4.13 The Conditional Operator

CONCEPT: **You can use the conditional operator to create short expressions that work like `if/else` statements.**

The conditional operator is powerful and unique. It provides a shorthand method of expressing a simple `if/else` statement. The operator consists of the question-mark (?) and the colon (:). Its format is:

```
expression ? expression : expression;
```

Here is an example of a statement using the conditional operator:

```
x < 0 ? y = 10 : z = 20;
```

The statement above is called a *conditional expression* and consists of three sub-expressions separated by the ? and : symbols. The expressions are `x < 0`, `y = 10`, and `z = 20`, as illustrated here:

```
x < 0   ?   y = 10   :   z = 20;
```

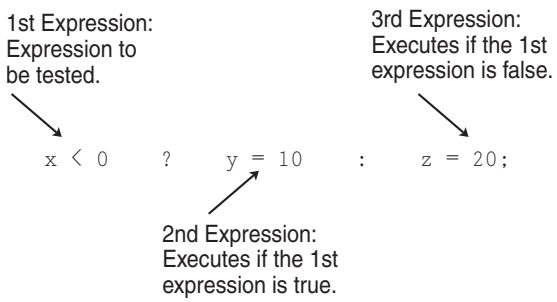
NOTE: Since it takes three operands, the conditional operator is considered a *ternary* operator.

The conditional expression above performs the same operation as the following `if/else` statement:

```
if (x < 0)
   y = 10;
else
   z = 20;
```

The part of the conditional expression that comes before the question mark is the expression to be tested. It's like the expression in the parentheses of an `if` statement. If the expression is true, the part of the statement between the ? and the : is executed. Otherwise, the part after the : is executed. Figure 4-10 illustrates the roles played by the three sub-expressions.

Figure 4-10



If it helps, you can put parentheses around the sub-expressions, as in the following:

```
(x < 0) ? (y = 10) : (z = 20);
```

Using the Value of a Conditional Expression

Remember, in C++ all expressions have a value, and this includes the conditional expression. If the first sub-expression is true, the value of the conditional expression is the value of the second sub-expression. Otherwise it is the value of the third sub-expression. Here is an example of an assignment statement using the value of a conditional expression:

```
a = x > 100 ? 0 : 1;
```

The value assigned to `a` will be either 0 or 1, depending upon whether `x` is greater than 100. This statement could be expressed as the following `if/else` statement:

```
if (x > 100)
   a = 0;
else
   a = 1;
```

Program 4-22 can be used to help a consultant calculate her charges. Her rate is $50.00 per hour, but her minimum charge is for five hours. The conditional operator is used in a statement that ensures the number of hours does not go below five.

Program 4-22

```
1  // This program calculates a consultant's charges at $50
2  // per hour, for a minimum of 5 hours. The ?: operator
3  // adjusts hours to 5 if less than 5 hours were worked.
4  #include <iostream>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10     const double PAY_RATE = 50.0; // Hourly pay rate
11     const int MIN_HOURS = 5;      // Minimum billable hours
12     double hours,                 // Hours worked
13            charges;               // Total charges
14
15     // Get the hours worked.
16     cout << "How many hours were worked? ";
17     cin >> hours;
18
19     // Determine the hours to charge for.
20     hours = hours < MIN_HOURS ? MIN_HOURS : hours;
21
22     // Calculate and display the charges.
23     charges = PAY_RATE * hours;
24     cout << fixed << showpoint << setprecision(2)
25          << "The charges are $" << charges << endl;
26     return 0;
27 }
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 10 [Enter]
The charges are $500.00
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 2 [Enter]
The charges are $250.00
```

Notice that in line 11 a constant named `MIN_HOURS` is defined to represent the minimum number of hours, which is 5. Here is the statement in line 20, with the conditional expression:

```
hours = hours < MIN_HOURS ? MIN_HOURS : hours;
```

If the value in `hours` is less than 5, then 5 is stored in `hours`. Otherwise `hours` is assigned the value it already has. The `hours` variable will not have a value less than 5 when it is used in the next statement, which calculates the consultant's charges.

As you can see, the conditional operator gives you the ability to pack decision-making power into a concise line of code. With a little imagination it can be applied to many other programming problems. For instance, consider the following statement:

```
cout << "Your grade is: " << (score < 60 ? "Fail." : "Pass.");
```

If you were to use an `if/else` statement, the statement above would be written as follows:

```
if (score < 60)
   cout << "Your grade is: Fail.";
else
   cout << "Your grade is: Pass.";
```

NOTE: The parentheses are placed around the conditional expression because the `<<` operator has higher precedence than the `?:` operator. Without the parentheses, just the value of the expression `score < 60` would be sent to `cout`.

Checkpoint

4.24 Rewrite the following `if/else` statements as conditional expressions:

A)
```
if (x > y)
      z = 1;
   else
      z = 20;
```

B)
```
if (temp > 45)
      population = base * 10;
   else
      population = base * 2;
```

C)
```
if (hours > 40)
      wages *= 1.5;
   else
      wages *= 1;
```

D)
```
if (result >= 0)
      cout << "The result is positive\n";
   else
      cout << "The result is negative.\n";
```

4.25 The following statements use conditional expressions. Rewrite each with an `if/else` statement.

A) `j = k > 90 ? 57 : 12;`
B) `factor = x >= 10 ? y * 22 : y * 35;`
C) `total += count == 1 ? sales : count * sales;`
D) `cout << (((num % 2) == 0) ? "Even\n" : "Odd\n");`

4.26 What will the following program display?

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

int main()
{
   const int UPPER = 8, LOWER = 2;
   int num1, num2, num3 = 12, num4 = 3;

   num1 = num3 < num4 ? UPPER : LOWER;
   num2 = num4 > UPPER ? num3 : LOWER;
   cout << num1 << " " << num2 << endl;
   return 0;
}
```

4.14 The `switch` Statement

CONCEPT: **The `switch` statement lets the value of a variable or expression determine where the program will branch.**

A branch occurs when one part of a program causes another part to execute. The `if/else if` statement allows your program to branch into one of several possible paths. It performs a series of tests (usually relational) and branches when one of these tests is true. The `switch` statement is a similar mechanism. It, however, tests the value of an integer expression and then uses that value to determine which set of statements to branch to. Here is the format of the `switch` statement:

```
switch (IntegerExpression)
{
   case ConstantExpression:
        // place one or more
        // statements here

   case ConstantExpression:
        // place one or more
        // statements here

    // case statements may be repeated as many
    // times as necessary

   default:
        // place one or more
        // statements here
}
```