Ch 6:Decision Making and Branching

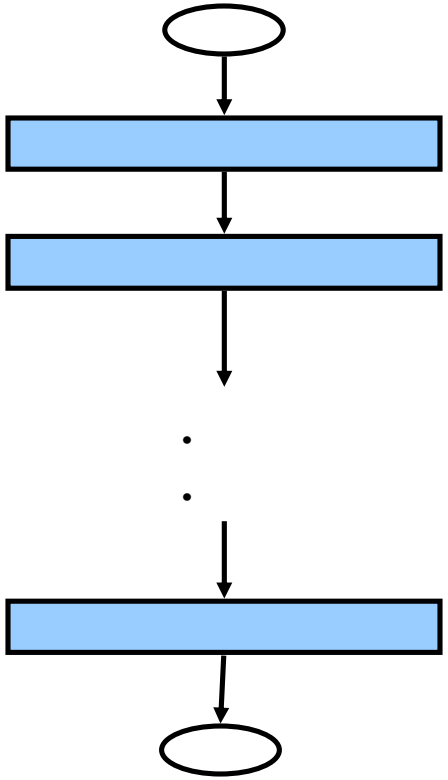
By : Ansari Rafiya

Sequential Programming Revisited

```
1   //  Addition.cs
2   // An addition program.
3
4   using System;
5
6   class Addition
7   {
8     static void Main( string[] args )
9     {
10       string firstNumber,   // first string entered by user
11              secondNumber;  // second string entered by user
12
13       int number1,          // first number to add
14           number2,          // second number to add
15           sum;              // sum of number1 and number2
16
17       // prompt for and read first number from user as string
18       Console.Write( "Please enter the first integer: " );
19       firstNumber = Console.ReadLine();
20
21       // read second number from user as string
22       Console.Write( "\nPlease enter the second integer: " );
23       secondNumber = Console.ReadLine();
24
25       // convert numbers from type string to type int
26       number1 = Int32.Parse( firstNumber );
27       number2 = Int32.Parse( secondNumber );
28
29       // add numbers
30       sum = number1 + number2;
31
32       // display results
33       Console.WriteLine( "\nThe sum is {0}.", sum );
34
35    } // end method Main
36
37  } // end class Addition
```

```
17      // prompt for and read first number from user as string
18      Console.Write( "Please enter the first integer: " );
19      firstNumber = Console.ReadLine();
20
21      // read second number from user as string
22      Console.Write( "\nPlease enter the second integer: " );
23      secondNumber = Console.ReadLine();
24
25      // convert numbers from type string to type int
26      number1 = Int32.Parse( firstNumber );
27      number2 = Int32.Parse( secondNumber );
28
29      // add numbers
30      sum = number1 + number2;
31
32      // display results
33      Console.WriteLine( "\nThe sum is {0}.", sum );
```

Sequence Structure (Flowchart)



Each of these statements could be:

- a variable declaration
- an assignment statement
- a method call (e.g., `Console.WriteLine( …);`)

More Interesting: Control Statements

- Selection (conditional statements): decide whether or not to execute a particular statement; these are also called the selection statements or decision statements
  - **`if`** selection (one choice)
  - **`if/else`** selection (two choices)
    - Also: the ternary conditional operator $\mathbf{e_1?e_2:e_3}$
  - **`switch`** statement (multiple choices)

- Repetition (loop statements): repeatedly executing the same statements (for a certain number of times or until a test condition is satisfied).
  - **`while`** structure
  - **`do/while`** structure
  - **`for`** structure
  - **`foreach`** structure (Chapter 12)

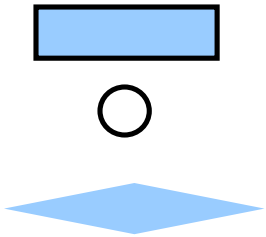
Why Control Statements ?

- Last few classes: a sequence of statements
  - *Sequential programming*
- Most programs need more flexibility in the order of statement execution
- The order of statement execution is called the flow of control

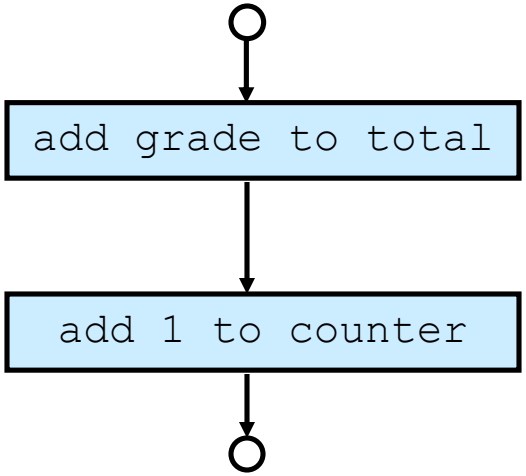
Pseudocode & Flowcharts to Represent Flow of Control

- Pseudocode
  - Artificial and informal language
  - Helps programmers to plan an algorithm
  - Similar to everyday English
  - Not an actual programming language

- Flowchart --- *a graphical way of writing pseudocode*
  - Rectangle used to show action
  - Circles used as connectors
  - Diamonds used as decisions



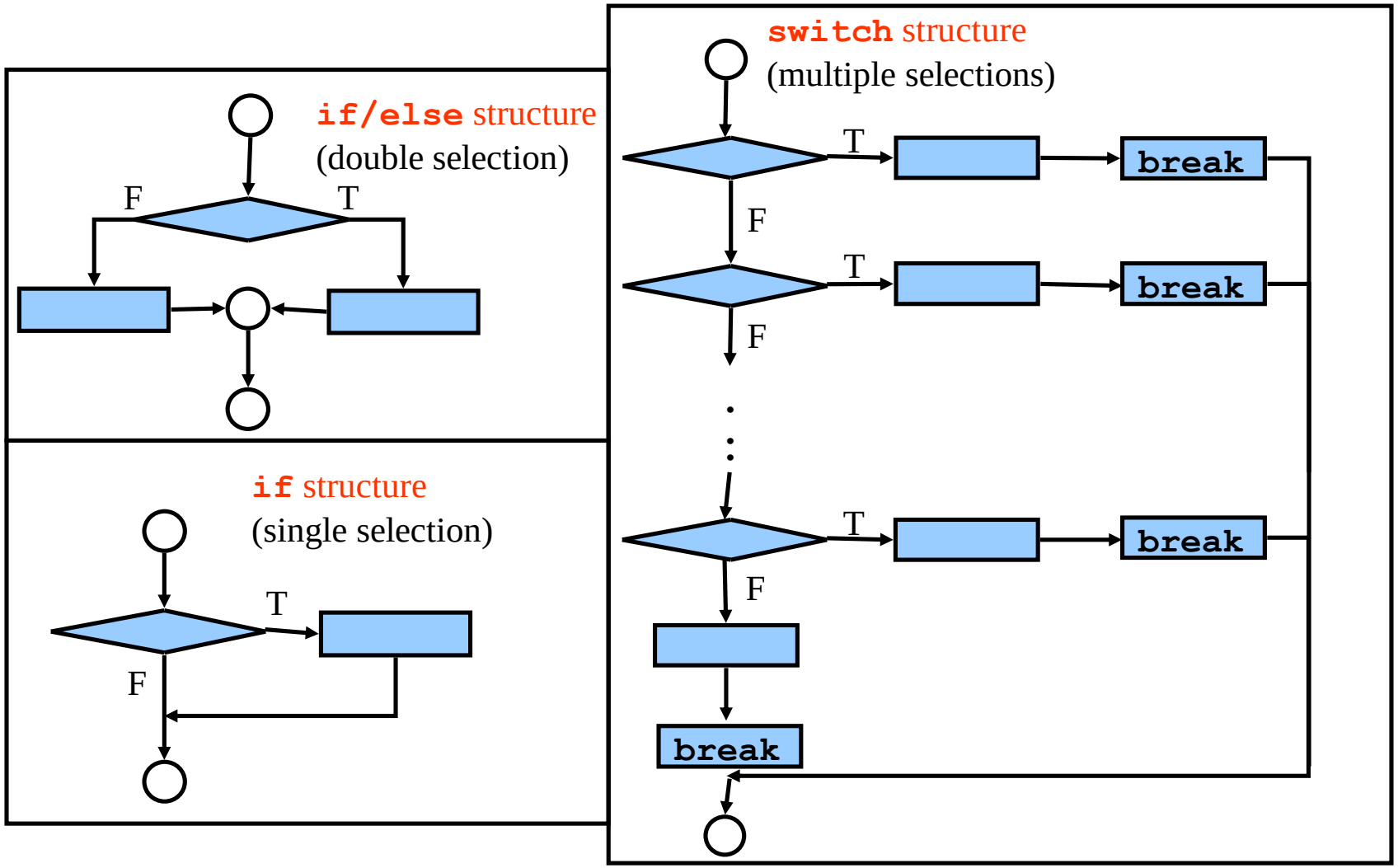
Sequence Structure



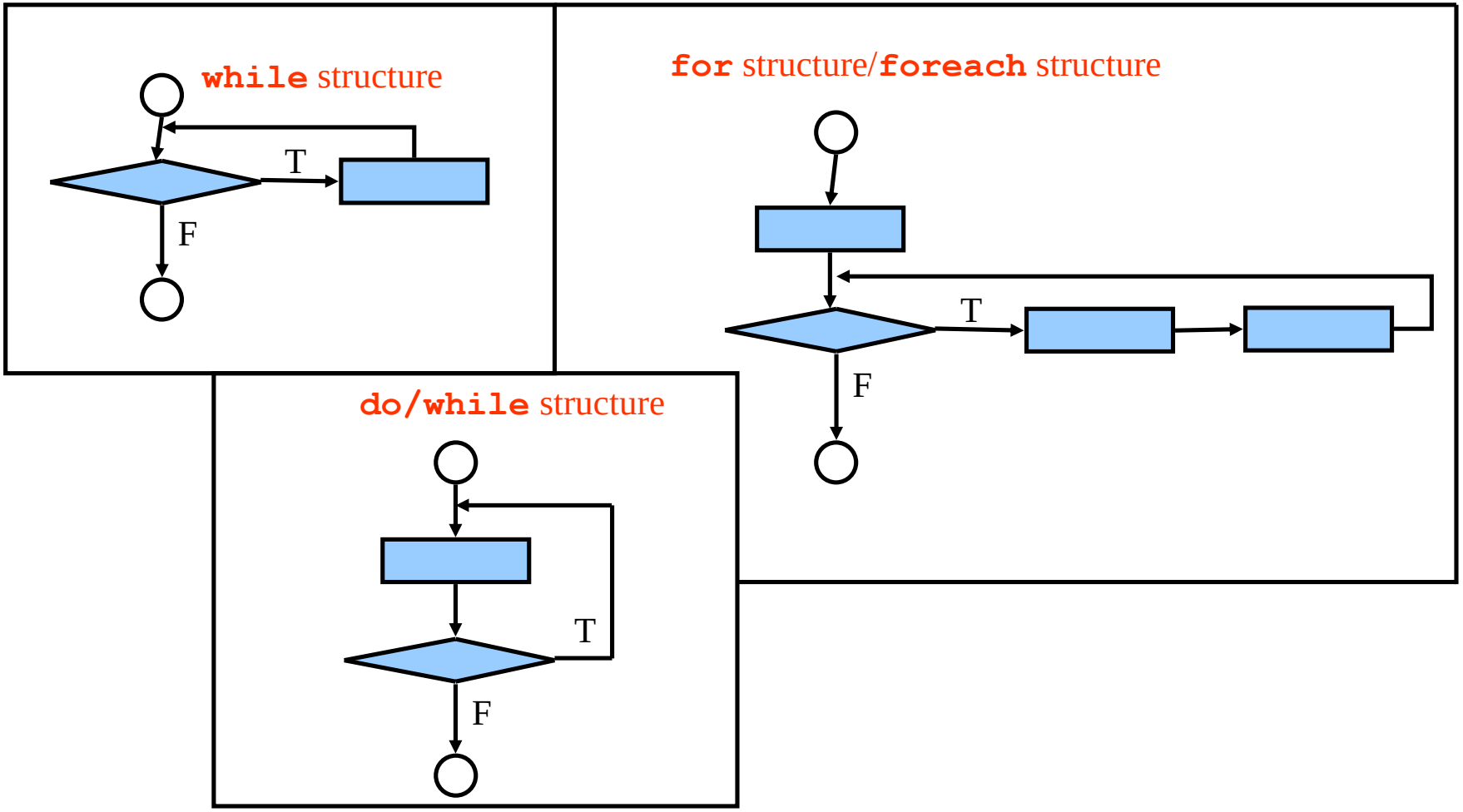

```
total = total + grade;

counter = counter + 1;
```

C# Control Structures: Selection



C# Control Structures: Repetition



The `if` Statement

- The *`if` statement* has the following syntax:

`if` is a C# reserved word

The condition must be a *boolean expression*. It must evaluate to either true or false.

```
if ( condition )
    statement;
```

If the condition is true, the statement is executed. If it is false, the statement is skipped.

`if` Statement

```
if ( <test> )
    <code executed if <test> is true> ;
```

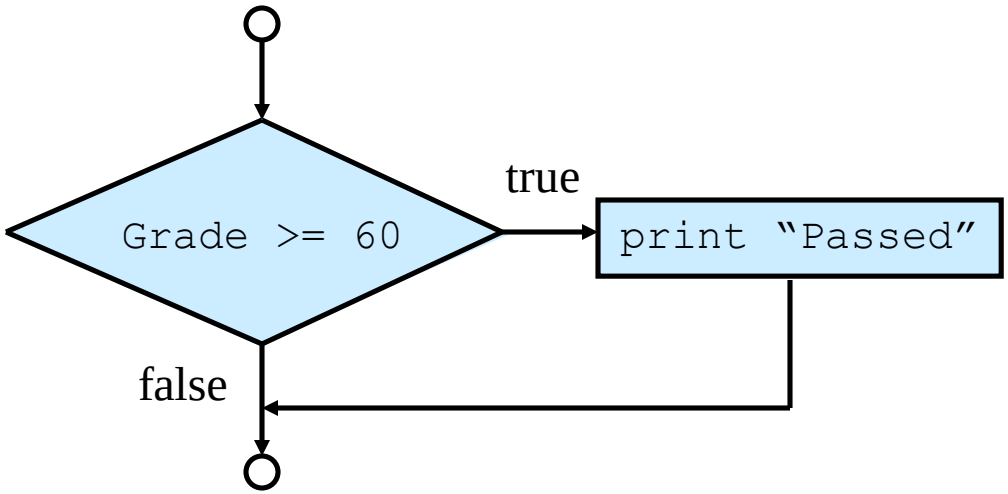
- The **`if`** statement
  - Causes the program to make a selection
  - Chooses based on conditional
    - *<test>:* any expression that evaluates to a **`bool`** type
    - **True**: perform an action
    - **False**: skip the action
  - Single entry/exit point
  - *No semicolon after the condition*

if Statement (cont'd)

```
if ( <test> )
{
    <code executed if <test> is true>  ;
    ......
    <more code executed if <test> is true>  ;
}
```

- The body of the branch can also be a block statement!
- *No semicolon after the condition*
- *No semicolon after the block statement*

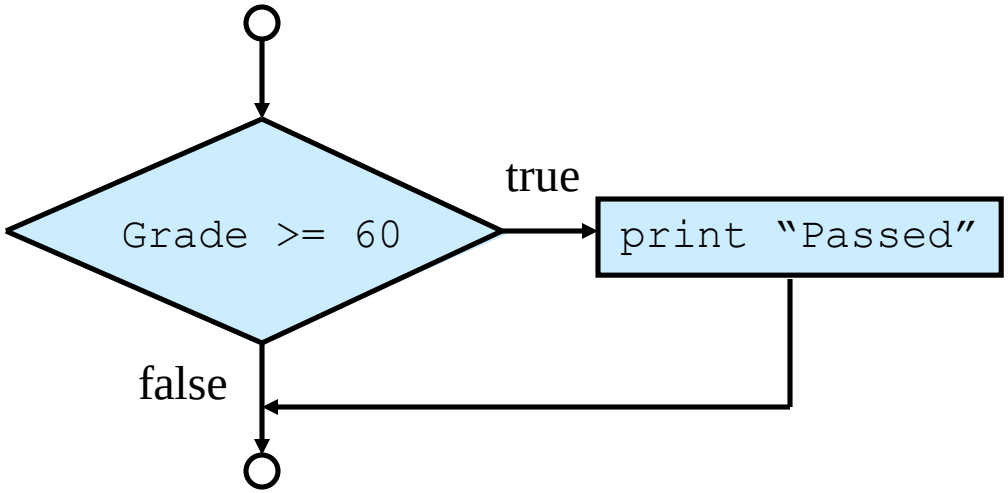
if Statement (cont'd)




```
if (studentGrade >= 60)
  Console.WriteLine ("Passed");


// beginning of the next statement
```

if Statement (cont'd)




```
if (studentGrade >= 60)
{
  Console.WriteLine ("Passed");
}


// beginning of the next statement
```

`if/else` Statement

```
if ( <test> )
   <code executed if <test> is true>  ;
else
   <code executed if <test> is false>  ;
```

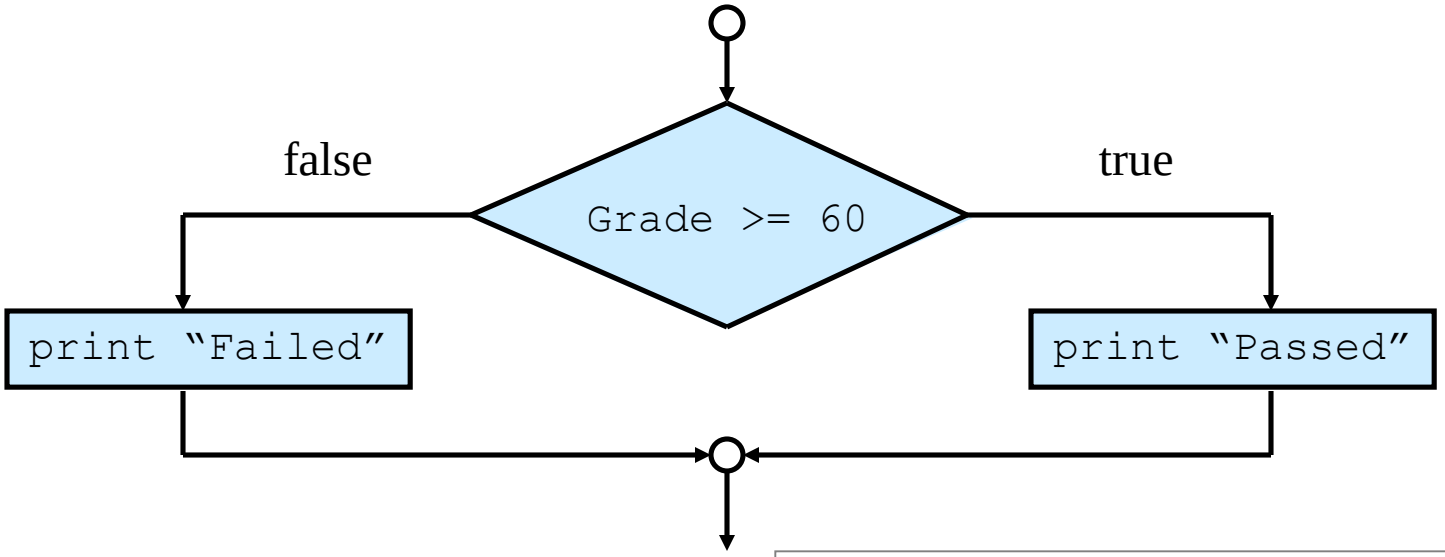
- The **`if/else`** structure
  - Alternate courses can be taken when the statement is false
  - Rather than one action there are two choices
  - Nested structures can test many cases

if/else Statement (cont'd)

```
if ( <test> )
{
   <code executed if <test> is true>  ;
   ......
}
else
{
   <code executed if <test> is false>  ;
   ......
}
```

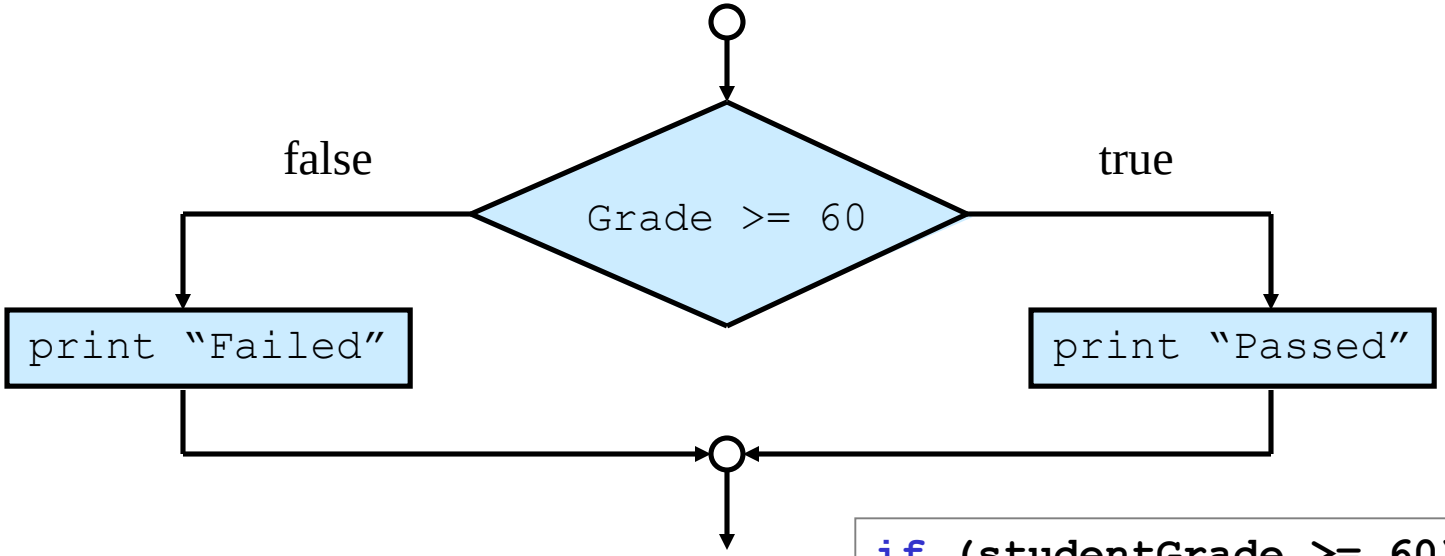
- Can use the block statement inside either branch

if/else Statement (cont'd)




```
if (studentGrade >= 60)
  Console.WriteLine ("Passed");
else
  Console.WriteLine ("Failed");
// beginning of the next statement
```

if/else Statement (cont'd)




```
if (studentGrade >= 60)
{
  Console.WriteLine ("Passed");
}
else
  Console.WriteLine ("Failed");
// beginning of the next statement
```

Nested `if/else` Statements

- The statement executed as a result of an if statement or else clause could be another if statement
- These are called *nested if /else statements*

```
if (studentGrade >= 90)
  Console.WriteLine("A");
else if (studentGrade >= 80)
  Console.WriteLine("B");
else if (studentGrade >= 70)
  Console.WriteLine("C");
else if (studentGrade >= 60)
  Console.WriteLine("D");
else
  Console.WriteLine("F");


// beginning of the next statement
```

Unbalanced `if-else` Statements

```
if (favorite == "apple")
   if (price <= 10 )
      Console.WriteLine("10");
   else
      Console.WriteLine("1");
```

```
if (favorite == "apple")
   if (price <= 10 )
      Console.WriteLine("10");
else
   Console.WriteLine("not my favorite");
```

Ternary Conditional Operator (?:)

- Conditional Operator (e_1?e_2:e_3)
  - C#'s only ternary operator
  - Can be used to construct expressions
  - Similar to an **`if/else`** structure

```
string result;
int numQ;
…….
result = (numQ==1) ? "Quarter" : "Quarters";


// beginning of the next statement
```

The `switch` Statement

- The `switch` *statement* provides another means to decide which statement to execute next
- The switch statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement list associated with the first value that matches

The `switch` Statement: Syntax

The general syntax of a switch statement is:

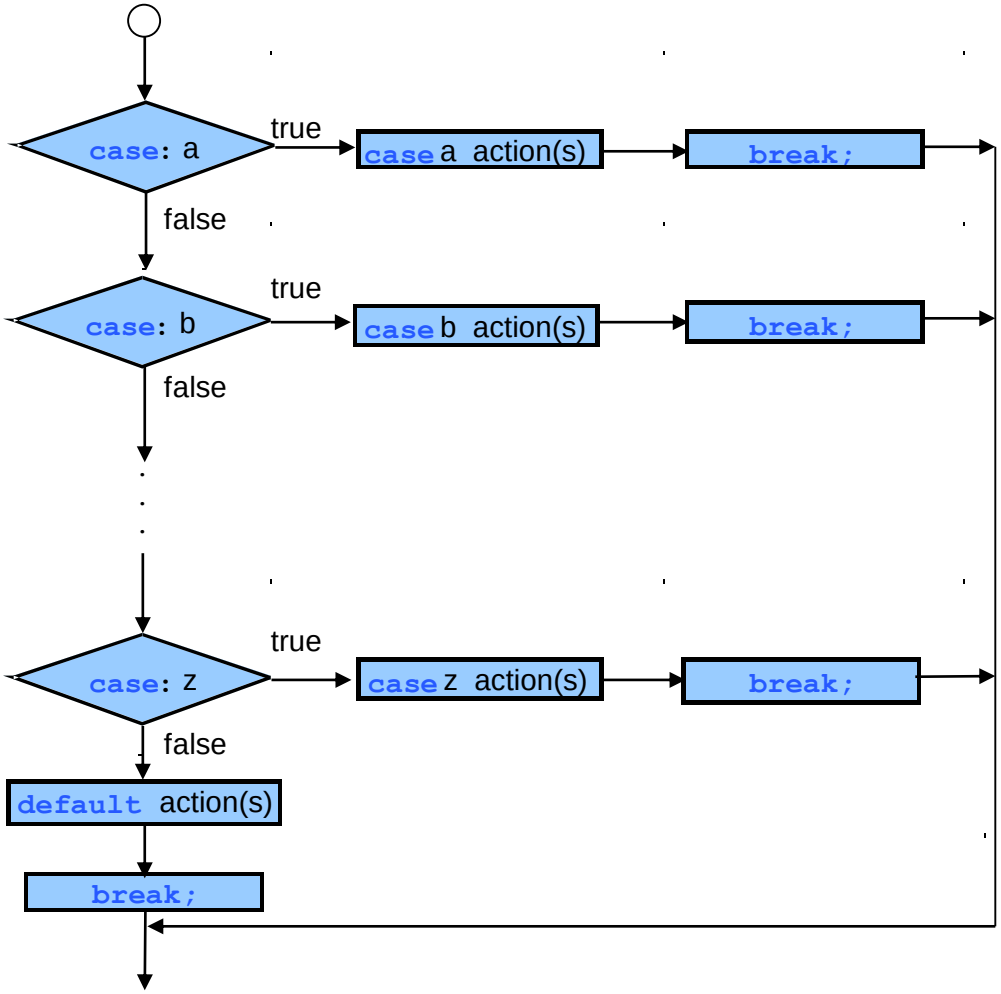
switch and case and default are reserved words

```
switch ( expression )
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case  ...

    default :
        statement-list
}
```

If *expression* matches *value2*, control jumps to here

The `switch` Statement



The `switch` Statement

- The expression of a switch statement must result in an *integral data type*, like an integer or character or a *string*

- Note that the implicit boolean condition in a switch statement is equality – it tries to match the expression with a value

The `switch` Statement

- A switch statement can have an optional *default case* as the last case in the statement
  - The default case has no associated value and simply uses the reserved word `default`
  - If the default case is present, control will transfer to it if no other case value matches
  - If there is no default case, and no other value matches the expression, control falls through to the statement after the switch

- A *break statement* is used as the last statement in each case's statement list
  - A `break` statement causes control to transfer to the end of the switch statement