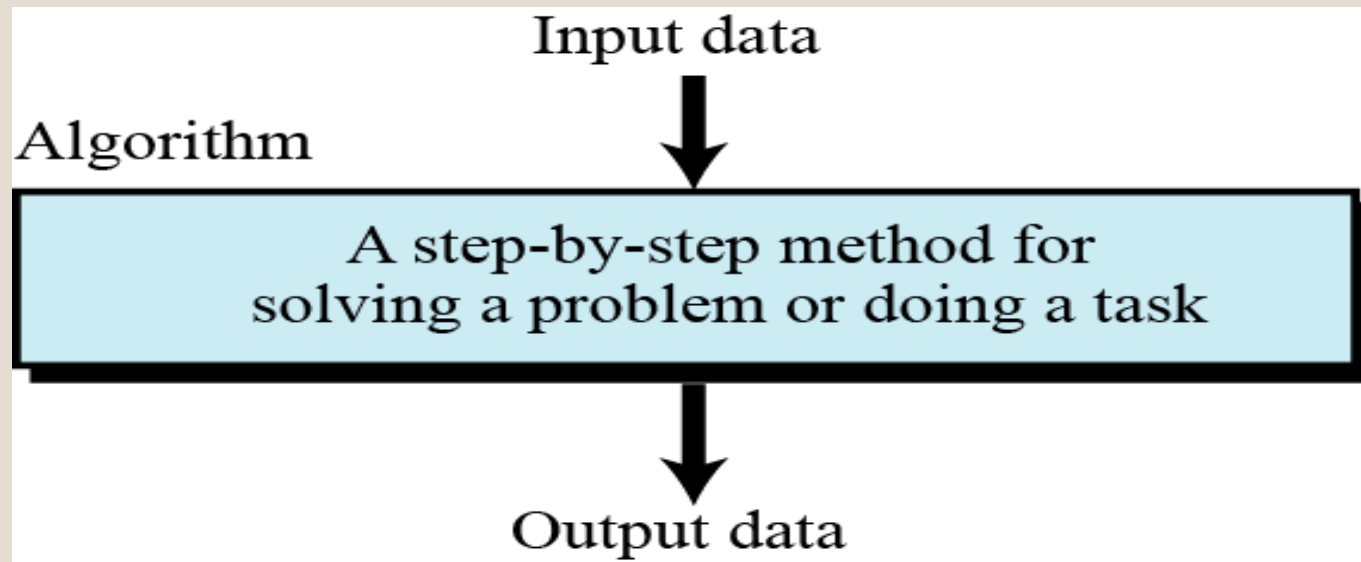


Algorithms, Pseudocode, Activity Diagrams & Control Structures

Algorithms

An informal definition of an algorithm is:

Algorithm: a step-by-step method for solving a problem or doing a task.



Informal definition of an algorithm used in a computer^{8.2}

Algorithms (cont)

An algorithm, deals with:

1. The actions to be executed and
2. The order in which these actions are to be executed

Algorithms (Cont.)

The following example demonstrates the importance of correctly specifying the order in which the actions are to be executed. Consider the following Algorithms for getting out of bed and going to work:

- (1) get out of bed,
- (2) take off pajamas,
- (3) take a shower,
- (4) get dressed,
- (5) eat breakfast and
- (6) Go to work.

This routine prepares an employee for a productive day at the office.

Algorithms (Cont.)

Suppose that the same steps are performed in a slightly different order:

- (1) get out of bed,
- (2) take off pajamas,
- (3) get dressed,
- (4) take a shower,
- (5) eat breakfast,
- (6) Go to work.

Algorithms (Cont.)

In this case, our person will show up for work soaking wet.

This Example Indicates the appropriate sequence in which to execute actions is equally crucial in computer programs.

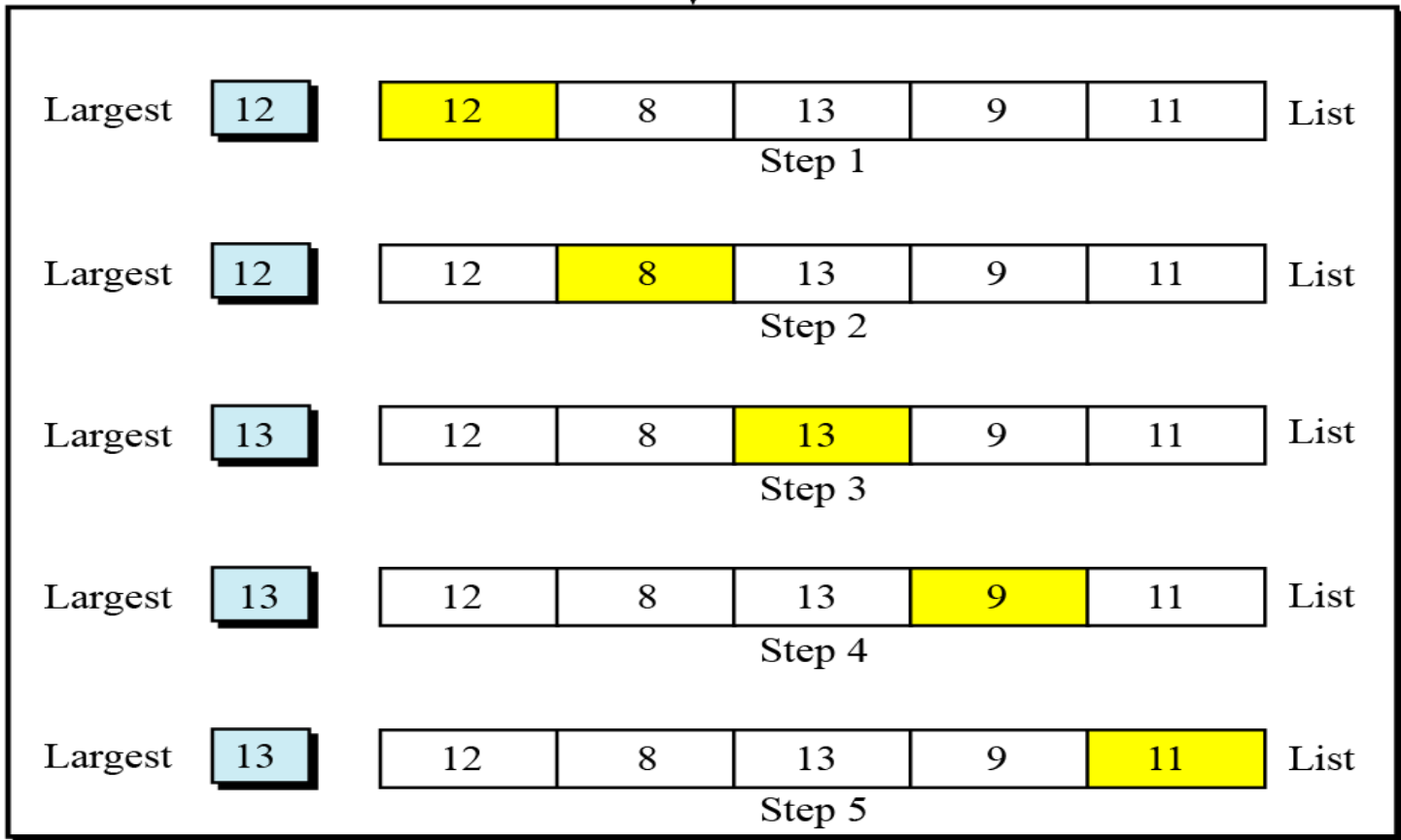
Example 2

We want to develop an algorithm to find the largest integer among a list of any values (for example 5, 1000, 10,000, 1,000,000). The algorithm should be general and not depend on the number of integers.

To solve this problem, we need an intuitive approach. First use a small number of integers (for example, five), then extend the solution to any number of integers.

To solve this problem. We develop the algorithm *FindLargest*. Each algorithm has a name to distinguish it from other algorithms. The algorithm receives a list of five integers as input and gives the largest integer as output.

(12 8 13 9 11) **Input data**



FindLargest



(13) **Output data**

Finding the largest integer among five integers

Pseudocode (cont)

- Pseudocode is an informal language that is used to describe and formulate algorithms.
- Pseudocode statements are not executed on computers.
- Pseudocode is convenient and user friendly
- Pseudocode is NOT an actual computer-programming language.
- Pseudocode Helps you think out an application before attempting to write it in a programming language.

(12 8 13 9 11) **Input data**



Set Largest to the first number.

Step 1

If the second number is greater than Largest, set Largest to the second number.

Step 2

If the third number is greater than Largest, set Largest to the third number.

Step 3

If the fourth number is greater than Largest, set Largest to the fourth number.

Step 4

If the fifth number is greater than Largest, set Largest to the fifth number.

Step 5



(13) **Output data**

FindLargest

7.3 Pseudocode (Cont.)

Example of a pseudocode statement:

- Assign 0 to the counter

The pseudocode statement above can be converted to the following statement:

| | |
|-----------|--------------|
| In VB | counter = 0 |
| In Pascal | counter := 0 |
| In c | counter = 0; |

7.3 Pseudocode (Cont.)

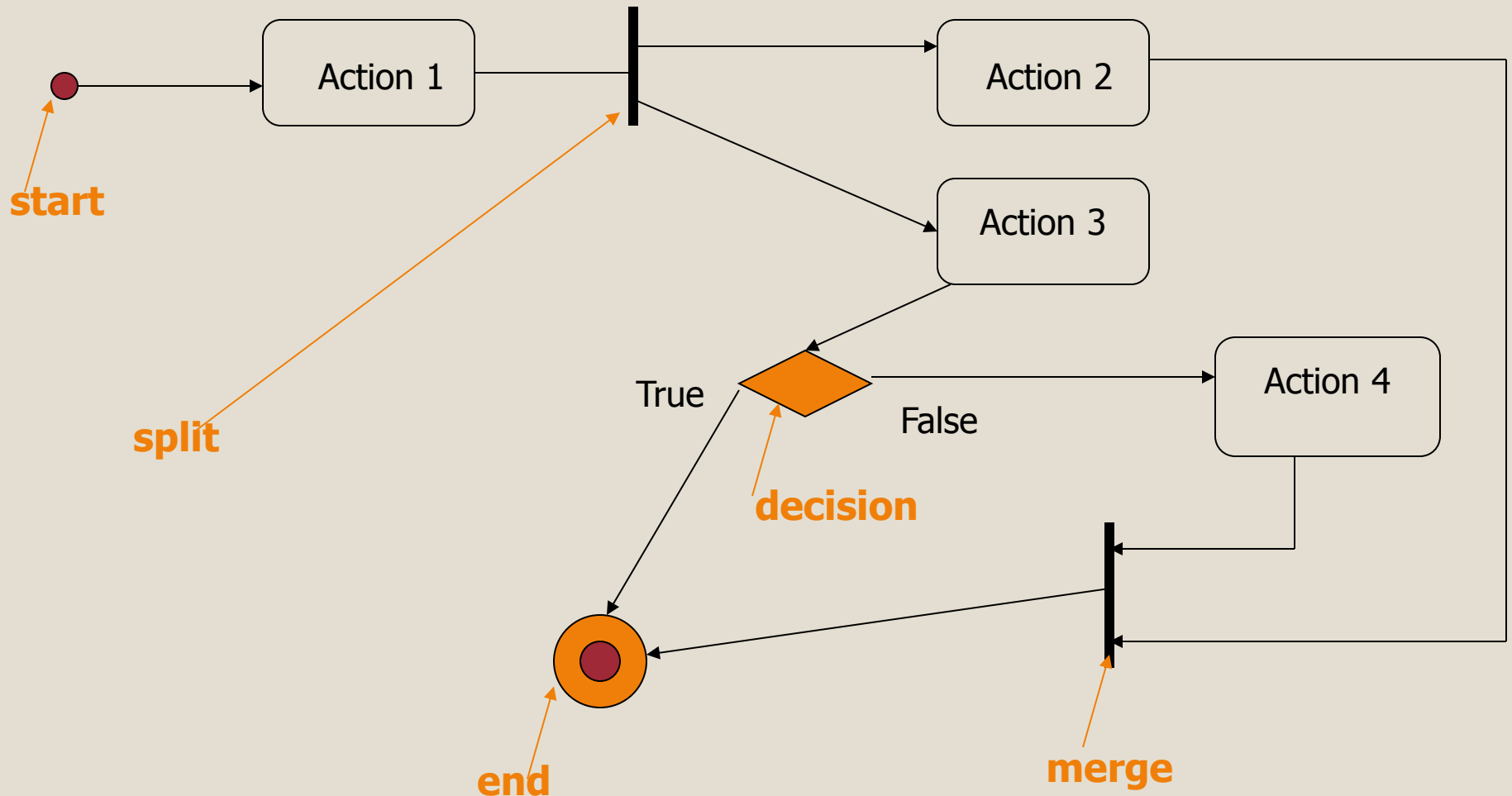
Pseudocode normally describes only executable statements—the actions performed when the corresponding Visual Basic application is run.

An example of a programming statement that is not in pseudocode is declaration.

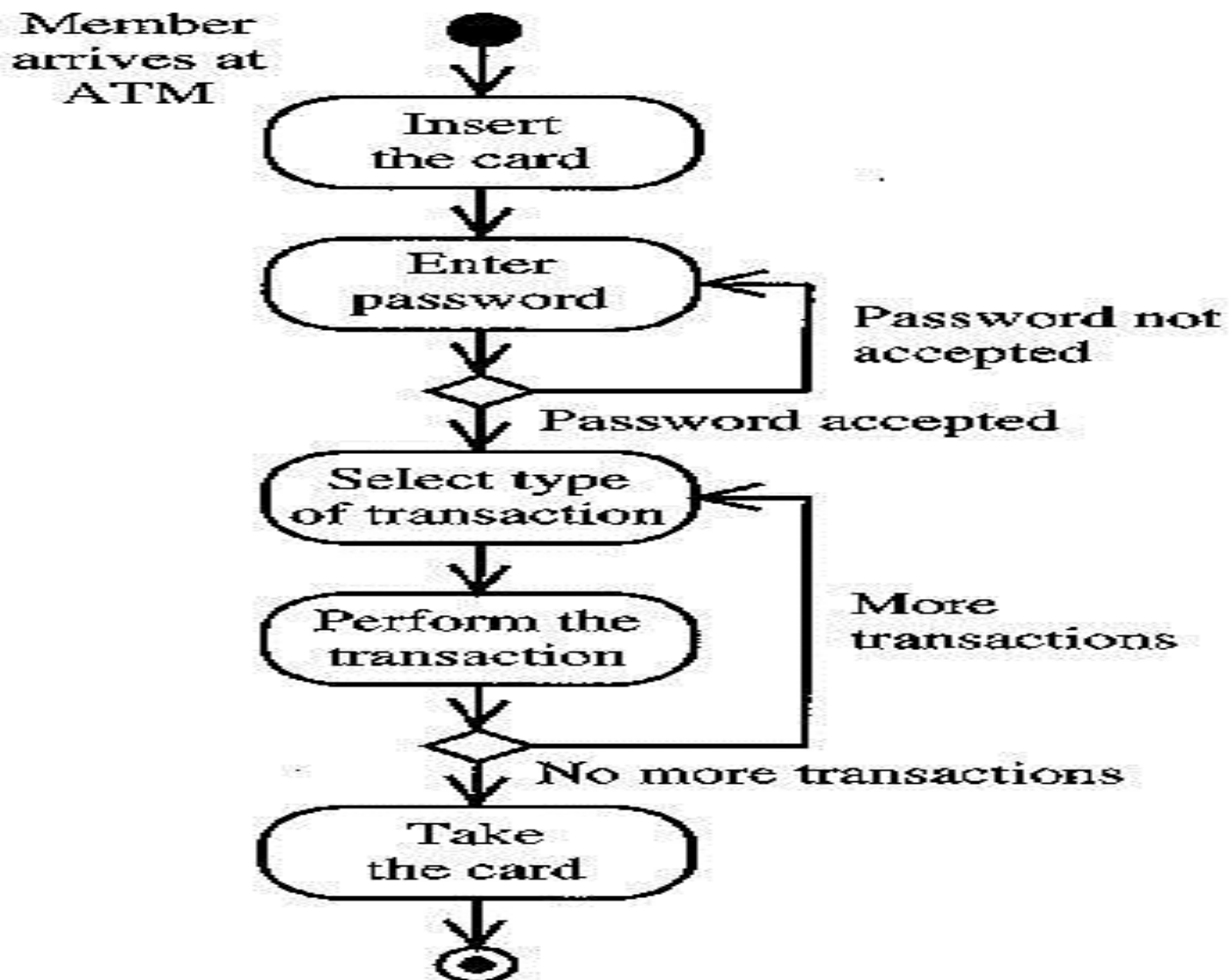
7.3 Activity Diagrams

- **Activity diagrams** are graphical representations of Algorithms
- Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:
 - *Rounded rectangles* represent *actions*;
 - *Diamonds* represent *decisions*;
 - *Bars* represent the start (*split*) or end (*join*) of concurrent activities;
 - A *black circle* represents the start (*initial state*) of the workflow;
 - An *encircled black circle* represents the end (*final state*).

Activity Diagram – Basic Symbols



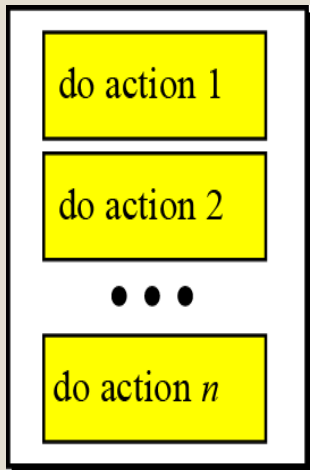
7.3 Activity Diagram(Cont.)



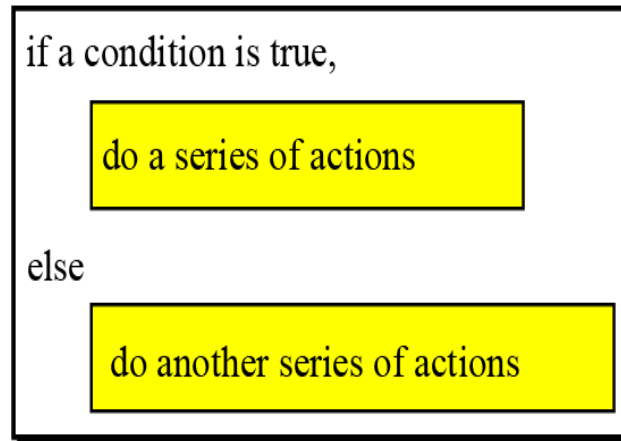
7.4 Control Statements

All programs can be written in terms of only three control statements:

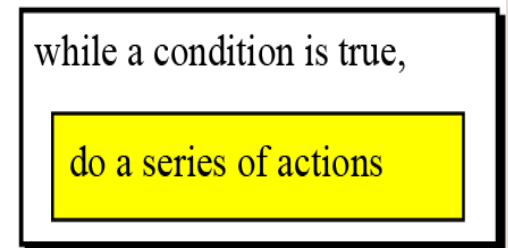
- 1. The sequence statements,**
- 2. The Decision statements and**
- 3. The repetition statements.**



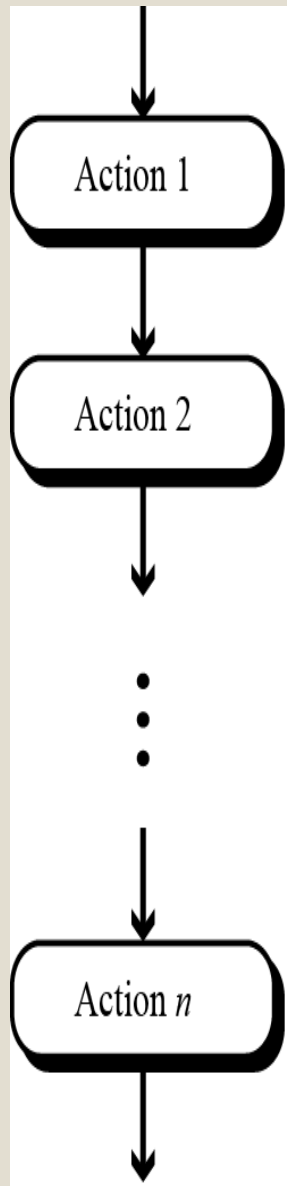
a. Sequence



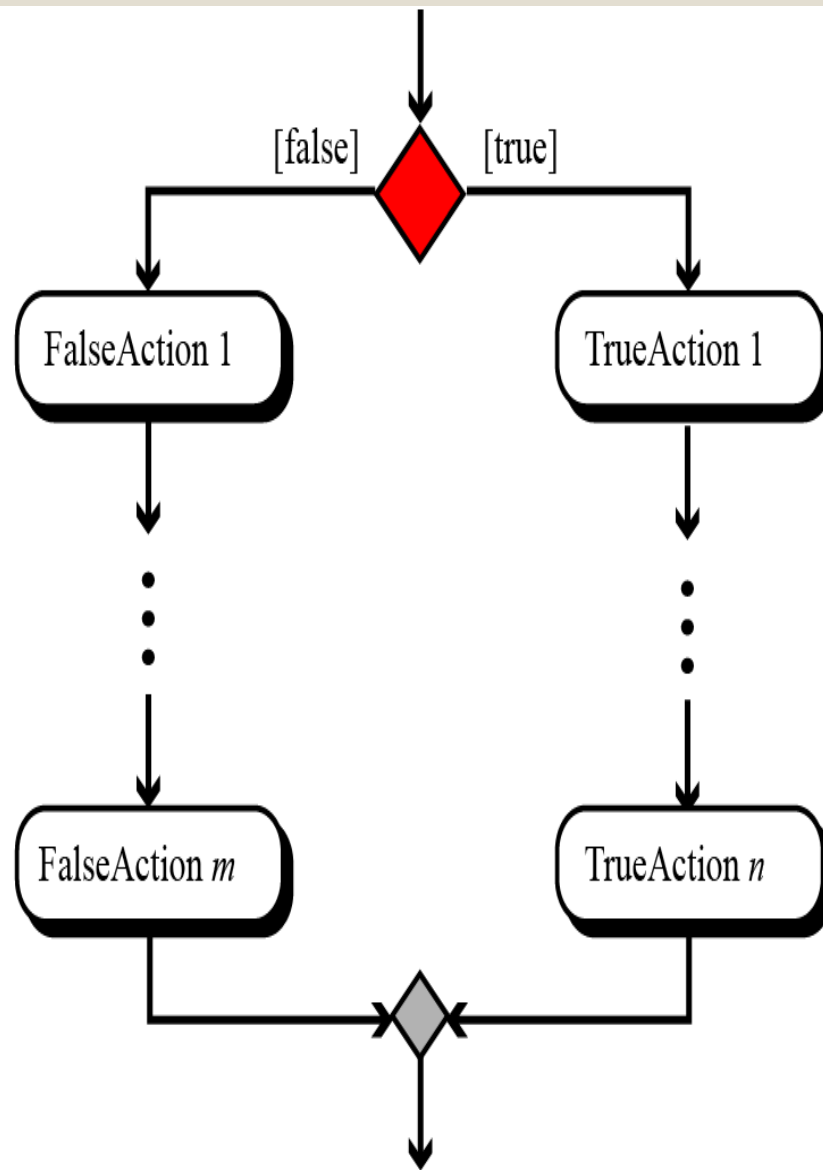
b. Decision



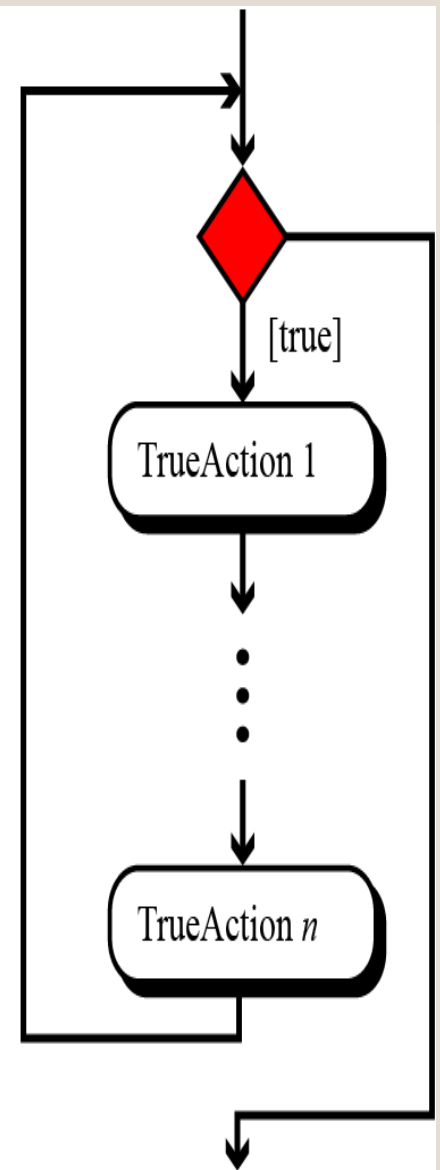
c. Repetition



a. Sequence



b. Decision



c. Repetition

7.4 Control Statements (Cont.)

7.4.1 The sequence statement

The sequence statement executes Visual Basic statements sequentially—that is, one after the other in the order in which they appear in the application.

The activity diagram in Fig. 7.3 illustrates a typical sequence statement, in which two calculations are performed in order.

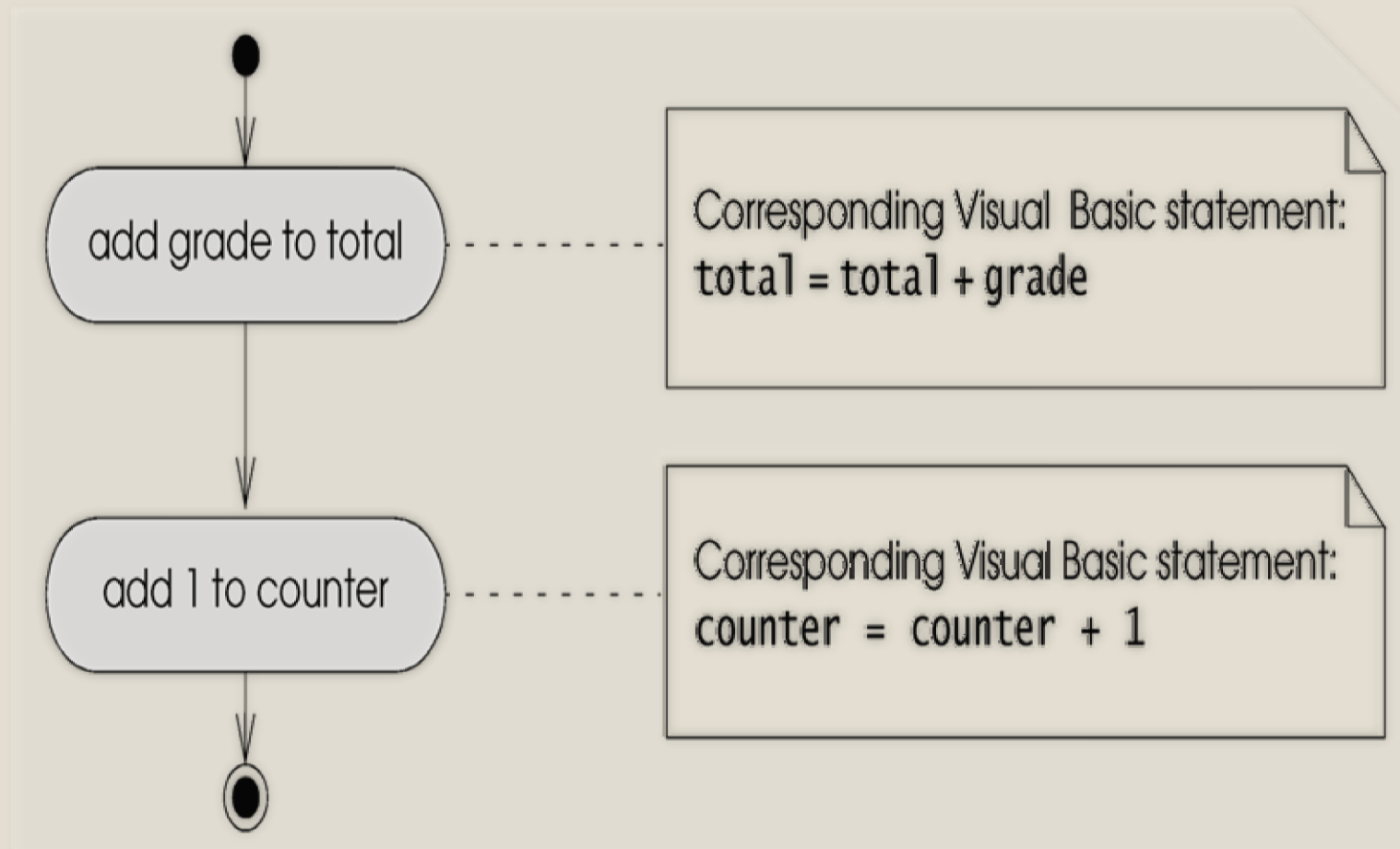


Figure 7.3 Sequence statement activity diagram.

7.4 Control Statements (Cont.)

7.4.2 Decision Statements

A condition is an expression with a true or false value that is used to make a decision. Conditions are evaluated to determine whether their value is true or false.

These values are of data type Boolean Specified in Visual Basic code by using the keywords
True and False.

7.4 Control Statements (Cont.)

Visual Basic provides 3 types of selection statements.

7.4.2.1. The If...Then selection statement performs (selects) an action (or sequence of actions) based on a condition.

If the condition evaluates to True, the actions specified by the If...Then statement are executed.

If the condition evaluates to False, the actions specified by the If...Then statement are skipped.

7.4 Control Statements (Cont.)

7.4.2.2. The If...Then...Else selection statement

Performs an action (or sequence of actions) if a condition is true

Performs a different action (or sequence of actions) if the condition is false.

7.4.2.3. The Nested if /Select Case statement

performs one of many actions (or sequences of actions), depending on the value of an expression

7.4 Control Statements (Cont.)

➤ ***In Summary***

- ***The If...Then*** statement is called a ***single-selection statement*** because it ***selects or ignores a single action (or a sequence of actions)***.
- ***The If...Then...Else*** statement is called a ***double-selection statement*** because it ***selects between two different actions (or sequences of actions)***.
- ***The Nested if / Select Case*** statement is called a ***multiple-selection statement*** because it ***selects among many different actions or sequences of actions***.

7.5.1 If...Then Selection Statement (Cont.)

- Figure 7.5 shows the syntax of the If...Then statement.
- A statement's syntax specifies how the statement must be formed to compile.

Syntax

```
If condition Then  
    [ statements ]  
End If
```

Figure 7.5 If...Then statement syntax.

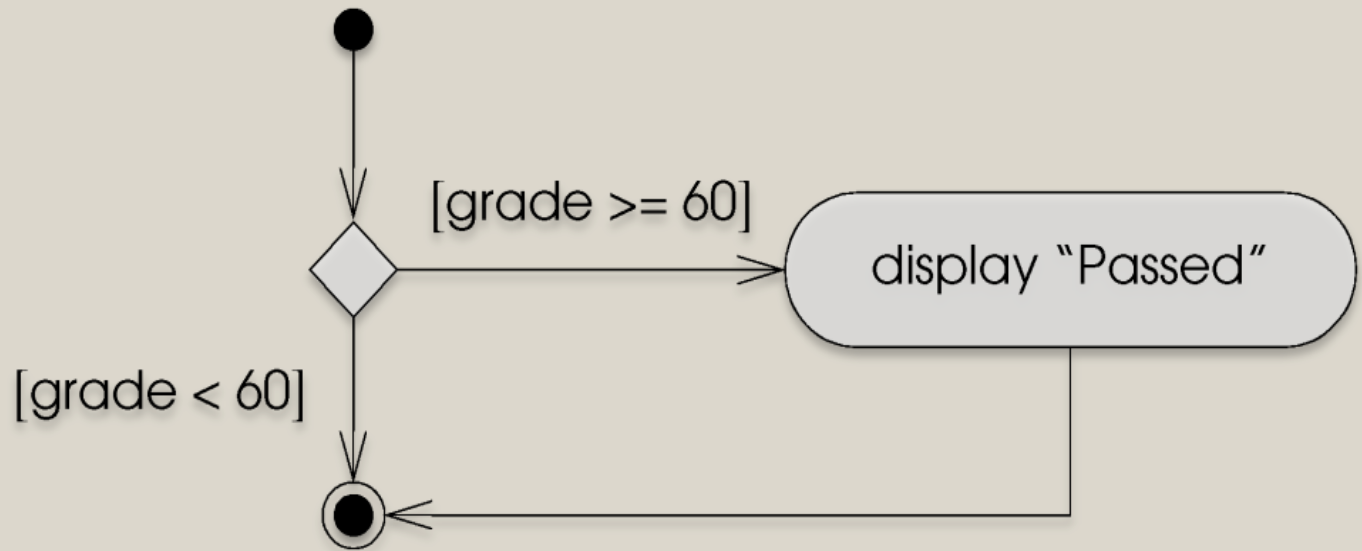


Figure 7.6 If...Then single-selection statement activity diagram

7.5.1 If...Then Selection Statement (Cont.)

- ***Figure 7.6 illustrates the single-selection If . . . Then statement activity diagram .***
- ***This activity diagram contains the diamond, or decision symbol, which indicates that a decision is to be made.***
- ***The two sets of square brackets above or next to the arrows leading from the decision symbol are called guard conditions.***

7.5.1 If...Then Selection Statement Example

- If student's grade is greater than or equal to 50

Display "Passed"

- The preceding pseudocode If statement may be written in Visual Basic as

- - If studentGrade >= 50 Then
 - displayLabel.Text = "Passed"
 - End If

| Algebraic equality or relational operator | Visual Basic equality or relational operator | Example of Visual Basic condition | Meaning of Visual Basic condition |
|---|--|-----------------------------------|-----------------------------------|
| <i>Relational operators</i> | | | |
| > | > | $x > y$ | x is greater than y |
| < | < | $x < y$ | x is less than y |
| \geq | >= | $x \geq y$ | x is greater than or equal to y |
| \leq | <= | $x \leq y$ | x is less than or equal to y |
| <i>Equality operators</i> | | | |
| = | = | $x = y$ | x is equal to y |
| \neq | <> | $x \neq y$ | x is not equal to y |

Figure 7.4 Equality and relational operators.

Table 4-2 VB .NET logical operators

| Operator | Description |
|----------------|--|
| Not | Negates an expression |
| And | Logically joins two expressions; if both expressions evaluate to true , then And returns true ; otherwise it returns false |
| Or | Logically joins two expressions; if both expressions evaluate to false , then Or returns false ; otherwise it returns true |
| Xor | Logically joins two expressions; if both expressions evaluate to true or both expressions evaluate to false , Xor returns false ; otherwise it returns true |
| AndAlso | Same as And except it employs short-circuit logic |
| OrElse | Same as Or except it employs short-circuit logic |



Common Programming Error

Omitting the **Then** keyword in an **If...Then** statement is a syntax error. The IDE helps prevent this error by inserting the **Then** keyword after you write the condition.



Common Programming Error

Adding spaces between the symbols in the operators `<>`, `>=` and `<=` (as in `< >`, `> =`, `< =`) is a syntax error that Visual Basic fixes automatically.



Common Programming Error

Reversal of the operators `<>`, `>=` and `<=` (as in `><`, `=>`, `=<`) is a syntax error that Visual Basic fixes automatically.



Good Programming Practice

The IDE indents the statements inside `If...Then` statements to improve readability.

7..5.2 *If...Then...Else* Selection Statement

Syntax

```
If condition Then  
    [ statements ]  
Else  
    [ statements ]  
End If
```

Figure 7.7 If...Then...Else statement syntax.

7.5.2 If...Then...Else Selection Statement

- **Figure 7.8 illustrates the flow of control in the *If...Then...Else* double-selection statement.**

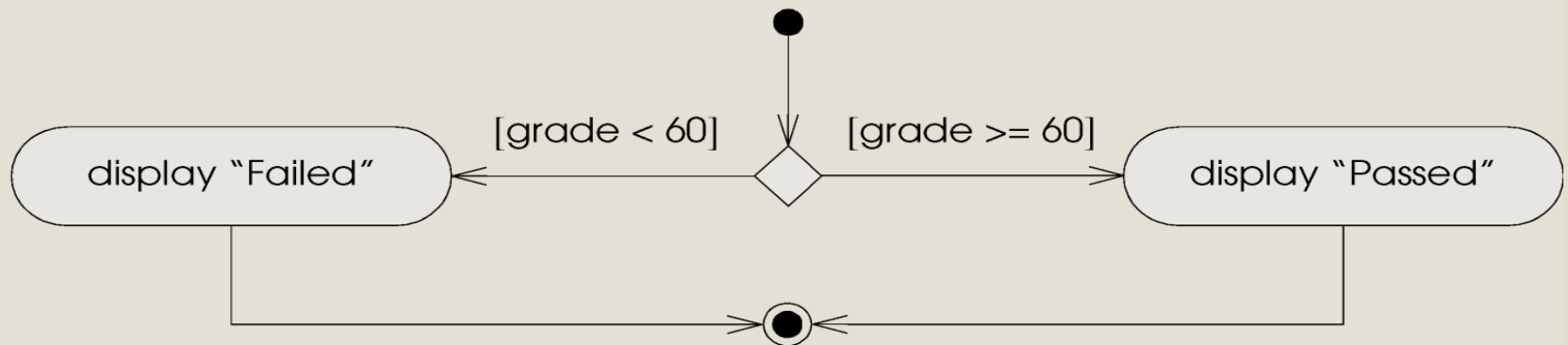


Figure 7.8 If...Then...Else double-selection statement activity diagram.

7.5.2 If...Then...Else Selection Statement (cont)

If student's grade is greater than or equal to 60

Display "Passed"

Else

Display "Failed"

The preceding pseudocode may be written in Visual Basic as

```
If studentGrade >= 60 Then
    displayLabel.Text = "Passed"
Else
    displayLabel.Text = "Failed"
End If
```

7.5.3 Nested If...Then...Else statements

Nested If...Then...Else statements test for multiple conditions by placing If...Then...Else statements inside other If...Then...Else statements.

```
If student's grade is greater than or equal to 90
    Display "A"
Else
    If student's grade is greater than or equal to 80
        Display "B"
    Else
        If student's grade is greater than or equal to 70
            Display "C"
        Else
            If student's grade is greater than or equal to 60
                Display "D"
            Else
                Display "F"
```

```
1  If studentGrade >= 90 Then
2      displayLabel.Text = "A"
3  Else
4      If studentGrade >= 80 Then
5          displayLabel.Text = "B"
6      Else
7          If studentGrade >= 70 Then
8              displayLabel.Text = "C"
9          Else
10             If studentGrade >= 60 Then
11                 displayLabel.Text = "D"
12             Else
13                 displayLabel.Text = "F"
14             End If
15         End If
16     End If
17 End If
```

Figure 7.9 Visual Basic code converted from pseudocode.

7.5.3 Nested If...Then...Else Selection Statement (Cont.)

- *Most Visual Basic programmers prefer to use the ***ElseIf*** keyword to write the preceding *If...Then...Else* statement, as shown in Fig. 7.10.*

```
1  If studentGrade >= 90 Then
2      displayLabel1.Text = "A"
3  ElseIf studentGrade >= 80 Then
4      displayLabel1.Text = "B"
5  ElseIf studentGrade >= 70 Then
6      displayLabel1.Text = "C"
7  ElseIf studentGrade >= 60 Then
8      displayLabel1.Text = "D"
9  Else
10     displayLabel1.Text = "F"
11 End If
```

Figure 7.10 If...Then...Else statement using the ElseIf keyword

7.5.3 Nested *If...Then...Else* Selection Statement (Cont.)

- **The two statements are equivalent, but you should use the latter statement—it avoids deep indentation of the code.**
- **The final portion of the *If...Then...Else* statement uses the *Else* keyword to handle all the remaining possibilities.**
- **The *Else* clause must always be last in an *If...Then...Else* statement—following an *Else* clause with another *Else* or *ElseIf* clause is a syntax error (Also it has no condition).**
- **Note that the latter statement requires only one *End If*.**

7.5.4 Conditional *If* expression

- The preceding *If . . . Then . . . Else* statement can also be written as

*displayLabel.Text = If(studentGrade >= 60,
"Passed", "Failed")*

- A conditional *If* expression starts with the keyword *If* and is followed by three expressions in parentheses—a condition, the value if the condition is true and the value if the condition is false.

7.4 Control Statements (Cont.)

7.4.3 Repetition Structures

Visual Basic provides seven types of repetition statements for performing a statement or group of statements repeatedly:

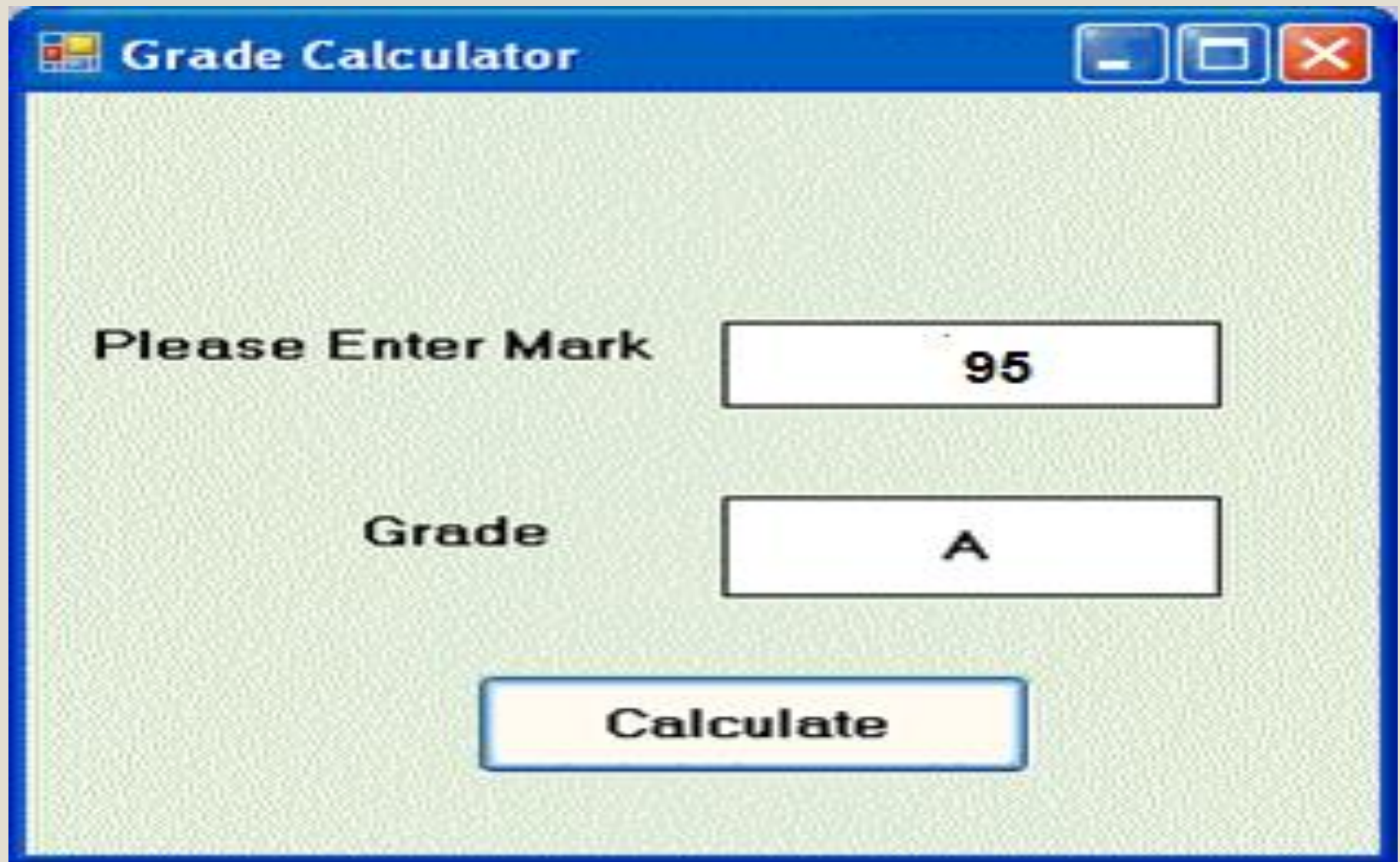
1. While...End While
2. *Do While...Loop*
3. Do Until...Loop
4. Do...Loop While
5. Do...Loop Until
6. For...Next
7. For Each...Next

7.4 Control Structures (Cont.)

From the previous discussions, we conclude that Visual Basic has **11 control statements**

- ➔ **1** sequence structure,
 - ➔ **3** types of selection statements and
 - ➔ **7** types of repetition statements.
-
- All Visual Basic apps are formed by combining as many of each type of control statement as is necessary.

Homework



A screenshot of a Windows-style application window titled "Grade Calculator". The window has a blue title bar with standard minimize, maximize, and close buttons. The main area has a light green background. It contains two input fields: one for "Please Enter Mark" with the value "95", and one for "Grade" with the value "A". Below these is a yellow "Calculate" button.

| Input Label | Input Value |
|-------------------|-------------|
| Please Enter Mark | 95 |
| Grade | A |

Calculate