Integer Flags

Integer variables may also be used as flags. This is because in C++ the value 0 is considered false, and any nonzero value is considered true. In the sales commission program previously described, we could define the `salesQuotaMet` variable with the following statement:

```
int salesQuotaMet = 0;       // 0 means false.
```

As before, we initialize the variable with 0 because we do not yet know if the sales quota has been met. After the sales have been calculated, we can use code similar to the following to set the value of the `salesQuotaMet` variable:

```
if (sales >= QUOTA_AMOUNT)
   salesQuotaMet = 1;
else
   salesQuotaMet = 0;
```

Later in the program we might test the flag in the following way:

```
if (salesQuotaMet)
   cout << "You have met your sales quota!\n";
```

4.8 Logical Operators

CONCEPT: Logical operators connect two or more relational expressions into one or reverse the logic of an expression.

In the previous section you saw how a program tests two conditions with two `if` statements. In this section you will see how to use logical operators to combine two or more relational expressions into one. Table 4-6 lists C++'s logical operators.

Table 4-6

Operator	Meaning	Effect
`&&`	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
`||`	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
`!`	NOT	The ! operator reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

The `&&` Operator

The `&&` operator is known as the logical AND operator. It takes two expressions as operands and creates an expression that is true only when both sub-expressions are true. Here is an example of an `if` statement that uses the `&&` operator:

```
if (temperature < 20 && minutes > 12)
   cout << "The temperature is in the danger zone.";
```

In the statement above the two relational expressions are combined into a single expression. The `cout` statement will only be executed if `temperature` is less than 20 AND `minutes` is greater than 12. If either relational test is false, the entire expression is false, and the `cout` statement is not executed.

TIP: You must provide complete expressions on both sides of the `&&` operator. For example, the following is not correct because the condition on the right side of the `&&` operator is not a complete expression.

```
temperature > 0 && < 100
```

The expression must be rewritten as

```
temperature > 0 && temperature < 100
```

Table 4-7 shows a truth table for the `&&` operator. The truth table lists all the possible combinations of values that two expressions may have, and the resulting value returned by the `&&` operator connecting the two expressions.

Table 4-7

Expression	Value of Expression
`true && false`	`false (0)`
`false && true`	`false (0)`
`false && false`	`false (0)`
`true && true`	`true  (1)`

As the table shows, both sub-expressions must be true for the `&&` operator to return a true value.

NOTE: If the sub-expression on the left side of an `&&` operator is false, the expression on the right side will not be checked. Since the entire expression is false if only one of the sub-expressions is false, it would waste CPU time to check the remaining expression. This is called *short circuit evaluation*.

The `&&` operator can be used to simplify programs that otherwise would use nested `if` statements. Program 4-15 performs a similar operation as Program 4-11, which qualifies a bank customer for a special interest rate. This program uses a logical operator.

Program 4-15

```
// This program demonstrates the && logical operator.
#include <iostream>
using namespace std;

int main()
{
    char employed,    // Currently employed, Y or N
         recentGrad; // Recent graduate, Y or N

    // Is the user employed and a recent graduate?
    cout << "Answer the following questions\n";
    cout << "with either Y for Yes or N for No.\n";

```

(program continues)

Program 4-15 *(continued)*

```
      cout << "Are you employed? ";
      cin >> employed;

      cout << "Have you graduated from college "
           << "in the past two years? ";
      cin >> recentGrad;

      // Determine the user's loan qualifications.
      if (employed == 'Y' && recentGrad == 'Y')
      {
         cout << "You qualify for the special "
              << "interest rate.\n";
      }
      else
      {
         cout << "You must be employed and have\n"
              << "graduated from college in the\n"
              << "past two years to qualify.\n";
      }
      return 0;
  }
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? **Y [Enter]**
Have you graduated from college in the past two years? **N [Enter]**
You must be employed and have
graduated from college in the
past two years to qualify.
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? **N [Enter]**
Have you graduated from college in the past two years? **Y [Enter]**
You must be employed and have
graduated from college in the
past two years to qualify.
```

Program Output with Example Input Shown in Bold

```
Answer the following questions
with either Y for Yes or N for No.
Are you employed? **Y [Enter]**
Have you graduated from college in the past two years? **Y [Enter]**
You qualify for the special interest rate.
```

The message "`You qualify for the special interest rate`" is displayed only when both the expressions `employed == 'Y'` and `recentGrad == 'Y'` are true. If either of these is false, the message "`You must be employed and have graduated from college in the past two years to qualify.`" is printed.

NOTE: Although it is similar, Program 4-15 is not the logical equivalent of Program 4-11. For example, Program 4-15 doesn't display the message "You must be employed to qualify."

The || Operator

The || operator is known as the logical OR operator. It takes two expressions as operands and creates an expression that is true when either of the sub-expressions are true. Here is an example of an `if` statement that uses the || operator:

```
if (temperature < 20 || temperature > 100)
   cout << "The temperature is in the danger zone.";
```

The `cout` statement will be executed if `temperature` is less than 20 OR `temperature` is greater than 100. If either relational test is true, the entire expression is true and the `cout` statement is executed.

TIP: You must provide complete expressions on both sides of the || operator. For example, the following is not correct because the condition on the right side of the || operator is not a complete expression.

```
temperature < 0 || > 100
```

The expression must be rewritten as

```
temperature < 0 || temperature > 100
```

Table 4-8 shows a truth table for the || operator.

Table 4-8

Expression	Value of the Expression
`true || false`	`true  (1)`
`false || true`	`true  (1)`
`false || false`	`false (0)`
`true || true`	`true  (1)`

All it takes for an OR expression to be true is for one of the sub-expressions to be true. It doesn't matter if the other sub-expression is false or true.

NOTE: The || operator also performs short circuit evaluation. If the sub-expression on the left side of an || operator is true, the expression on the right side will not be checked. Since it's only necessary for one of the sub-expressions to be true, it would waste CPU time to check the remaining expression.

Program 4-16 performs different tests to qualify a person for a loan. This one determines if the customer earns at least $35,000 per year, or has been employed for more than five years.

Program 4-16

```cpp
// This program demonstrates the logical || operator.
#include <iostream>
using namespace std;

int main()
{
    // Constants for minimum income and years
    const double MIN_INCOME = 35000.0;
    const int MIN_YEARS = 5;

    double income; // Annual income
    int years;     // Years at the current job

    // Get the annual income
    cout << "What is your annual income? ";
    cin >> income;

    // Get the number of years at the current job.
    cout << "How many years have you worked at "
         << "your current job? ";
    cin >> years;

    // Determine the user's loan qualifications.
    if (income >= MIN_INCOME || years > MIN_YEARS)
        cout << "You qualify.\n";
    else
    {
        cout << "You must earn at least $"
             << MIN_INCOME << " or have been "
             << "employed more than " << MIN_YEARS
             << " years.\n";
    }
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 40000 [Enter]
How many years have you worked at your current job? 2 [Enter]
You qualify.
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 20000 [Enter]
How many years have you worked at your current job? 7 [Enter]
You qualify.
```

Program Output with Example Input Shown in Bold

```
What is your annual income? 30000 [Enter]
How many years have you worked at your current job? 3 [Enter]
You must earn at least $35000 or have been employed more than 5 years.
```

The message "`You qualify\n.`" is displayed when either or both the expressions `income >= 35000` or `years > 5` are true. If both of these are false, the disqualifying message is printed.

The `!` Operator

The `!` operator performs a logical NOT operation. It takes an operand and reverses its truth or falsehood. In other words, if the expression is true, the `!` operator returns false, and if the expression is false, it returns true. Here is an `if` statement using the `!` operator:

```
if (!(temperature > 100))
   cout << "You are below the maximum temperature.\n";
```

First, the expression `(temperature > 100)` is tested to be true or false. Then the `!` operator is applied to that value. If the expression `(temperature > 100)` is true, the `!` operator returns false. If it is false, the `!` operator returns true. In the example, it is equivalent to asking "is the temperature not greater than 100?"

Table 4-9 shows a truth table for the `!` operator.

Table 4-9

Expression	Value of the Expression
`!true`	`false (0)`
`!false`	`true  (1)`

Program 4-17 performs the same task as Program 4-16. The `if` statement, however, uses the `!` operator to determine if the user does *not* make at least $35,000 or has *not* been on the job more than five years.

Program 4-17

```
// This program demonstrates the logical ! operator.
#include <iostream>
using namespace std;

int main()
{
    // Constants for minimum income and years
    const double MIN_INCOME = 35000.0;
    const int MIN_YEARS = 5;

   double income; // Annual income
   int years;     // Years at the current job

   // Get the annual income
   cout << "What is your annual income? ";
   cin >> income;

   // Get the number of years at the current job.
   cout << "How many years have you worked at "
        << "your current job? ";
```

(program continues)

Program 4-17 *(continued)*

```
    cin >> years;

    // Determine the user's loan qualifications.
    if (!(income >= MIN_INCOME || years > MIN_YEARS))
    {
       cout << "You must earn at least $"
            << MIN_INCOME << " or have been "
            << "employed more than " << MIN_YEARS
            << " years.\n";
    }
    else
       cout << "You qualify.\n";
    return 0;
}
```

The output of Program 4-17 is the same as Program 4-16.

Precedence and Associativity of Logical Operators

Table 4-10 shows the precedence of C++'s logical operators, from highest to lowest.

Table 4-10

Logical Operators in Order of Precedence
`!`
`&&`
`||`

The `!` operator has a higher precedence than many of the C++ operators. To avoid an error, you should always enclose its operand in parentheses unless you intend to apply it to a variable or a simple expression with no other operators. For example, consider the following expressions:

```
!(x > 2)
!x > 2
```

The first expression applies the `!` operator to the expression `x > 2`. It is asking, "Is `x` not greater than 2?" The second expression, however, applies the `!` operator to `x` only. It is asking, "Is the logical negation of `x` greater than 2?" Suppose `x` is set to 5. Since 5 is nonzero, it would be considered true, so the `!` operator would reverse it to false, which is 0. The > operator would then determine if 0 is greater than 2. To avoid a catastrophe like this, always use parentheses!

The `&&` and `||` operators rank lower in precedence than the relational operators, so precedence problems are less likely to occur. If you feel unsure, however, it doesn't hurt to use parentheses anyway.

`(a > b) && (x < y)`	is the same as	`a > b && x < y`
`(x == y) || (b > a)`	is the same as	`x == y || b > a`