CS118 – Programming Fundamentals

Lecture # 09
Tuesday, September 17, 2019
FALL 2019
FAST – NUCES, Faisalabad Campus

Zain Iqbal

Example


```
//Input Failure program

#include <iostream>

using namespace std;

int main()
{
    int a = 10;                                          //Line 1
    int b = 20;                                          //Line 2
    int c = 30;                                          //Line 3
    int d = 40;                                          //Line 4

    cout << "Line 5: Enter four integers: ";             //Line 5
    cin >> a >> b >> c >> d;                             //Line 6
    cout << endl;                                        //Line 7
    cout << "Line 8: The numbers you entered are:"
         << endl;                                        //Line 8
    cout << "Line 9: a = " << a << ", b = " << b
         << ", c = " << c << ", d = " << d << endl;      //Line 9

    return 0;
}
```

Output

Sample Run 1

Line 5: Enter four integers: 34 K 67 28

Line 8: The numbers you entered are:
Line 9: a = 34, b = 20, c = 30, d = 40

Sample Run 2

Line 5: Enter four integers: 43 225.56 39 61

Line 8: The numbers you entered are:
Line 9: a = 43, b = 225, c = 30, d = 40

Increment and Decrement Operators

- Increment operator(**++**): increment variable by 1
  - **Pre-increment:** ++variable
  - **Post-increment:** variable++
  - int j = i++; // j will contain i, i will be incremented.
  - int j = ++i; // i will be incremented, and j will contain i+1.
- Decrement operator (**--**): decrement variable by 1
  - **Pre-decrement:** --variable
  - **Post-decrement:** variable--
- What is the difference between the following?

```
x = 9;
y = ++x;
```

```
x = 9;
y = x++;
```

More on Assignment Statements

- C++ has special assignment statements called compound assignments

  +=, -=, *=, /=, and %=

- Example:

  x = x * y ;

  as

  x *= y ;

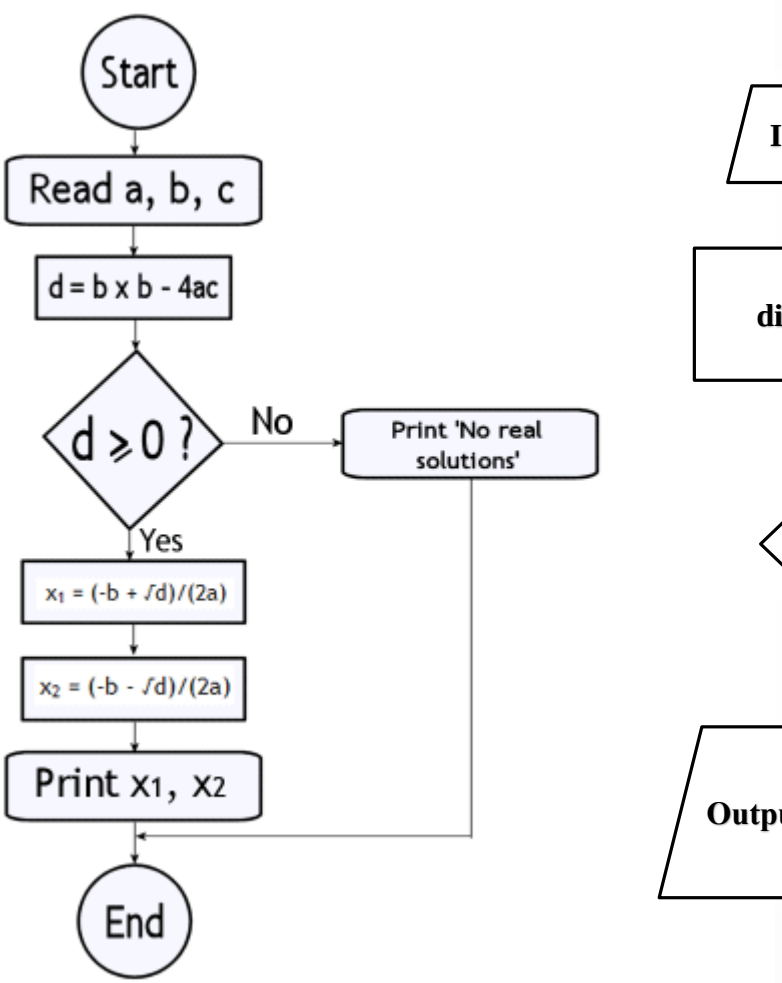
Example

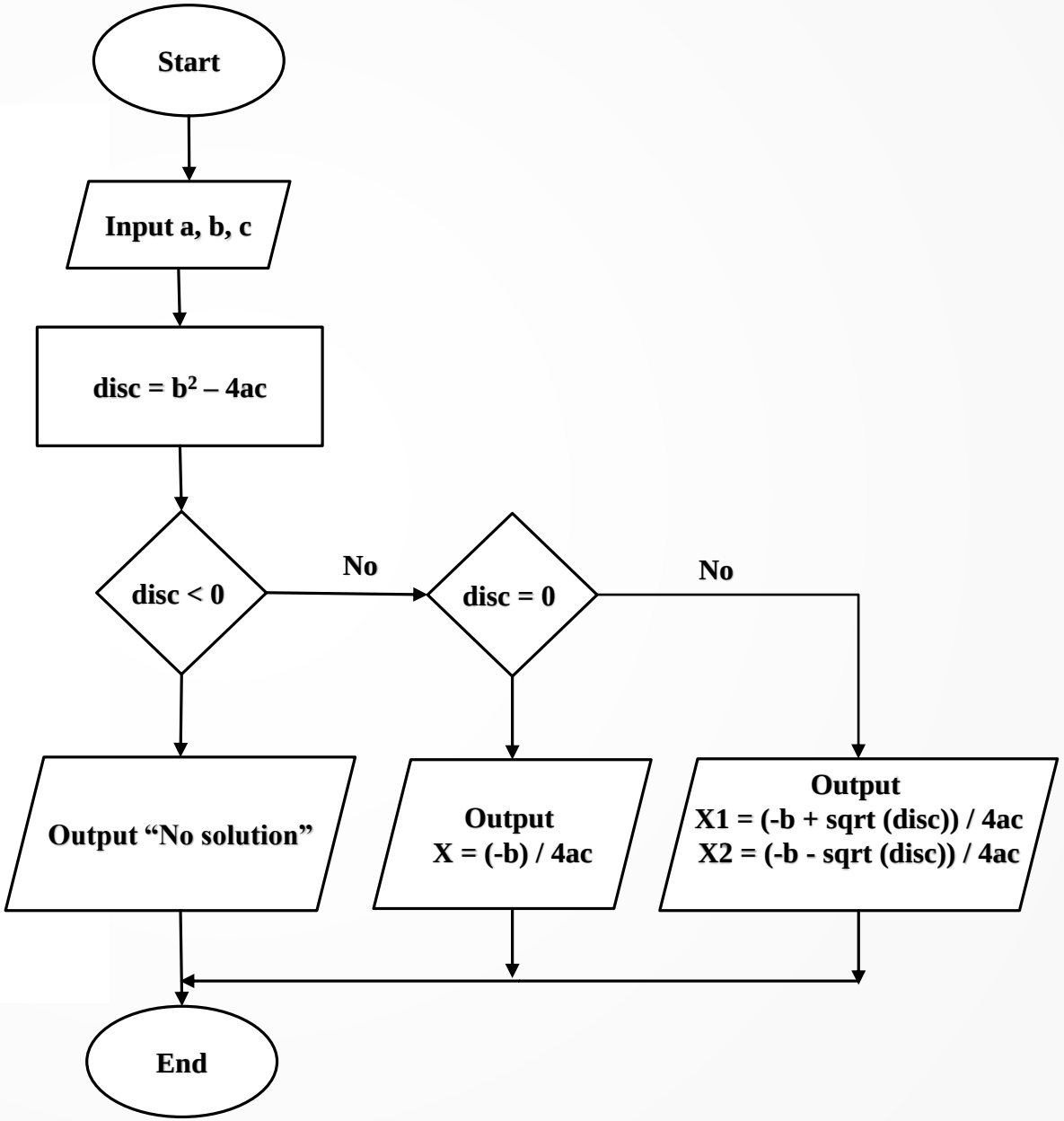
EXAMPLE 2-31

This example shows several compound assignment statements that are equivalent to simple assignment statements.

Simple Assignment Statement	Compound Assignment Statement
`i = i + 5;`	`i += 5;`
`counter = counter + 1;`	`counter += 1;`
`sum = sum + number;`	`sum += number;`
`amount = amount * (interest + 1);`	`amount *= interest + 1;`
`x = x / ( y + 5);`	`x /= y + 5;`

Control Structures





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 met if it evaluates to true

Control Structures (cont'd.)

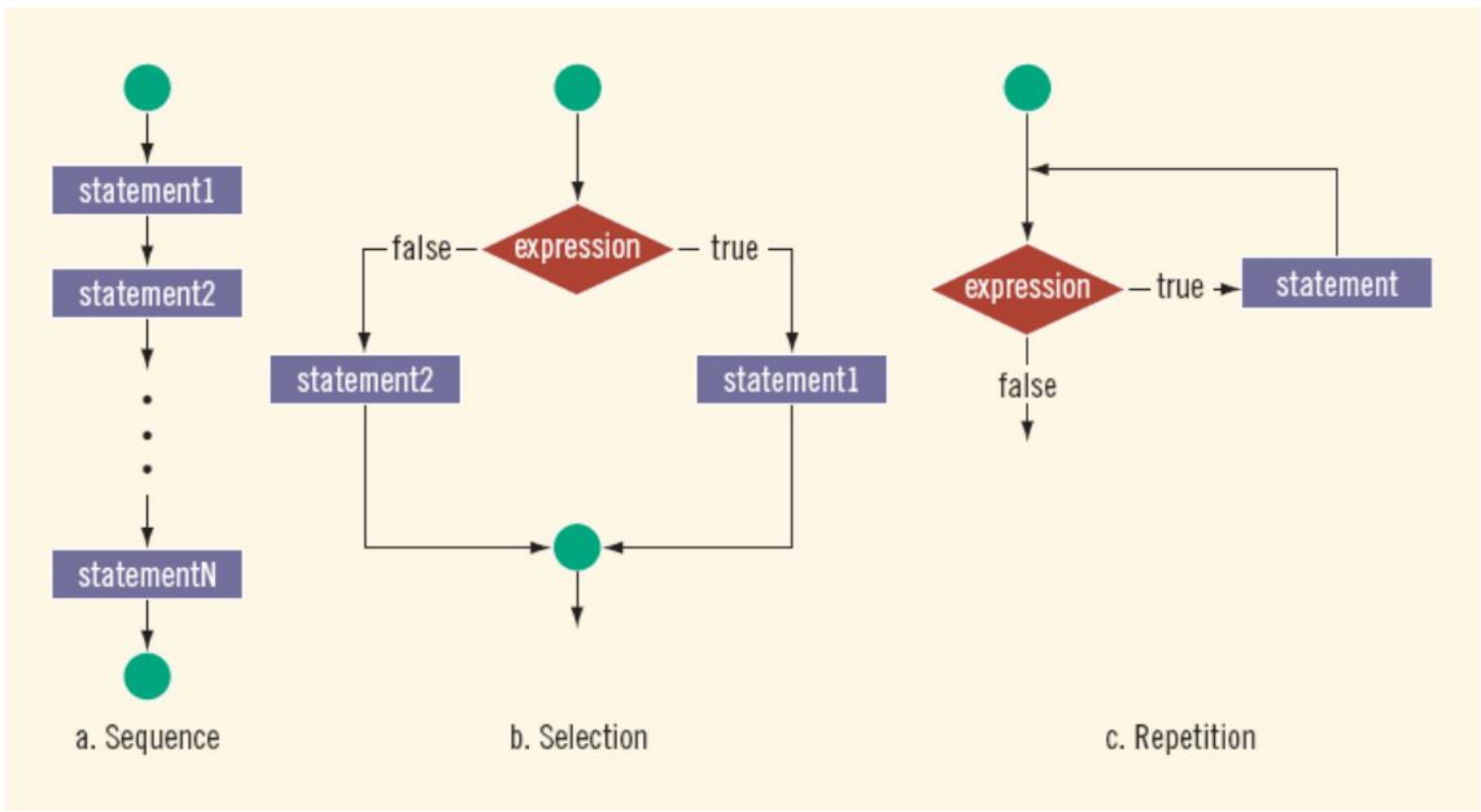


FIGURE 4-1 Flow of execution

Control Structures (cont'd.)

- You must understand the nature of conditional statements and how to use them.
- Consider the following three statements:

```
1. if (score is greater than or equal to 90)
      grade is A
2. if (hours worked are less than or equal to 40)
      wages = rate * hours
   otherwise
      wages = (rate * 40) + 1.5 * (rate * (hours - 40))
3. if (temperature is greater than 70 degrees and it is not
   raining)
      Go golfing!
```

Decision Making: Equality and Relational Operators

if structure

- Decision based on truth or false of condition
  - If condition met, body executed
  - Else, body not executed

Equality and relational operators

- Equality operators
  - Same level of precedence
- Relational operators
  - Same level of precedence
- Associate left to right

Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Relational Operators and Simple Data Types

Expression	Meaning	Value
`8 < 15`	`8` is less than `15`	`true`
`6 != 6`	`6` is not equal to `6`	`false`
`2.5 > 5.8`	`2.5` is greater than `5.8`	`false`
`5.9 <= 7.5`	`5.9` is less than or equal to `7.5`	`true`

Comparing Characters

- Expression with relational operators
  - Depends on machine's collating sequence
  - ASCII character set
- Logical (Boolean) expressions
  - Expressions such as 4 < 6 and 'R' > 'T'
  - Returns an integer value of 1 if the logical expression evaluates to true
  - Returns an integer value of 0 otherwise

Comparison of Characters

- For characters
  - Respective ASCIII values are compared
- 'R' > 'T' is false (82 > 84)
- '+' < '*' is false (43 < 42)
- 'A' <= 'a' is true (65<97)

Relational and Equality Operators (cont.)

- The relational operators have very low precedence and associate left-to-right
- The equality operators have very-very low precedence and associate left-to-right
- Some examples:

```
17 < x          foo == 3.14
age != 21       x+1 >= 4*y-z
```

Precedence

Operators	Precedence
()	highest (applied first)
* / %	
+ -	
< <= > >=	
== !=	
=	lowest (applied last)

Logical (Boolean) Operators and Logical Expressions

Operator	Description
!	NOT
&&	AND
||	OR

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

Conti…

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`.

- AND & OR operators work just like AND/OR-Gate as you studied in Physics in intermediate
- AND = True iff all conditions are TRUE
- OR = False iff all results are FASLE

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`.

Order of precedence

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

- Suppose you have the following declarations

```
bool found = true ;
int age = 20 ;
double hours = 45.30 ;
double overtime = 15.00 ;
int count = 20;
char ch = 'B';
```

Expression	Value / Expression
!found	**false** Because found is **true**, !found is **false**
hours > 40.0	**true** Because hours is 45.3 and 45.3 > 40.0 is **true**, the expression hours > 40.0 evaluates to **true**
!age	**false** Age is 20, which is non zero so age is **true.** Therefore !age is **false**
!found && (age >=18)	**false** !found is **false;** age >= 18 is 20 >= 18 is **true.** Therefore !found && (age >=18) is **false** && **true**, which evaluates to **false**

```
bool found = true ;
int age = 20 ;
double hours = 45.30 ;
double overTime = 15.00 ;
int count = 20;
char ch = 'B';
```

Expression	Value / Expression
hours + overTime <= 75.0	**true** hours + overTime is 45.30 + 15.00 = 60.30 and 60.30 <= 75.0 is true, it follows that hours + overtime <= 75 evaluates to **true**
(count >= 0) && (count <= 100)	**true** Now count is 20, Because 20 >= 0 is **true**, count >=0 is **true**. Also 20 <= 100 is **true**, count <=100 is **true**. Therefore (count >= 20) && (count <= 100) is **true** & **true**, which evaluates to **true**
('A' <= ch && ch <= 'Z')	**true** Here ch is 'B'. Because 'A' <= 'B' is **true**, 'A' <= ch evaluates to **true**. Also, because 'B' <= 'Z' is **true**, ch <= 'Z' evaluates to **true**. Therefore ('A' <= ch && ch <= 'Z') is **true** && **true** evaluates to **true**.

Relational Operators

- A condition is represented by a logical (Boolean) expression that can be **true** or **false**
- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Evaluate to true or false

Relational Operators (cont'd.)

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

Relational Operators and Simple Data Types

- You can use the relational operators with all three simple data types:
  - 8 < 15 evaluates to **true**
  - 6 != 6 evaluates to **false**
  - 2.5 > 5.8 evaluates to **false**
  - 5.9 <= 7.5 evaluates to **true**

Relational Operators and the string Type

- 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

```
Bill >= Billy    //False
```

Relational Operators and the string Type (cont'd.)

- Suppose we have the following declarations:

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

Relational Operators and th
string Type (cont'd.)

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

Expression	Value /Explanation
`str1 < str2`	`true` `str1 = "Hello"` and `str2 = "Hi"`. The first characters 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`.
`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 `str2` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is `true`.

Relational Operators and th
string Type (cont'd.)


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

Expression	Value/Explanation
`str4 >= "Billy"`	`false` `str4 = "Bill"`. It has four characters, and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"`, and `"Billy"` is the larger string. Therefore, `str4 >= "Billy"` is `false`.
`str5 <= "Bigger"`	`true` `str5 = "Big"`. It has three characters, and `"Bigger"` has six characters. Therefore, `str5` is the shorter string. All three characters of `str5` are the same as the corresponding first three characters of `"Bigger"`, and `"Bigger"` is the larger string. Therefore, `str5 <= "Bigger"` is `true`.

Logical (Boolean) Operators and Logical Expressions (cont'd.)

TABLE 4-4 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 4-4

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.

Logical (Boolean) Operators and Logical Expressions (cont'd.)

TABLE 4-5 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 4-5

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`.

int **Data Type and Logical (Boolean) Expressions**

- 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 to manipulate logical (Boolean) expressions

int Data Type and Logical (Boolean) Expressions


```
int legalAge;
int age;
```

and the assignment statement:

```
legalAge = 21;
```

If you regard `legalAge` as a logical variable, the value of `legalAge` assigned by this statement is `true`.

The assignment statement:

```
legalAge = (age >= 21);
```

assigns the value `1` to `legalAge` if the value of `age` is greater than or equal to `21`. The statement assigns the value `0` if the value of `age` is less than `21`.

The bool Data Type and Logical (Boolean) Expressions

- 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**

```
legalAge= (age>=21);
```

Programming Example:


```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    bool found = true;
    int age = 20;
    double hours = 45.30;
    double overTime = 15.00;
    int count = 20;
    char ch = 'B';

    cout << fixed << showpoint << setprecision(2);
    cout << "found = " << found << ", age = " << age
         << ", hours = " << hours << ", overTime = " << overTime
         << "," << endl << "count = " << count
         << ", ch = " << ch << endl << endl;

    cout << "!found evaluates to " << !found << endl;
    cout << "hours > 40.00 evaluates to " << (hours > 40.00) << endl;
    cout << "!age evaluates to " << !age << endl;
    cout << "!found && (hours >= 0) evaluates to "
         << (!found && (hours >= 0)) << endl;
```

```
    cout << "!(found && (hours >= 0)) evaluates to "
         << (!(found && (hours >= 0))) << endl;
    cout << "hours + overTime <= 75.00 evaluates to "
         << (hours + overTime <= 75.00) << endl;
    cout << "(count >= 0) && (count <= 100) evaluates to "
         << ((count >= 0) && (count <= 100)) << endl;
    cout << "('A' <= ch && ch <= 'Z') evaluates to "
         << ('A' <= ch && ch <= 'Z') << endl;

    return 0;
}
```

Sample Run:

```
found = 1, age = 20, hours = 45.30, overTime = 15.00,
count = 20, ch = B

!found evaluates to 0
hours > 40.00 evaluates to 1
!age evaluates to 0
!found && (hours >= 0) evaluates to 0
!(found && (hours >= 0)) evaluates to 0
hours + overTime <= 75.00 evaluates to 1
(count >= 0) && (count <= 100) evaluates to 1
('A' <= ch && ch <= 'Z') evaluates to 1
```

Selection: if **and** if...else

- One-Way Selection
- Two-Way Selection
- Compound (Block of) Statements
- Multiple Selections: Nested **if**
- Comparing **if...else** Statements with a Series of **if** Statements

One-Way Selection

- The syntax of one-way selection is:

```
if (expression)
    statement
```

- The statement is executed if the value of the expression is **true**
- The statement is bypassed if the value is **false**; program goes to the next statement
- **i**f is a reserved word

One-Way Selection (cont'd.)

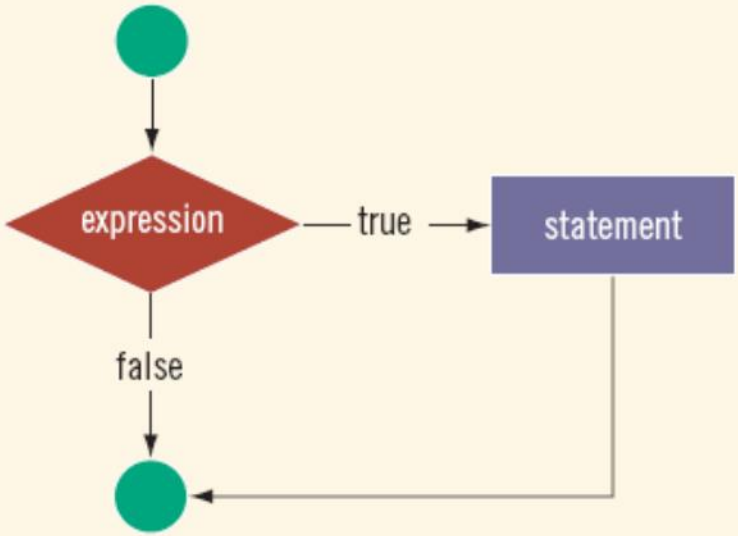


FIGURE 4-2 One-way selection

One-Way Selection (cont'd.)

EXAMPLE 4-7

```
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 4-8

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
```

One-Way Selection (cont'd.)

EXAMPLE 4-9

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 4-10

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.

Questions