Programming Fundamentals
Control Structures in C++

Lecture 16

Objectives

In this chapter you will:

- Learn about control structures
- Examine relational and logical operators
- Explore how to form and evaluate logical (Boolean) expressions
- Discover how to use the selection control structures `if`, `if`...`else`, and `switch` in a program

Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- Some statements are executed only if certain conditions are met
- A condition is represented by a logical (Boolean) expression that can be true or false
- A condition is met if it evaluates to true

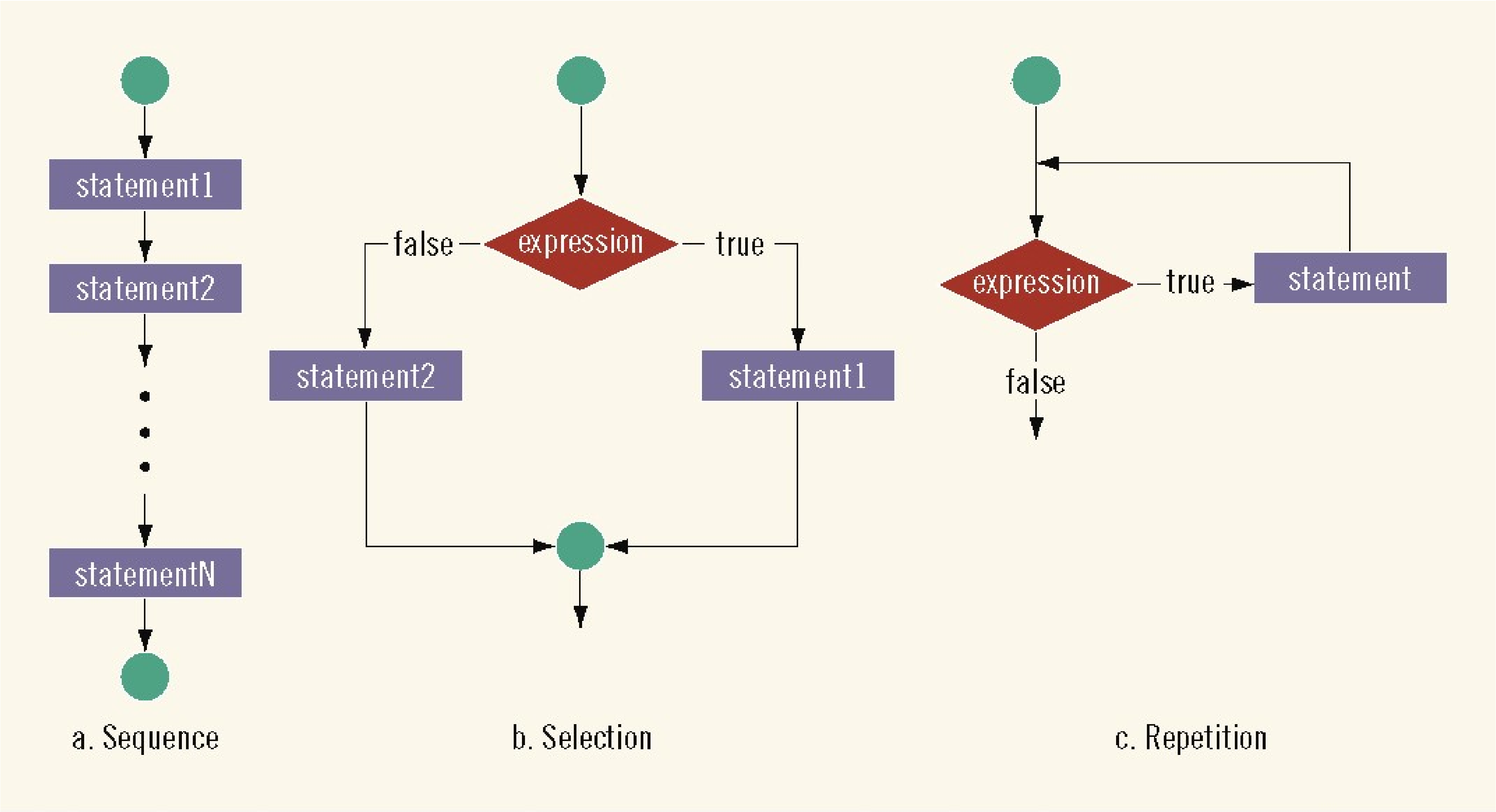
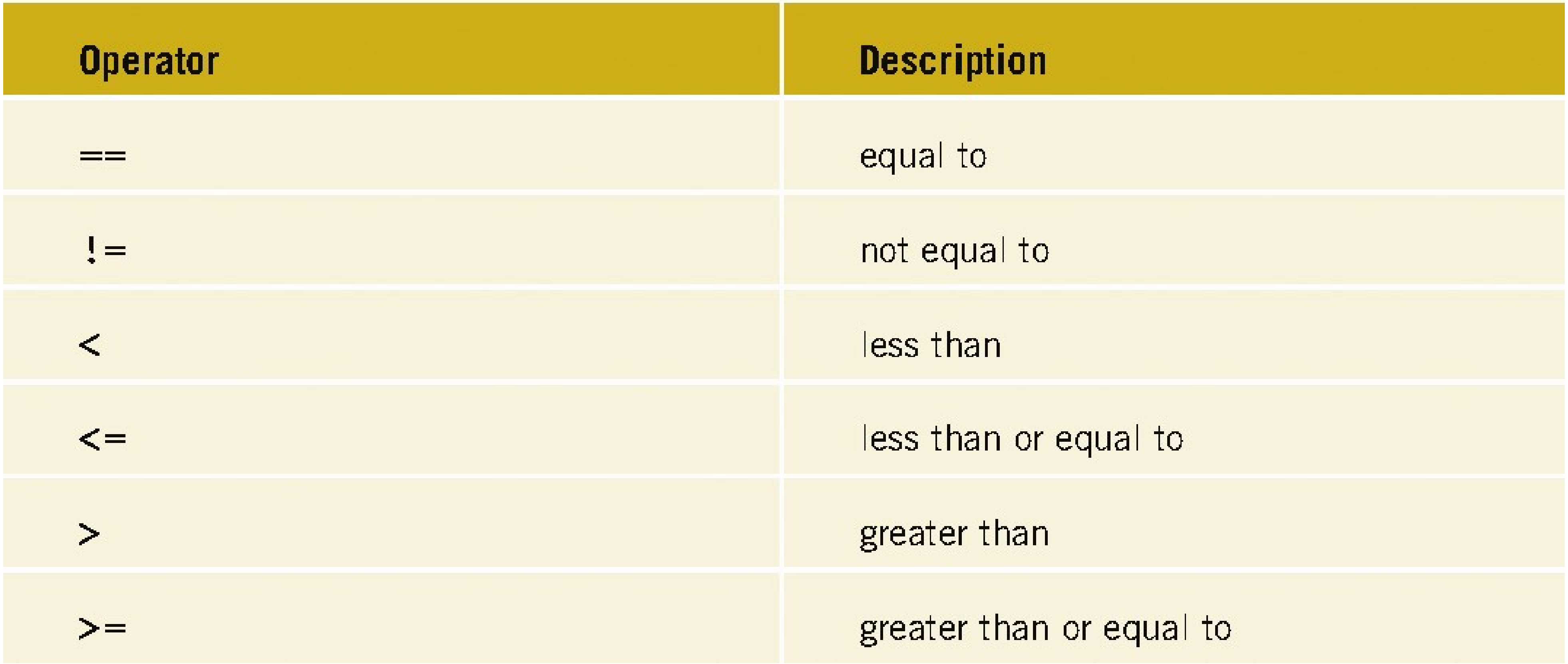


FIGURE 4-1 Flow of execution

Relational Operators

- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Return `1` if expression is `true`, `0` otherwise
- Comparing values of different data types may produce unpredictable results
  - For example, `8 < '5'` should not be done
- Any nonzero value is treated as `true`

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

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

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

Comparing `string` Types

- 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

string Comparison Example

- Suppose we have the following declarations:

```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
```

TABLE 4-3 Evaluating Logical Expressions with string Variables

Expression	Value	Explanation
str1 < str2	true	str1 = "Hello" and str2 = "Hi". The first character of str1 and str2 are the same, but the second character 'e' of str1 is less than the second character 'i' of str2. Therefore, str1 < str2 is true.
str1 > "Hen"	false	str1 = "Hello". The first two characters of str1 and "Hen" are the same, but the third character 'l' of str1 is less than the third character 'n' of "Hen". Therefore, str1 > "Hen" is false.
str3 < "An"	true	str3 = "Air". The first characters of str3 and "An" are the same, but the second character 'i' of "Air" is less than the second character 'n' of "An". Therefore, str3 < "An" is true.

TABLE 4-3 Evaluating Logical Expressions with `string` Variables (continued)

Expression	Value	Explanation
`str1 == "hello"`	`false`	`str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the ASCII value of `'H'` is 72, and the ASCII value of `'h'` is 104. Therefore, `str1 == "hello"` is `false`.
`str3 <= str4`	`true`	`str3 = "Air"` and `str4 = "Bill"`. The first character `'A'` of `str3` is less than the first character `'B'` of `str4`. Therefore, `str3 <= str4` is `true`.
`str2 > str4`	`true`	`str2 = "Hi"` and `str4 = "Bill"`. The first character `'H'` of `str3` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is `true`.

Logical (Boolean) Operators

- Logical (Boolean) operators enable you to combine logical expressions
- Three logical (Boolean) operators:

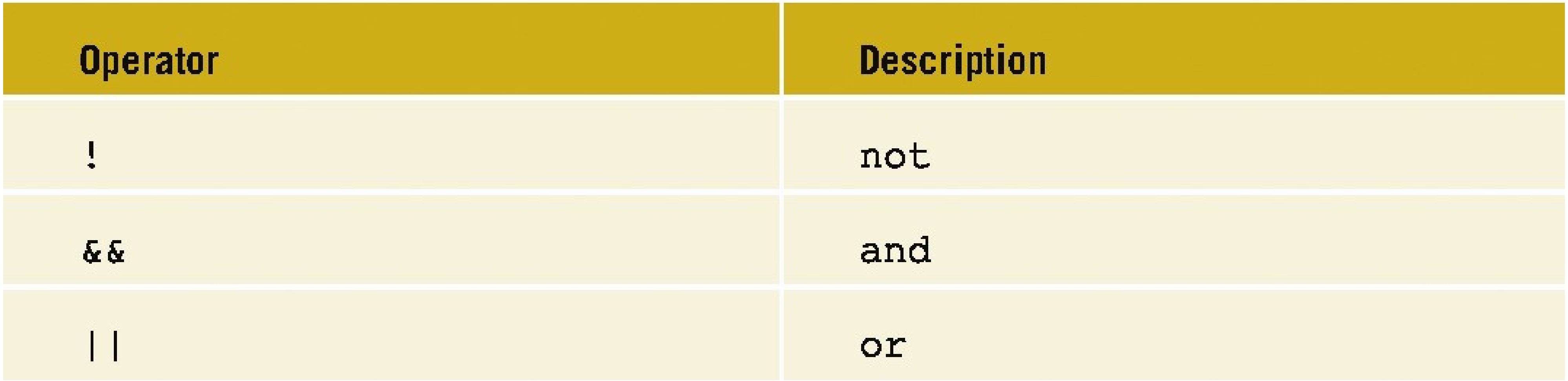
  `!` - not

  `&&` – and

  `||` - or

- Logical operators take logical values as operands and yield logical values as results
- `!` is unary; `&&` and `||` are binary operators
- Putting `!` in front of a logical expression reverses its value

TABLE 4-5 Logical (Boolean) Operators in C++



Operator	Description
!	not
&&	and
||	or

TABLE 4-6 The ! (Not) Operator

Expression	!(Expression)
`true (nonzero)`	`false (0)`
`false (0)`	`true (1)`

EXAMPLE 4-2

Expression	Value	Explanation
`!('A' > 'B')`	`true`	Because `'A' > 'B'` is `false`, `!('A' > 'B')` is `true`.
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

TABLE 4-7 The && (And) Operator

Expression1	Expression2	Expression1 && Expression2
`true` (nonzero)	`true` (nonzero)	`true` (1)
`true` (nonzero)	`false` (0)	`false` (0)
`false` (0)	`true` (nonzero)	`false` (0)
`false` (0)	`false` (0)	`false` (0)

EXAMPLE 4-3

Expression	Value	Explanation
`(14 >= 5) && ('A' < 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' < 'B')` is `true`, and `true && true` is `true`, the expression evaluates to `true`.
`(24 >= 35) && ('A' < 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' < 'B')` is `true`, and `false && true` is `false`, the expression evaluates to `false`.

TABLE 4-8 The || (Or) Operator

Expression1	Expression2	Expression1 || Expression2
`true` (nonzero)	`true` (nonzero)	`true` (1)
`true` (nonzero)	`false` (0)	`true` (1)
`false` (0)	`true` (nonzero)	`true` (1)
`false` (0)	`false` (0)	`false` (0)

EXAMPLE 4-4

Expression	Value	Explanation
`(14 >= 5) || ('A' > 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' > 'B')` is `false`, and `true || false` is `true`, the expression evaluates to `true`.
`(24 >= 35) || ('A' > 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' > 'B')` is `false`, and `false || false` is `false`, the expression evaluates to `false`.
`('A' <= 'a') || (7 != 7)`	`true`	Because `('A' <= 'a')` is `true`, `(7 != 7)` is `false`, and `true || false` is `true`, the expression evaluates to `true`.

Precedence of Operators

- Relational and logical operators are evaluated from left to right
- The associativity is left to right
- Parentheses can override precedence

TABLE 4-9 Precedence of Operators

Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

EXAMPLE 4-5

Suppose you have the following declarations:

```
bool found = true;
bool flag = false;
int num = 1;
double x = 5.2;
double y = 3.4;
int a = 5, b = 8;
int n = 20;
char ch = 'B';
```

Logical (Boolean) Expressions (continued)

- Logical expressions can be unpredictable
- The following expression appears to represent a comparison of `0`, `num`, and `10`:

```
0 <= num <= 10
```

- It always evaluates true because `0 <= num` evaluates to either `0` or `1`, and `0 <= 10` is `true` and `1 <= 10` is `true`
- A correct way to write this expression is:

```
0 <= num && num <= 10
```

One-Way (`if`) Selection

- The syntax of one-way selection is:

```
if (expression)
    statement
```

- Statement is executed if the value of the expression is `true`
- Statement is bypassed if the value is `false`; program goes to the next statement

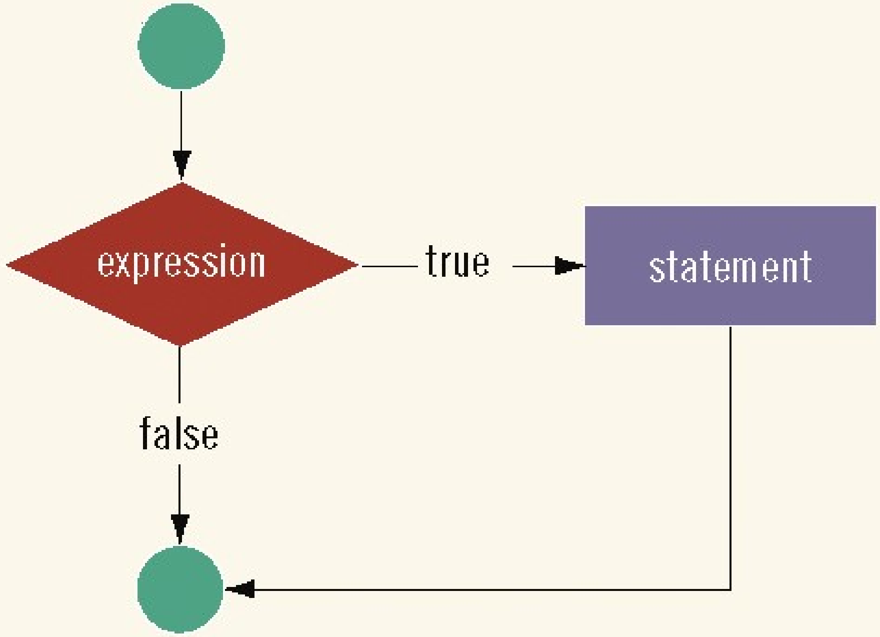


FIGURE 4-2 One-way selection

EXAMPLE 4-9

```
if (score >= 90)
    grade = 'A';
```

In this code, if the expression (score >= 90) evaluates to **true**, the assignment statement, grade = 'A';, executes. If the expression evaluates to **false**, the statements (if any) following the **if** structure execute. For example, if the value of score is 95, the value assigned to the variable grade is 'A'.

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: Please 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: Please enter an integer: -6734
Line 7: The absolute value of -6734 is 6734
```

Two-Way (if…else) Selection

- Two-way selection takes the form:

```
if (expression)
     statement1
else
     statement2
```

- If expression is `true`, statement1 is executed otherwise statement2 is executed
- statement1 and statement2 are any C++ statements
- `else` is a reserved word

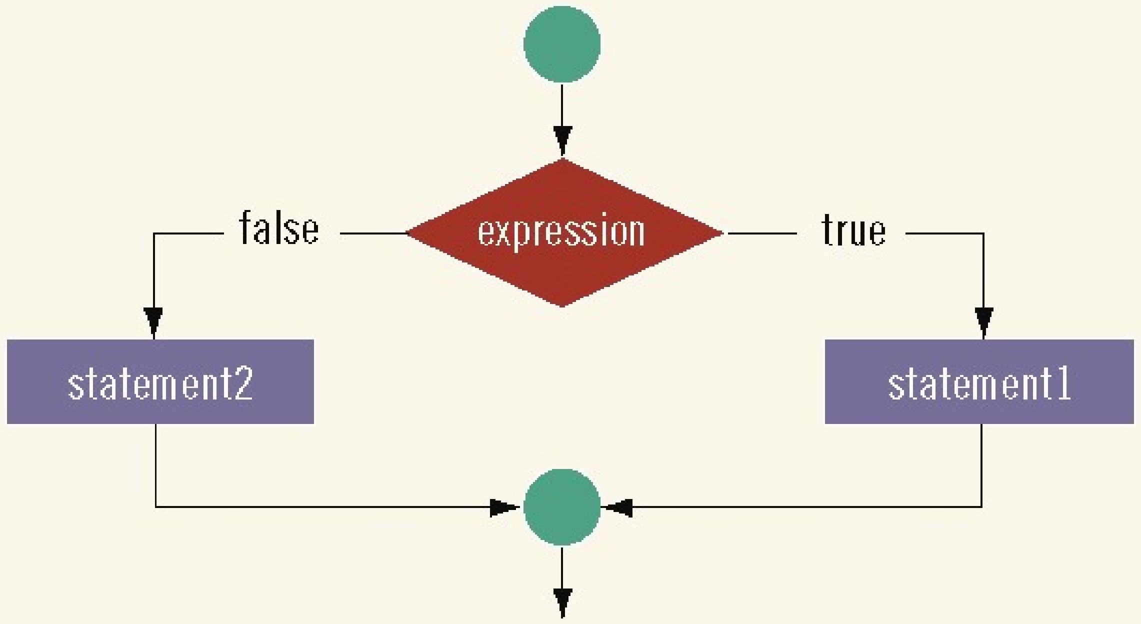


FIGURE 4-3 Two-way selection

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.

Compound (Block of) Statement

- Compound statement (block of statements):

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

- A compound statement is a single statement

Compound Statement Example

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

Nested if

- Nesting: one control statement in another
- An `else` is associated with the most recent `if` that has not been paired with an `else`

EXAMPLE 4-19

Assume that score is a variable of type int. Based on the value of score, the following code outputs the grade:

```
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```

Conditional Operator (?:)

- Conditional operator (`?:`) takes three arguments (ternary)
- Syntax for using the conditional operator:

```
expression1 ? expression2 : expression3
```

- If `expression1` is `true`, the result of the conditional expression is `expression2`. Otherwise, the result is `expression3`

`switch` Structures

- `switch` structure: alternate to if-else
- `switch` expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector

switch Structures (continued)

- Expression value can be only integral
- Its value determines which statement is selected for execution
- A particular case value should appear only once

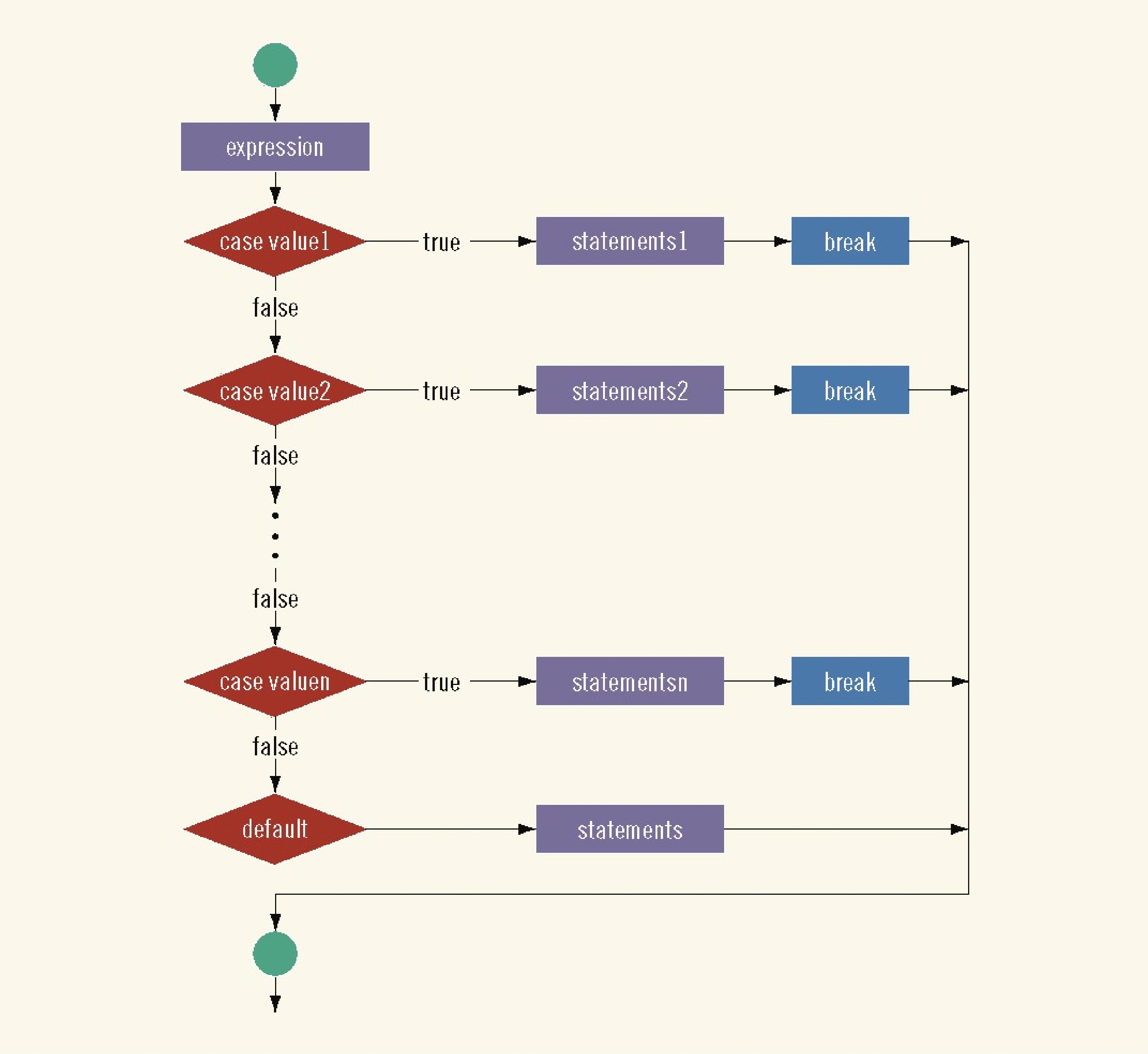


FIGURE 4-4 **switch** statement

switch Structures (continued)

- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- The break statement may or may not appear after each statement
- switch, case, break, and default are reserved words

```
#include <iostream>
 using namespace std;
int main()
{
char grade; cout << "Enter your grade: "; cin >> grade; cout << endl;
switch (grade)
        {
        case 'A':
                cout << "Your grade is A." << endl;
                break;
        case 'B':
                cout << "Your grade is B." << endl;
                break;
        case 'C':
                cout << "Your grade is C." << endl;
                break;
        case 'F':
         case 'f':
                cout << "Your grade is C." << endl;
                break;
        default:
                cout<<" The grade is invalid."<<endl;
        }
return 0;
}
```

Summary

- Control structures alter normal control flow
- Most common control structures are selection and repetition
- Relational operators: `==`, `<`, `<=`, `>`, `>=`, `!=`
- Logical expressions evaluate to `1` (`true`) or `0` (`false`)
- Logical operators: `!` (not), `&&` (and), `||` (or)

Summary (continued)

- Two selection structures: one-way selection and two-way selection
- The expression in an `if` or `if`...`else` structure is usually a logical expression
- No `else` statement in C++. Every `else` has a related `if`
- A sequence of statements enclosed between braces, `{` and `}`, is called a compound statement or block of statements

Summary (continued)

- Using assignment in place of the equality operator creates a semantic error
- `switch` structure handles multiway selection
- `break` statement ends `switch` statement