- Office Location: room 339
- Email: xidong@pvamu.edu
- Office Hour:

Monday to Wednesday

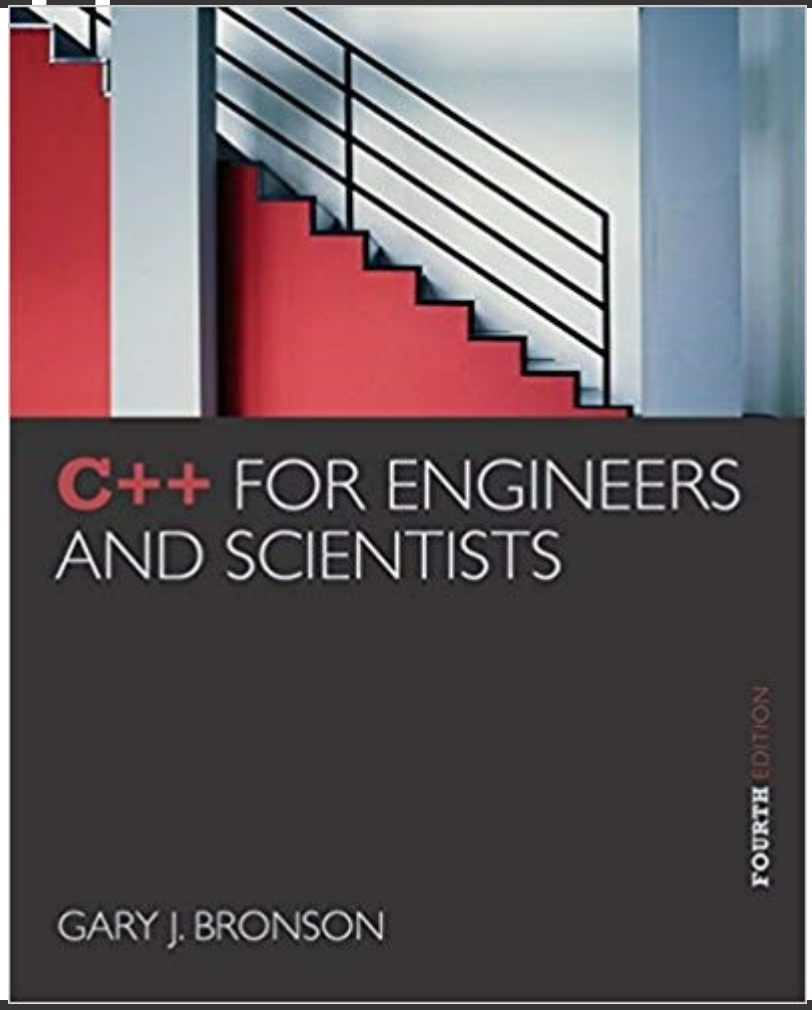
12:00 pm to 15:00 pm

Tuesday

10:00 am to 11:00am

Recommendation: Meeting Appointment with email

ELEG 1043
Computer Applications in Engineering



Chapter 4:
Selection Structures

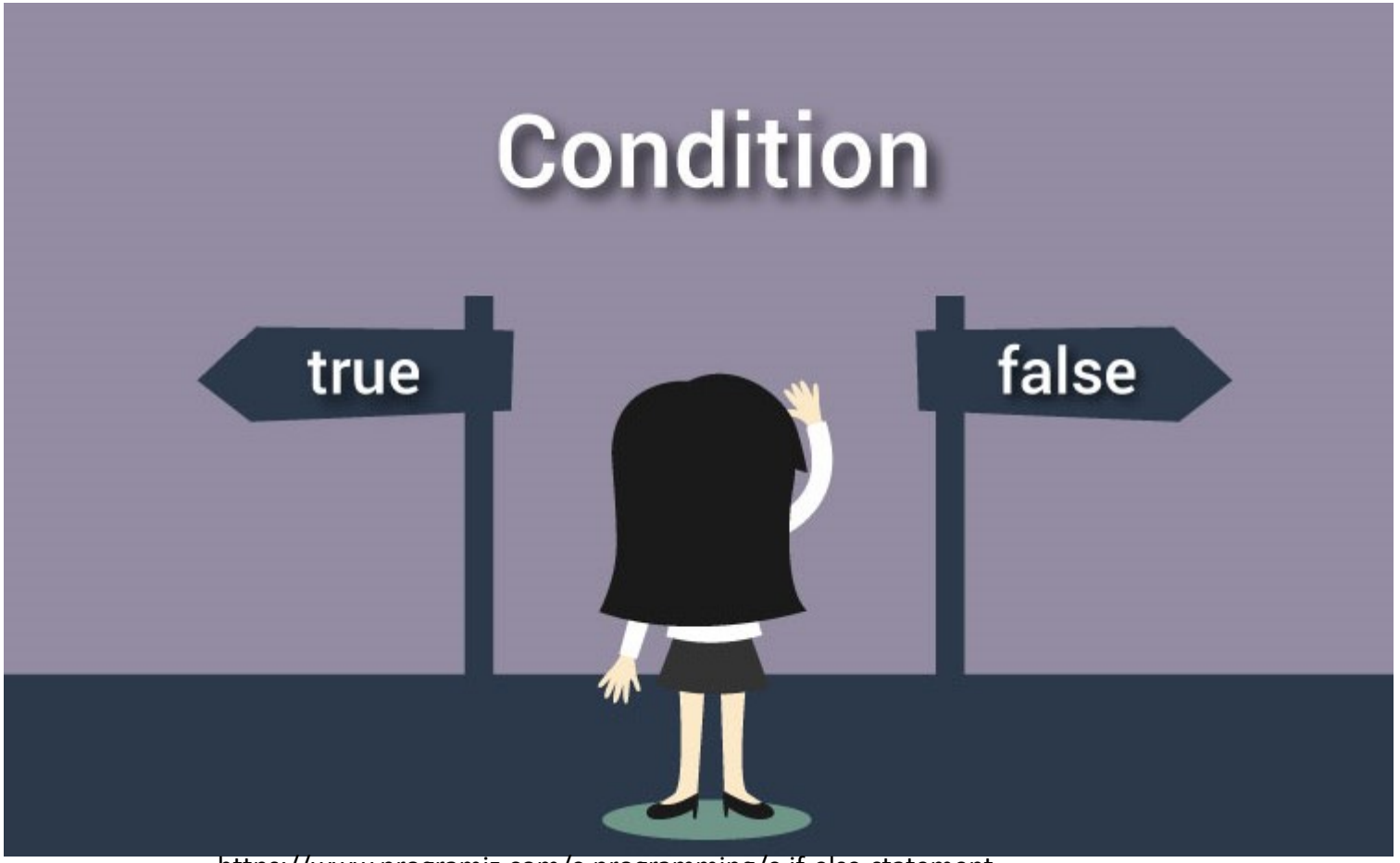
Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Objectives

- In this chapter, you will learn about:
  - Selection criteria
  - The **`if-else`** statement
  - Nested **`if`** statements
  - The **`switch`** statement
  - Program testing
  - Common programming errors

Selection Criteria



https://www.programiz.com/c-programming/c-if-else-statement

Selection Criteria

- **`if-else`** statement: Implements a decision structure for two alternatives

  Syntax:

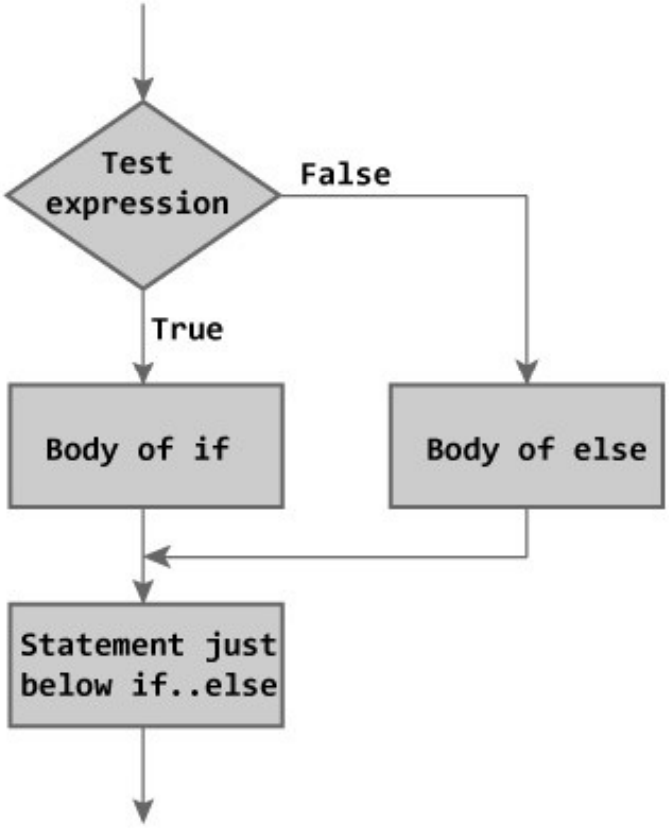
  if (condition)

  statement executed if condition is true;

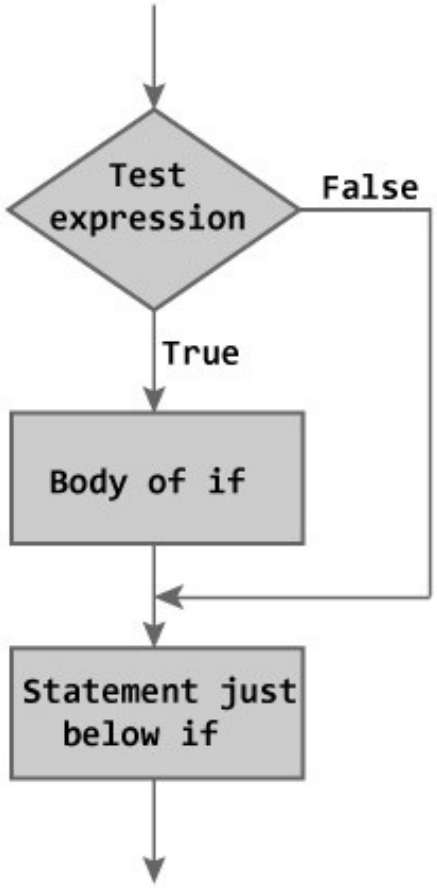
  else

  statement executed if condition is false;

Flowchart of if statement



Flowchart of if statement



Selection Criteria (continued)

- The condition is evaluated to its numerical value:
  - A non-zero value is considered to be **true**
  - A zero value is considered to be **false**
- The **`else`** portion is optional
  - **Executed only if the condition is false**
- The condition may be any valid C++ expression

Relational Operators

- **Relational expression:** Compares two operands or expressions using **relational operators**

Relational Operator	Meaning	Example
<	Less than	`age < 30`
>	Greater than	`height > 6.2`
<=	Less than or equal to	`taxable <= 20000`
>=	Greater than or equal to	`temp >= 98.6`
==	Equal to	`grade == 100`
!=	Not equal to	`number != 250`

Table 4.1 C++'s Relational Operators

Logical Operators

- AND (`&&`): Condition is true only if both expressions are true
- OR (`||`): Condition is true if either one or both of the expressions is true
- NOT (`!`): Changes an expression to its opposite state; true becomes false, false becomes true

One-Way Selection

- **One-way selection:** An **`if`** statement without the optional **`else`** portion

```
int a = 1;
if(a > 0)
{
    cout<<a;
}
```

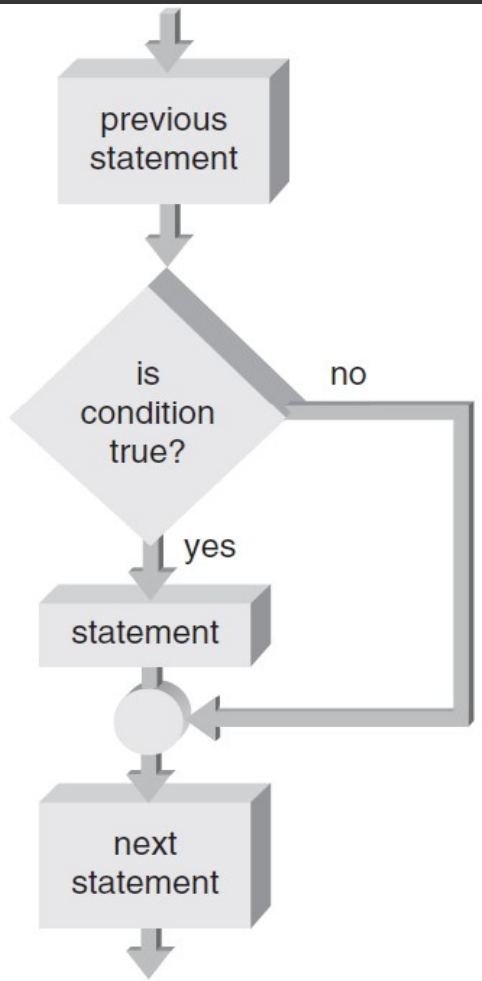


Figure 4.3 A one-way selection **`if`** statement

Problems Associated with the `if-else` Statement

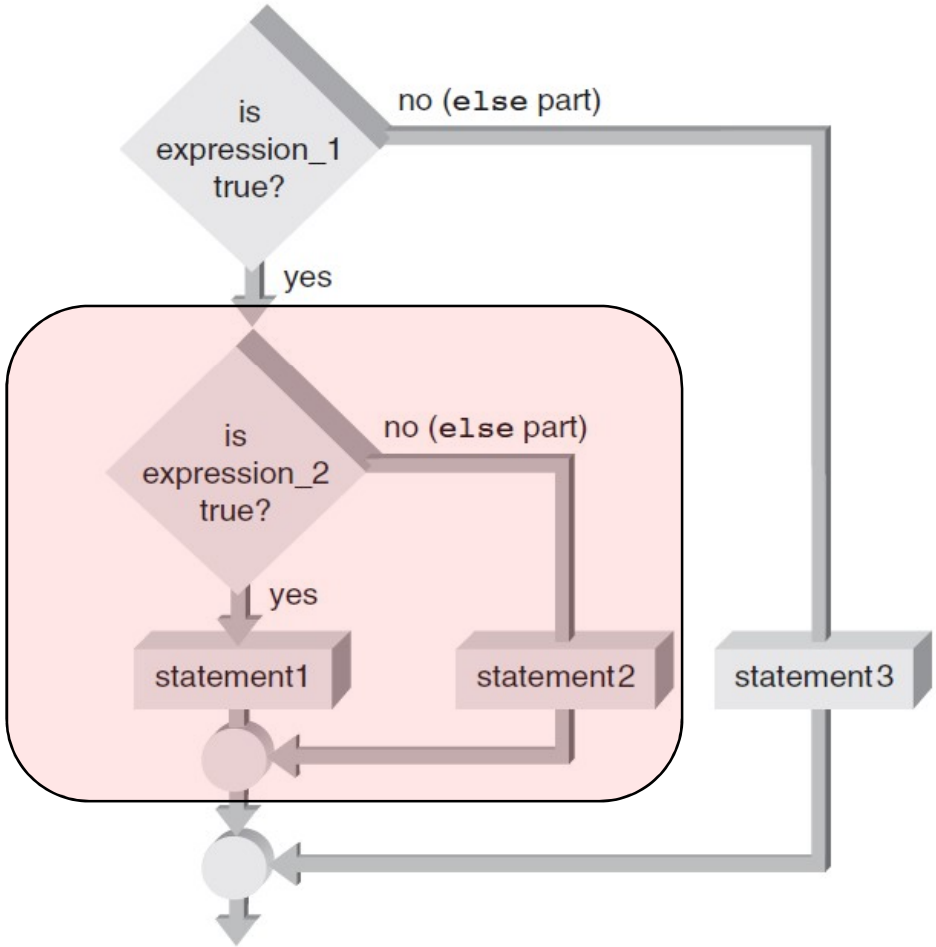
- Common problems with **`if-else`** statements:
  - Misunderstanding what an expression is
  - Using the assignment operator (**=**) instead of the relational operator (**==**)

Nested `if` Statements

- **`if-else`** statement can contain any valid C++ statement, including another **`if-else`**
- Nested **`if`** statement: an **`if-else`** statement completely contained within another **`if-else`**
- Use braces to block code, especially when inner **`if`** statement does not have its own **`else`**

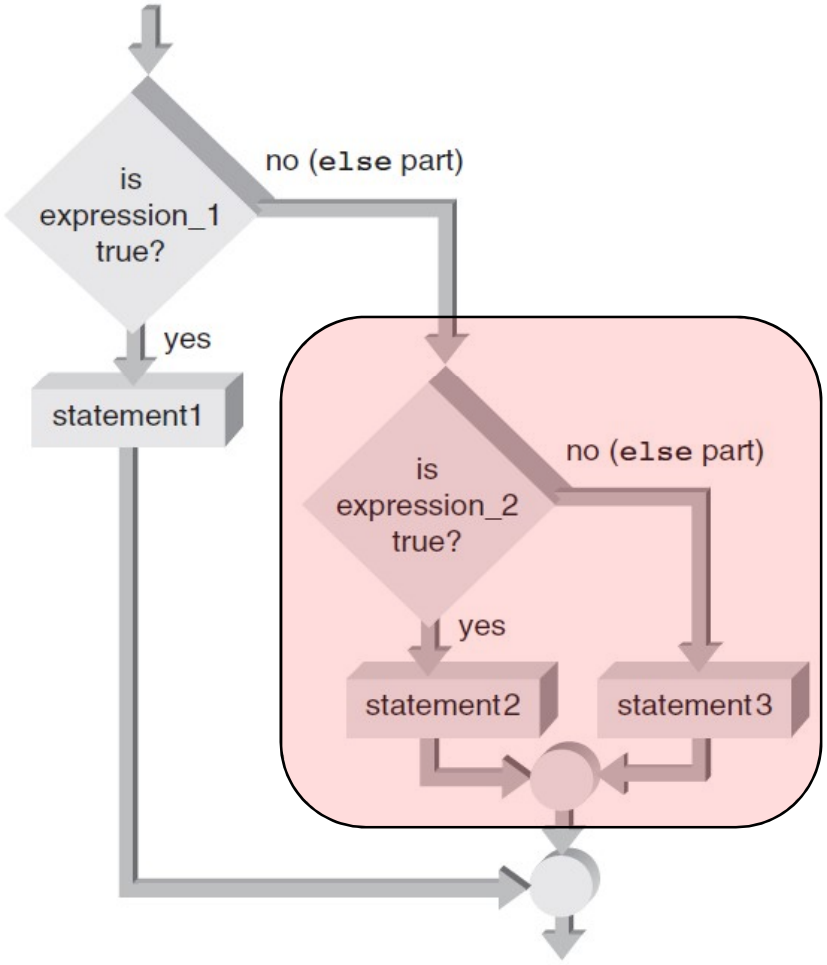
Nested `if` Statements (continued)

Figure 4.4a
Nested within the `if` part



Nested `else` Statements (continued)

Figure 4.4b
Nested within the **`else`** part



The `if-else` Chain

- If any condition is true, the corresponding statement is executed and the chain terminates
- Final **`else`** is only executed if no conditions were true
  - Serves as a catch-all case
- **`if-else`** chain provides one selection from many possible alternatives

The `if-else` Chain (continued)

- General form of an **`if-else`** chain

```
if (expression_1)
  statement1;
else if (expression_2)
  statement2;
else if (expression_3)
  statement3;
      .
      .
      .
else if (expression_n)
  statementn;
else
  last_statement;
```

Example 3

```
#include <iostream>
using namespace std;

int main()
{
    int number1, number2;
    cout<<"Enter two integers: \n";
    cin>>number1>>number2;

    if( number1 ==  number2)
          cout<<number1<<“  is equal to “<<number2;
    else if( number1 >  number2)
          cout<<number1<<“  is larger than“<<number2;
    else
          cout<<number1<<“  is smaller than“<<number2;

    return 0;

}
```

The `switch` Statement

https://www.programiz.com/c-programming/c-switch-case-statement

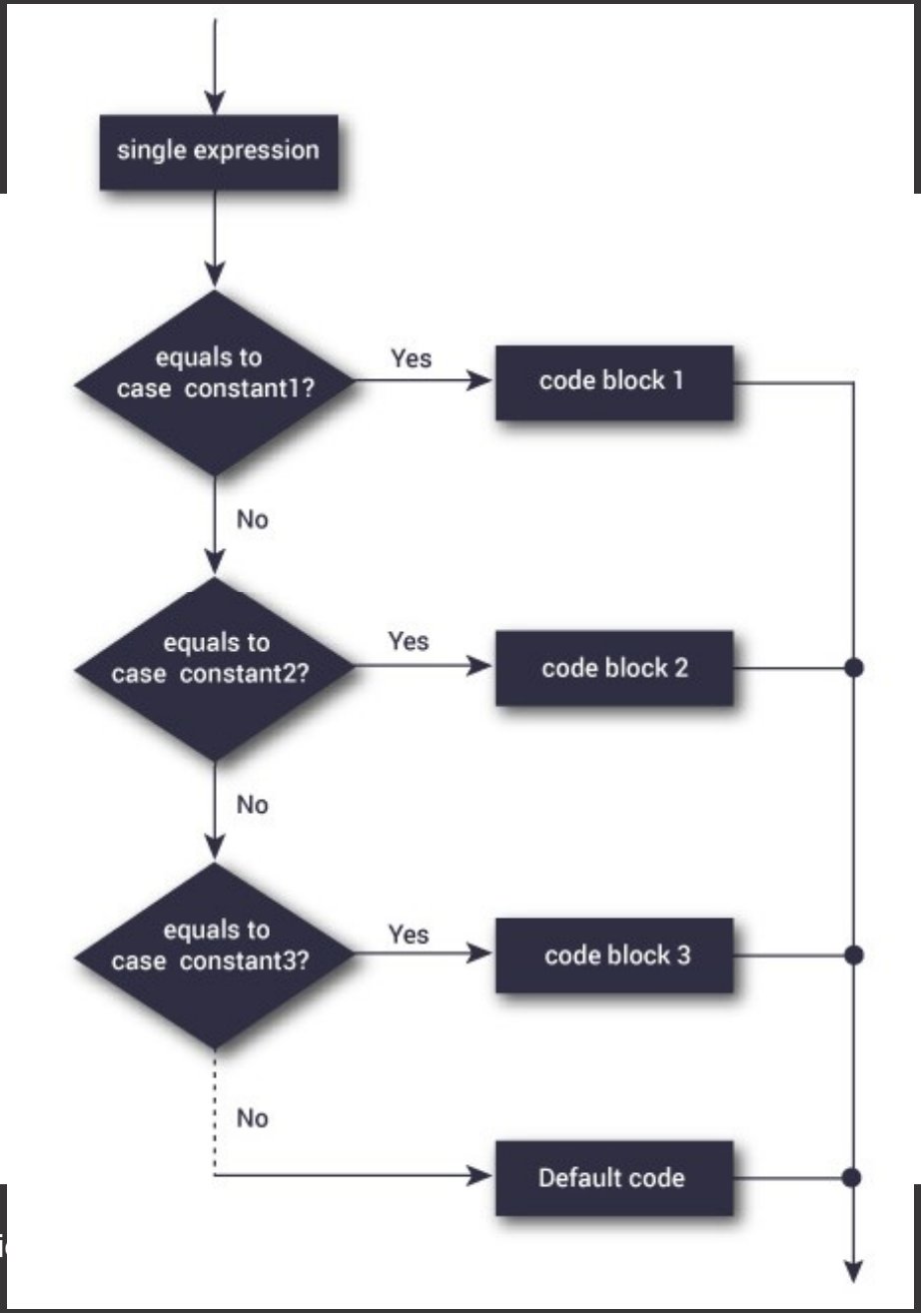
The `switch` Statement

- **`switch`** statement: Provides for one selection from many alternatives
- **`switch`** keyword starts the statement
  - Is followed by the expression to be evaluated
- **`case`** keyword identifies a value to be compared to the switch expression
  - When a match is found, statements in this **`case`** block are executed

The `switch` Statement (continued)

- **`default`** case is executed if no other case value matches were found
- **`default`** case is optional

The `switch` Statement



The `switch` Statement (continued)

```
 switch(expression)
{
    case constant-expression :
        statement(s);
        break;
    case constant-expression :
        statement(s);
        break;
     …
    default :
        statement(s); }
```

Example 4

```
#include <iostream>
using namespace std;

int main()
{
     int num=2;

    switch(num)
    {
            case 1:
                        cout<<"Case1: Value is: ”<<num;  break;
            case 2:
                        cout<<"Case2: Value is: ”<<num;  break;
            case 3:
                        cout<<"Case3: Value is: ”<<num;  break;
            default:
                        cout<<“Default: Value is: ”<<num;
    }

    return 0;

}
```

A Case Study: Solving Quadratic Equations

- **Data validation:** Use defensive programming techniques to validate user input
  - Includes code to check for improper data before an attempt is made to process it further
- **Solving quadratic equations:** Use the software development procedure to solve for the roots of a quadratic equation

A Closer Look: Program Testing

- Theory: A comprehensive set of test runs would test all combinations of input and computations, and would reveal all errors
- Reality: There are too many combinations to test for any program except a very simple one
- Example:
  - One program with 10 modules, each with five **`if`** statements, always called in the same order
  - There are 2^5 paths through each module, and more than 2^{50} paths through the program!

A Closer Look: Program Testing (continued)

- Conclusion: there is no error-free program, only one in which no errors have recently been encountered

Common Programming Errors

- Using the assignment operator (=) instead of the relational operator (==) for an equality test
- Placing a semicolon immediately after the condition
- Assuming a structural problem with an `if-else` causes the error instead of focusing on the data value being tested

Summary

- Relational expressions, or conditions, are used to compare operands
- If the relation expression is true, its value is `1`; if false, its value is `0`
- Use logical operators `&&` (AND), `||` (OR), and `!` (NOT) to construct complex conditions
- **`if-else`** allows selection between two alternatives

Summary (continued)

- An `if` expression that evaluates to `0` is false; if non-zero, it is true
- `if` statements can be nested
- Chained `if` statement provides a multiway selection
- Compound statement: contains any number of individual statements enclosed in braces

Summary (continued)

- **`switch`** statement: Provides a multiway selection
- **`switch`** expression: Evaluated and compared to each **`case`** value
  - If a match is found, execution begins at that case's statements and continues unless a **`break`** is encountered