Chapter 4
Selection Structures: Making Decisions

4.1 An Introduction to Selection Structures

Single-alternative (`If-Then`)

- A single block of statements to be executed or skipped

Dual-alternative (`If-Then-Else`)

- Two blocks of statements, one of which is to be executed, while the other one is to be skipped

Multiple-alternative (`If-Then-Else-If` or `Case/Switch`)

- More than two blocks of statements, only one of which is to be executed and the rest skipped

Single Alternative

```
If something is true Then
     Do something (any
     number of statements)
End If
```

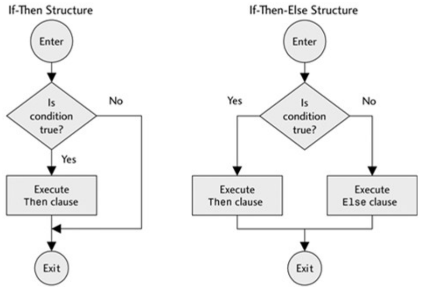
```
Write "How old are you?"
Input Age
If Age >= 18
     Set Eligibility = "Yes"
     Write "You can vote."
     Do other things…
End If
```

Dual Alternative

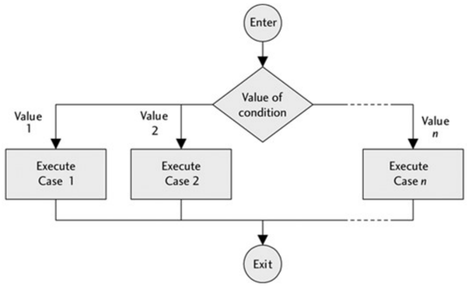
```
If something is true Then
     Do something
Else
     Do something else
End If
```

```
Write "How old are you?"
Input Age
If Age >= 18
     Set Eligibility = "Yes"
     Write "You can vote."
Else
     Set Eligibility = "No"
     Write "You're too young."
End If
```

Flowcharts for Selection Structures: Single Alternative and Dual Alternatives



Flowcharts for Selection Structures: Multiple Alternative



Guidelines

- An **`Else`** condition does not have to exist. Sometimes we only want to do something if something is true and do nothing if it is not true.
- Do not manufacture alternative conditions.
- In an **`If`** statement, the body of the **`If`** is executed if, and only if, the test condition is true. Otherwise, no action is taken.
- Be sure to indent for readability.

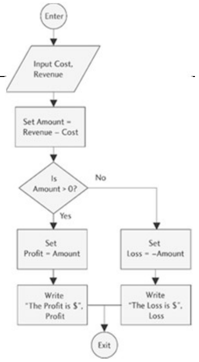
Example: two ways to write a test condition

```
Write "Do you have any children? ↵
        Type Y for yes, N for no"
Input Response
If Response is "Y" Then
     Write "How many?"
     Input NumberChildren
End If
Write "Questionnaire is complete.↵
        Thank you."
```

```
Write "Do you have any children? ↵
        Type Y for yes, N for no"
Input Response
If Response is not "N" Then
     Write "How many?"
     Input NumberChildren
End If
Write "Questionnaire is complete.↵
        Thank you."
```

Example: Multiple Alternatives

```
Write "Enter total cost:"
Input Cost
Write "Enter total revenue:"
Input Revenue
Set Amount = Revenue - Cost
If Amount > 0 Then
      Set Profit = Amount
      Write "The profit is $" + Profit
Else
      Set Loss = -Amount
      Write "The loss is $" + Loss
End If
```


Enter
Input Cost, Revenue
Set Amount = Revenue – Cost
Is Amount > 0?
No
Yes
Set Profit = Amount
Set Loss = –Amount
Write "The Profit is $", Profit
Write "The Loss is $", Loss
Exit

4.2 Relational and Logical Operators

Decision making involves testing a condition

To help construct these conditions use

- relational operators
- logical operators

Relational Operators

Relational operators are the symbols used in the condition to be evaluated in `If` statements:

`==`	is the same as (the comparison operator)
`!=`	is not the same as (not equal to)
`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

Example

Assume the variables **A** and **B** have the values:

```
A = 9, B = 6
```

Then the `If` statement:

```
If A > B Then
     Write A + "is greater than" + B
End If
```

can be read:

"If it is true that the value of the variable **A** is greater than the value of the variable **B**, then write the value of **B** to the screen."

The display will be:

```
9 is greater than 6
```

More examples:

Given: `A = 23, B = 16`

Then:

`A > B` is `true`

`A < B` is `false`

`A >= B` is `true`

`A <= B` is `false`

`A != B` is `true`

`A == B` is `false`

Comparison *vs.* Assignment Operators

There is a significant difference between the use of an equals sign (**=**) as the assignment operator and a double equals sign (**==**) as the comparison operator.

As an **assignment operator**, the equals sign sets the value of an expression on the right side to the variable on the left side.

As a **comparison operator**, the double equals sign asks the question, "Is the value of the variable on the left side the same as the value of the expression, number, or variable on the right side?"

a single equals sign (**=**) signifies the **assignment operator**

a double equals sign (**==**) signifies the **comparison operator**

The Assignment Operator

Given: `A = 14, B = 27`

In programming code, the assignment statement:

`A = B`

sets the value of `B` to the variable `A`.

In other words, after this statement is executed, both `A = 27` and `B = 27`.

In this case, the equals sign is used as an **assignment operator**.

The Comparison Operator

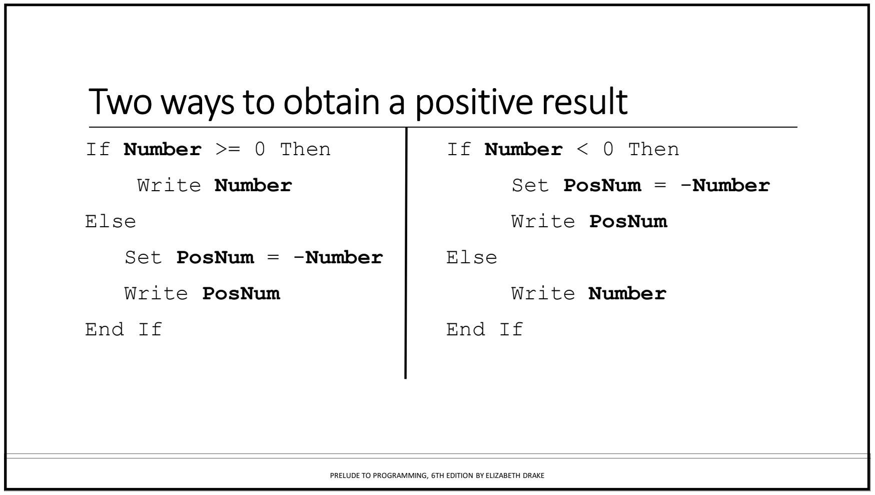
Given: `A = 14, B = 27`

Using **relational operators**, the statement:

`A == B`

is a **comparison**.

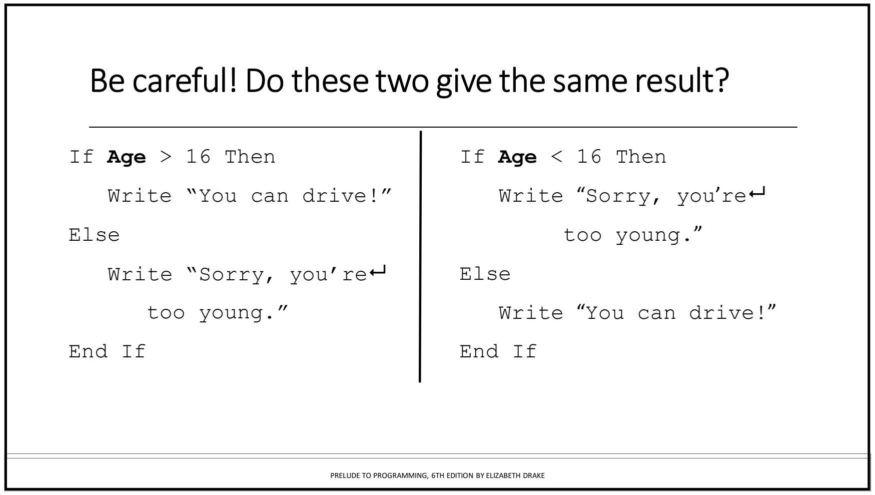
This statement asks the question, "Is the value of `A` the same as the value of `B`?" In this case, since `A` and `B` have different values, the answer is "no" and the statement would result in a value of `False.`



Two ways to obtain a positive result

```
If Number >= 0 Then
    Write Number
Else
   Set PosNum = -Number
   Write PosNum
End If
```

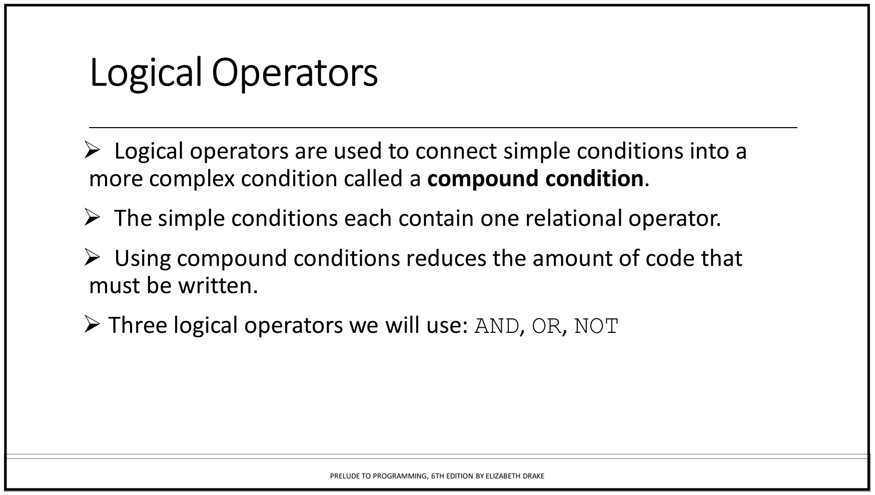
```
If Number < 0 Then
     Set PosNum = -Number
     Write PosNum
Else
     Write Number
End If
```



Be careful! Do these two give the same result?

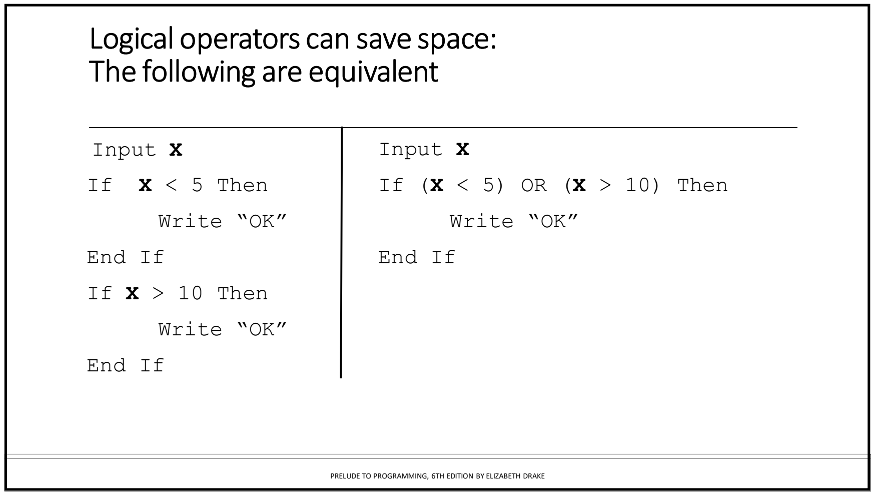
```
If Age > 16 Then
   Write "You can drive!"
Else
   Write "Sorry, you're↵
      too young."
End If
```

```
If Age < 16 Then
   Write "Sorry, you're↵
        too young."
Else
   Write "You can drive!"
End If
```



Logical Operators

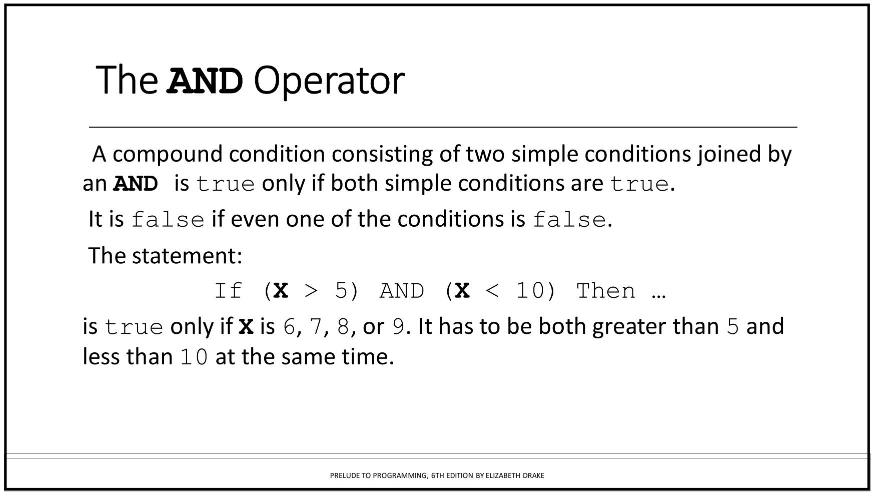
- Logical operators are used to connect simple conditions into a more complex condition called a **compound condition**.
- The simple conditions each contain one relational operator.
- Using compound conditions reduces the amount of code that must be written.
- Three logical operators we will use: AND, OR, NOT



Logical operators can save space: The following are equivalent

```
Input X
If  X < 5 Then
     Write "OK"
End If
If X > 10 Then
     Write "OK"
End If
```

```
Input X
If (X < 5) OR (X > 10) Then
     Write "OK"
End If
```



The **AND** Operator

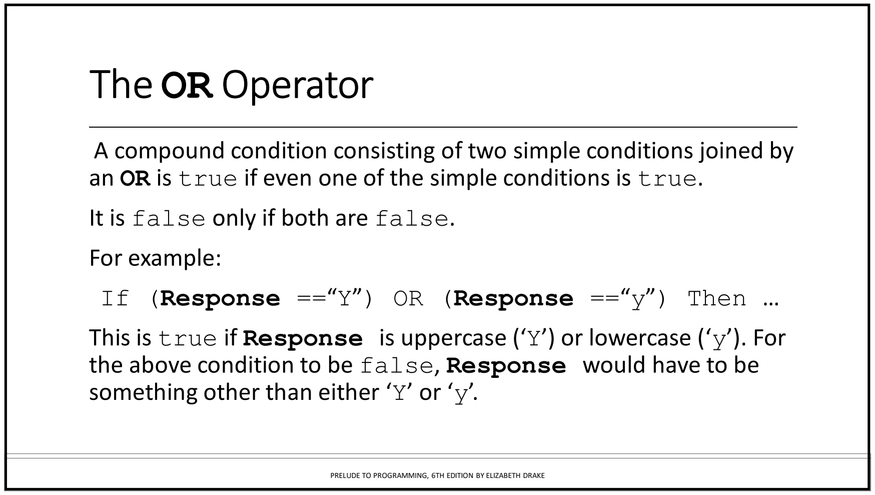
A compound condition consisting of two simple conditions joined by an **AND** is `true` only if both simple conditions are `true`.

It is `false` if even one of the conditions is `false`.

The statement:

```
If (X > 5) AND (X < 10) Then …
```

is `true` only if **X** is `6`, `7`, `8`, or `9`. It has to be both greater than `5` and less than `10` at the same time.



The **OR** Operator

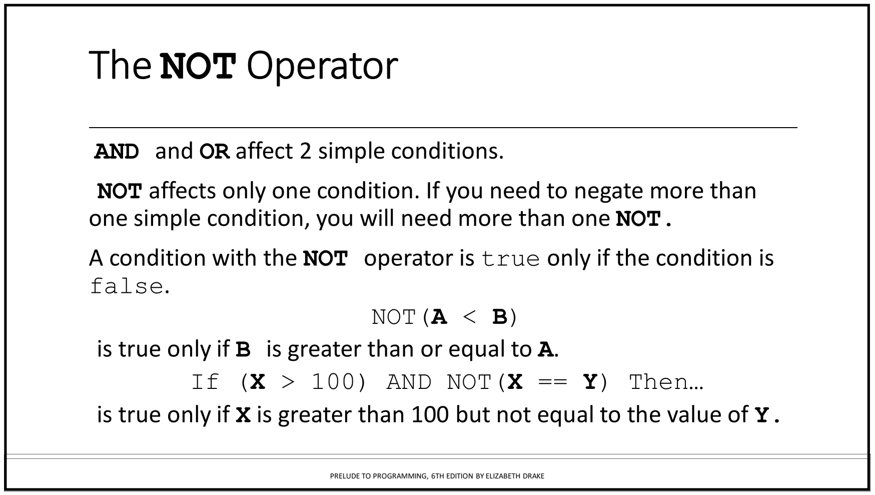
A compound condition consisting of two simple conditions joined by an **OR** is `true` if even one of the simple conditions is `true`.

It is `false` only if both are `false`.

For example:

```
If (Response ==“Y”) OR (Response ==“y”) Then …
```

This is `true` if **Response** is uppercase (‘`Y`’) or lowercase (‘`y`’). For the above condition to be `false`, **Response** would have to be something other than either ‘`Y`’ or ‘`y`’.



The **NOT** Operator

AND and **OR** affect 2 simple conditions.

NOT affects only one condition. If you need to negate more than one simple condition, you will need more than one **NOT**.

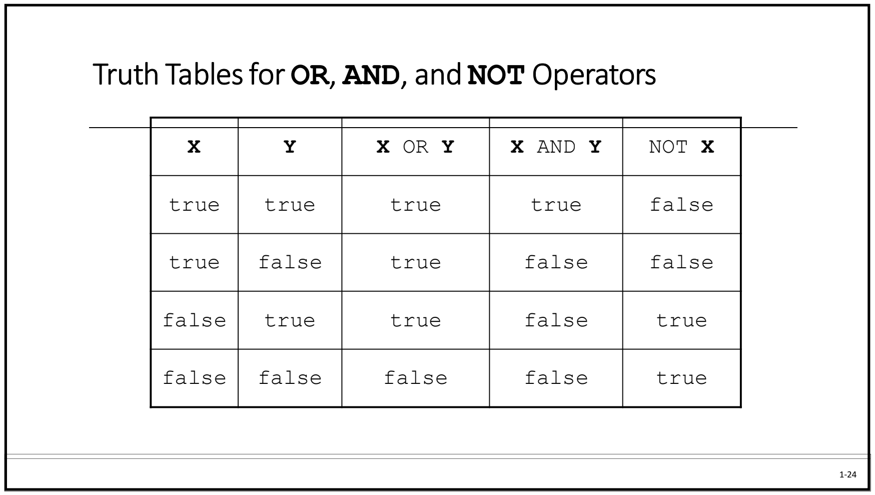
A condition with the **NOT** operator is `true` only if the condition is `false`.

```
NOT(A < B)
```

is true only if **B** is greater than or equal to **A**.

```
If (X > 100) AND NOT(X == Y) Then…
```

is true only if **X** is greater than 100 but not equal to the value of **Y**.



Truth Tables for **OR**, **AND**, and **NOT** Operators

X	**Y**	**X** OR **Y**	**X** AND **Y**	NOT **X**
true	true	true	true	false
true	false	true	false	false
false	true	true	false	true
false	false	false	false	true

Hints

In a compound condition, it is necessary to use complete simple conditions.

This is correct:

```
If (X < 5) OR (X > 10) Then …
```

This is not correct:

```
If (X < 5 OR > 10) Then …
```

Example: First way using **AND**

Workers who earn less than $10 per hour are paid 1.5 times their normal rate for overtime hours.
Workers who earn $10 or more per hour are paid their regular hourly rate regardless of number hours worked.
Working more than 40 hours per week is considered overtime.

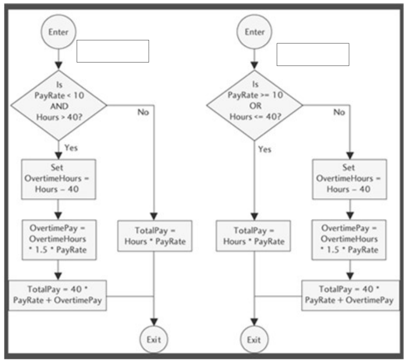
```
If (PayRate < 10) AND (Hours > 40) Then
     Set OvertimeHours = Hours – 40
     Set OvertimePay = OvertimeHours * 1.5 * PayRate
     Set TotalPay = 40 * PayRate + OvertimePay
Else
     Set TotalPay = Hours * PayRate
End If
```

Example: Second way using **OR**

Workers who earn less than $10 per hour are paid 1.5 times their normal rate for overtime hours.
Workers who earn $10 or more per hour are paid their regular hourly rate regardless of number hours worked.
Working more than 40 hours per week is considered overtime.

```
If (PayRate >= 10) OR (Hours <= 40) Then
      Set TotalPay = Hours * PayRate
Else
     Set OvertimeHours = Hours – 40
     Set OvertimePay = OvertimeHours * 1.5 * PayRate
     Set TotalPay = 40 * PayRate + OvertimePay
End If
```

Flowcharts for the two ways:


Enter
Is PayRate < 10 AND Hours > 40?
No
Yes
Set OvertimeHours = Hours – 40
OvertimePay = OvertimeHours * 1.5 * PayRate
TotalPay = Hours * PayRate
TotalPay = 40 * PayRate + OvertimePay
Exit
Enter
Is PayRate >= 10 OR Hours <= 40?
No
Yes
Set OvertimeHours = Hours – 40
TotalPay = Hours * PayRate
OvertimePay = OvertimeHours * 1.5 * PayRate
TotalPay = 40 * PayRate + OvertimePay
Exit

Hierarchy of Operations (Order of Precedence)

Description	Symbol
Arithmetic Operators are evaluated first in the order listed	
First: Parentheses	( )
Second: Exponents	^
Third: Multiplication / Division / Modulus	*, /, %
Fourth: Addition / Subtraction	+ –
Relational Operators are evaluated second and all relational operators have the same precedence	
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
The same as, equal to	==
Not the same as	!=
Logical Operators are evaluated last in the order listed	
First: NOT	! or NOT or not
Second: AND	&& or AND or and
Third: OR	|| or OR or or

Combining Logical and Relational Operators

Example:

Let **Q** = 3 and let **R** = 5

Is the following expression `true` or `false`?

```
NOT Q > 3 OR R < 3 AND Q – R <= 0
```

Step 1: `(NOT(false)) OR (false AND true)`
Step 2: `true OR (false AND true)`
Step 3: `true OR false`
Step 4: `true`

The Boolean Type

Most programming languages allow variables to be of logical (or `Boolean`) type.

A `Boolean` variable may only be either `true` or `false`.

Example: Can declare a variable, **`Answer`**, to be of `Boolean` type and use it in a statement anywhere that a value of `true` or `false` is valid, such as the following C++ snippet:

```
bool Answer;
Answer = true;
if(Answer) cout << "Congratulations!";
```

This means: If the value of **`Answer`** is `true`, then write `"Congratulations!"` to the screen.

The following C++ statement is equivalent to the `if` statement shown above and may be clearer:

```
if (Answer == "true") cout << "Congratulations!";
```

4.3 ASCII Code and Comparing Strings

- A character can be defined as any symbol that can be typed on the keyboard.
- These symbols include special characters like asterisks (*), ampersands (&), @ signs, as well as letters, digits, punctuation marks, and blank spaces.
- There is a more precise definition of a character using how characters are represented in a computer's memory.
- Relational operators <, <=, >, !=,==, and >= can be applied to any string of characters.

ASCII Code

- A programming language uses a scheme to associate each character with a number.
- The standard is the **American Standard Code for Information Interchange (ASCII code).**
- All data, including characters, are stored in the computer’s memory in binary form.
- It is pronounced “askey.”
- Each character is associated with a number from `0` to `127`.

Examples:

Uppercase `A` is `65`

Uppercase `Z` is `90`

Digits have codes from `48` (“`0`”) to `57 (“9”)`

Ordering Arbitrary Strings

- Letters are in alphabetical order.
- All uppercase letters precede all lowercase letters.
- Digits (viewed as characters) retain their natural order.

For example, **`“1” < “2”, “2” < “3”`**

- The blank precedes all digits and letters.

Examples: All of the following conditions are true:

```
“a” > “B”
“1” <= “5”
“2” >= “2”
```

Rules for Ordering Strings

Two strings, **`S1`** and **`S2`**, are equal (**`S1`** == **`S2`**) if they have exactly the same characters in exactly the same order; they are not equal (**`S1`** != **`S2`**) otherwise. To see which of two unequal strings comes first, use the following procedure:

1. Scan strings from left to right, stopping at the first position for which the corresponding characters differ or when one of the strings ends.

2. If two corresponding characters differ before either string ends, the ordering of these characters determines the ordering of the given strings.

3. If one string ends before any pair of corresponding characters differ, then the shorter string precedes the longer one.

When applying this procedure, the following is true:

- If string **`S1`** precedes string **`S2`**, then **`S1`** < **`S2`**.
- If string **`S1`** follows string **`S2`**, then **`S1`** > **`S2`**.

Beware of Strings of Digits

- A character string may be numbers like `“123”.` But `“123”` is *not* the same as `123`.
- The numeric constant is stored in memory by storing the binary equivalent of `123`, but the string constant is stored in memory by successively storing the ASCII codes for `“1”`, `“2”`, and `“3”`.

There is no mechanism to compare numbers with strings so, if the variable (**`Num`**) is a numeric variable, statements like the following:

```
Num == “123”
```

make no sense, and will lead to an error message if used in a program.

Also, using the procedure given for ordering strings, we see that:

`”123” < “25”` and `“24.0” != “24”`

are both `true` statements!

4.4 Selecting from Several Alternatives

To handle more than two options in a program we use multiple **`If-Then`** statements or multiple **`If-Then-Else`** statements.

```
If (something is true) Then
      Do something
Else
      If (something else is true) Then
            Do something else
      Else
            Do a different something else
      End If
End If
```

Example

```
If Age >= 18 Then
   Set Eligibility = "Yes"
Else
   If Age > 15 Then
        Set Eligibility = "Maybe"
   Else
        Set Eligibility = "No"
   End If
End If
```

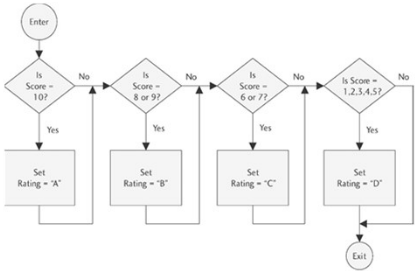
Hints

- ➢ The number of **`End If's`** must equal the number of **`If's`**.
- ➢ You can draw a line to connect them to check.
- ➢ In the previous example, the check for **`Age`** `> 15` will never be done if the **`Age`** is `>= 18`.
- ➢ Regardless of how many possible conditions are included, only one will ever be executed.

Assigning Ratings the Long Way

```
Declare Score As Integer
Declare Rating As Character
Write "Enter score: "
Input Score
If Score == 10 Then
   Set Rating = "A"
End If
If (Score == 8) OR (Score == 9) Then
   Set Rating = "B"
End If
If (Score == 6) OR (Score == 7) Then
   Set Rating = "C"
End If
If (Score >= 1) AND (Score <= 5) Then
   Set Rating = "D"
End If
```

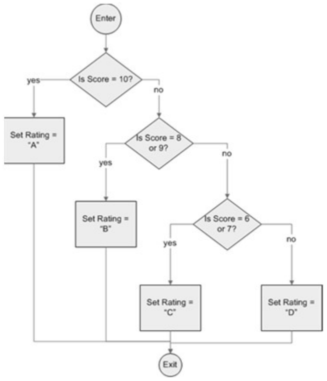
Assigning Ratings the Long Way



Assigning Ratings the Nested `If-Then-Else` Way

```
Declare Score As Integer
Declare Rating As Character
Write "Enter score: "
Input Score
If Score == 10 Then
   Set Rating = "A"
Else
   If (Score == 8) OR (Score == 9) Then
          Set Rating = "B"
   Else
          If (Score == 6) OR (Score == 7) Then
                 Set Rating = "C"
          Else
                 Set Rating = "D"
          End If
   End If
 End If
```

Assigning Ratings the Nested `If-Then-Else` Way



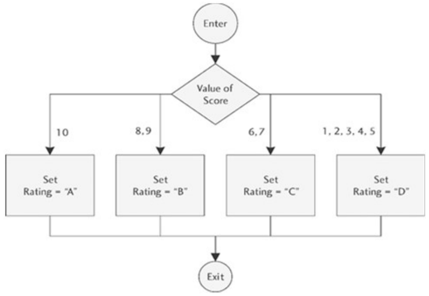
Using `Case`-Like (or `Switch`) Statements

```
Select Case Of ???                          test expression to be evaluated
Case 1st value:
              Statements                    execute if test expression is a match
              Break out of Cases
Case 2nd value:
              Statements                    execute if test expression is a match
              Break out of Cases
Case 3rd value:
              Statements                    execute if test expression is a match
              Break out of Cases
  .
  .                                         all the rest of the values that can be chosen
  .
Case nth value:
              Statements                    execute if test expression is a match
              Break out of Cases
Default:
              Statements                    execute if test expression does not match any of the above
End Case
```

Assigning Ratings Using a `Case` Statement

```
Declare Score As Integer
Declare Rating As Character
Write “Enter score: “
Input Score
Select Case Of Score
    Case 10:
            Set Rating = “A”
            Break
    Case 8, 9:
            Set Rating = “B”
            Break
    Case 6, 7:
            Set Rating = “C”
            Break
    Case 1-5:
            Set Rating = “D”
            Break
 End Case
```

Assigning Ratings using
A `Case` Statement



4.5 Applications of Selection Structures

Defensive Programming:

Program defensively in order to prevent bad data from entering our program. To do this, set **error traps**.

If our program should accept only a `Cost` greater than **0**, we can stop any other value from entering with the following trap:

```
Input Cost
If Cost <= 0 Then
      Write “Invalid cost”
Else
      Write “The cost is ” + Cost
End If
```

Error Trap for a Division by Zero Error

The reciprocal of a number is 1 divided by that number. But division by zero is not allowed ao the reciprocal of 0 is undefined. The following example will display the reciprocal of any number:

```
Write “Enter a number.”
Write “This program will display its reciprocal.”
Input Number
If Number != 0 Then
      Set Reciprocal = 1/Number
      Write “The reciprocal of “+ Number + ” is “+ Reciprocal
Else
      Write “The reciprocal of 0 is not defined.”
End If
```

Avoiding Illegal Operations: the `Sqrt()` function

The following program segment displays the square root of a number unless the number is negative:

```
Write “Enter a number:”
Write “This program will display its square root.”
Input Number
If Number >= 0 Then
      Write “The square root of “ + Number + “ is “+ ↵
                  Sqrt(Number)
Else
      Write “The square root of “ + Number + ” is not defined.”
End If
```

Defensive Programming

- ➢ Be sure to test your program by “**playing computer**.”
- ➢ This is also called “**desk checking**.”
- ➢ Perform all calculations multiple times manually or with a calculator.
- ➢ Use data that will show the results when each branch of each selection structure is executed at least once.
- ➢ Check for division by zero, negative values for a square root function, and any other special conditions.

Menu-Driven Programs

- ➢ A major goal of a programmer is to create **user friendly** programs.
- ➢ When the user has many options, **menus** are often used.
- ➢ Such programs are **menu-driven**.
- ➢ Menus are usually arranged in a row near the top of the screen.
- ➢ The user clicks the mouse on the desired choice.
- ➢ The **main menu** (a list of the program’s major functions) is usually the first thing the user sees.

Sample Menu

The Legendary Lawn Mower Company Inventory Control

Leave the program .... Enter 0
Add an item to the list .... Enter 1
Delete an item from the list ...... Enter 2
Change an item on the list .... Enter 3