Engi231020p2c1

Tuesday, April 09, 2013
4:37 PM

Falta

Pero vamos a adelantarnos al capítulo 4 y luego regresamos al 2.12

4.1 Relational Operators

CONCEPT: **Relational operators allow you to compare numeric and `char` values and determine whether one is greater than, less than, equal to, or not equal to another.**

So far, the programs you have written follow this simple scheme:

- Gather input from the user.
- Perform one or more calculations.
- Display the results on the screen.

Computers are good at performing calculations, but they are also quite adept at comparing values to determine if one is greater than, less than, or equal to, the other. These types of operations are valuable for tasks such as examining sales figures, determining profit and loss, checking a number to ensure it is within an acceptable range, and validating the input given by a user.

Table 4-1 Relational Operators

Relational Operators	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to

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

ojo

The == operator determines whether the operand on its left is equal to the operand on its right. If both operands have the same value, the expression is true. Assuming that a is 4, the following expression is true:

```
a == 4
```

But the following is false:

```
a == 2
```

WARNING! Notice the equality operator is two = symbols together. Don't confuse this operator with the assignment operator, which is one = symbol. The == operator determines if a variable is equal to another value, but the = operator assigns the value on the operator's right to the variable on its left. There will be more about this later in the chapter.

Table 4-2 Example Relational Expressions (Assume x is 10 and y is 7.)

Expression	Value
`x < y`	false, because x is not less than y.
`x > y`	true, because x is greater than y.
`x >= y`	true, because x is greater than or equal to y.
`x <= y`	false, because x is not less than or equal to y.
`y != x`	true, because y is not equal to x.

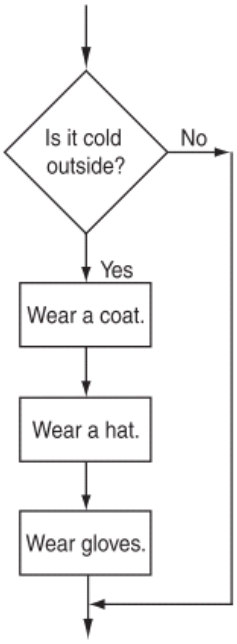
4.2 The `if` Statement

CONCEPT: The `if` statement can cause other statements to execute only under certain conditions.

VideoNote
Using an `if` Statement

You might think of the statements in a procedural program as individual steps taken as you are walking down a road. To reach the destination, you must start at the beginning and take each step, one after the other, until you reach the destination. The programs you have written so far are like a "path" of execution for the program to follow.

If crea una bifurcación en el programa. El siguiente diagrama de flujo lo muestra:



Programación del if statement:

Figure 4-3

```
if (condition)
{
      statement 1;
      statement 2;
         .
         .
      statement n;
}
```

or

```
if (condition) {
      statement 1;
      statement 2;
         .
         .
      statement n;
}
```

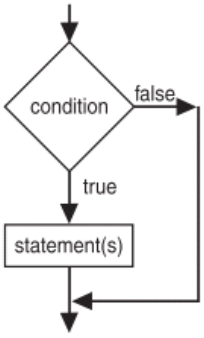
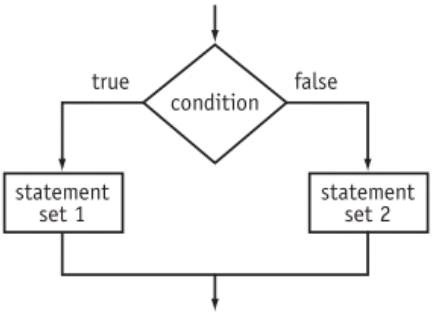


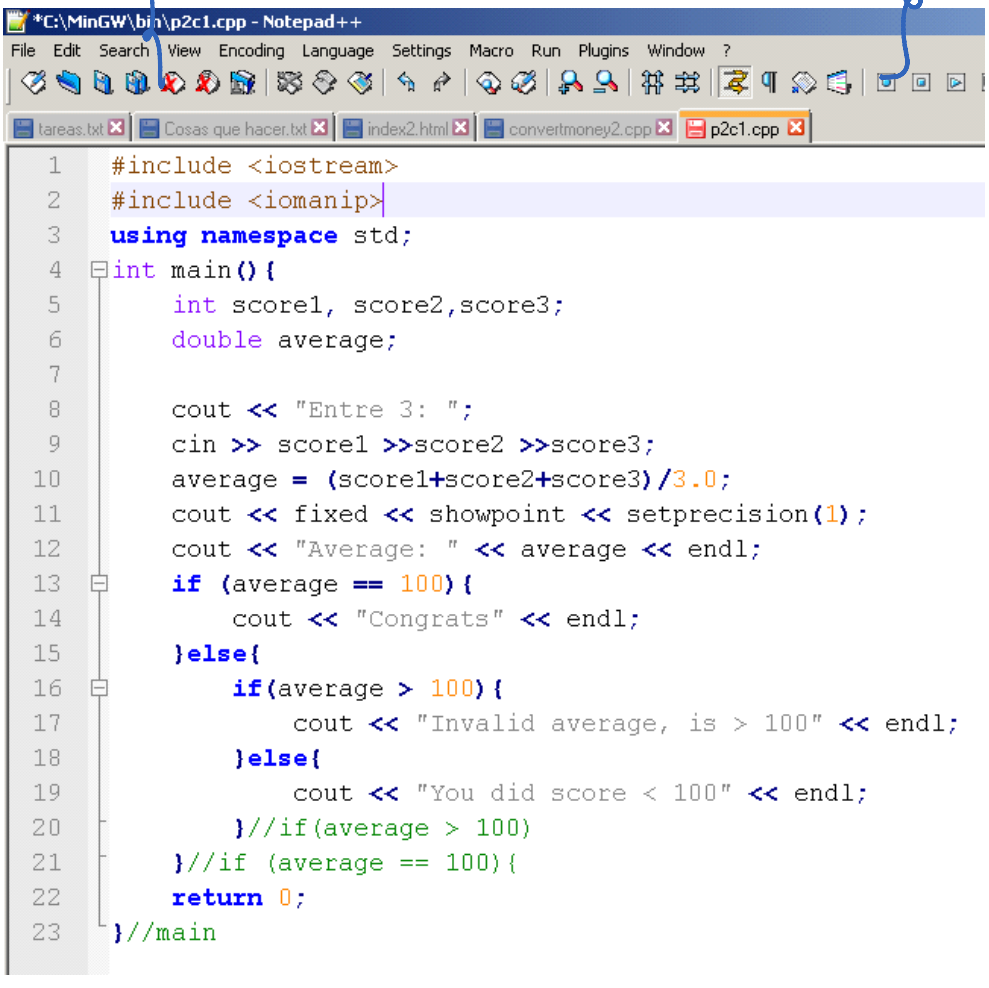
Figure 4-4

```
if (condition)
{
      statement set 1;
}
else
{
      statement set 2;
}
```



Program 4-2

```
1  // This program correctly averages 3 test scores.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  int main()
7  {
8     int score1, score2, score3;
9     double average;
10
11    // Get the three test scores
12    cout << "Enter 3 test scores and I will average them: ";
13    cin  >> score1 >> score2 >> score3;
14
15    // Calculate and display the average score
16    average = (score1 + score2 + score3) / 3.0;
17    cout << fixed << showpoint << setprecision(1);
18    cout << "Your average is " << average << endl;
19
20    // If the average equals 100, congratulate the user
21    if (average == 100)
22    {  cout << "Congratulations! ";
23       cout << "That's a perfect score!\n";
24    }
25    return 0;
26 }
```



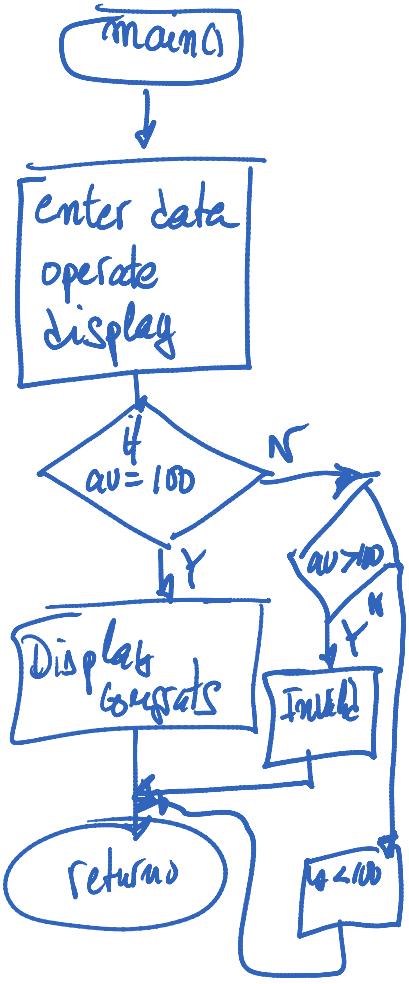


Table 4-5 Example `if` Statements

Statements	Outcome

`if (hours > 40)`
`{  overTime = true;`
`  payRate *= 2;`
`}`
	Assigns `true` to Boolean variable `overTime` and doubles `payRate` only when `hours` is greater than 40. Because there is more than one statement in the conditionally executed block, braces `{}` are required.

`if (temperature > 32)` `  freezing = false;`	Assigns `false` to Boolean variable `freezing` only when `temperature` is greater than 32. Because there is only one statement in the conditionally executed block, braces `{}` are optional.

Be Careful with Semicolons

Semicolons do not mark the end of a line, but the end of a complete C++ statement. The `if` statement isn't complete without the conditionally executed statement that comes after it. So, you must not put a semicolon after the `if (condition)` portion of an `if` statement.

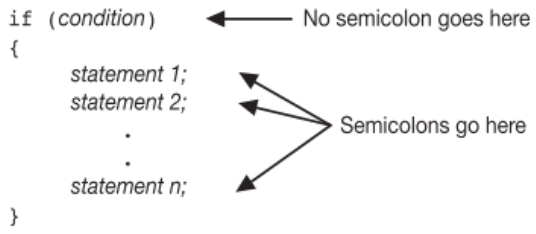
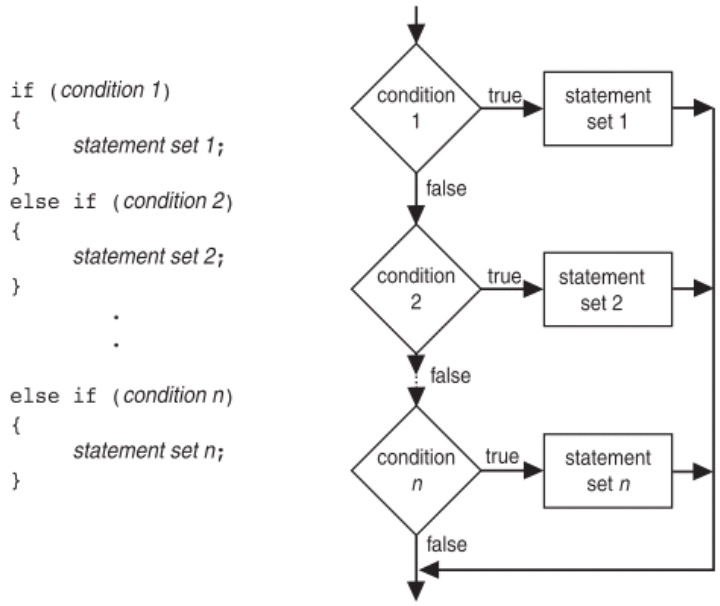
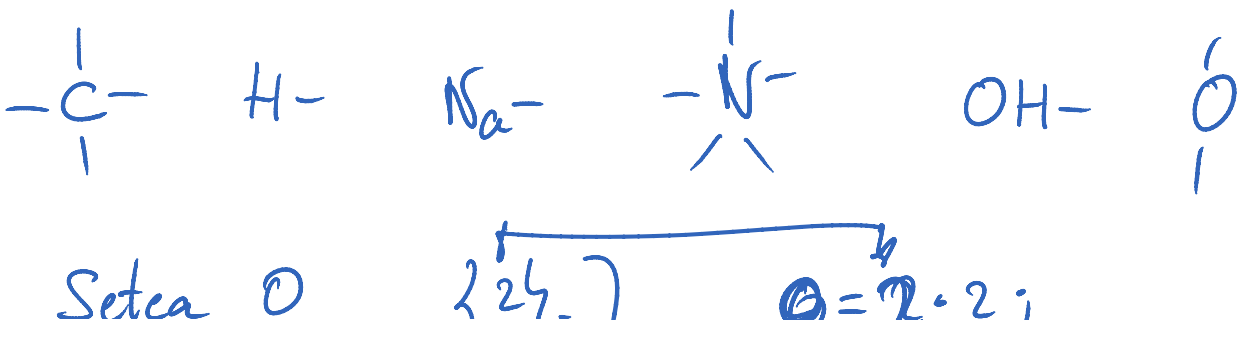
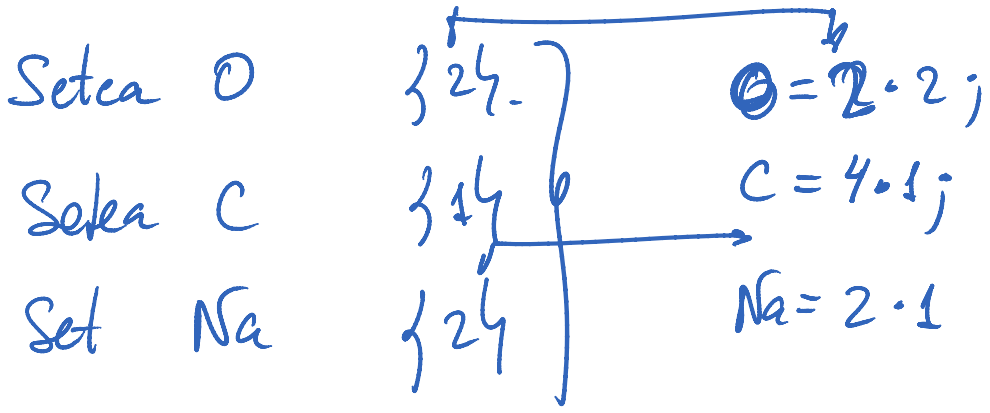


Figure 4-5








```
if (D < C)
    enlaces = C - D;
else
    enlaces = C;
```

if enlaces > 0
if (Na < enlaces)