Misr International University (MIU)

Faculty of Computer Science

Computer Science Program

CSC 105: Introduction to Programming & Problem Solving

Dr. Ayman Ezzat and Dr. Ashraf AbdelRaouf

Lecture 1

Review

(Input / Output, Control Structure, Functions)

Escape Sequences

TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

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

Two-Way (if…else) Selection

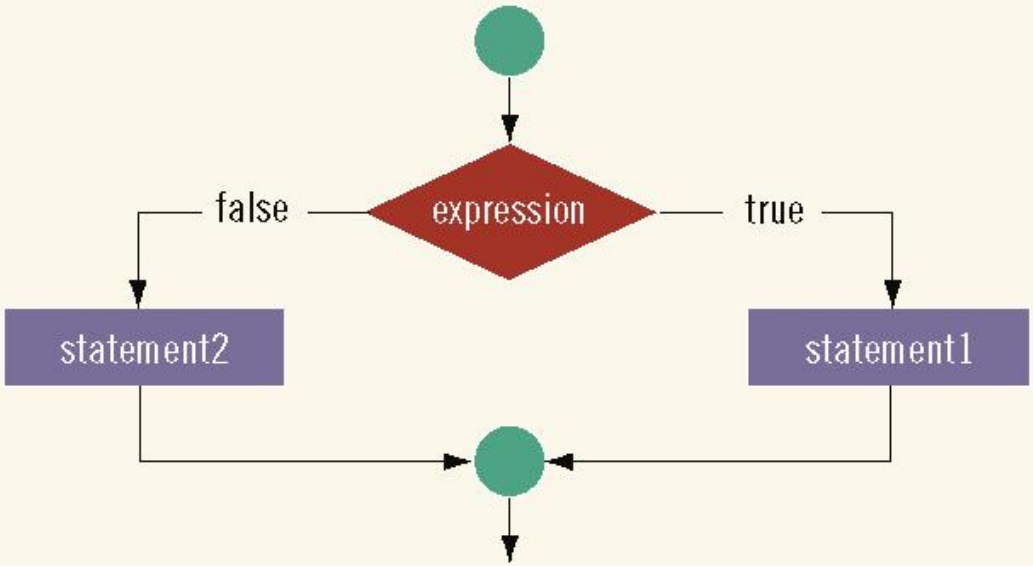

FIGURE 4-3 Two-way selection

Example: if Statement

3. Write a program that prompts the user to input an integer between 0 and 35. If the number is less than or equal to 9, the program should output the number; otherwise, it should output A for 10, B for 11, C for 12 . . . and Z for 35. (*Hint:* Use the cast operator, `static_cast<char>( )`, for numbers >= 10.)

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`

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`

```
cout<<((No%2==0)?"Odd":"Even")<<endl;
```

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

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

switch Structures

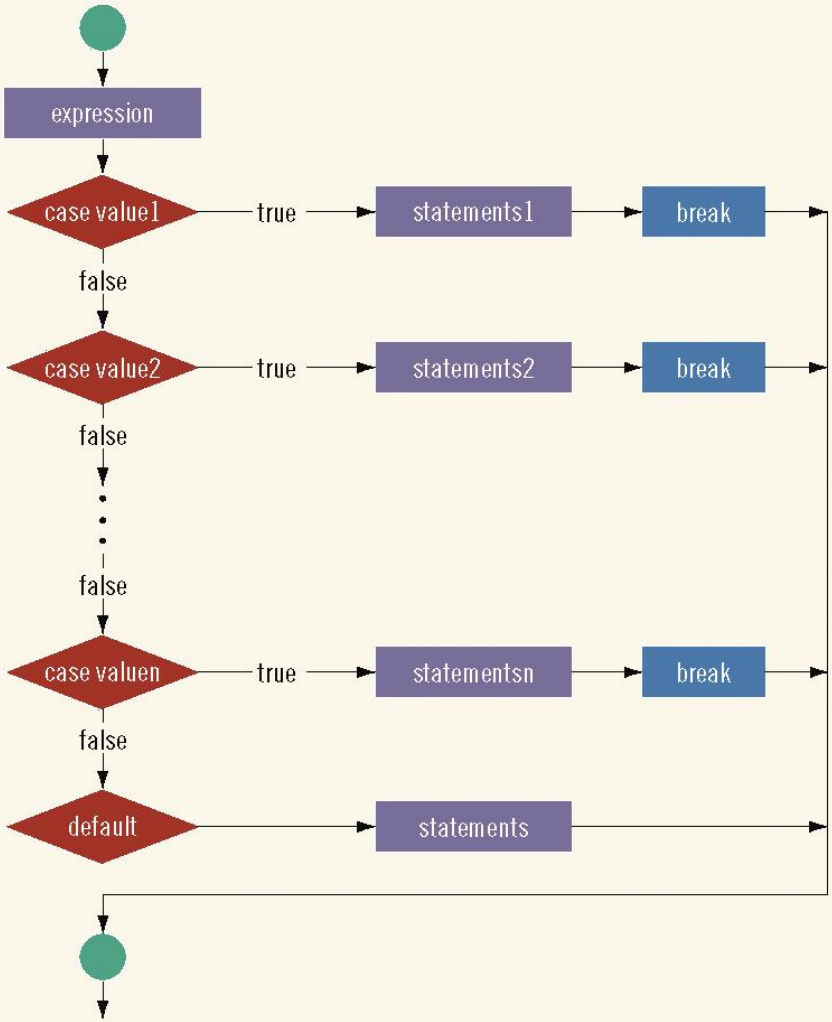

FIGURE 4-4 switch statement

Example: switch

9. Write a program that mimics a calculator. The program should take as input two integers and the operation to be performed. It should then output the numbers, the operator, and the result. (For division, if the denominator is zero, output an appropriate message.) Some sample outputs follow:

```
3 + 4 = 7
13 * 5 = 65
```

for Looping (Repetition) Structure

- The general form of the for statement is:

```
for (initial statement; loop condition; update statement)
    statement
```

- The initial statement, loop condition, and update statement are called for loop control statements
  - initial statement usually initializes a variable (called the for **loop control**, or for **indexed**, **variable**)
- In C++, for is a reserved word

for Looping (Repetition) Structure (continued)

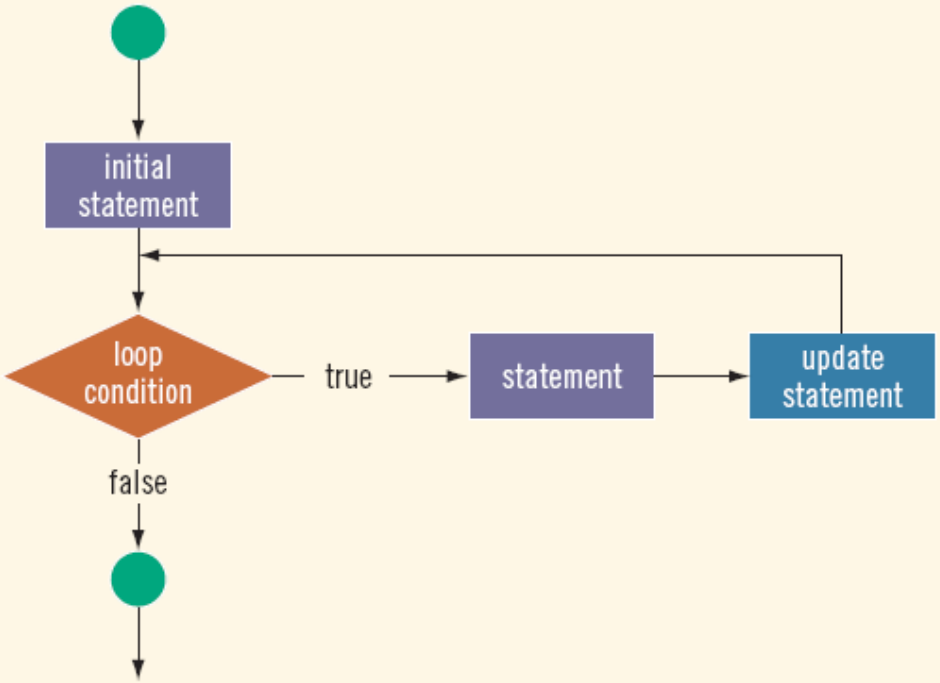

FIGURE 5-2 for loop

Example: for Looping

4.14 ***(Factorials)*** The *factorial* function is used frequently in probability problems. The factorial of a positive integer *n* (written *n*! and pronounced "*n* factorial") is equal to the product of the positive integers from 1 to *n*. Write a program that evaluates the factorials of the integers from 1 to 5. Print the results in tabular format. What difficulty might prevent you from calculating the factorial of 20?

while Looping (Repetition) Structure

- The general form of the while statement is:

```
while (expression)
    statement
```

  while is a reserved word
- Statement can be simple or compound
- Expression acts as a decision maker and is usually a logical expression
- Statement is called the body of the loop
- The parentheses are part of the syntax

`while` Looping (Repetition) Structure (continued)

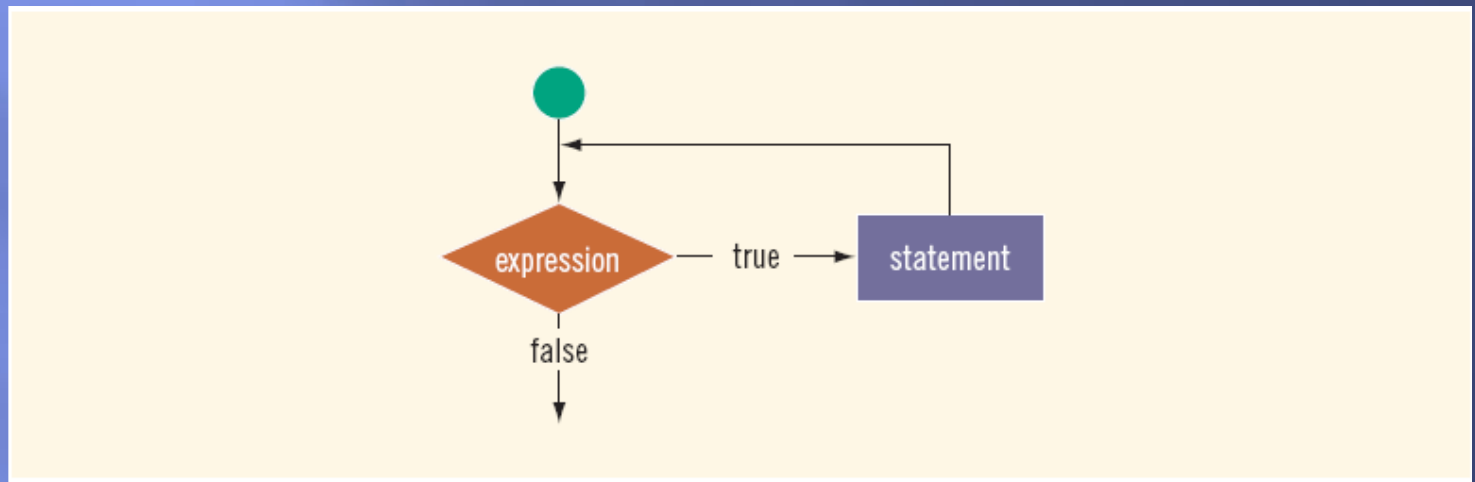

FIGURE 5-1 `while` loop

- Infinite loop: continues to execute endlessly
  - Avoided by including statements in loop body that assure exit condition is eventually `false`

do...while Looping (Repetition) Structure

- General form of a do...while:

```
do
    statement
while (expression);
```

- The statement executes first, and then the expression is evaluated
- To avoid an infinite loop, body must contain a statement that makes the expression false
- The statement can be simple or compound
- Loop always iterates at least once

do...while Looping (Repetition) Structure (continued)

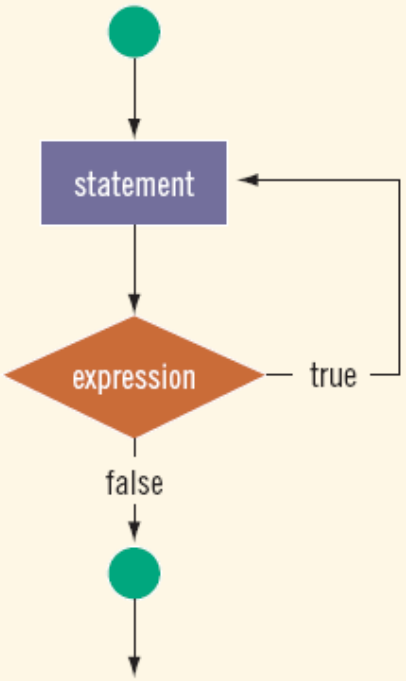

FIGURE 5-3 do...while loop

Choosing the Right Looping Structure

- All three loops have their place in C++
  - If you know or can determine in advance the number of repetitions needed, the `for` loop is the correct choice
  - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a `while` loop
  - If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a `do...while` loop

Predefined Functions

- Some of the predefined mathematical functions are:

```
sqrt(x)
pow(x, y)
floor(x)
```

- Predefined functions are organized into separate libraries
- I/O functions are in `iostream` header
- Math functions are in `cmath` header

Example: Predefined Functions

5.8 Show the value of x after each of the following statements is performed:

a) `x = fabs( 7.5 );`
b) `x = floor( 7.5 );`
c) `x = fabs( 0.0 );`
d) `x = ceil( 0.0 );`
e) `x = fabs( -6.4 );`
f) `x = ceil( -6.4 );`
g) `x = ceil( -fabs( -8 + floor( -5.5 ) ) );`

User-Defined Functions

- Value-returning functions: have a return type
  - Return a value of a specific data type using the `return` statement
- Void functions: do not have a return type
  - *Do not* use a `return` statement to return a value

Function Prototype

- Function prototype: function heading without the body of the function
- Syntax:

```
functionType functionName(parameter list);
```

- It is not necessary to specify the variable name in the parameter list
- The data type of each parameter must be specified

Void Functions

- Void functions and value-returning functions have similar structures
  - Both have a heading part and a statement part
- User-defined void functions can be placed either before or after the function `main`
- If user-defined void functions are placed after the function `main`
  - The function prototype must be placed before the function `main`
- A void function does not have a return type
  - `return` statement without any value is typically used to exit the function early
- Formal parameters are optional
- A call to a void function is a stand-alone statement

Example: Functions

5.41 ***(Distance Between Points)*** Write function `distance` that calculates the distance between two points *(x1, y1)* and *(x2, y2)*. All numbers and return values should be of type `double`.

A player rolls two dice. Each die has six faces. These faces contain 1, 2, 3, 4, 5, and 6 spots. After the dice have come to rest, the sum of the spots on the two upward faces is calculated. If the sum is 7 or 11 on the first throw, the player wins. If the sum is 2, 3, or 12 on the first throw (called "craps"), the player loses (i.e., the "house" wins). If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, then that sum becomes the player's "point." To win, you must continue rolling the dice until you "make your point." The player loses by rolling a 7 before making the point.

Value Parameters

- If a formal parameter is a value parameter
  - The value of the corresponding actual parameter is copied into it
- The value parameter has its own copy of the data
- During program execution
  - The value parameter manipulates the data stored in its own memory space

Reference Variables as Parameters

- If a formal parameter is a reference parameter
  - It receives the memory address of the corresponding actual parameter
- During program execution to manipulate data
  - Changes to formal parameter will change the corresponding actual parameter

Reference Variables as Parameters (cont'd.)

- Reference parameters are useful in three situations:
  - Returning more than one value
  - Changing the actual parameter
  - When passing the address would save memory space and time

Recursive Definitions

- Recursion: solving a problem by reducing it to smaller versions of itself
- $0! = 1$ (1)

  $n! = n \times (n-1)!$ if $n > 0$ (2)
- The definition of factorial in equations (1) and (2) is called a recursive definition
- Equation (1) is called the base case
- Equation (2) is called the general case