Input and Output

• Inputs make data available for processing
• Inputs from
  – Users (keyboard, mouse, microphone, etc)
  – Files
• Outputs display the results of processing
• Outputs to
  – screen
  – printer
  – file
  – audio devices

The InputBox Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputBox</td>
<td>Displays an input box for user to enter input. Result is the string the user types in. The string is stored in a variable</td>
</tr>
</tbody>
</table>

Lecture Outline

• Input
  – Input boxes
  – Text boxes
• Output
  – Message boxes
  – labels
The Text Box Control

- Text Boxes allow user input on a VB.NET Form
  - Can also be used for output

- Properties:
  - (Name) – name of the control, e.g. txtCustomerAge
  - Text – its current contents as shown on screen (a String)
  - Font – the font to use for the characters typed/displayed
  - Many Others

- Events:
  - Click – generated if mouse is clicked inside it
  - Enter – generated when you arrive at this text box
  - Leave – generated when you leave from the text box
  - Many Others

Output

- A Message Box allows a String to be displayed to the user in a simple pop-up window:

```
MessageBox.Show("Hello " & strUsername, "Salutation")
```

MessageBox.Show

- Procedure
- Title

```
MessageBox.Show("Hello " & strUsername, "Salutation")
```

The Label Control

- Labels allow output to a VB.NET Form
  - Cannot be modified by the user

- Properties:
  - (Name) – the name of the control, e.g. lblTotalPrice
  - Text – its current contents as shown on screen (a String)
  - Font – the font to use for the characters typed/displayed
  - Many Others

- Events:
  - Click – generated if mouse is clicked on the label
  - DoubleClick – Occurs when you click twice in short time period
  - Many Others

Questions/Reading

- Unlock any questions
- Lock in the key concepts.
  - What’s an input box used for?
  - What’s the difference between input and message box.
  - Reading
    - Zak Chpt 3
    - Zakp193-197, p143 148
    - Unit Guide, SG 3
Overview

- Conditional expressions
  - Relational operators
  - Logical operators
  - Compound Conditions
- Decision Structures
  - IF statements
  - Nested IF statements
  - CASE statements

Overview

- Conditional expressions
  - Relational operators
  - Logical operators
  - Compound Conditions
- Decision Structures
  - IF statements
  - Nested IF statements
  - CASE statements

Relational Operators

- Allows the program to compare data items
  - Same value
  - Less than
  - More than
  - Less than or same
  - More than or same
- Data comparisons may involve:
  - Variables
  - Numerical operators
  - Functions
- E.g.: 
  - \( a < b \) asks “Is the value of \( a \) less than the value of \( b \)?”

Relational Operators

- Allows the program to compare data items
  - Same value
  - Less than
  - Greater than
  - Less than or same
  - Greater than or same
- Data comparisons may involve:
  - Variables
  - Numerical operators
  - Functions
- E.g.: 
  - \( a < b \) asks “Is the value of \( a \) less than the value of \( b \)?”

Numeric Comparisons

- Given a statement like:
  - \( \text{retailPrice} = 34.99 \)
  - the condition \( \text{retailPrice} = 50 \) can be tested,
  - and in this case it evaluates to ????

Numeric Comparisons

- Given a statement like:
  - \( \text{retailPrice} = 34.99 \)
  - the condition \( \text{retailPrice} = 50 \) can be tested,
  - and in this case it evaluates to ????

String Comparisons

- String comparisons are evaluated according to the ASCII code of the characters
  - Each character is considered in turn
  - i.e. the first character of both strings are compared.
  - if the first characters match, the 2nd characters are compared, etc.
  - When the compared characters differ, the one with the earlier ASCII code is considered to be an earlier string.
- digits < CAPITAL letters < lower case letters
  - “1Cat” is considered less than “OneCat”
  - “OneCat” is considered less than “oneCat”
  - “OneCat” and “oneCat” are considered to be not the same

String Comparisons

- String comparisons are evaluated according to the ASCII code of the characters
  - Each character is considered in turn
  - i.e. the first character of both strings are compared.
  - If the first characters match, the 2nd characters are compared, etc.
  - When the compared characters differ, the one with the earlier ASCII code is considered to be an earlier string.
- digits < CAPITAL letters < lower case letters
  - “1Cat” is considered less than “OneCat”
  - “OneCat” is considered less than “oneCat”
  - “OneCat” and “oneCat” are considered to be not the same

Relational Comparison Examples

- \( -5 < 5 \) TRUE
- \( 5 < 5 \) FALSE
- “cat” >= “car” TRUE
- “dog” < “Dog” FALSE

Relational Comparison Examples

- \( -5 < 5 \) TRUE
- \( 5 < 5 \) FALSE
- “cat” >= “car” TRUE
- “dog” < “Dog” FALSE

If a variable named myAge contains value 50:
- myAge< 60 TRUE
- myAge<= 50 FALSE
- myAge<> 50 FALSE
- myAge= 50 TRUE
- myAge<> 50 FALSE
Compound Conditions

- We can combine several conditions to form compound conditions
- Use logical operators to form compound conditions
- Logical Operators:
  - AND – require that the conditions are all true
  - OR – require that at least one of the conditions is true
  - NOT – reverse of the condition
- Truth Tables used to evaluate these
- Each part evaluated according to the order of precedence

Evaluation Order

- Conditions and expressions are evaluated according to an operator hierarchy:
  - arithmetic operations
  - >, <, >=, <=, >=
  - NOT, then AND then OR
- Brackets aid control of evaluation order
- Function calls, if they appear, are evaluated and their result effectively replaces the expression of that function
  - This occurs prior to any relational operators which have the function expression as part of it.
  - E.g.: Val(TextBox.text) < 50 causes the textbox’s value to be obtained (function call) before the comparison can be done.

Logical Operator - AND

- Requires both conditions (on either side of AND) to evaluate TRUE, in order for the AND-expression to evaluate TRUE
- Consider two conditions, represented A and B

<table>
<thead>
<tr>
<th>Condition A</th>
<th>Condition B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Logical Operator - OR

- Requires either or both conditions to evaluate TRUE, in order for the OR-expression to evaluate TRUE

<table>
<thead>
<tr>
<th>Condition A</th>
<th>Condition B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Logical Operator - NOT

- The NOT operator can negate the truth-value of a condition
- If a condition is normally TRUE, applying the NOT operator will result in FALSE
- If a condition is normally FALSE, applying the NOT operator will result in TRUE

<table>
<thead>
<tr>
<th>Condition A</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Using NOT with AND

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>(A AND B)</th>
<th>NOT (A AND B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Using NOT with AND

Contrast the previous Truth Table with the following:

<table>
<thead>
<tr>
<th>A</th>
<th>NOT A</th>
<th>B</th>
<th>NOT B</th>
<th>(NOT A) AND (NOT B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>FALSE</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>FALSE</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>FALSE</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Decisions / Selection

• Choice of two paths through a program
  - TRUE branch/path
  - FALSE branch/path
• Choice of branch taken is determined by evaluation of a condition at run-time
  - We do not know at coding-time whether a branch will be taken
• Use these key words:
  - IF condition THEN
  - ELSE
  - ENDIF

Decisions in Algorithms

Situation 1 – Do something sometimes:
IF a condition is true THEN
perform a set of actions
ENDIF

Get in Car
Turn car on
IF (Petrol tank is nearly empty) THEN
Go to Petrol Station
Fill up Tank
ENDIF
Drive to destination

Get in Car
Turn car on
IF (It is raining) THEN
Turn on Windscreen Wiper
Wait for Windscreen to become clear
ELSE
Open the Sun Roof
Wind Down the Windows
ENDIF
Drive to destination

Decision Control Structures

Graphical Representation:

IF condition = true THEN
YES Actions
ELSE
NO Actions
ENDIF

Nested Conditions

• Sometimes we have more than 2 courses of action
• We can use Nested IFs
• 2 types:
  - Linear – Used when more than 2 mutually exclusive cases
  - Non-Linear – Used when several conditions combine to describe a specific case.

Example – Linear Nested IF

IF mark >= 80 THEN
grade = “High Distinction”
ELSE
IF mark >=70 THEN
grade = “Distinction”
ELSE
IF mark >= 60 THEN
grade = “Credit”
ELSE
IF mark >= 50 THEN
grade = “Pass”
ELSE
grade = “Fail”
ENDIF
ENDIF
ENDIF

Decisions are nested like Russian dolls. You only reach the next level if you don’t perform the action at the current level.

Test this algorithm with mark = 55 and mark = 75
Example – Non-Linear Nested IF

IF it is Winter THEN
  Wear a jumper and long pants
ELSE
  Take an umbrella
ENDIF

ELSE
  Turn central heating on
ENDIF

ELSE
  Wear shorts and T-Shirt
IF I am going outside THEN
  Wear Hat and Sunscreen
ELSE
  Turn on Air Conditioner
ENDIF

Desk-checking: Algorithms with Selection

• Aim of Desk Checking is to check the logic of program
• Selection structures mean that some of the algorithm won’t be executed
  – If the TRUE-block is done, the FALSE-block is not done, etc.
• Need to select test data sets such that all paths through algorithm will be checked at least once.

Test Data for Nested-IF example

• For the University Grading example:

<table>
<thead>
<tr>
<th>Test Data (mark)</th>
<th>Expected Result (grading)</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>“High Distinction”</td>
</tr>
<tr>
<td>20</td>
<td>“Fail”</td>
</tr>
<tr>
<td>65</td>
<td>“Credit”</td>
</tr>
<tr>
<td>51</td>
<td>“Pass”</td>
</tr>
<tr>
<td>79</td>
<td>“Distinction”</td>
</tr>
</tbody>
</table>

University Grades Example (1) – Nested IFs

If intMark >= 80 Then
  strGrade = "High Distinction"
Else If intMark >= 70 Then
  strGrade = "Distinction"
Else
  If intMark >= 60 Then
    strGrade = "Credit"
      Else If intMark >= 50 Then
        strGrade = "Pass"
          Else
            strGrade = "Fail"
          End If
      End If
  End If
End If

IF .. THEN .. ELSEIF ... ELSE .. ENDIF

• VB.NET Provides the ElseIf keyword as an alternative to Else
• Eliminates the need to have linear nested IFs
• Must provide a new condition

University Grades Example (2) – Cascading IF

If intMark >= 80 Then
  strGrade = "High Distinction"
Else If intMark >= 70 Then
  strGrade = "Distinction"
ElseIf intMark >= 60 Then
  strGrade = "Credit"
ElseIf intMark >= 50 Then
  strGrade = "Pass"
Else
  strGrade = "Fail"
End If
Case Statements

• Sometimes you have many conditions which test the value of a common variable, called the Selector
  – E.G. ‘mark’ in the previous examples
• Case statements simplify decisions with more than two possible branches for a single selector
• Useful for menus

Algorithmic form (as in Robertson):

```
CASE OF variableName
  value1: statements
  value2: statements
  ...
  otherwise: statements
ENDCASE
```

Case Statements

Syntax for Visual Basic:

```
Select Case Selector
  Case values
    Actions
  Case values
    Actions
  Case Else
    Actions
End Select
```

Case statements in Visual Basic .NET

• Values can be
  – A constant, variable, or literal of a matching type
  – An expression which evaluates to true
  – A range
• Example:
  ```
  Select Case dayOfWeek
    Case 1, 7
      TextBox.Text = "Weekend"
    Case 2 To 6
      textBox.Text = "Weekday"
    Case Else
      TextBox.Text = "Invalid Day"
  End Select
  ```

University Grades
Example (3) – Using Case

```
Select Case intMark
  Case Is = 100
    strGrade = "Perfect Score"
  Case Is >= 80
    strGrade = "High Distinction"
  Case Is >= 70
    strGrade = "Distinction"
  Case Is >= 60
    strGrade = "Credit"
  Case Is >= 50
    strGrade = "Pass"
  Case Else
    strGrade = "Fail"
End Select
```
Which structure do I use?

- Single option (null/absent ELSE branch)
  IF condition THEN
  ENDIF ...
- Two options
  IF condition THEN
  ELSE
  ENDIF ...

Three options
- Use nested IF for algorithm
  IF condition THEN
  ELSE
  IF condition THEN
  ENDIF
  ENDIF
- May use nested or cascading IF in code
- Four or more options - Case statement

Questions/Reading

- Take a minute to note what you covered today
  - key points
  - major concepts
  - any questions?
- Reading
  - Study Guide 4
  - Zak, Chpt 4
  - Robertson, Chpt 4