COMSATS University
Islamabad
(Lahore Campus)

CSC103- Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 4:
Control Structures I (Selection)

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

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 the `string` Type (cont'd.)

```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str5 = "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**.

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

```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str5 = "Big";
```

`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 the `string` Type (cont'd.)

```
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str5 = "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`.

Compound (Block of) Statements

- Compound statement (block of statements):

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

- A compound statement consists of one or more statements enclosed in curly braces, { and }.
- 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
- in C++, there is no stand-alone else statement. Every else must be paired with an if. The rule to pair an else with an if is as follows:
  - Pairing an else with an if: In a nested if statement, C++ associates an else with the most recent incomplete if—that is, the most recent if that has not been paired with an else.

Multiple Selections: Nested `if` (cont'd.)

EXAMPLE 4-17

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 == 4)                     //Line 7
    cout << "April" << endl;             //Line 8
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;
```

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
```

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 = 0.9999999999999989`
- Solution: use a tolerance value
  - Example: `if fabs(x - y) < 0.000001`

Associativity of Relational Operators: A Precaution

```
#include <iostream>

using namespace std;

int main()
{
     int num;

     cout << "Enter an integer: ";
     cin >> num;
     cout << endl;

     if (0 <= num <= 10)
         cout << num << " is within 0 and 10." << endl;


   else
       cout << num << " is not within 0 and 10." << endl;

   return 0;
}
```

Associativity of Relational Operators: A Precaution (cont'd.)

- `num = 5`

`0 <= num <= 10`	`= 0 <= 5 <= 10`	
	`= (0 <= 5) <= 10`	(Because relational operators are evaluated from left to right)
	`= 1 <= 10`	(Because `0 <= 5` is `true`, `0 <= 5` evaluates to 1)
	`= 1 (true)`	

- `num = 20`

`0 <= num <= 10`	`= 0 <= 20 <= 10`	
	`= (0 <= 20) <= 10`	(Because relational operators are evaluated from left to right)
	`= 1 <= 10`	(Because `0 <= 20` is `true`, `0 <= 20` evaluates to 1)
	`= 1 (true)`	

Input Failure and the `if` Statement

- If input stream enters a fail state
  - All subsequent input statements associated with that stream are ignored
  - Program continues to execute
  - May produce erroneous results
- Can use `if` statements to check status of input stream
- If stream enters the fail state, include instructions that stop program execution

Confusion Between the Equality (==) and Assignment (=) Operators

- C++ allows you to use any expression that can be evaluated to either `true` or `false` as an expression in the `if` statement:

```
if (x = 5)
    cout << "The value is five." << endl;
```

- The appearance of = in place of == resembles a *silent killer*
  - It is not a syntax error
  - It is a logical error

Conditional Operator (?:)

- Conditional operator (`?:`)
  - Ternary operator: takes 3 arguments
- Syntax for the conditional operator:

```
expression1 ? expression2 : expression3
```

- If `expression1` is `true`, the result of the conditional expression is `expression2`
  - Otherwise, the result is `expression3`
- Example: `max = (a >= b) ? a : b;`

Nested Conditional Operator (?:)

- The conditional operator can also be “nested” just like other selection statements. See example below:
- Assume grade and gpa already declared:

```
grade=='a' ? gpa=4.0 : grade=='b' ? gpa=3.0 : grade=='c' ? gpa=2.0 : gpa=0.0;
```

- Each differently colored statement above starts a new conditional operator.

Conditional Operator and output statement

- The conditional operator can also be used with the cout statement as below.

```
cout << (a >= b) ? a : b;
```

OR

```
cout << (a >= b) ? "A" : "B" << " is larger!";
```

- In the second example above, the result of the conditional expression is either "A" or "B", so either one of them will be printed.

Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques

- Must use concepts and techniques correctly
  - Otherwise solution will be either incorrect or deficient
- If you do not understand a concept or technique completely
  - Don’t use it
  - Save yourself an enormous amount of debugging time

Program Style and Form (Revisited): Indentation

- A properly indented program:
  - Helps you spot and fix errors quickly
  - Shows the natural grouping of statements
- Insert a blank line between statements that are naturally separate
- Two commonly used styles for placing braces
  - On a line by themselves
  - Or left brace is placed after the expression, and the right brace is on a line by itself

Using Pseudocode to Develop, Test, and Debug a Program

- Pseudocode, or just pseudo
  - Informal mixture of C++ and ordinary language
  - Helps you quickly develop the correct structure of the program and avoid making common errors
- Use a wide range of values in a walk-through to evaluate the program

`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
}
```

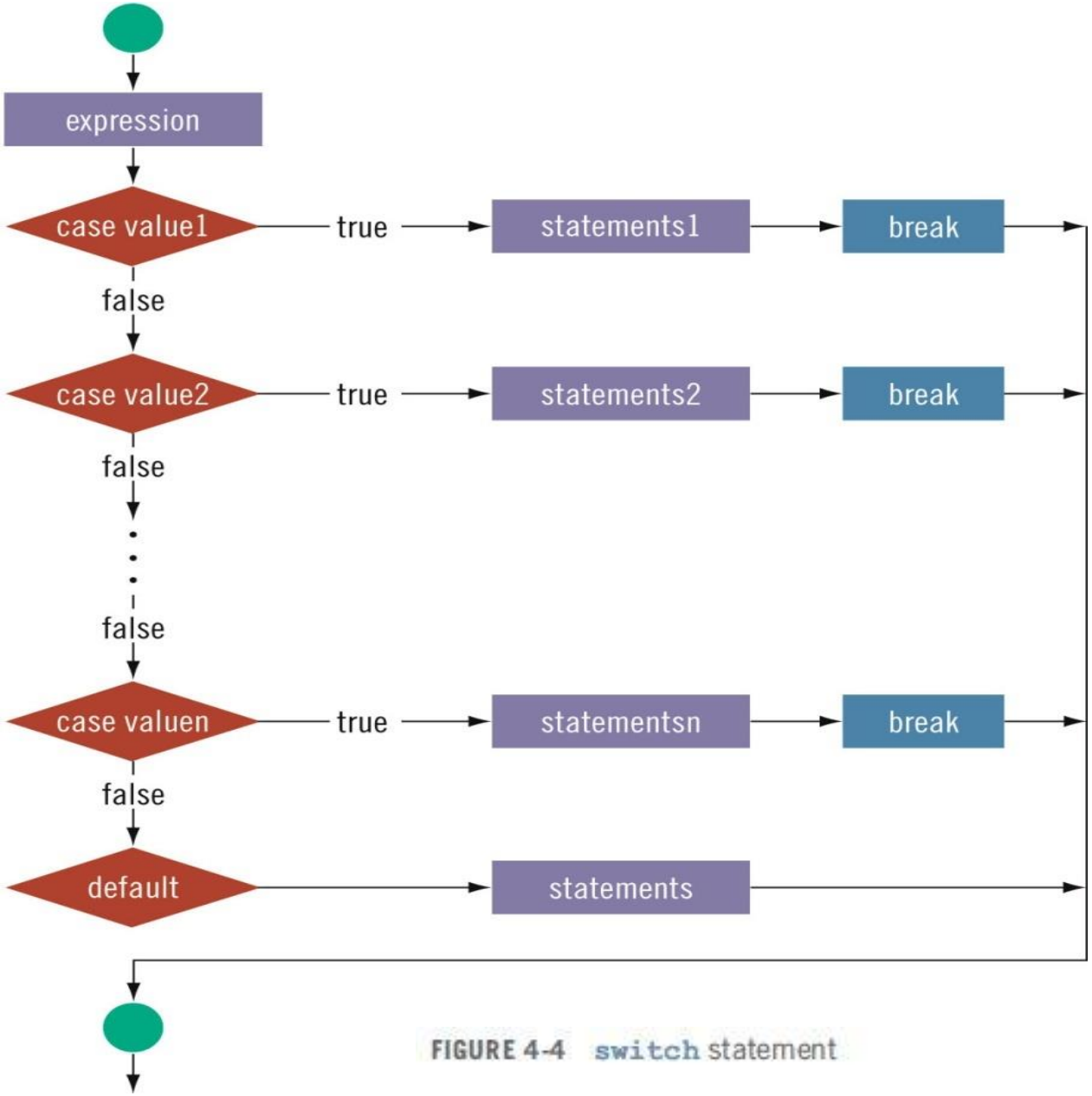

FIGURE 4-4 `switch` statement

`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

EXAMPLE 4-22

Consider the following statements, in which grade is a variable of type char:

```
switch (grade)
{
case 'A':
    cout << "The grade point is 4.0.";
    break;
case 'B':
    cout << "The grade point is 3.0.";
    break;
case 'C':
    cout << "The grade point is 2.0.";
    break;
case 'D':
    cout << "The grade point is 1.0.";
    break;
case 'F':
    cout << "The grade point is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

`Switch`: The Fall-through effect

- Missing a break statement in a case inside the switch statement may result in the so-called **fall through effect**.
  - All the subsequent cases will run (no matter the case is matched or not), unless another break statement appears.
  - Example on Next slide

switch Structures (cont'd.)

- A break statement missing in case 'B' and 'C'.
  - If grade contains 'B', then case 'B' will be matched.
  - But, as there are no break statement after case 'B', so case 'C' and case 'D' will also run. The break after case 'D' will then end the switch statement.

EXAMPLE 4-22

Consider the following statements, in which grade is a variable of type char:

```
switch (grade)
{
case 'A':
    cout << "The grade point is 4.0.";
    break;
case 'B':
    cout << "The grade point is 3.0.";

case 'C':
    cout << "The grade point is 2.0.";

case 'D':
    cout << "The grade point is 1.0.";
    break;
case 'F':
    cout << "The grade point is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

Avoiding Bugs: Revisited

- To output results correctly
  - Consider whether the `switch` structure must include a `break` statement after each `cout` statement

Terminating a Program with the `assert` Function

- Certain types of errors are very difficult to catch
  - Example: division by zero
- `assert` function: useful in stopping program execution when certain such errors occur

The `assert` Function

- Syntax:

```
assert(expression);
```

  - `expression` is any logical expression
- If `expression` evaluates to `true`, the next statement executes
- If `expression` evaluates to `false`, the program terminates and indicates where in the program the error occurred
- To use `assert`, include `cassert` header file

The `assert` Function

```
int numerator;
int denominator;
int quotient;

assert(denominator);
quotient = numerator / denominator;
```

- Now, if denominator is 0, the assert statement halts the of the program with an error message similar to the following:

```
Assertion failed: denominator, file c:\temp\assert
function\assertfunction.cpp, line 20
```

The `assert` Function (cont'd.)

- `assert` is useful for enforcing programming constraints during program development
- After developing and testing a program, remove or disable `assert` statements
- The preprocessor directive `#define NDEBUG` must be placed before the directive `#include <cassert>` to disable the `assert` statement

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 (cont'd.)

- 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 stand-alone `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 (cont'd.)

- Using assignment in place of the equality operator creates a semantic error
- `switch` structure handles multiway selection
- `break` statement ends `switch` statement
- Use `assert` to terminate a program if certain conditions are not met