© zennie/iStockphoto.

Chapter Three: Decisions

Chapter Goals

- To be able to implement decisions using `if` statements
- To learn how to compare integers, floating-point numbers, and strings
- To understand the Boolean data type
- To develop strategies for validating user input

Topic 1

1. The `if` statement
2. Comparing numbers and strings
3. Multiple alternatives
4. Nested branches
5. Problem solving: flowcharts
6. Problem solving: test cases
7. Boolean variables and operators
8. Application: input validation
9. Chapter summary

The `if` Statement

- The `if` statement is used to implement a decision.
  - When a condition is fulfilled,
    one set of statements is executed.
  - Otherwise,
    another set of statements is executed.

- Like a fork in the road

The `if` Statement: Elevator Example

We must write the code to control the elevator.

How can we skip the 13th floor?

`if()` Elevator Example Code

If the user inputs 20,
the program must set the actual floor to 19.
Otherwise,
we simply use the supplied floor number.

We need to decrement the input only under a certain condition:

```
cin >> floor;
int actual_floor;
if (floor > 13)   //never put a semicolon after the parentheses!!
{
   actual_floor = floor - 1; //
}
else
{
   actual_floor = floor;
}
```

Syntax of the `if()` Statement

```
if (condition)//never put a semicolon after the parentheses!!
{
  statement1; //executed if condition is true
}
else  //the else part is optional
{
  statement2; //executed if condition false
} //braces are optional but recommended
```

The `if` Statement, Example without an `else`

Here is another way to write this code:

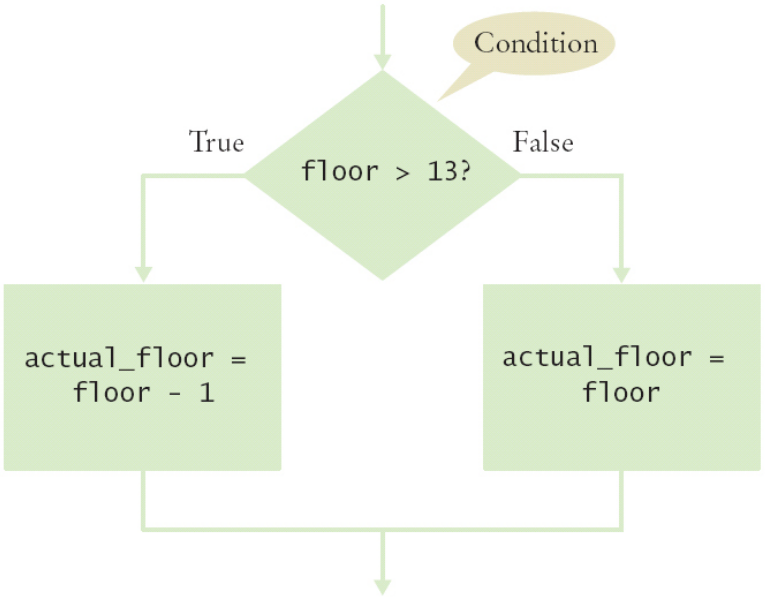
We only need to decrement
when the floor is greater than 13.

We can set **`actual_floor`** before testing:

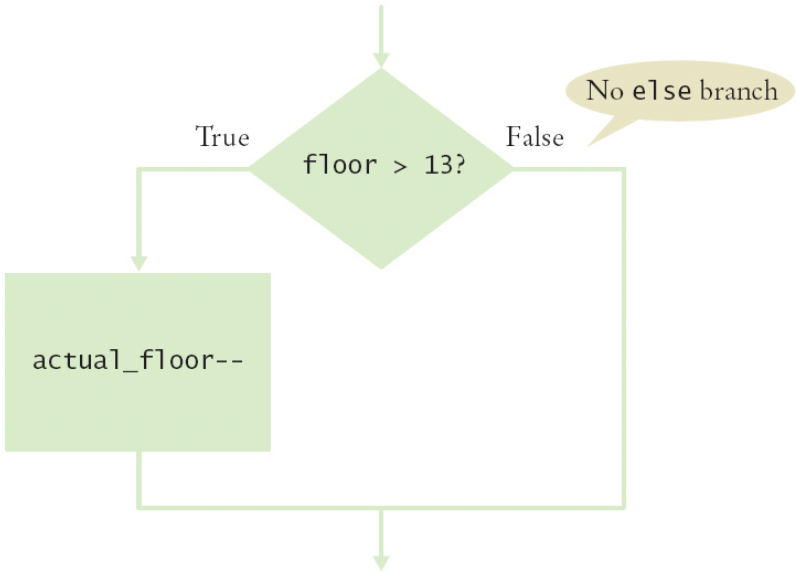
```
int actual_floor = floor;
if (floor > 13)
{
   actual_floor--;
} // No else needed
```

(And you'll notice we used the decrement operator this time.)

The `if` Statement Flowcharts



With else



Without else

The `if` Statement – A Complete Elevator Program

ch03/elevator1.cpp

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

int main()
{
   int floor;
   cout << "Floor: ";
   cin >> floor;
   int actual_floor;
   if (floor > 13)
   {
      actual_floor = floor - 1;
   }
   else
   {
      actual_floor = floor;
   }

   cout << "The elevator will travel to the actual floor "
      << actual_floor << endl;

   return 0;
}
```

The `if` Statement – Brace Layout

- Making your code easy to read is good practice.
- Lining up braces vertically helps.

```
if (floor > 13)
{
   floor--;
}
```

The `if` Statement – Brace Layout, continued

- As long as the ending brace clearly shows what it is closing, there is no confusion.

```
if (floor > 13) {
   floor--;
}
```

- Some programmers prefer this style—it saves a physical line in the code.

The `if` Statement – Always Use Braces

- When the body of an **`if`** statement consists of a single statement, you need not use braces:

```
if (floor > 13)
   floor--;
```

- However, it is a good idea to always include the braces:
  - the braces makes your code easier to read, and
  - you are less likely to make errors such as …

The if Statement – Common Error – The Do-nothing Statement

```
if (floor > 13) ; // ERROR ?
{
   floor--;
}
```

- This is *not* a compiler error.
- The compiler does not complain.
- It interprets this `if` statement as follows:

If floor is greater than 13, execute the *do-nothing statement*.
(semicolon by itself is the do nothing statement)

- Then execute the code enclosed in the braces.
- Any statements enclosed in the braces are no longer a part of the `if` statement.

The `if` Statement – Indent when Nesting

Block-structured code has the property that *nested* statements are indented by one or more levels.

```
int main()
{
   int floor;
   ...
   if (floor > 13)
   {
      floor--;
   }
   ...
   return 0;
}
0  1  2
```

Indentation level

Indent when Nesting

- Using the tab key is a way to get this indentation

but …

not all tabs are the same width!

- Luckily most development environments have settings to automatically convert all tabs to spaces.

- The IDE may also automatically indent your nested statements.

?: The Conditional Operator

- C++ has the conditional operator of the form

```
condition ? value1 : value2
```

- The value of that expression is either **`value1`** if the test passes or **`value2`** if it fails.

The Conditional Operator vs. an `if()`

For example, we can compute the actual floor number as

```
actual_floor = floor > 13 ? floor - 1 : floor;
```

which is equivalent to

```
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
```

The `if` Statement – Removing Duplication

```
if (floor > 13)
{
   actual_floor = floor - 1;
   cout << "Actual floor: " << actual_floor << endl;
}
else
{
   actual_floor = floor;
   cout << "Actual floor: " << actual_floor << endl;
}
```

Do you find anything redundant in this code?

The `if` Statement – Duplication Removed

```
if (floor > 13)
{
   actual_floor = floor - 1;
   }
else
{
   actual_floor = floor;
}
cout << "Actual floor: " << actual_floor << endl;
```

You can remove the duplication by moving the two identical **`cout`** statements outside of and after the braces, and of course deleting one of the two.