

Control Structures

- A program can proceed:
 - Sequentially
 - Selectively (branch) - making a choice
 - Repetitively (iteratively) - looping

Conditional Execution

- `if` is a reserved word
- The most basic syntax for `if`:

```
if( condition )  
    statement
```

- The statement is executed if the condition evaluates to `true`
- The statement is bypassed if the condition evaluates to `false`

bool Data Type and Conditions

- A condition can be a `bool` variable
- The data type `bool` has logical (Boolean) values `true` and `false`
- `bool`, `true`, and `false` are reserved words
- The identifier `true` has the value 1
- The identifier `false` has the value 0

`int` Data Type and Conditions

- Earlier versions of C++ did not provide built-in data types that had Boolean values
- Logical expressions evaluate to either 1 or 0
 - The value of a logical expression was stored in a variable of the data type `int`
- You can use the `int` data type as a condition

Logical Expressions

- General syntax for `if`:

```
if( logical-expression )  
    statement
```

- A logical expression is any expression that evaluates to `true` or `false`
 - A literal (anything but `0` is true)
 - A variable (any built-in type)
 - A function (should return `bool` or `int`)
 - Any expression that evaluates to `bool` or `int`

Logical Expressions

- Arithmetic expressions
 - Built with arithmetic operators
 - Evaluate to numbers (integer or floating-point)

```
3 + 5
```

```
(7 / 2) * 4.0
```

- Logical expressions
 - Built with relational operators
 - Evaluate to `true` or `false`

```
3 == 3
```

```
"hello" < "goodbye"
```

Relational Operators

TABLE 4-1 Relational Operators in C++

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Comparing Numbers

- Integer and floating-point types can be compared
 - `8 < 15` evaluates to `true`
 - `6 != 6` evaluates to `false`
 - `2.5 > 5.8` evaluates to `false`
 - `5.9 <= 7` evaluates to `true`

Comparing Characters

TABLE 4-2 Evaluating Expressions Using Relational Operators and the ASCII Collating Sequence

Expression	Value of Expression	Explanation
' ' < 'a'	<code>true</code>	The ASCII value of ' ' is 32, and the ASCII value of 'a' is 97. Because 32 < 97 is <code>true</code> , it follows that ' ' < 'a' is <code>true</code> .
'R' > 'T'	<code>false</code>	The ASCII value of 'R' is 82, and the ASCII value of 'T' is 84. Because 82 > 84 is <code>false</code> , it follows that 'R' > 'T' is <code>false</code> .
'+' < '*'	<code>false</code>	The ASCII value of '+' is 43, and the ASCII value of '*' is 42. Because 43 < 42 is <code>false</code> , it follows that '+' < '*' is <code>false</code> .
'6' <= '>'	<code>true</code>	The ASCII value of '6' is 54, and the ASCII value of '>' is 62. Because 54 <= 62 is <code>true</code> , it follows that '6' <= '>' is <code>true</code> .

Comparing strings

- Relational operators can be applied to strings
- Strings are compared character by character, starting with the first character
- Comparison continues until either a mismatch is found or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
 - The shorter string is less than the larger string
- Note: this does not work for comparing 2 string literals!

Examples

EXAMPLE 4-9

```
if (score >= 60)
    grade = 'P';
```

In this code, if the expression (`score >= 60`) evaluates to **true**, the assignment statement, `grade = 'P';`, executes. If the expression evaluates to **false**, the statements (if any) following the **if** structure execute. For example, if the value of `score` is 65, the value assigned to the variable `grade` is 'P'.

EXAMPLE 4-10

The following C++ program finds the absolute value of an integer:

```
//Program: Absolute value of an integer

#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";           //Line 1
    cin >> number;                                   //Line 2
    cout << endl;                                    //Line 3

    temp = number;                                   //Line 4

    if (number < 0)                                  //Line 5
        number = -number;                            //Line 6

    cout << "Line 7: The absolute value of "
         << temp << " is " << number << endl;      //Line 7

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

Line 1: Enter an integer: -6734

Line 7: The absolute value of -6734 is 6734

Common Syntax Errors

EXAMPLE 4-11

Consider the following statement:

```
if score >= 60      //syntax error
    grade = 'P';
```

This statement illustrates an incorrect version of an `if` statement. The parentheses around the logical expression are missing, which is a syntax error.

EXAMPLE 4-12

Consider the following C++ statements:

```
if (score >= 60);      //Line 1
    grade = 'P';      //Line 2
```

Because there is a semicolon at the end of the expression (see Line 1), the `if` statement in Line 1 terminates. The action of this `if` statement is null, and the statement in Line 2 is not part of the `if` statement in Line 1. Hence, the statement in Line 2 executes regardless of how the `if` statement evaluates.

Two-way Conditional Execution

- `if` can be paired with `else`

```
if( logical-expression )
```

```
    statement1
```

```
else
```

```
    statement2
```

- If the condition is `true`, `statement1` is executed
- If the condition is `false`, `statement2` is executed

One-Way Selection

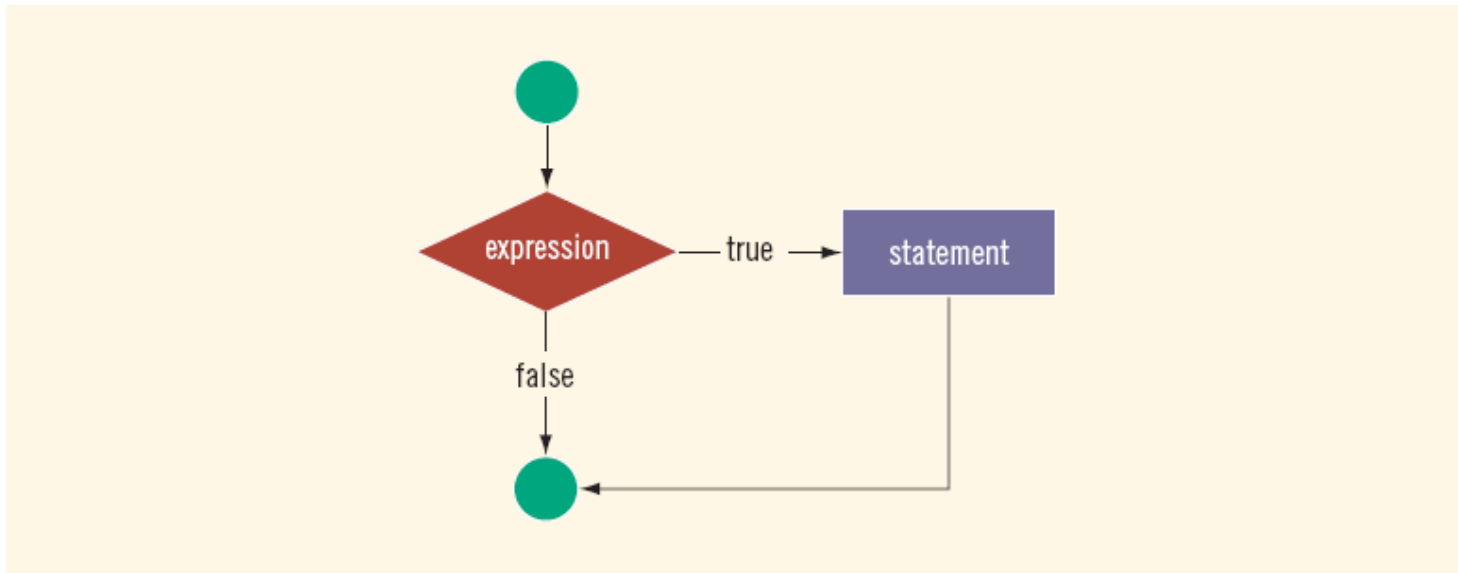


FIGURE 4-2 One-way selection

Two-Way Selection

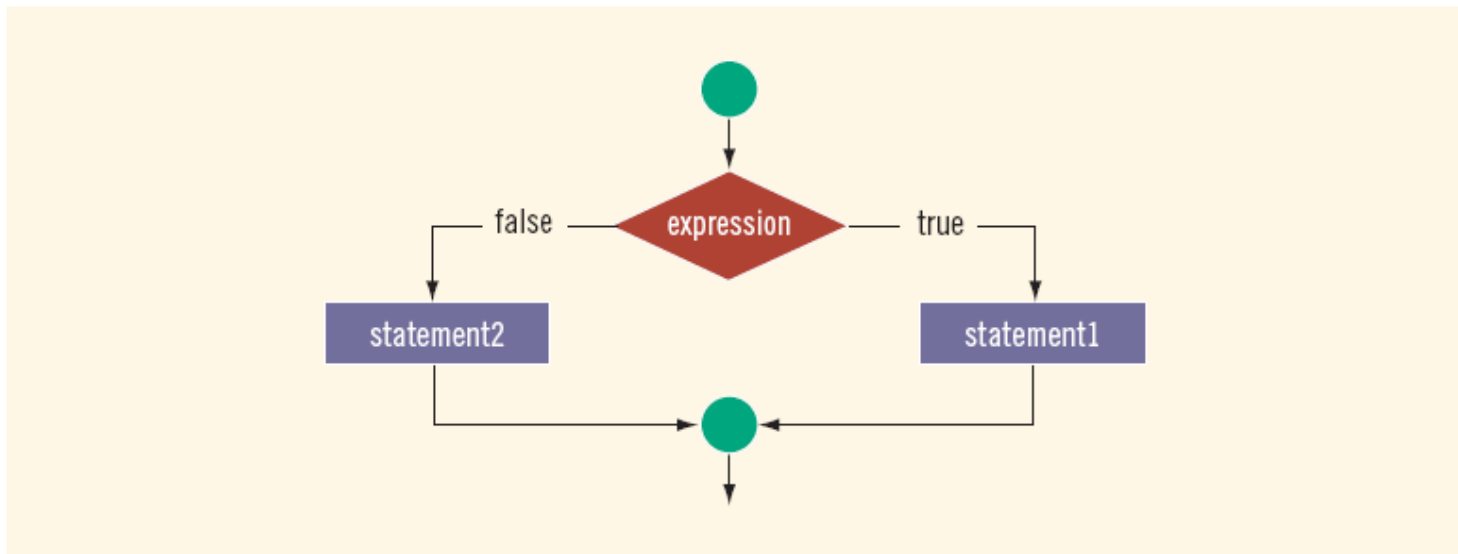


FIGURE 4-3 Two-way selection

Example

EXAMPLE 4-13

Consider the following statements:

```
if (hours > 40.0)           //Line 1
    wages = 40.0 * rate +
        1.5 * rate * (hours - 40.0); //Line 2
else                         //Line 3
    wages = hours * rate;    //Line 4
```

If the value of the variable `hours` is greater than `40.0`, then the `wages` include overtime payment. Suppose that `hours` is `50`. The expression in the `if` statement, in Line 1, evaluates to `true`, so the statement in Line 2 executes. On the other hand, if `hours` is `30`, or any number less than or equal to `40`, the expression in the `if` statement, in Line 1, evaluates to `false`. In this case, the program skips the statement in Line 2 and executes the statement in Line 4—that is, the statement following the reserved word `else` executes.

Common Syntax Errors

EXAMPLE 4-14

The following statements show an example of a syntax error:

```
if (hours > 40.0); //Line 1
    wages = 40.0 * rate +
        1.5 * rate * (hours - 40.0); //Line 2
else //Line 3
    wages = hours * rate; //Line 4
```

The semicolon at the end of the `if` statement (see Line 1) ends the `if` statement, so the statement in Line 2 separates the `else` clause from the `if` statement. That is, `else` is all by itself. Because there is no stand-alone `else` statement in C++, this code generates a syntax error.

Block Statements

- A block (or compound) statement looks like:

```
{
    statement1
    statement2
    .
    .
    .
    statementn
}
```

- A block can be used anywhere a statement can be used

Conditional Block Statements

```
if (age > 18)
    cout << "No longer a minor." << endl;
else
    cout << "Still a minor." << endl;
```

Conditional Block Statements

```
if (age > 18)
{
    cout << "No longer a minor." << endl;
}
else
{
    cout << "Still a minor." << endl;
}
```

Conditional Block Statements

```
if (age > 18)
{
    cout << "No longer a minor." << endl;
    cout << "Eligible to vote." << endl;
}
else
{
    cout << "Still a minor." << endl;
    cout << "Not eligible to vote." << endl;
}
```

More Than 2 Choices

- Series of `if` statements:

```
if( logical-expression1 )
{
    statement1
}
if( logical-expression2 )
{
    statement2
}
if( logical-expression3 )
{
    statement3
}
```

- Checks all three conditions
- Can't have a default `else` condition
- Used for statements that are not mutually exclusive

More Than 2 Choices

- For mutually exclusive conditions, use an `if...else` tree
 - Stops when a condition is true
 - Can have a default `else` condition

```
if( logical-expression1 )
{
    statement1
}
else if( logical-expression2 )
{
    statement2
}
else if( logical-expression3 )
{
    statement3
}
else
{
    statement4
}
```


Example: Date Conversion

- Input
 - Date in the form *yyyy-mm-dd*
 - (e.g. 2009-09-24)
- Output
 - Date in the form *month day, year*
 - (e.g. September 24, 2009)

Example: Large Joe's

Write a simple fast food drive-through ordering program for Large Joe's restaurant. The menu is:

Triple Burger:	\$4.99
Fried Chicken	\$6.99
French Fries	\$2.29

Sample run (user input in **bold**):

```
== Welcome to Large Joe's, can I take your order? ==
```

```
For a triple burger, press 1
```

```
For a heap of fried chicken, press 2
```

```
Your order: 1
```

```
Would you like fries with that? (y/n): y
```

```
Your total is $7.28, please drive through.
```

Example: Large Joe's

- Large Joe's now has bacon!
 - Adding bacon to your burger costs \$0.99
 - Modify the program so that if the customer orders a burger, the program asks them:

```
Would you like bacon on your burger? (y/n): y
```

- If they answer yes, add the cost to their order
- Don't ask about bacon if they didn't order a burger!