `accountActive` is?

The `switch` Statement

We have already seen how `if` statements can affect the branching of a program during execution. Another way to do this is using the **`switch` statement**. It is also a conditional statement. The `switch` statement uses the value of an integer expression to determine which group of statements to branch through. The sample program below illustrates the syntax.

Sample Program 4.3:

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

int main()
{
    char grade;

    cout << "What grade did you earn in Programming I?" << endl;
    cin >> grade;

    switch( grade ) // This is where the switch statement begins
    {
        case 'A':cout << "an A - excellent work!" << endl;
             break;
        case 'B':cout << "you got a B - good job" << endl;
             break;
        case 'C':cout << "earning a C is satisfactory" << endl;
             break;
        case 'D':cout << "while D is passing, there is a problem" << endl;
             break;
        case 'F':cout << "you failed - better luck next time" << endl;
             break;
        default:cout << "You did not enter an A, B, C, D, or F" << endl;
    }

    return 0;

}
```

Note the use of the curly braces that enclose the cases and the use of `break`; after each `case`. Also, consider the variable `grade`. It is defined as a character data type and the case statements have character arguments such as `'B'`. This seems to contradict what we said above, namely that the `switch` statement uses the value of integer expressions to determine branching. However, this apparent contradiction is resolved by the compiler automatically converting character data into the integer data type. Finally, notice the role of the `default` statement. The `default` branch is followed if none of the case expressions match the given `switch` expression.

Character & string comparisons

So far, relational operators have been used to compare numeric constants and variables. Characters and string objects can also be compared with the same operators. For example:

```
char letter = 'F';
string word = "passed";

switch(letter)
{
    case 'A': cout << "Your grade is A." << endl;
         break;
    case 'B': cout << "Your grade is B." << endl;
         break;
    case 'C: cout << "Your grade is C." << endl;
         break;
    case 'D': cout << "Your grade is D." << endl;
         break;
    case 'F': word = "failed";
         break;
    default: cout << "You did not enter an A,B,C,D or F" << endl;
}

if (word == "passed")
    cout << "You passed" << endl;
else
    cout << "You failed" << endl;
```

What is printed ?

Review #4 CS 151 SEC# _____ Name _____

Fill-in-the-Blank Questions

1. The two possible values for a relational expression are _____ and _____.

2. C++ uses the _____ symbol to represent the AND operator.

3. The `switch` statement and `if` statements are examples of _____statements.

4. In C++ is the meaning of the OR logical operator inclusive or exclusive? _____

5. C++ uses the _____ symbol to represent the OR operator.

6. It is good programming practice to do what to the operand after the NOT operator?

   _____

7. The `switch` statement uses the value of a(n) _____ expression to determine which group of statements to branch through.

8. In a `switch` statement the _____ branch is followed if none of the case expressions match the given `switch` expression.

9. C++ allows the programmer to compare numeric values using _____.

10. The C++ symbol for equality is _____.