

Introduction to Programming

Lecture 4: C++ Flow Control (Selection)

Dr. Nizar Saadi Dahir

Control Structures

- *A computer can proceed:*
 - *In sequence*
 - *Selectively (branch): making a choice*
 - *Repetitively (iteratively): looping*
 - *By calling a function*
- *Two most common control structures:*
 - *Selection*
 - *Repetition*

Control Structures (cont'd.)

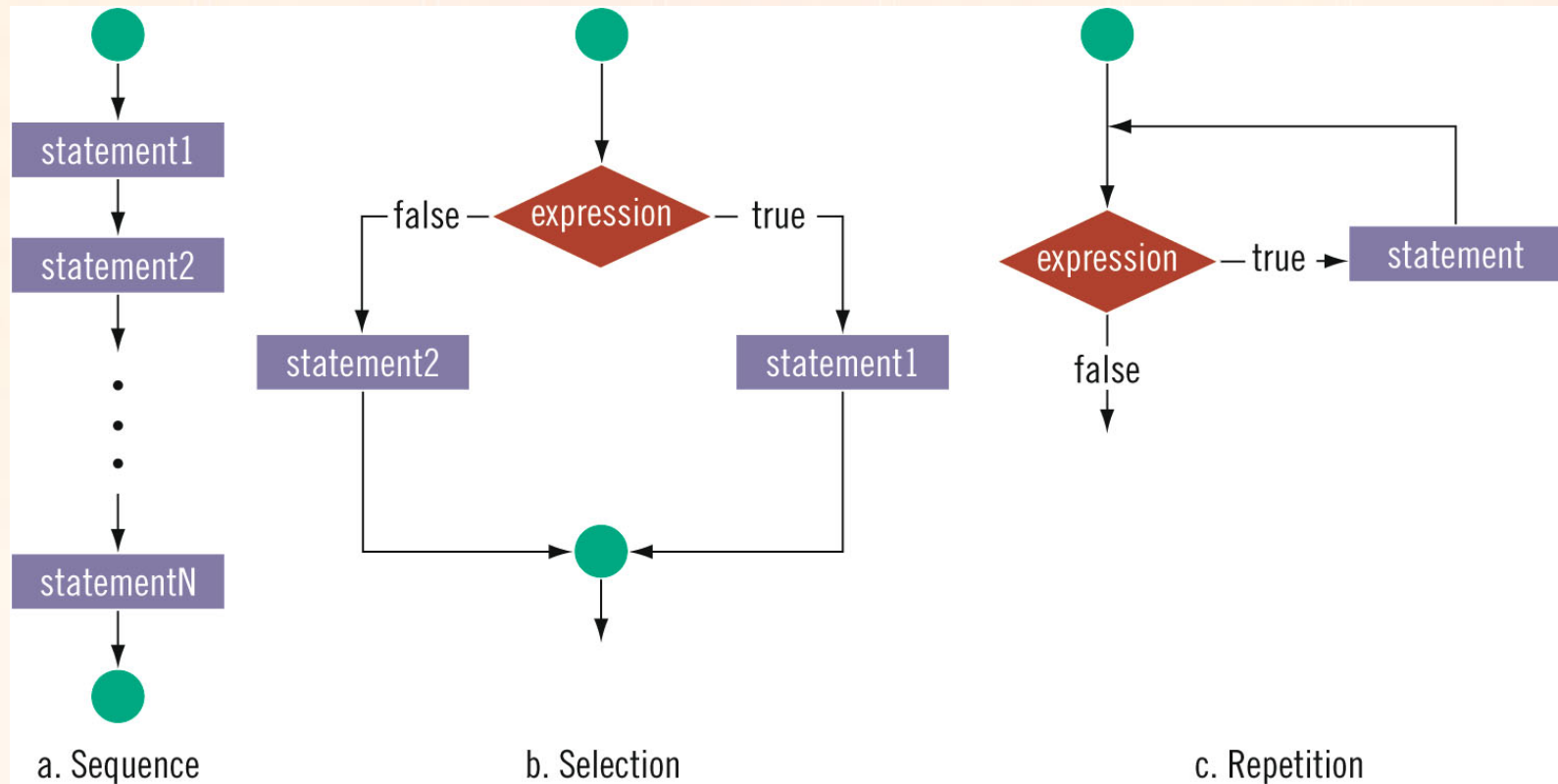


FIGURE 4-1 Flow of execution

Relational Operators

- *Conditional statements: only executed if certain conditions are met*
- *Condition: represented by a logical (Boolean) expression that evaluates to a logical (Boolean) value of *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

- *Relational operators can be used with all three simple data types:*
 - *$8 < 15$ evaluates to true*
 - *$6 \neq 6$ evaluates to false*
 - *$2.5 > 5.8$ evaluates to false*
 - *$5.9 \leq 7.5$ evaluates to true*

Comparing Characters

- *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*

- # Comparing Characters
- *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*

Logical (Boolean) Operators and Logical Expressions

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

TABLE 4-2 Logical (Boolean) Operators in C++

Operator	Description
!	not
&&	and
	or

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

TABLE 4-3 The ! (Not) Operator

Expression	!(Expression)
<code>true</code> (nonzero)	<code>false</code> (0)
<code>false</code> (0)	<code>true</code> (1)

EXAMPLE 4-3

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

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 <code> </code> Expression2
<code>true</code> (nonzero)	<code>true</code> (nonzero)	<code>true</code> (1)
<code>true</code> (nonzero)	<code>false</code> (0)	<code>true</code> (1)
<code>false</code> (0)	<code>true</code> (nonzero)	<code>true</code> (1)
<code>false</code> (0)	<code>false</code> (0)	<code>false</code> (0)

EXAMPLE 4-5

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

Short-Circuit Evaluation

- Short-circuit evaluation: evaluation of a logical expression stops as soon as the value of the expression is known
- *Example:*

```
(age >= 21) || ( x == 5)    //Line 1  
(grade == 'A') && (x >= 7) //Line 2
```

Class activity (5 min)

- *Determine the scenarios where short-circuit evaluation takes place in the following lines of C++ code ?*

```
(age >= 21) || ( x == 5)           //Line 1
```

```
(grade == 'A') && (x >= 7)        //Line 2
```

Comparing Floating-Point Numbers for Equality: A Precaution

- *Comparison of floating-point numbers for equality may not behave as you would expect*
 - *Example:*
 - $1.0 == 3.0/7.0 + 2.0/7.0 + 2.0/7.0$ evaluates to false
 - *Why?* $3.0/7.0 + 2.0/7.0 + 2.0/7.0 = 0.999999999999999989$
 - *Solution: use a tolerance value*
 - *Example:* $\text{if } \text{fabs}(x - y) < 0.000001$

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

Selection: `if` and `if...else`

- *`if` and `if...else` statements can be used to create:*
 - *One-way selection*
 - *Two-way selection*
 - *Multiple selections*

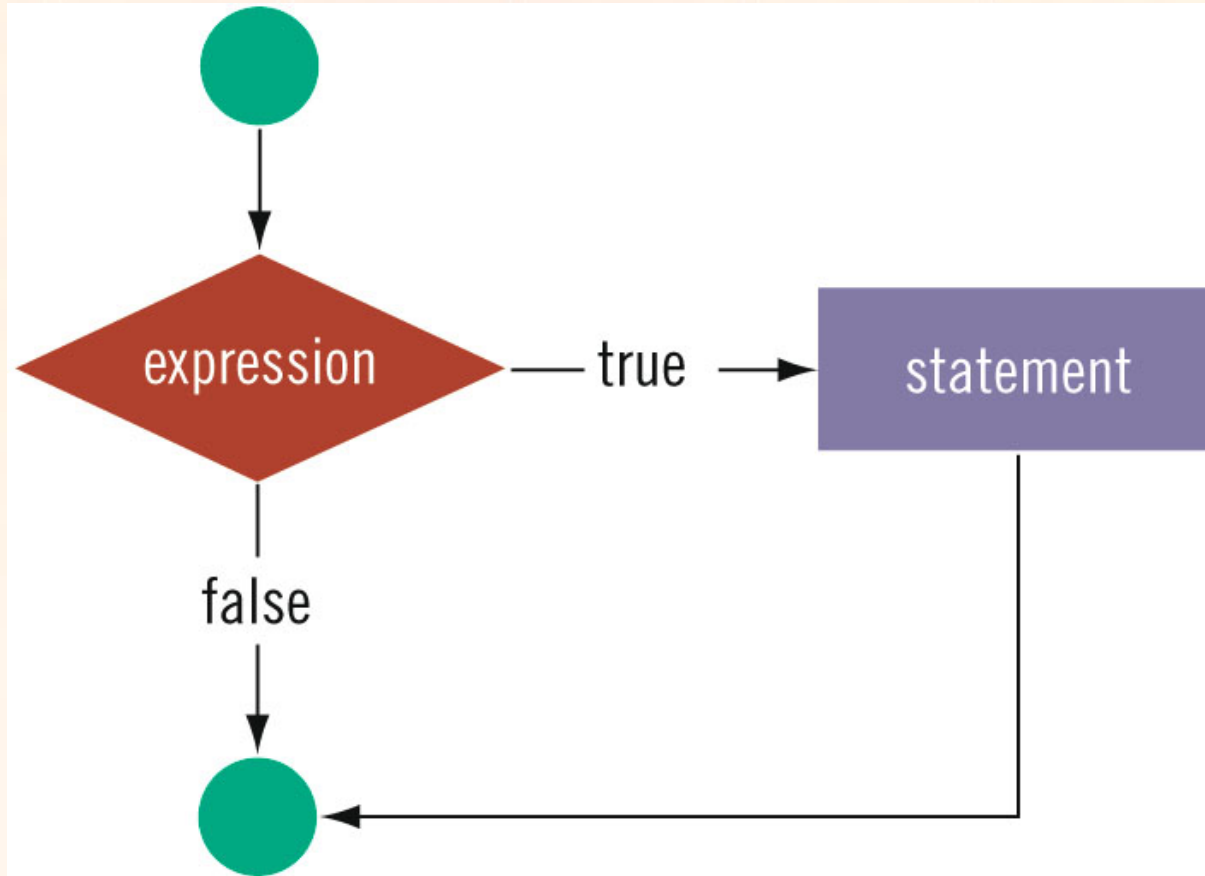
One-Way Selection

- *One-way selection syntax:*

```
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*
- *Expression is called a decision maker*

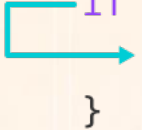
One-Way Selection (cont'd.)



One-Way Selection (cont'd.)

Condition is true

```
int number = 5;  
  
if (number > 0) {  
    // code  
}  
  
// code after if
```



Condition is false

```
int number = 5;  
  
if (number < 0) {  
    // code  
}  
  
// code after if
```



One-Way Selection (cont'd.)

```
// Program to print positive number entered by the user
// If the user enters a negative number, it is skipped

#include <iostream>
using namespace std;

int main() {

    int number;

    cout << "Enter an integer: ";
    cin >> number;

    // checks if the number is positive
    if (number >= 0) {
        cout << "You entered a positive integer: " << number << endl;
    }

    cout << "This statement is always executed.";

    return 0;
}
```

Two-Way Selection

- *Two-way selection syntax:*

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

- *If expression is true, statement1 is executed; otherwise, statement2 is executed*
 - *statement1 and statement2 are any C++ statements*

Two-Way Selection (cont'd.)

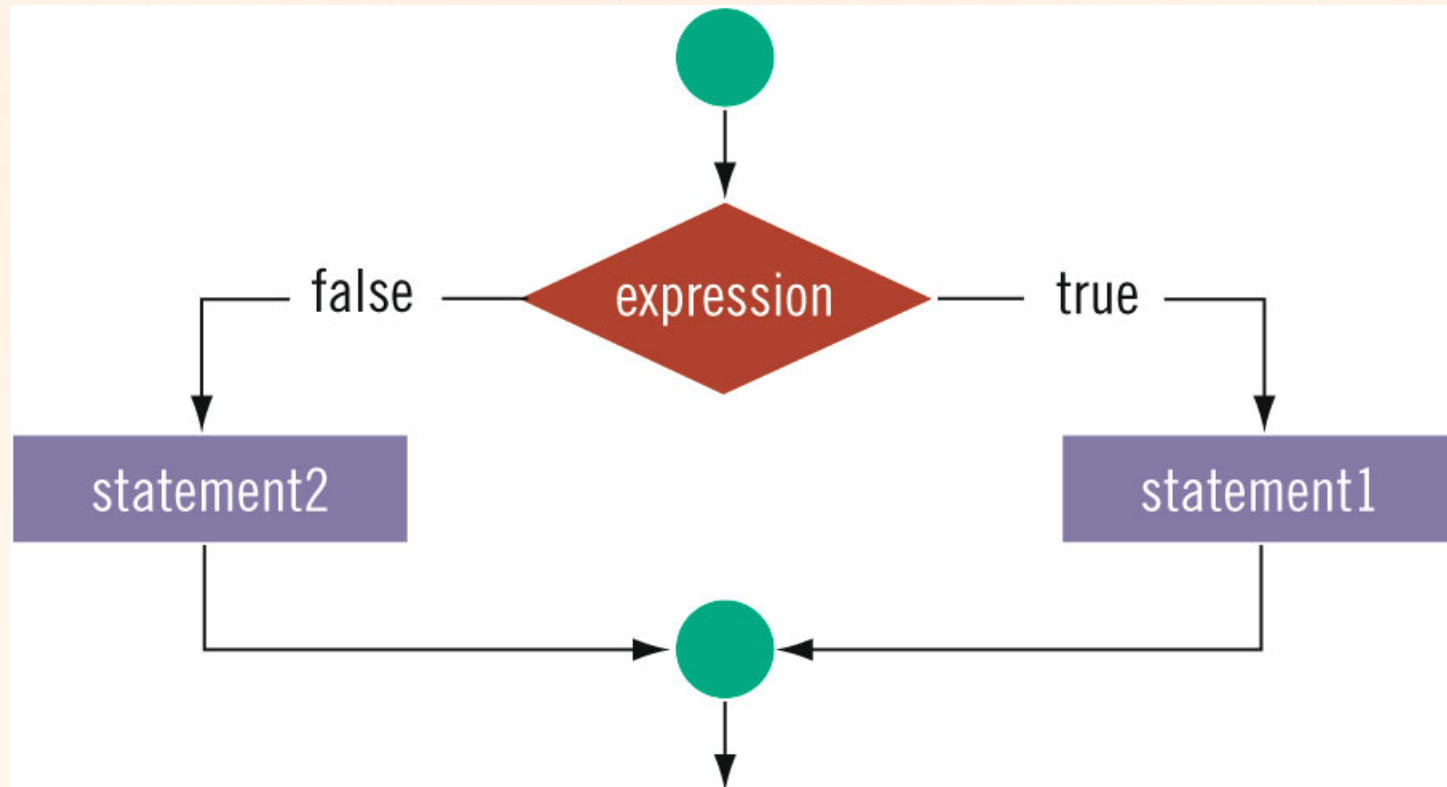


FIGURE 4-3 Two-way selection

Two-Way Selection (cont'd.)

Condition is true

```
int number = 5;
```

```
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if...else
```



Condition is false

```
int number = 5;
```

```
if (number < 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if...else
```



Two-Way Selection (cont'd.)

```
// Program to check whether an integer is positive or negative
// This program considers 0 as a positive number

#include <iostream>
using namespace std;

int main() {

    int number;

    cout << "Enter an integer: ";
    cin >> number;

    if (number >= 0) {
        cout << "You entered a positive integer: " << number << endl;
    }
    else {
        cout << "You entered a negative integer: " << number << endl;
    }

    cout << "This line is always printed.";

    return 0;
}
```

Compound (Block of) Statements

- Compound statement (block of statements):

```
{
    statement_1
    statement_2
    .
    .
    .
    statement_n
}
```

- *A compound statement functions like a single statement*

Compound (Block of) Statements (cont'd.)

```
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;
}
```

Multiple Selections: Nested if

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

Multiple Selections: Nested if (cont'd.)

EXAMPLE 4-16

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;
```

Comparing if...else Statements with a Series of if Statements

```
a.  if (month == 1)                                //Line 1
    cout << "January" << endl;                    //Line 2
else if (month == 2)                                //Line 3
    cout << "February" << endl;                   //Line 4
else if (month == 3)                                //Line 5
    cout << "March" << endl;                       //Line 6
else if (month == 5)                                //Line 9
    cout << "May" << endl;                          //Line 10
else if (month == 6)                                //Line 11
    cout << "June" << endl;                         //Line 12
```

Comparing if...else Statements with if Statements (cont'd.)

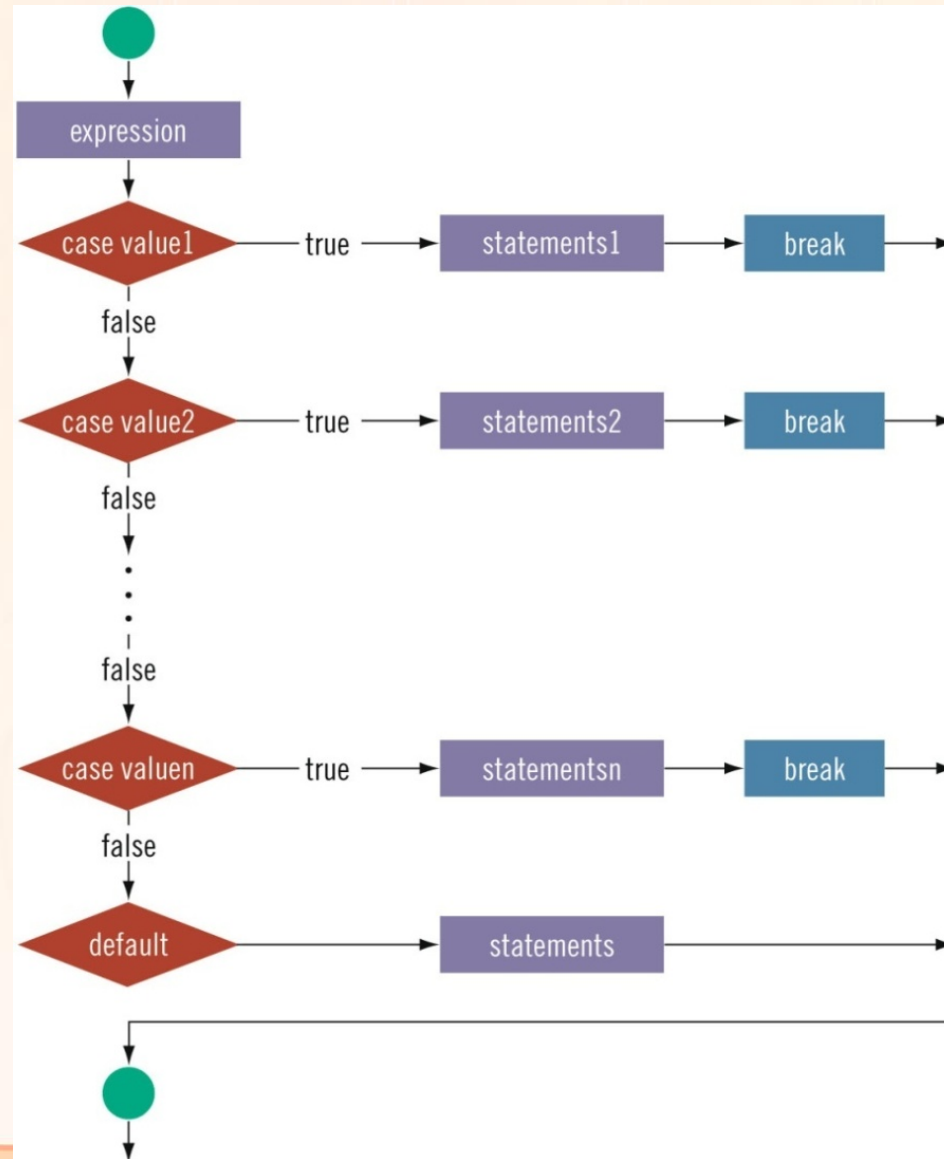
```
b.  if (month == 1)
        cout << "January" << endl;
    if (month == 2)
        cout << "February" << endl;
    if (month == 3)
        cout << "March" << endl;
    if (month == 4)
        cout << "April" << endl;
    if (month == 5)
        cout << "May" << endl;
    if (month == 6)
        cout << "June" << endl;
```

switch Structures

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

```
switch (expression)
{
    case value1:
        statements1
        break;
    case value2:
        statements2
        break;
    .
    .
    .
    case valuen:
        statementsn
        break;
    default:
        statements
}
```

switch Structures (cont'd.)



switch Structures (cont'd.)

- *One or more statements may follow a case label*
- *Braces are not needed to turn multiple statements into a single compound statement*
- *When a case value is matched, all statements after it execute until a break is encountered*
- *The break statement may or may not appear after each statement*
- *switch, case, break, and default are reserved words*

switch Structures (cont'd.)

```
// Program to build a simple calculator using switch Statement
#include <iostream>
using namespace std;

int main() {
    char oper;
    float num1, num2;
    cout << "Enter an operator (+, -): ";
    cin >> oper;
    cout << "Enter two numbers: " << endl;
    cin >> num1 >> num2;

    switch (oper) {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1 + num2;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1 - num2;
            break;
        default:
            // operator is doesn't match any case constant (+, -)
            cout << "Error! The operator is not recognized";
            break;
    }

    return 0;
}
```

Challenge

- Write a program to run as basic calculator
- There are four operations: addition +, subtraction -, multiplication *, and division /.
- Return the result of the operation specified by *op* on the numbers *num1* and *num2*.
- Use two methods:
 1. if ... else if
 2. switch case
- Example runs:

```
Enter a arithmetic expression: 6*34  
6 * 34 = 204
```

```
Enter a arithmetic expression: 33.5/11  
33.5 / 11 = 3.04545
```