Visual Basic in Detail

CMPT 110

Adapted with permission from Dr. Imad Rahal, College of St. Benedict.

COLLEGE OF Saint Benedict

Tentative Syllabus

Week	Topic
1 (Sept. 4th)	Introduction to Course
2 (11th)	Introduction to Programming
3 (18th)	Programming in VB
4 (25th)	Events
5 (Oct. 2nd)	Representing and Storing Values
6 (9th)	Subprograms
7 (16th)	MIDTERM
8 (23rd)	Decisions
9 (30th)	Iteration
10 (Nov. 6th)	Arrays
11 (13th)	I/O
12 (20th)	Graphics
13 (27th)	Review

KNOW THIS PRESENTATION!

Summary to Date

- We have learned
  - The basics of computing science,
  - The basics of algorithms,
  - OOP,
  - Introduction to VB

0 - Admin-110.pptx
1 - Introduction.pptx
2 - Algorithms.pptx
3R - The High-Speed Electronic Digital C...
Algorithm Flow Chart.pdf

4 - Object Oriented Introduction.pptx
5 - Programming in Visual Basic.pptx

  - Now, let us look at the details of VB

Review of Computer Programming

- Different types of high-level language approaches
  - **Imperative** programming paradigm
    - A program is a **sequence of instructions**
    - Machine language/assembly language
  - **Object-oriented** sub-paradigm (just as we, humans, think)
    - Visual Basic or VB uses the OOP
  - **Event-Driven**

VB is all of theses things…

Review of Computer Programming

- Different types of high-level language approaches
  - **Imperative** programming paradigm
    - A program is a **sequence of instructions**
    - Machine language/assembly language
  - **Object-oriented** sub-paradigm (just as we, humans, think)
    - Visual Basic or VB uses the OOP
  - **Event-Driven**

- Objects each having properties and actions: *Imagine a computer screen with several icons*
  - Each icon is an object – also desktop is an object
  - Each icon has its own attributes and actions
    - Properties: Icon image/shape and color
    - Actions:
      - Single-clicked ➔ Darken Color (affect itself)
      - Double-clicked ➔
        - Run corresponding program (Program is another object)
        - Change desktop look (Desktop is another object)

Review of OOP

- The program is collection of interacting objects
- In VB, actions are triggered by events
  - Done by users (interactive): e.g. CLICK or DOUBLE_CLICK
- Objects sharing properties/methods are **grouped into classes**
  - ***Class***: An Icon (Student)
  - ***Object***: My Documents, My Computer (John, Lynn, Cathy, *etc.*)

Introduction

- Basic is an imperative language developed in 1960s
- Visual Basic (VB) --- developed by Microsoft
  - OO version of Basic
  - Underlying ALL MS office programs
    - You could do a lot
      - Write your own programs
      - Tailor office applications
  - Visually oriented
    - Programs have a GUI for input/output
    - First thing to do to build a VB program is to design the interface

- Intro to VB

Hungarian Notation

- **A must in this course**
- Every object used MUST be renamed including the form(s) using the following rules
  - Form ➔ frm***F***orm***N***ame
    - E.g. **frmT**emperature
    - ***Also change caption***
  - Textbox ➔ txt***T***ext***B***ox***N***ame
    - E.g. **txtI**nput**B**ox
    - ***Also change Text (to empty)***
  - Label ➔ lbl***L***abel***N***ame
    - E.g. **lblD**escribe**I**nput
    - ***Also change caption***
  - Button ➔ cmd***B***utton***N***ame
    - E.g**. cmdC**ompute**B**utton
    - ***Also change caption***
  - PictureBox ➔ pic***P***icture***B***ox***N***ame
    - E.g. **picO**utput**B**ox

Simple Calculation Example

Simple Calculations

- Algorithm
  - Get temperature in Celsius, call it Cel
  - Convert using Fah = 9/5*Cel+32
  - Print result in picture box

- **Show code: *option Explicit***

```
Private Sub cmdConvertButton_Click()
   'program to convert Celsius to Fahrenheit
   Dim Cel As Single     'declare a variable of type float ... no space in name
   Dim Fah As Single     'declare a variable of type float... MUST Starts with a letter
   Cel = txtInputText.Text    'Read value input in Textbox into variable C
   Fah = 32 + 9 / 5 * Cel      'Compute Fah
   picOutputPicture.Print Cel; " degrees Celsius is"; Fah; " degrees Fahrenheit."  'Print result
End Sub
```

Simple Calculations

- **'program to convert Celsius to Fahrenheit**
  - a comment describing what the subroutine does…for the reader
  - Everything starting with a quotation mark (‘) is ignored by the computer
  - For longer programs, we need more comments spread throughout the program

- Cel & Fah are variables that represents the memory registers used to store the floating-point values

Variable Declarations

- Declaring variables
  - At the beginning of the code
  - **Dim** *variable_name* **As** *Type*
    - Dim C As Single
- The variable name
  - or a string of characters (letters, numbers, and others)
    - MUST Start with a letter
    - NO SPACES
  - Name related to the value it is storing (**REQUIRED**)

(read https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/variables/how-to-create-a-new-variable)

Variable Declarations

- **Type**
  - **Integer** for short integers (2's complements)
    - 2 bytes
    - **-2,147,483,648 through 2,147,483,647 (signed)**
  - **Long** for long integers (double precision 2's complements)
    - 4 bytes
    - **9,223,372,036,854,775,808 through 9,223,372,036,854,775,807**
  - **Single** for short real numbers (floating-point)
    - 4 bytes
    - **3.4028235E+38 through -1.401298E-45 for negative**
    - **1.401298E-45 through 3.4028235E+38 for positive**
  - **Double** for long real numbers (double precision floating-point)
    - 8 bytes
    - **-1.79769313486231570E+308 through -4.94065645841246544E-324 for negative**
    - **4.94065645841246544E-324 through 1.79769313486231570E+308 for positive**
  - **Boolean**: True or False
    - 2 bytes
  - **String** for character strings (ASCII)
    - Variable length
    - **0 to approximately 2 billion Unicode characters**

Variable Declarations

- After declaring a variable, we **initialize** it
  - Dim A As Integer
    A = 10
- If not provided
  - All numbers are set to 0
  - Strings to empty
  - Booleans to false
- VB does not force us to declare them
  - Using **Option Explicit**
    - Placed at the top of code module sheet
    - Forces all variables to be declared on that form
    - **A MUST IN THIS COURSE**

Simple Calculations

- **Fah = 32 + 9 / 5 * Cel**
  - We need to
    - divide by 5
    - multiply by Cel
      - Whenever a variable is used, the computer uses the corresponding value stored in the variable
    - add 32
    - BUT IN WHAT ORDER?

Order of Precedence

- Arithmetic operations

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Exponentiation	**^**

- What happens when we have more than 1 in the same equation?

- F = 32 + 9 / 5 * C
- (), ^, * and /, + and –
  - Handled from left to right
- A = 3*8/2^2+2-5
  - 2^2 = 4 ➔ A = 3*8/4+2-5
  - 3*8 = 24 ➔ A = 24/4+2-5
  - 24/4 = 6 ➔ A = 6+2-5
  - 6+2 = 8 ➔ A = 8-5
  - 8-5 = 3 ➔ A = 3

Simple Calculations

- **picOutputPicture.Print Cel; "degrees Celsius is"; Fah; "degrees Fahrenheit."**
  - An object function
  - The value in Cel is printed first
  - A semi-colon is used to separate the numbers and the following or preceding words
  - Text between double quotes is printed as is
  - All printing is done in the Picture Box (pic**OutputPicture**)
- Every time we click the button, the program prints the required output
  - What happens if we click it twice?
    - i.e. what happens to the old output when there is new output
  - **picOutputPicture.Cls**
    - At the beginning of the code
    - Clears all previous output

Simple Calculations

- To add code for the cmdExitButton
  - We need to double click it in design mode
- It is required to include an cmdExitButton on every form
  - Good programming practice
- **Indentation** makes things clearer for the reader

```
Private Sub ExitButton_Click()
    End
End Sub
```

Numeric Functions

- Sqr(X)
  - Square root
  - Sqr(4)
- Int(X)
  - Int(2.1234)
- Round(X)
  - To the nearest integer
  - Round(2.45)
- Abs(X)
  - Abs(-34)
- Log(X)
  - Log(4)

- Program: calculate the roots of a quadratic equation given a, b and c

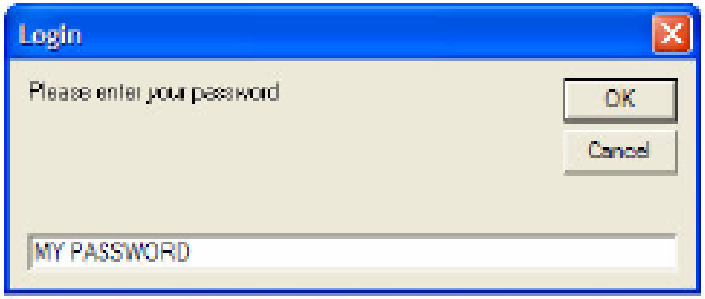
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

- **Private Sub Quadraticbutton_Click()**

```
Dim A As Integer, B As Integer, C As Integer
Dim R1 As Double, R2 As Double
A = txtA.Text
B = txtB.Text
C = txtC.Text
R1 = (-B+Sqr(B^2-4*A*C))/(2*A)
R2 = (-B-Sqr(B^2-4*A*C))/(2*A)
Results.Print "Roots are:";R1;R2
```

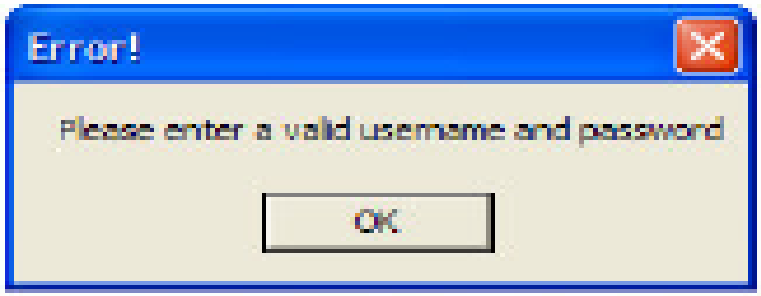
- **End Sub**

Input



- Textboxes
- Input Boxes
  - Different than textboxes
    - Good for small amount of input (form full of textboxes is not nice)
  - *X = Inputbox("prompt message", "title message")*
    - The input is assigned to variable X when the use hits OK
  - A = Inputbox("Enter an exam score","Exam 1")

Output



- Pictureboxes
- Message Boxes
  - For output specially in situations of errors
  - different than pictureboxes
  - ***MsgBox "prompt message", , "title message"***
  - MsgBox "Sorry but you must enter a positive number" , , "Error"

Output Display (*pictureboxes*)

- Spacing
  - , (comma) … ALIGN OUTPUT
    - Every picture box is divided into 14-space print zones
    - Jumps to the next zone (does not jump 14 spaces necessarily!)
      - From its current location (its current 14-space zone)
    - Results.Print 5,30,2
  - Tab function: **Tab(**X**)**
    - Followed by a ; (automatically inserted by VB)
    - leaves X spaces from start
    - Pushes any overlap to new lines
      - Results.Print Tab(2); "Hi!"
      - Results.Print Tab(2); "Hi!"; Tab(2); "Bi!";
        - What should the second 2 be to print on the same line
- A comma or semi-colon at the end of a print statement prints next print statement on the same line

Output Display

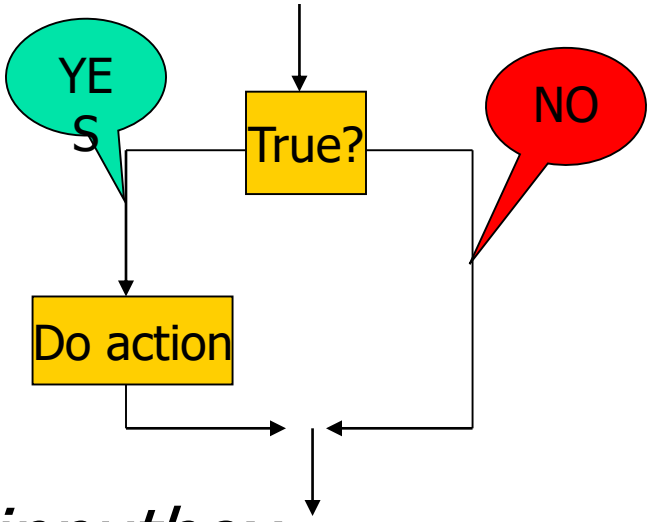
- **FormatCurrency(x,y) --- y is 2 by default**
  - x is the decimal number to be formatted
  - y is the number of digits allowed after the decimal point (rounding)
    - Extra 0s are appended
  - Adds a $ sign and commas if needed
  - FormatCurrency(1234.234,2) = $1,234.23
  - FormatCurrency(1234.2,2) = ?
- **FormatPercent(x,y) --- y is 2 by default**
  - x is the decimal number (less than 1) to be formatted
  - y is the number of digits allowed after the decimal point (rounding)
    - Extra 0s are appended
  - FormatPercent(0.5235, 3) = 52.400%
- **FormatNumber(x,y)** --- y is 2 by default
  - Rounds x to the nearest number of digits after the decimal point specified by y
  - FormatNumber(0.5235) = 0.52
  - FormatNumber(0.5235,3) = 0.524

Variable Scope

- Where we declare a variable defines its **scope**
  - i.e. where it is known
  - Determines what subroutines can access it
    - Subroutine-level: Within a subroutine
      - ➔ only in that subroutine
    - Form-level: At top of the form (before the code for the first subroutine under Option Explicit)
      - Not within any subroutine
      - Accessible by all SUBROUTINES for that form

If Statements

- We want to execute something only when a certain condition is met
  - `If condition(s) then`
    - `Statements to execute`
  - `End If`



- Read student grade using an *inputbox*
  - If 90 or more, print the following in a *message box*
    - "Congratulations on your hard work!"

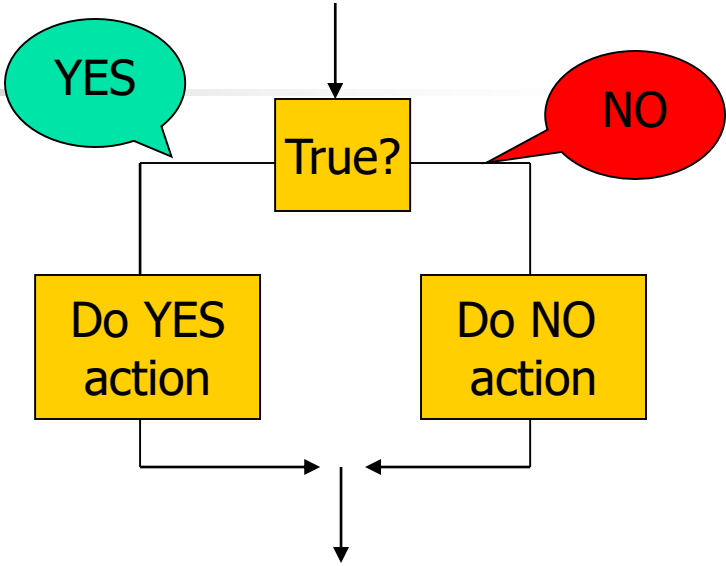
If Statements

- Private Sub cmdGrade_Click()
  - Dim grade As Single
  - grade = Inputbox ("Please enter a score")
  - If grade >= 90 then
    - MsgBox "Congratulations on your hard work!"
  - End If
- End Sub

Allowed Comparison operations

Equals	**=**
Greater than	**>**
Less than	**<**
Greater than or equal	**>=**
Less than or equal	**<=**
Not equal (i.e. different)	**<>**

If Statements



- What if grade is less than 90?
  - Nothing happens
  - Not very good!
- If statement with the else clause
  - `If condition(s) then`
    - `Statements to execute if condition is met`
  - `Else`
    - `Statements to execute if condition is NOT met`
  - `End If`
- If grade>=90 then display congrats message; otherwise, display support message

If Statements (not indented)

```
Private Sub Grade_Click()
Dim grade As Single
grade = Inputbox ("Please enter a score","Score")
If grade >= 90 then
MsgBox "Congratulations on your hard work!", ,"YES"
Else
MsgBox "DUDE!!! Try harder next time!", ,"NO"
End If
End Sub
```

If Statements (not indented)

```
Private Sub Grade_Click()
  Dim grade As Single
  grade = Inputbox ("Please enter a score","Score")
  If grade >= 90 then
      MsgBox "Congratulations on your hard work!", ,"YES"
  Else
      MsgBox "DUDE!!! Try harder next time!", ,"NO"
  End If
End Sub
```

Notice the indentation

- Private Sub/End Sub
- IF/End If/Else

If Statements

- Compound statements for compound conditions
- More than one condition can appear in a statement which can be joined by either
  - NOT ➔ the opposite
  - AND ➔ both must be true
  - OR ➔ at least one must be true
  - NOT takes precedence over AND and AND takes precedence over OR

- **Detect whether input grade is valid or invalid** (i.e. outside [0,100])
  - If grade>=90 then display congrats message; otherwise, display support message
  - Nested Ifs

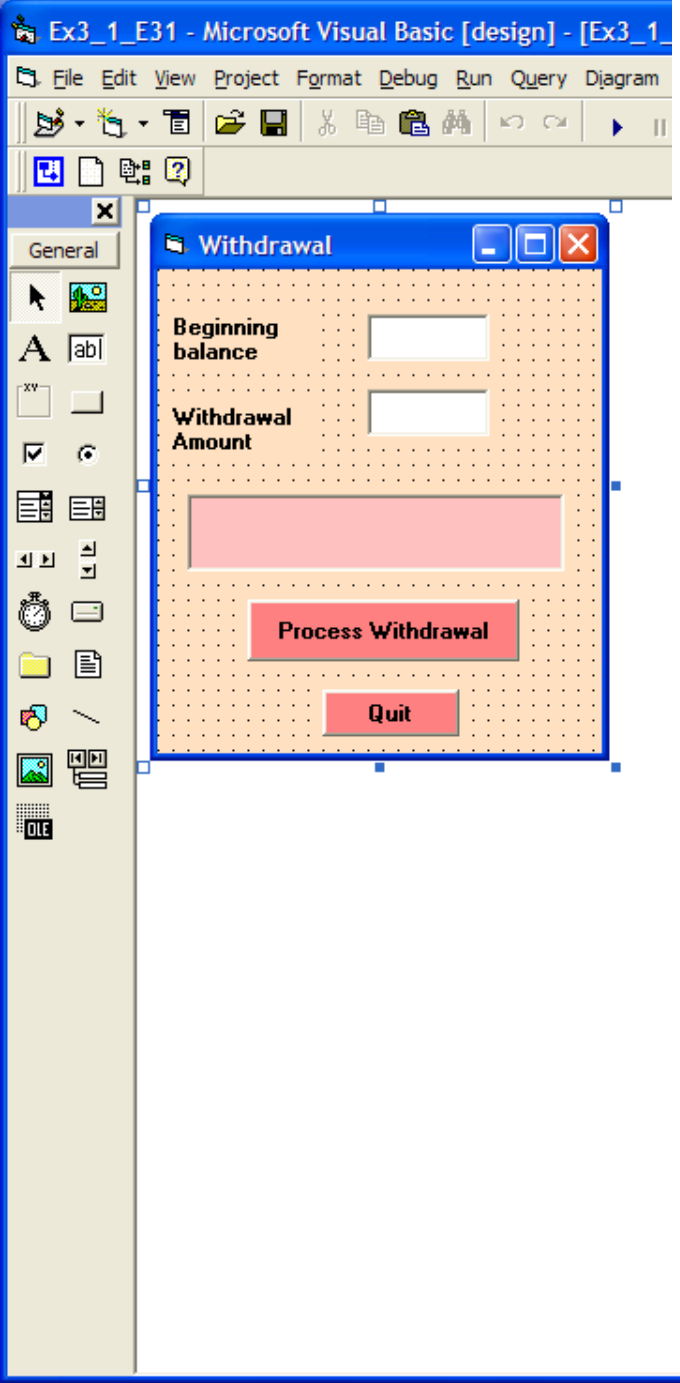
- Show Cel/Fah conversion but limit input to -100,+100
  - If not valid, inform user

Find Output

- Dim X As Integer, Y as Integer, Z as String
- X = 5
- Y = 2*X
- Z = “CSCI-130”
- If Z = “PHIL-330” OR X<>4 AND Y>9 Then
  - Msgbox “Good Job”
- Else
  - Msgbox “Even Better”
- End If

Numeric to Letter grade

- **If** grade >= 90 Then
  - Msgbox grade & " ==> A"
- **ElseIf** grade >= 80 Then
  - Msgbox grade & " ==> B"
- **ElseIf** grade >= 70 Then
  - Msgbox grade & " ==> C"
- **ElseIf** grade >= 60 Then
  - Msgbox grade & " ==> D"
- **Else**
  - Msgbox grade & " ==> F"
- **End If**



1. A VB project to perform simple bank withdrawals

2. If the withdrawn amount is larger than the beginning balance, the user is warned and nothing happens to the balance … display warning in a ***message box***:

   ERROR: Withdrawal denied!

3. Otherwise, the new balance is shown in the picture box like the following:

   The new balance is $889.12

4. In the last case above, modify your program to use a nested IF so that if the new balance is below $150.00, the user is informed

   The new balance is $89.12

   WARNING: Balance below $150!

Do Loops – Week 8 Topic

- Stopping/looping condition but not sure how many iterations
  - good with flags (or sentinels)
- **Do While --- LOOPING CONDITION**
  - `Dim InputNum As Single, Sum As Single`
  - `Sum = 0`
  - `InputNum=inputbox("Enter a number to add, 0 to stop")`
  - `Do While InputNum<>0`
    - `Sum = Sum + InputNum`
    - `InputNum=inputbox("Enter a number to add, 0 to stop")`
  - `Loop`
  - `Msgbox Sum`
- **What would happen if I don't read input again from user inside loop?**
- **Print all even numbers between 2 and 100 in a picturebox**

Do Loops

```
Dim counter As Integer
counter=2
Do While counter <=100
    picResults.Print counter
    counter = counter +2
Loop
```

- *Initialization phase*
- *Exit conditions*
- *Action* (Loop body: Do … Loop)
  - Must include progress

Do Loops

- Program to compute the average for any exam in any class at CSBSJU (number of students differs)
  - Enter any number of grades
  - When the flag (-1) is provided (-1 is known as sentinel value)
    - end of input and print average
- Algorithm + program
  - Show program

Do Loops

- Program to get the average of any number of grades (enter -1 to exit)
  - Sum = 0, count=0, average=0
  - grade=input by user
  - While grade <> -1
    - Add grade to sum
    - Add 1 to count
    - Enter another grade
  - Divide sum by count to obtain the average
  - Print average

Do Loops

```
Private Sub cmdAverageButton_Click()
    'program to compute the average exam score
    Dim count As Integer
    Dim grade As Single, sum As Single, avg As Single
    grade = InputBox ("Enter the first exam score:", "Exam Scores")
    Do While grade <> -1
        sum = sum + grade
        count = count + 1
        grade = InputBox("Enter another score Type -1 to end.", "Exam Scores")
    Loop
    avg = sum/count
    MsgBox "The average is: " & FormatNumber(avg)
End Sub
```

For Next Loops

- When we know how many times we need to repeat the loop
  - **With a consistent** increase or decrease ➔ For Next Loops are better

- **Display values between 1 to 5** (vs. **until user inputs -1**)
  - `Dim CTR As Integer`
  - `For CTR = 1 to 5`
    - `picResults.Print CTR`
  - `Next CTR`
  - After every loop, the loop counter is incremented by 1 (default)
- Initialization: `CTR=1`
- Exit Condition: `CTR>5`
- Action: `picResults.Print CTR  (NO NEED TO MAKE PROGRESS, DONE AUTOMATICALLY FOR YOU)`
- Combines the initialization and exit conditions into one line
  - Previously initialization was done before loop

For Next Loops

- After every loop, the loop counter is incremented by 1 (default)
  - Can be changed by specifying **steps (display even numbers from 2 to 100)**
    - `For CTR = 2 to 100 Step 2`
      - `picResults.Print CTR`
    - `Next CTR`
  - Can be positive or negative **(display even numbers from 100 to 2)**
    - `For CTR = 100 to 2 Step -2`
      - `picResults.Print CTR`
    - `Next CTR`

For Next Loops

- The steps don't have to be integers
  - `For CTR = 1 to 3 Step .5`
    - `picResults.Print CTR`
  - `Next CTR`
- Suppose we want to display all even numbers between 2 and another number specified by the user
  - `Dim CTR as Integer, N As Integer`
  - `N = txtEndBox.Text`
  - `For CTR = 2 to N Step 2`
    - `picResults.Print CTR`
  - `Next CTR`
- Design a VB program that displays in a picture box the first N multiples of an input integer

For Next Loops

- **Input 19 lab scores for each of 26 students**
  - `For Student = 1 to 26`
    - `For score= 1 to 19`
      - `Grade = InputBox("Enter a score for lab" & score)`
    - `Next score`
  - `Next Student`
- The nested (inner) loop is executed completely for each execution of the outer loop --- how many?
  - Outer 26, inner 26*19=494
- We can nest loops but they may not overlap
  - i.e. if the start of a loop L1 starts after that of another one, L2, then L1's end must come before L2's

Input & Output

- Data Files
  - vs. *Textboxes* and *Inputboxes*
  - When we have a lot of input --- not convenient to enter all at runtime
    - Average for exam scores in a class of 30 students

- Open some text editor (Notepad)
  - Enter data separating individual pieces of data by commas and rows by placing them on separate lines
  - Save the file with its own name
  - **NO EMPTY LINES AT THE END**

Input & Output

- File must be in the project's folder
  - create project and save it first and then save the data file in the same folder
  - `Open App.Path & "\FILE NAME" For Input As #1`
  - `Do While Not(EOF(1))`
    - `Input#1, col1, col2, col3,…`
  - `Loop`
  - `Close #1`

- Looping used with files:
  - Read until a flag is met
  - **`EOF(x)`** is a special flag that tells the program to loop until the end of the file (whose number is provided in parentheses)

Arrays – Week 9 Topic

- We have been using variables
  - pointing to a single memory register
  - holding a single value
- What if we have to store a **list of related data**
  - E.g. exam-1 scores for this class
    - declare 30 variables?
- An array is a variable that holds multiple data pieces such as a list or table
  - A block of memory registers
- Declare it by giving a name and a size
  - `Dim ArrayName(1 to MaxSize) As DataType`
- Refer to stored data by their positions

Arrays

- *Dim* ***Runner(1 to 75)*** *As String*
  - 75 memory registers are set up for array Runner
  - Each holds a name string
  - To print the 15th runner: *picResults.Print* ***Runner(15)***
- **Declare two arrays for exam-1 students and grades for this class**
- **Display student 10 with his/her grade in a messagebox**
  - **Parallel arrays**
- Average the contents of the runner times array
  - `Dim` **`Times(1 to 75)`** `As Single, Pos As Integer`
  - `For Pos = 1 to 75`
    - `Sum = Sum + Times(Pos)`
  - `Next Pos`
  - `Avg = Sum/75`

Arrays and Files (Week 10)

- **Runners and times program**
  - **Parallel arrays**
  - **1 - Reads names & times into arrays**
    - **used multiple times**
    - **done in a separate step**
    - **maintain CTR as an array index**
  - **2- Find average**
- size vs. capacity
- **Change to display all runners with less time than an average**

Sequential Search

- Two types of array **sequential** searching problems
  - (1) Match-and-stop (first match)
    - Begin at first array element
    - Compare search value with each data element
    - If there a match , stop and return match or its position
    - Otherwise, go to next element and repeat
    - Stop looking when a condition is satisfied or when we have finished the list in which case we return a failure
      - Usually we use a Do Loop

Match-and-Stop Search

- Use Boolean variable to denote whether a match has been found or not
  - Found initially **False**
  - If a match is found, we set it to **True**

```
Dim Found As Boolean
Found = False
POS = 0
SearchValue = Inputbox…
Do While (Found=false and POS<(Array size))
    POS = POS +1
    If Array(POS) = SearchValue Then
        Found=True
        Print success at location POS
    End If
Loop
If Found=False then
    Print Failure
End If
```

Match-and-Stop Search

- Read an array of 75 runner names and array of 75 runner times from a file and search for a specific name input through an *inputbox*
  - If found, display rank and time
  - Else, display not found
- What would happen if we didn't use `POS<(Array size)`?
  - Try a successful and an unsuccessful search
- What would happen if the `else` is included in the loop and not outside

Sequential Search

- (2) Exhaustive search (must check all array elements)
  - E.g. find all values less than a given value, maximum, or minimum
    - Must scan whole array
    - We use a For Loop
    - **For Pos = 1 to (CTR)**
      - **If condition is true then do action**

Exhaustive Search

- Write a program that reads numbers from a file (unknown size but say less than 100) into an array
- Uses the array to find and display all numbers larger than an input minimum threshold

Exhaustive Search

```
Private Sub cmdSearchButton_Click()
    Dim CTR As Integer, Pos As Integer, Min As Integer
    Dim Numbers(1 to 100) As Integer
    Found = False
    Open App.Path & "\Numbers. txt" For Input As #1
    CTR = 0
    Do While not(EOF(1))
          CTR = CTR + 1
          Input #1, Numbers(CTR)
    Loop
    Min = InputBox("Enter your minimum threshold please" , "INPUT")
    For Pos = 1 to CTR
          If Min < Numbers (Pos) Then
                   Found = true
                   picResults.Print Numbers(Pos)
          End If
    Next Pos
    If found=false then
           picResults.Print "There were no matches"
    End if
End Sub
```

Multiple Forms

- We've been designing projects with single forms
- Very easy to add new forms to your project if needed
  - A form to load data from a file into an array
  - A form to search for entries in the array
  - A form to sort the array and display it sorted
- Different backgrounds and designs
- How to move from one form to another
  - **Visible properties**
    - frmFirst.Visible = False
    - frmSecond.Visible = True
  - Similar to buttons (cmdMyButton.Visible)
  - Hide/Show actions
    - frmFirst.Hide
    - frmSecond.Show

Multiple Forms

- All forms except the initial has this property set to Hide
- `Private Sub cmdChangeformbutton_Click()`
  - `cmdInitialForm.Hide`
  - `cmdNewForm.Show`
- `End Form`
- `Private Sub cmdChangeformbutton_Click()`
  - `cmdInitialForm.Visible = False`
  - `cmdNewForm.Visible = True`
- `End Form`
- Click on the Select project ➔ Add Form ➔ Form
  - Click on **Add Form**
  - Will create a new form in the same project

Multiple Forms

- Variable scope categories
  - Subroutine-level vs. form-level vs. project-level

- To use the **same variables on more than one form** we need to declare them differently
  - Add new code module
  - At the top of a code module as public
  - In this module, we list all declarations of all variables that be shared among multiple forms
  - To add a code module
    - Select project ➔ Add Module ➔ Open
    - Enter declarations
    - Public ***SomeVar*** As ***SomeType***
      - Private keyword

Dynamic Picture Loading

- `picResults.Picture = loadPicture(App.Path & "\" & imageFileName)`

Presentation Terminated