Table of Contents – Advanced Visual Basic

Review of Important Programming Concepts

Overview

- ***Sequence*** (statements executed one after the other), ***selection*** and ***repetition*** are the main structures of programming.
- ***Selection*** ("**If**" statements in VB) is used whenever programs need to make ***decisions***.
- **Loops** (repetition structures) are used whenever groups of statements need to be repeated.
- **Loops** are useful in processing large amounts of data, adding up lists of numbers, finding averages, animating objects and a wide variety of other applications.
- **Loops** are classified as ***"counted"*** or ***"conditional."***
- ***Counted*** loops ("**For** … **Next**" loops in VB) are used whenever the number of repetitions is known at design-time. Counted loops automatically increment the ***counter variable***.
- ***Conditional*** loops ("**Do** … **Loop**" loops in VB) are used whenever the number of repetitions is not known at design-time. These loops continue ***while*** a certain ***condition*** is true or ***until*** a certain ***condition*** is true.
- Within a form (object) module or a code module, ***variables*** can be declared ***locally*** (called ***"at procedure level"*** in VB) or ***globally*** (called ***"at module level"*** in VB).
- A variable that is declared ***locally*** exists (i.e. is visible) only within the **`Sub`** in which it is declared. It is created when the **`Sub`** or **`Function`** is ***invoked*** (called) and destroyed when the **`Sub`** or **`Function`** ***returns*** (is finished executing). Local variables cannot be accessed outside the **`Sub`** or **`Function`** of declaration. Since local variables are destroyed as soon as the **`Sub`** or **`Function`** finishes executing, memory can be freed for other parts of the program or for other applications. Furthermore, local variables help to decrease debugging time (because bugs are localized to **`Subs`** and **`Functions`**) and they help to make code reusable. ***Whenever possible, declare variables as local variables!***
- A variable that is declared ***globally*** (at module level) is accessible to all **`Subs`** or **`Function`** within the module. Moreover, if the variable is declared as **`Public`**, then it is accessible to **`Subs`** or **`Functions`** in all modules of the program. ***Use global variables only when necessary (especially if they are*** **`Public`**). ***If global variables are used in a careless manner, programs can become extremely difficult to understand, modify and debug. In addition, the overuse of global variables makes it difficult to design reusable code.***
- In C and C++, local variables are called ***automatic variables*** and global variables are called ***external variables.***
- Use names like ***InsertionPoint*** instead of *insertionpoint*, *INSERTIONPOINT*, *insertion_point* or *INSERTION_POINT*. This practice is known as "UpperCamelCase." (See excerpt from Wikipedia article on the next page.)
- Use names that ***clearly describe the purpose of a variable, constant, sub procedure*** or ***function procedure***.
- Using ***meaningful, descriptive*** names will allow you to write programs that are for the most part self-explanatory. This means that you do not need to include too many comments. However, ***comments should still be considered an integral part of the software development process***. Comments should be included as you write your code, not after it is written!
- Generally, include ***comments for major blocks of code*** and for any ***code that is not self-explanatory***.
- Use global variables only when necessary! All other variables should be declared either within procedures or as parameters of procedures.
- ***Avoid repetitive code*** by writing sub procedures or function procedures and calling them whenever they are needed.
- ***Consider several different algorithms*** and implement the one that best suits your needs.
- ***Indent your code properly*** as you write it! Do not consider indentation an afterthought.
- ***Test your code thoroughly under extreme conditions***. Allow other people to conduct some of the testing and note all bugs.

Excerpt from Wikipedia Article on CamelCase

For the full text of the article, visit http://en.wikipedia.org/wiki/CamelCase .

CamelCase

From Wikipedia, the free encyclopedia

CamelCase, **camel case** or **medial capitals** is the practice of writing compound words or phrases where the words are joined without spaces, and each word is capitalized within the compound. The name comes from the uppercase "bumps" in the middle of the compound word, suggestive of the humps of a camel.

A road sign with CamelCase

This practice is known by a large variety of names, including **camelBack**, **BiCapitalization**, **InterCaps**, **MixedCase**, etc., and many of its users do not ascribe a name to it at all.

CamelCase is a standard identifier naming convention for several programming languages, and has become fashionable in marketing for names of products and companies. Outside these contexts, however, CamelCase is rarely used in formal written English, and most style guides recommend against it.

Variations and synonyms

There are two common varieties of CamelCase, distinguished by their handling of the initial letter of what would otherwise be the first of separate words. Where the first letter is capitalized is commonly called **UpperCamelCase**, **PascalCase** (references: WikiWikiWeb, Brad Abrams), **BiCapitalized**, or **WalkingCamel** (in reference to the position of a camel's head when it is walking). Where the first letter is left in lowercase is commonly called **lowerCamelCase**. This variant has also been occasionally called **camelBack**, **dromedaryCase**, **drinkingCamel** (in reference to the position of a camel's head when it is drinking), or simply **camelCase**. For clarity, this article will use the terms **UpperCamelCase** and **lowerCamelCase**, respectively.

```
camelCaseLooksLikeThis
lowerCamelCaseLooksTheSame
UpperCamelCaseLooksLikeThis
```

The term *StudlyCaps* is similar — but not necessarily identical — to CamelCase. It is sometimes used in reference to CamelCase but can also refer to random mixed capitalization (as in *"MiXeD CaPitALiZaTioN"*) as popularly used in online culture.

Other synonyms include:

- **camelBack**
- **BumpyCaps**
- **BumpyCase**
- **camelBase Case**
- **CamelCaps**
- **CamelHumpedWord**
- **CapWords** in Python (reference)
- **mixedCase** (for lowerCamelCase) in Python (reference)
- **CICI** (Capital-lower Capital-lower) and sometimes **CIC**
- **HumpBackNotation**
- **InterCaps**
- **InternalCapitalization**
- **NerdCaps**
- **WordMixing**
- **WordsStrungTogether *or* WordsRunTogether**

The name *CamelCase* is not related to the "Camel book" (*Programming Perl*), which uses all-lowercase identifiers with underscores in its sample code.

Review Questions

1. How can you tell that a program might require loops?

2. What is the difference between a ***local variable*** and a ***global variable***? What is a "**Static**" local variable?

3. Explain why it is wise to avoid global variables whenever possible.

4. Describe a situation in programming that makes the use of global variables necessary.

5. Explain the terms "design-time," "run-time" and "compile-time."

6. Explain the rules of indentation. Why is proper indentation so important to the software development process?

7. When is it appropriate to use comments? Why is it a bad idea to omit comments altogether? Is it ever possible to include too many comments?

8. Suppose that you notice that the same or similar code is used in several places throughout a program. What would you do to make the program far more streamlined?

Run-Time Error Handling

Two Different Methods of Correcting the Bugs in Time Converter 1.0 Alpha

The Code for Version 1.0 Beta

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' PROGRAMMER'S NAME: Nick E. Nolfi      VERSION: Time Converter Version 1.0 Beta
'
' PURPOSE OF PROGRAM: Convert a time given in seconds to the format hours : minutes : seconds (h:m:s).
'
' LIMITATIONS and BUGS
' This version corrects the bugs in Version 1.0.  Now any error, including any user
' input errors, are handled (which prevents this program from crashing).
'
' NOTE
' It is a good idea to set the "MaxLength" property of the "txtSeconds" text box to 10.  This
' prevents the user from entering more than 10 digits (2^31 – 1 = 2147483647, which is 10 digits long).
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

Const CtrlC=3, CtrlV=22, CtrlX=24
Private Sub cmdClose_Click()
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                                              vbYesNo + vbDefaultButton2, "Leaving so soon?")

    If Response = vbYes Then
        Unload Me
        End
    End If
End Sub

'Convert a time specified in seconds to the format hours:minutes:seconds.
Private Sub cmdConvert_Click()

    On Error GoTo ErrorHandler

    'Memory
    Dim SecondsRemaining As Long, Hours As Long, Minutes As Byte

    'Input
    SecondsRemaining = Val(txtSeconds.Text)

    If SecondsRemaining >= 0 Then
        'Processing
        Hours = SecondsRemaining \ 3600
        SecondsRemaining = SecondsRemaining Mod 3600
        Minutes = SecondsRemaining \ 60
        SecondsRemaining = SecondsRemaining Mod 60
        'Output
        lblHoursMinutesSeconds.Caption = CStr(Hours) & " : " & _
                                         CStr(Minutes) & " : " & _
                                         CStr(SecondsRemaining)
    Else
        MsgBox "You must enter a positive value." _
                                         , vbExclamation, "Oops!"
    End If

    Exit Sub
ErrorHandler:
    If Err.Number = 6 Then
       MsgBox "The number you have entered is too large.", vbExclamation, "Oops!"
    Else
       MsgBox "An unexpected error has occurred: " & Err.Description & ". Error Number: " _
                                              & Err.Number, vbCritical, "What happened?"
    End If
End Sub
```

What are the words shown in blue boldface called? Why?

This means that if a ***run-time error*** like "overflow" occurs, the VB programming environment will not take control of your program, halt its execution and display an error message.

Instead, the program will branch to the lines of code labelled "ErrorHandler." In other words, the program itself intercepts the error and generates its own error messages.

Note that "ErrorHandler" is a name chosen by the programmer. It is not a VB keyword.

Why is this **"Exit Sub"** statement needed?

How do we know that the "overflow" error is error number 6? What is "Err?"

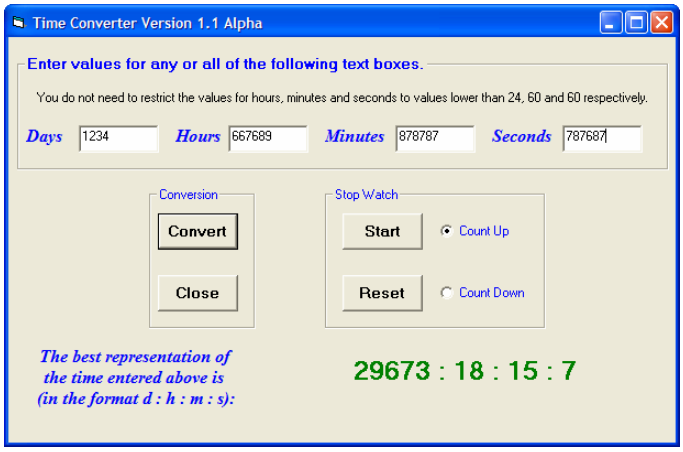
The Code for Version 1.0 Final Release

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' PROGRAMMER'S NAME: Nick E. Nolfi      VERSION: Time Converter Version 1.0 Final Release
'
' PURPOSE OF PROGRAM: Convert a time given in seconds to the format hours : minutes : seconds
'
' LIMITATIONS and BUGS
' This version corrects the bugs in Version 1.0 Alpha.  Now any error, including any user input
' errors, are trapped, which prevents crashing. Note that this version is an improvement over
' version 1.0 Beta because the input in the text box is restricted to the digits 0 – 9
'
' NOTE
' It is a good idea to set the "MaxLength" property of the "txtSeconds" text box to 10.  This stops
' the user from entering more than 10 digits (2^31 – 1 = 2147483647, which is 10 digits long).
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit
Const CtrlC=3, CtrlV=22, CtrlX=24
Private Sub cmdClose_Click()
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                                vbYesNo + vbDefaultButton2 + vbQuestion, "Leaving so soon?")
    If Response = vbYes Then
        End
    End If
End Sub
'Convert a time specified in seconds to the format hours:minutes:seconds.
Private Sub cmdConvert_Click()
    On Error GoTo ErrorHandler
    'Memory
    Dim SecondsRemaining As Long, Hours As Long, Minutes As Byte
    'Input
    SecondsRemaining = Val(txtSeconds.Text)
    'Processing
    Hours = SecondsRemaining \ 3600
    SecondsRemaining = SecondsRemaining Mod 3600
    Minutes = SecondsRemaining \ 60
    SecondsRemaining = SecondsRemaining Mod 60
    'Output
    lblHoursMinutesSeconds.Caption = CStr(Hours) & " : " & _
                                     CStr(Minutes) & " : " & _
                                     CStr(SecondsRemaining)
    Exit Sub
ErrorHandler:
    If Err.Number = 6 Then
       MsgBox "The number you have entered is too large.", vbExclamation, "Oops!"
    Else
       MsgBox "An unexpected error has occurred: " & Err.Description & ". Error Number: " & _
                                                  Err.Number, vbCritical, "What happened?"
    End If
End Sub

' Reject any characters typed in the "txtSeconds" text box that do not lie between 0 and 9, except
'for the backspace key.
Private Sub txtSeconds_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack
                     And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
```

Why is it not necessary in this version to use an "**If**" statement that checks if a negative number has been entered? (refers to `SecondsRemaining = Val(txtSeconds.Text)`)

What is the purpose of this line of code? (refers to `KeyAscii = 0`)

Time Converter Version 1.1 Alpha




```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'
' PROGRAMMER'S NAME: Nick E. Nolfi
' VERSION: Time Converter Version 1.1 Alpha
'
'LIMITATIONS and BUGS
' This version correctly handles the conversion to
' the format d:h:m:s but the stop watch has not
' yet been implemented.
'
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit
Const SecsInMin = 60, SecsInHour = 3600
Const SecsInDay = 86400, HoursInDay = 24
Const MinsInDay = 1440, MinsInHour = 60
Const CtrlC=3, CtrlV=22, CtrlX=24

'Convert a time specified in d:h:m:s to the best d:h:m:s representation
Private Sub cmdConvert_Click()

    On Error GoTo ErrorHandler

    'MEMORY
    Dim Seconds As Long, Hours As Long
    Dim Days As Long, Minutes As Long

    'INPUT
    Seconds = Val(txtSeconds.Text)
    Minutes = Val(txtMinutes.Text)
    Hours = Val(txtHours.Text)
    Days = Val(txtDays.Text)

    'PROCESSING: Convert to d:h:m:s and find final value of 'Seconds'
    Days = Days + Seconds \ SecsInDay
    Seconds = Seconds Mod SecsInDay
    Hours = Hours + Seconds \ SecsInHour
    Seconds = Seconds Mod SecsInHour
    Minutes = Minutes + Seconds \ SecsInMin
    Seconds = Seconds Mod SecsInMin
    'Convert to d:h:m and find final value of 'Minutes'
    Days = Days + Minutes \ MinsInDay
    Minutes = Minutes Mod MinsInDay
    Hours = Hours + Minutes \ MinsInHour
    Minutes = Minutes Mod MinsInHour
    'Convert to d:h and find final values of 'Hours' and 'Days'
    Days = Days + Hours \ HoursInDay
    Hours = Hours Mod HoursInDay
    'OUTPUT
    lblTime.Caption = CStr(Days) & " : " & CStr(Hours) & _
                      " : " & CStr(Minutes) & " : " & CStr(Seconds)

    Exit Sub

ErrorHandler:
    If Err.Number = 6 Then
       MsgBox "The number you have entered is too large.", vbExclamation, "Oops!"
    Else
       MsgBox "An unexpected error has occurred: " & Err.Description & ". Error Number: " & _
                                                    Err.Number, vbCritical, "What happened?"
    End If
End Sub

'The code is continued on the next page
```

- These statements are called ***constant declarations***.
- ***Constant identifiers*** are just like ***variable identifiers*** except that the value of a constant is not allowed to change. Attempting to change the value of a constant generates a ***run-time*** error.
- Constant identifiers make programs ***easier to understand***.
- Constant identifiers make programs ***easier to modify*** (change).

Explain the purpose of this statement.

```
Private Sub cmdClose_Click()
    'Generate the "Unload" event (to be intercepted by "Form_Unload")
    Unload Me
End Sub

'Intercept the unloading of the form to prevent the user from accidentally quitting.  This sub
'procedure is invoked (called into action) whenever the "Close" button or the "X" (top right hand
' corner of form) is clicked.  This happens because both actions generate the "Unload" event.
Private Sub Form_Unload(Cancel As Integer)
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                      vbYesNo + vbDefaultButton2, "Leaving so soon?")
    If Response = vbYes Then
        End
    Else
        Cancel = 1  'Set "Cancel" to any non-zero value to cancel the unloading of the form.
    End If
End Sub

' Invalid character rejection subroutines.
Private Sub txtSeconds_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
Private Sub txtMinutes_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
Private Sub txtHours_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
Private Sub txtDays_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
```

Questions

1. Explain why the “cmdClose_Click” sub procedure contains only the statement “Unload Me.”

2. Explain the purpose of the “Form_Unload” sub procedure.

Time Converter Version 1.1 Beta

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' PROGRAMMER'S NAME: Nick E. Nolfi                    VERSION: Time Converter Version 1.1 Beta
'
' LIMITATIONS and BUGS
' This version correctly handles the conversion to the format d:h:m:s.  The stop watch appears to work correctly
' but, depending on the speed of the processor and the number of tasks running, the stop watch loses anywhere
' from a few to several minutes per hour. This is due to the fact that with an interval of 1000 ms, the
' processor receives a request to execute the code in the "tmrStopWatch" sub procedure every 1000 ms (1 s). If
' the processor is busy executing code that cannot be interrupted, the execution of "tmrStopWatch" is delayed.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

Const SecsInMin = 60, SecsInHour = 3600, SecsInDay = 86400, CtrlX = 24
Const MinsInDay = 1440, MinsInHour = 60, HoursInDay = 24, CtrlC = 3, CtrlV = 22

'The variables below must be declared globally.  (Why?)
Dim Seconds As Long, Hours As Long, Days As Long, Minutes As Long

'Start or stop the timer
Private Sub cmdStartStop_Click()
    If Trim(LCase(cmdStartStop.Caption)) = "start" Then
        tmrStopWatch.Enabled = True
        cmdStartStop.Caption = "Stop"
    Else
        tmrStopWatch.Enabled = False
        cmdStartStop.Caption = "Start"
    End If
End Sub

'Convert a time specified in d:h:m:s to the best d:h:m:s representation
Private Sub cmdConvert_Click()
    On Error GoTo ErrorHandler
    'INPUT
    Seconds = Val(txtSeconds.Text)
    Minutes = Val(txtMinutes.Text)
    Hours = Val(txtHours.Text)
    Days = Val(txtDays.Text)
    'PROCESSING: Convert to d:h:m:s and find final value of 'Seconds'
    Days = Days + Seconds \ SecsInDay
    Seconds = Seconds Mod SecsInDay
    Hours = Hours + Seconds \ SecsInHour
    Seconds = Seconds Mod SecsInHour
    Minutes = Minutes + Seconds \ SecsInMin
    Seconds = Seconds Mod SecsInMin
    'Convert to d:h:m and find final value of 'Minutes'
    Days = Days + Minutes \ MinsInDay
    Minutes = Minutes Mod MinsInDay
    Hours = Hours + Minutes \ MinsInHour
    Minutes = Minutes Mod MinsInHour
    'Convert to d:h and find final values of 'Hours' and 'Days'
    Days = Days + Hours \ HoursInDay
    Hours = Hours Mod HoursInDay
    'OUTPUT
    lblTime.Caption = CStr(Days) & " : " & CStr(Hours) & " : " & CStr(Minutes) & _
                                                                    " : " & CStr(Seconds)
    Exit Sub
ErrorHandler:
    If Err.Number = 6 Then
       MsgBox "The number you have entered is too large.", vbExclamation, "Oops!"
    Else
       MsgBox "An unexpected error has occurred: " & Err.Description & ". Error Number: " & Err.Number, _
                                                   vbCritical, "What happened?"
    End If
End Sub

Private Sub cmdClose_Click()
    'Generate the "Unload" event (to be intercepted by "Form_Unload")
    Unload Me
End Sub
```

```
' Intercept the unloading of the form to prevent the user from inadvertently quitting.
Private Sub Form_Unload(Cancel As Integer)
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                              vbYesNo + vbDefaultButton2 + vbQuestion, "Leaving so soon?")
    If Response = vbYes Then
        End
    Else
        'Set "Cancel" to any non-zero value to cancel the unloading of the form.
        Cancel = 1
    End If
End Sub

'This sub is automatically executed every 1000 ms once tmrStopWatch.Enabled is set to "True"
Private Sub tmrStopWatch_Timer()
    Dim UpOrDownOne As Integer

    '"UpOrDownOne" equals either 1 or -1 depending on whether "Count Up" or "Count Down" is chosen.
    UpOrDownOne = optCountUp.Value * (-1) + optCountDown.Value * 1
    Seconds = Seconds + UpOrDownOne

    If Seconds = SecsInMin Or Seconds = -1 Then
        Seconds = Seconds Mod SecsInMin - (Seconds = -1) * SecsInMin
        Minutes = Minutes + UpOrDownOne
        If Minutes = MinsInHour Or Minutes = -1 Then
            Minutes = Minutes Mod MinsInHour - (Minutes = -1) * MinsInHour
            Hours = Hours + UpOrDownOne
            If Hours = HoursInDay Or Hours = -1 Then
                Hours = Hours Mod HoursInDay - (Hours = -1) * HoursInDay
                Days = Days + UpOrDownOne
            End If
        End If
    End If

    'OUTPUT
    lblTime.Caption = CStr(Days) & " : " & CStr(Hours) & " : " & CStr(Minutes) _
                                                              & " : " & CStr(Seconds)
End Sub

' Invalid character rejection subroutines.
Private Sub txtSeconds_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub

Private Sub txtMinutes_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub

Private Sub txtHours_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub

Private Sub txtDays_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
                        And KeyAscii <> CtrlC And KeyAscii <> CtrlX And KeyAscii <> CtrlV Then
        KeyAscii = 0
    End If
End Sub
```

Questions

1. The code for the "tmrStopWatch_Timer" sub procedure is quite compact but it seems a little difficult to understand. Compare the code in version 1.1 Beta to the following alternative way of writing the code.

```
Private Sub tmrStopWatch_Timer()
    If optCountUp.Value = True Then
        Seconds = Seconds + 1
        If Seconds = SecsInMin Then
            Seconds = 0
            Minutes = Minutes + 1
            If Minutes = MinsInHour Then
                Minutes = 0
                Hours = Hours + 1
                If Hours = HoursInDay Then
                    Hours = 0
                    Days = Days + 1
                End If
            End If
        End If
    Else
        Seconds = Seconds - 1
        If Seconds = -1 Then
            Seconds = 59
            Minutes = Minutes - 1
            If Minutes = -1 Then
                Minutes = 59
                Hours = Hours -1
                If Hours = -1 Then
                    Hours = 23
                    Days = Days -1
                End If
            End If
        End If
    End If
    'OUTPUT
    lblTime.Caption = CStr(Days) & " : " & _
        CStr(Hours) & " : " & _
        CStr(Minutes) & " : " & CStr(Seconds)
End Sub
```

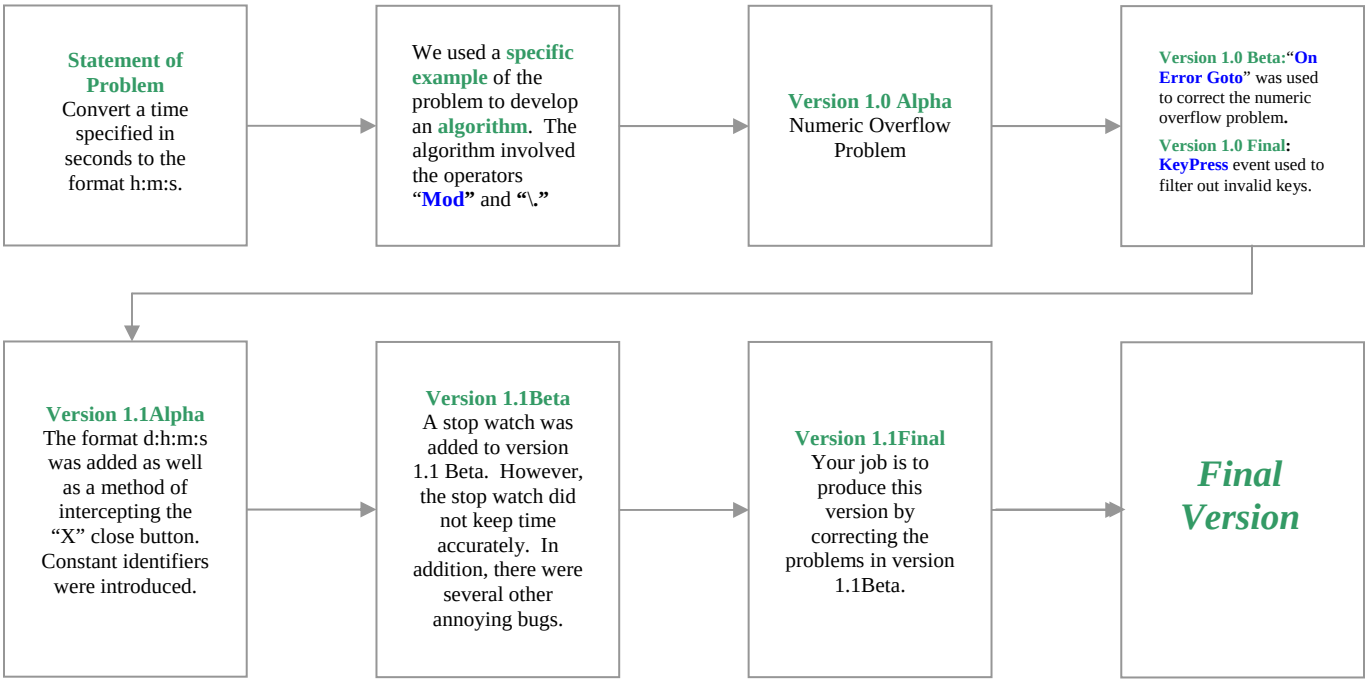
Strengths of this Version	*Weaknesses of this Version*
Strengths of Version 1.1 Beta	*Weaknesses of Version 1.1 Beta*

2. Thoroughly test version 1.1 Beta (you will find it in the usual place on the I: drive). Then complete the following table.

Bugs found in Version 1.1 Beta	*Improvements that are Required for Version 1.1 Final Release*

Creating the Final Version of Time Converter

Brief Summary of the Evolution of the Time Converter Program



Your Assignment

1. Use a word processor to create and complete a table that looks just like the following:

New Concepts Learned while Developing the Time Converter Program	*Explanations of New Concepts*	*Examples Involving the New Concepts*
(You supply the details.)	(You supply the details.)	(You supply the details.)

2. Use a word processor to create a list of all the bugs in version 1.1 Beta. Also, include a list of improvements that should be made to the program.

Bugs (Include the Cause of Each Bug)	*Improvements to be Made in Version 1.1 Final Release*
(You supply the details.)	(You supply the details.)

3. Create version "1.1 Final Release" of the "Time Converter" program. Incorporate all the bug fixes and improvements that you listed in question two.

Evaluation Guide for Time Converter 1.1 Final Release

Categories	Criteria	Descriptors					Level	Average
		Level 4	Level 3	Level 2	Level 1	Level 0		
Knowledge and Understanding (KU)	**Understanding of Programming Concepts**	Extensive	Good	Moderate	Minimal	Insufficient		
	Understanding of the Problem	Extensive	Good	Moderate	Minimal	Insufficient		
Application (APP)	**Correctness** To what degree is the output correct?	Very High	High	Moderate	Minimal	Insufficient		
	Declaration of Variables To what degree are the variables declared with appropriate data types?	Very High	High	Moderate	Minimal	Insufficient		
	Debugging To what degree has the student employed a logical, thorough and organized debugging method?	Very High	High	Moderate	Minimal	Insufficient		
Thinking, Inquiry and Problem Solving (TIPS)	**Degree of Improvement over Version 1.1 Beta** To what degree has the student incorporated significant improvements to Time Converter 1.1 Beta?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Design and Select Algorithms Independently To what degree has the student been able to design and select algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Implement Algorithms Independently To what degree is the student able to implement chosen algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Efficiency of Algorithms and Implementation To what degree does the algorithm use resources (memory, processor time, etc) efficiently?	Very High	High	Moderate	Minimal	Insufficient		
Communication (COM)	**Indentation of Code** **Insertion of Blank Lines in Strategic Places** (to make code easier to read)	Very Few or no Errors	A Few Minor Errors	Moderate Number of Errors	Large Number of Errors	Very Large Number of Errors		
	Comments • Effectiveness of explaining abstruse (difficult-to-understand) code • Effectiveness of introducing major blocks of code • Avoidance of comments for self-explanatory code	Very High	High	Moderate	Minimal	Insufficient		
	Descriptiveness of Identifier Names Variables, Constants, Objects, Functions, Subs, etc **Inclusion of Property Names with Object Names** (**e.g.** 'txtName.Text' instead of 'txtName' alone) **Clarity of Code** How easy is it to understand, modify and debug the code? **Adherence to Naming Conventions** (**e.g.** use "txt" for text boxes, "lbl" for labels, etc.)	Masterful	Good	Adequate	Passable	Insufficient		
	User Interface To what degree is the user interface well designed, logical, attractive and user-friendly?	Very High	High	Moderate	Minimal	Insufficient		

Counted Loops and Conditional Loops in VB

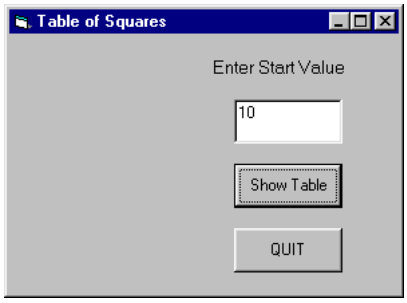
Conditional Loop Example	*Counted Loop Example*
Stir Coffee Until the Sugar has Fully Dissolved ' The following is not real VB. It is called "pseudo-code" which ' means false code. It is a mixture of VB and English and is a ' useful method for planning the overall structure of your ' programs. **Do** keep stirring **Loop Until** Sugar is Dissolved **Note** The number of repetitions of the code in this loop is dependent upon how long the sugar takes to dissolve. The number of repetitions is impossible to **predict**. You, as a programmer, will not in general be able to determine beforehand the number of repetitions of a conditional loop.	**Add Three Spoonfuls of Sugar to the Coffee** ' The following is not real VB. It is called "pseudo-code" which ' means false code. It is a mixture of VB and English and is a ' useful method for planning the overall structure of your ' programs. **For** I=1 **To** 3 add one spoonful of sugar **Next** I **Note** In this example, the number of repetitions is exactly three. This is easy to predict in advance because we know that the initial value of "I" is 1. After the first repetition, "I" becomes 2, after the second repetition, "I" becomes 3 and after the third repetition, the loop halts.

Various Conditional Loop Structures in Visual Basic			
Repeat Zero or More Times		***Repeat At Least Once***	

`Do While condition`
`  statements`
`Loop`
	`Do Until condition`
	`  statements`
	`Loop`
		`Do`
		`  statements`
		`Loop While condition`
			`Do`
			`  statements`
			`Loop Until condition`

Counted Loops in VB – "For…Next" Loops

The following program produces a TABLE of SQUARES. The user types in a START value and then clicks the "Show Table" button. A table of 10 values is printed on the form.




```
Private Sub cmdShow_Click()
    Dim I As Integer, Start As Integer, ISquared As Integer
    Start = Val(txtStart.Text)
    Me.Cls    'Clear the form
    For I = Start To Start + 9
        ISquared = I ^ 2
        Print I, ISquared    'I and its square are printed
    Next I
End Sub
```

The instruction "`For I = Start To Start + 9`" means that the counter variable "I" should have an initial value of "`Start`" and a final value of "`Start + 9.`"

The instruction "`Next I`" means that the value of "I" should be increased by 1 and that the loop should continue to repeat the group of statements enclosed between "**For**" and "**Next**."

ALWAYS DECLARE THE LOOP COUNTER VARIABLE AS A LOCAL VARIABLE!

"For…Next" (Counted Loop) Exercises

1. Given the ***variables*** shown below, describe ***in words*** what will happen when each of the ***program segments*** below is executed. Show a ***trace chart*** (memory map) for the variables used. (The first one is done for you.) ***Check your answers by using Visual Basic and break points!***

Size	Count	Score	Sum
11217	10	142	0

```
For X = 1 To 5
   Sum = Sum + X
   Count = Count + 1
   Print Sum
Next X
Print Count
```

Values Before Entering Loop →

Values After Exiting Loop →

X	Sum	Count
0	0	10
1	1	11
2	3	12
3	6	13
4	10	14
5	15	15
6	15	15

Output

```
1
3
6
10
15
15
```

```
For N = Count To 0 Step -2
   Sum = Size Mod Count
   Score = Score + Sum
   Size = Size - Sum
Next N
txtNum1.Text = CStr(Score)
txtNum2.Text = CStr(Size)
```

```
For N = 7 To Count
   Sum = Score Mod N
   If Sum = 0 Then
      Print Score
   End If
   Score = Score + Sum
Next N
```

```
For Num = 1 To Count
   Sum = Sum + Num
   Score = Score – Sum
   If Score Mod 2 = 0 Then
      Print "Even Score"
   Else
      Print "Odd Score"
   End If
Next Num
```

```
For X = 0 To 8 Step 2
   Score = Score – 1
   Sum = Score + X
   Print Sum
Next X
Print Score
```

2. Write a "**For...Next**" loop to perform each of the following tasks:

a) Print your name 20 times on the form

b) Add up the numbers from 1 to 50

c) Add up all even numbers less than 100

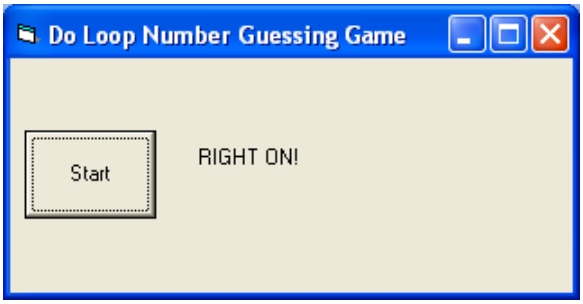
d) Print the sequence of squares, "1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256" on the form.

3. Design and code a program that computes the ***sum*** from `Lower` to `Upper`, where `Lower` and `Upper` are variables that store two numbers entered by the user into text boxes (`Lower` ≤ `Upper`). The result should be displayed in a label box.

"Do ... While" and "Do ... Until" Loop structures (Conditional Loops)

This Game demonstrates a "**Do … Loop Until**" loop structure.

Once the "Start" button is clicked, the player keeps entering guesses **Until** the entered guess is equal to the secret number.




```
'To try out this program, just load it from the I:drive
'I:\Out\Nolfi\Ics3mo\Do Loop Guessing Game With Multiple Forms
Option Explicit
Private Sub Form_Load()
    Randomize
End Sub
Private Sub cmdStart_Click()
    Dim Guess As Byte, SecretNumber As Byte
    SecretNumber = Int(100 * Rnd + 1)
    lblClue.Caption = ""
    Do 'Beginning of loop
        Guess = Val(InputBox("Enter a Guess", ""))
        If Guess > SecretNumber Then
            lblClue.Caption = "Too High!"
        ElseIf Guess < SecretNumber Then
            lblClue.Caption = "Too Low!"
        End If
    Loop Until Guess = SecretNumber 'End of loop
    lblClue.Caption = "RIGHT ON!"
End Sub
```

Questions

1. What is the purpose of the "Randomize" statement? Why is it used within a "Form_Load" sub procedure? Why would it be wasteful to include the "Randomize" statement in the "cmdStart_Click" sub procedure?

2. Why is it possible in this program to declare both "SecretNumber" and "Guess" as local (procedure level) variables. Why is it not necessary to use any global (module level) variables?

3. Why is the "Val" function used in conjunction with the "InputBox" function? What could go wrong if "Val" were omitted?

Here are the various **LOOP PATTERNS** you can use. The four variations shown below differ in subtle ways.

Do [*statements*] [**Exit Do**] [*statements*] **Loop Until** *condition*	**Do** [*statements*] [**Exit Do**] [*statements*] **Loop While** *condition*	**Do Until** *condition* [*statements*] [**Exit Do**] [*statements*] **Loop**	**Do While** *condition* [*statements*] [**Exit Do**] [*statements*] **Loop**

Question

Explain the subtle differences in the four loop structures shown above.

Examples

Example of **Adding Up a List of Numbers** using a ***Flag Variable*** to Signal the End of the Input

```
Sum = 0
Do
    Num = InputBox("Enter a Number","")
    Sum = Sum + Num
Loop Until Num = 0
```

Pseudo-Code

Initialize sum to 0
Begin the loop
    Get user input
    Add "Num" to "Sum"
Stop if user enters a ZERO

To use the same method to **Average** a list, we need to **Count** the number of inputs.

```
Sum = 0
NumEntries = 0
Do
   Num = InputBox("Enter a Number","")
   Sum = Sum + Num
   NumEntries = NumEntries + 1
Loop Until Num = 0
NumEntries = NumEntries – 1
Average = Sum / NumEntries
```

Pseudo-Code

Initialize *Sum* and *Count* to zero
Begin the loop
    Set *Num* to user input
    Add *Num* to *Sum*
    Add 1 to *NumEntries*
Stop if user enters a ZERO (when *Num*=0)
Remove zero from the count
Calculate *Average*

Example of **Rolling 2 Dice** until the Roll is a Seven

```
Do
    Die1 = Int(Rnd * 6) + 1
    Die2 = Int(Rnd * 6) + 1
    Roll = Die1 + Die2
Loop Until Roll = 7
```

Pseudo-Code

Begin the loop
    Roll first die
    Roll second die
    Add their values
Stop if *Roll* = 7

Example to find the **Smallest Divisor** (other than 1) of a Number

```
Num = InputBox("Enter a Number","")
SmallestDivisor = 1
Do
    SmallestDivisor = SmallestDivisor + 1
    Remainder = Num Mod SmallestDivisor
Loop Until Remainder = 0 Or SmallestDivisor = Num
If Remainder = 0 Then
    Print SmallestDivisor
Else
    Print "The number is prime."
End If
```

Pseudo-Code

User enters *number*
Set *smallestDivisor* to 1
Begin loop
    Add 1 to *smallestDivisor*
    Set *remainder* to *number* mod *smallestDivisor*
Stop if *remainder* =0 or *smallestDivisor* = *number*
If *remainder* =0
    Print *smallestDivisor*
Else
    Print "Number is prime"

"Do…Loop" (Conditional Loop) Exercises

1. Each of the following ***loops*** uses an "InputBox" to get data from the ***user***. In each question, a sample of user data is given. Create a trace chart (memory map) for each code segment and show the ***exact*** output.

User Data: 65, 54, 70, 68, 52, 81, 75, 0

```
Biggest = 0
Do
    Num = InputBox("Enter a Number","")
    If Num > Biggest Then
        Biggest = Num
    End if
Loop Until Num=0
Print Biggest
```

User Data: 65, 54, 70, 68, 52, 81, 75, 0

```
Sum = 0
Do
    Num = InputBox("Enter a Number","")
    Unit = Num Mod 10
    Sum = Sum + Unit
Loop Until Num=0
Print Sum
```

User Data: 23, 12, 5, 34, 88, 15, 120, 25

```
Count = 0
Do
    Num = InputBox("Enter a Number","")
    If Num >=15 Then
        Count = Count + 1
        Print Num;
    End If
Loop While Count < 100
Print "****"; Count
```

User Data: 22, 11, 5, 12, 4, 33, 16, 9, 3

```
Sum = 0
Do While Sum >= 0
    Num = InputBox("Enter a Number","")
    If Num Mod 2 = 0 Then
        Sum = Sum + Num
    Else
        Sum = Sum – Num
    End If
Loop
Print Sum
```

2. Write a "**Do…Loop**" to perform each of the following tasks:

- **a)** Add up the numbers 1 + 2 + 3 + 4 + ... until the **Sum** > **100**.
- **b)** Determine how many numbers 2 +4 + 6 + 8 + ... are needed to give a **Sum** > **1000**.
- **c)** Output all powers of 2 (i.e. 1, 2, 4, 8, 16, 32, …) that are less than 1000000.
- **d)** Output the smallest number (other than 1) that divides evenly into 2701.

3. Design and Code the programs described below.

- **a)** The USER enters numbers using an Input Box. A zero is used to flag the final input. The computer then uses a Label to show the AVERAGE of the Highest and Lowest numbers that were entered.

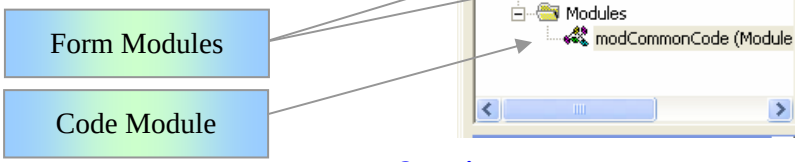
- **b)** The USER enters numbers using an Input Box. A zero is used to flag the final input. The computer then uses a Label Box to state whether the Even or Odd numbers had the largest total.

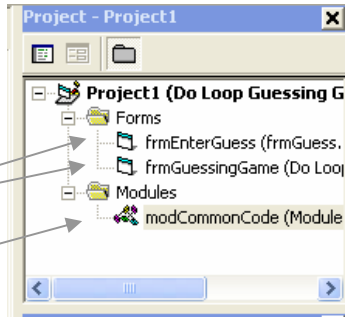
An Enhanced Version of the Do Loop Guessing Game

Note: you will find a copy of this program in the folder

I:\Out\Nolfi\Ics3m0\Do Loop Guessing Game with Multiple Forms

You are probably accustomed to writing Visual Basic programs with only ***one*** form. The enhanced version of the "do loop guessing game is an example of a program with ***two*** forms (more precisely, "form modules") and one code module.






```
'This is an example of a "Code Module ("modCommonCode").  It is used as
'a "storage area" for code that is required by 2 or more "Form Modules."
'Unlike form modules, code modules are not associated with any objects.
Option Explicit
Public Guess As Integer
Public GiveUp As Boolean, ValidGuess As Boolean
```

Questions

1. What is the purpose of the code stored in the ***code module*** "modCommonCode?"

```
'This program is designed to help you understand the following: Using
'multiple forms, using code modules to store code that is common to
'multiple forms, using "application modal" forms, using the enter
'key to signify the end of input.  (This form is "frmGuessingGame")
Option Explicit
Const ApplicationModal = 1

Private Sub Form_Load()
    Randomize
    Me.Show
    cmdStart.SetFocus
End Sub

Private Sub cmdStart_Click()
    Dim SecretNumber As Byte
    SecretNumber = Int(100 * Rnd + 1)
    lblClue.Caption = ""
    Do
        'The user's response is obtained from a different form.
        'This prevents the loop from becoming infinite.
        frmEnterGuess.Show (ApplicationModal)
        'The values of "GiveUp," " Guess" and "ValidGuess"
        'are assigned in the form "frmEnterGuess."
        If Not GiveUp And ValidGuess Then
            If Guess > SecretNumber Then
                MsgBox "Too HIGH"
            ElseIf Guess < SecretNumber Then
                MsgBox "Too LOW"
            End If
        End If
    Loop Until Guess = SecretNumber Or GiveUp
    If Guess = SecretNumber Then
        lblClue.Caption = _
             "Right on! You got it! Click START to play again."
    Else
        lblClue.Caption = _
             "You gave up! The secret number is" & _
             Str(SecretNumber) & ". Click START to try again."
    End If
End Sub
```

2. How does an "application modal" form differ from a non-modal form?

3. Why is it necessary to use an application modal form in this program to receive input from the user

```
'This form ("frmEnterGuess") is used to obtain a guess from the
'user or to allow the user to give up.  It is an "application
'modal"form, which means that the application is suspended until
'the user responds to this form.

Option Explicit

'The "Activate" event occurs every time a
'form becomes the active window.

Private Sub Form_Activate()
    txtGuess.Text = ""
    txtGuess.SetFocus
End Sub

Private Sub cmdEnterGuess_Click()
    Guess = Val(txtGuess.Text)
    If Guess >= 1 And Guess <= 100 Then
        ValidGuess = True
    Else
        MsgBox _
          "Your guess must be a whole number between 1 and 100.", _
                                                    vbExclamation
        ValidGuess = False
    End If
    GiveUp = False
    txtGuess.Text = ""
    txtGuess.SetFocus
    Me.Hide
End Sub

Private Sub cmdGiveUp_Click()
    GiveUp = True
    Me.Hide
End Sub

'Prevent the user from entering any characters other
'than the digits from 0 to 9.  It also allows the pressing of
'the ENTER key to signify the end of the input.
Private Sub txtGuess_KeyPress(KeyAscii As Integer)
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And _
                                    KeyAscii <> vbKeyBack Then
        KeyAscii = 0
    End If
End Sub
```

4. What makes it possible to use the ENTER key to enter the guess (instead of clicking on the “Enter Guess” button)?

5. How is the maximum length of the user’s input limited to three characters?

6. Why are the digits 0 to 9 and the BACKSPACE accepted while all other keys are rejected?

7. What is the difference between the “Activate” event and the “Load” event?

Using Visual Basic to Produce String Art

The String Art Algorithm

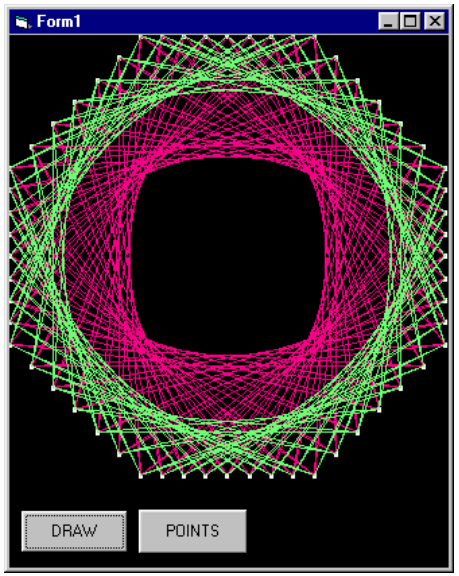
A set of N points is read in from a data file (or are defined from code) and connected according to the following *algorithm*. Note that the following *IS NOT* Visual Basic code! It is pseudo-code! Your job is to *translate* the pseudo-code into VB!

```
Initialize the values of A and B
Set A=1 and B=some value between
1 & N
loop
   join point A to point B
   add 1 to A
   join point B to point A
   add 1 to B
   if B > N
      set B=1
until A = N
```

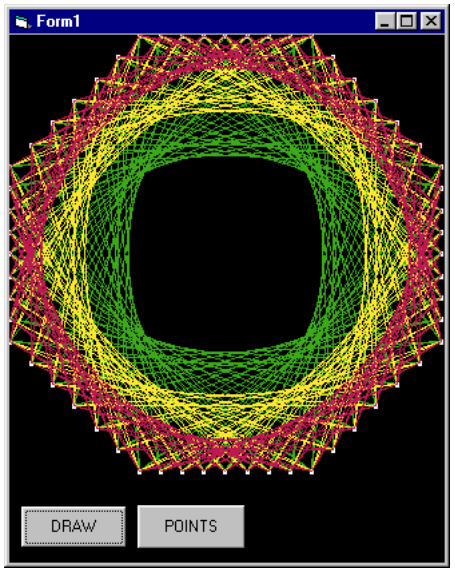
By changing the initial value of B (just before the loop) a different pattern can be produced.

Examples of String Art

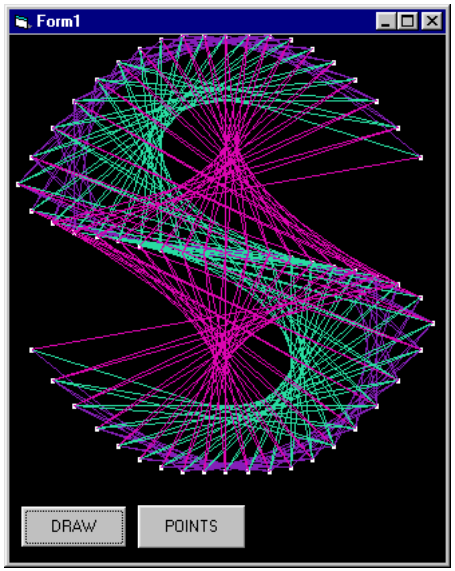
String Art Example 1



String Art Example 2



String Art Example 3



Exercises

1. How many points are used in string art example 1?

2. How many points are used in string art example 2?

3. How many points are used in string art example 3?

4. Explain the string art algorithm in plain English.

5. Write a VB program that can produce any string art given "N" points and an initial value of "B." Include a feature that allows the user to change the initial value of "B" and to select the colours used. Allow the user to select up to three colours.

Fractals

Fractal Geometry

Fractal geometry is the branch of mathematics that deals with producing extremely irregular curves or shapes for which any suitably chosen part is similar in shape to a given larger or smaller part when magnified or reduced to the same size (this property of fractals is known as ***self-similarity***.). A "picture" or "image" produced by a fractal geometry algorithm is usually called a ***fractal***. Fractal geometry is closely related to a branch of mathematics known as ***chaos theory***.

The Chaos Game

To gain a basic understanding of fractals, it is helpful to play a game called the ***chaos*** game. The game proceeds in its simplest form as follows. Place three dots at the vertices of any triangle. Colour the top vertex red, the lower left green and the lower right blue. Then take a die and colour two faces red, two green and two blue.

To play the game, you need a ***seed***, an arbitrary starting point in the plane. Starting with this point, the algorithm begins with a roll of the die. Then, depending upon which colour comes up, plot a point halfway between the seed and the appropriate coloured vertex. Repeat this process using the terminal point of the previous move as the seed for the next. ***To obtain the best possible results, do not plot the first 15 (or so) points generated by this algorithm! Only begin plotting after the first 15 points have been generated!***

For example, *Figure 1* shows the moves associated with rolling red, green, blue and blue in order.

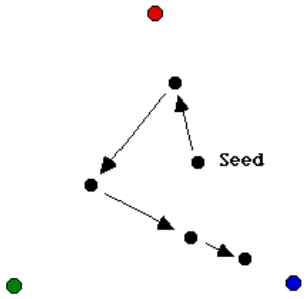
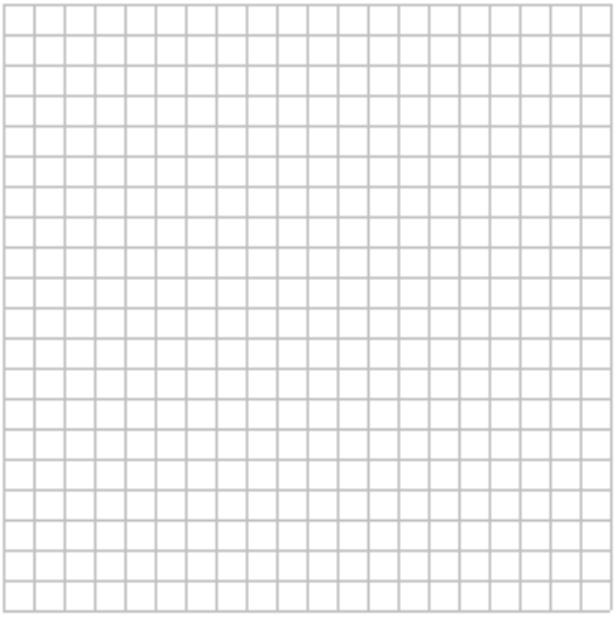


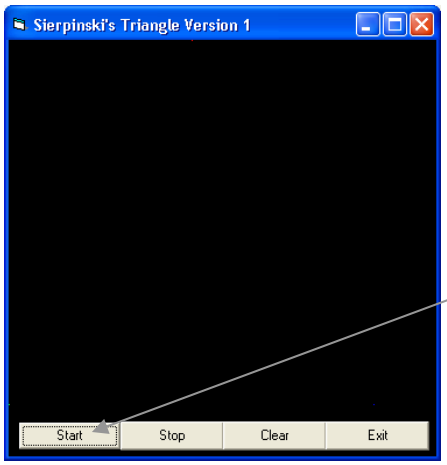
Figure1 Playing the chaos game with rolls of red, green, blue, blue.



People who have never played this game are always surprised and amazed at the result! Most expect the algorithm to yield a blur of points in the middle of the triangle. Some expect the moving point to fill the whole triangle. Surprisingly, however, the result is anything but a random mess. The resulting picture is one of the most famous of all fractals, the *Sierpinski triangle*.

The Sierpinski Triangle

Now to see the *Chaos Game* in action, run the program "Sierpinski's Triangle.vbp" in the folder **"I:\Out\Nolfi\Drawing, Graphics, Game Program Examples\Sierpinski's Triangle V1 and V2."**



You must be patient once you click on the "Start" button!

It takes a few minutes for this program to generate Sierpinski's triangle. However, it will be well worth the wait! You ***will*** be amazed by the figure generated by this seemingly random and chaotic algorithm!

Assignment (To be handed in)

1. Use the Internet (or whatever other resources that you wish to use) to find algorithms that produce the following fractals:

   a. Sierpinski Triangle (this one is easy because I have already given it to you)
   b. Sierpinski Pentagon
   c. Sierpinski Hexagon
   d. Sierpinski Carpet
   e. Koch Snowflakes
   f. Any other fractal that is not too difficult to code

   Then create a word processor document that gives a brief outline of each algorithm. Include diagrams to supplement the description of each algorithm.

2. Using a Web browser, load the Java applet with URL http://math.bu.edu/DYSYS/applets/fractalina.html. Experiment with this applet for a few minutes to familiarize yourself with its various features. Then write a Visual Basic program that is similar to the "Fractalina" applet. Your program must be able to generate the following fractals:

   a. Sierpinski Triangle
   b. Sierpinski Pentagon
   c. Sierpinski Hexagon
   d. Sierpinski Carpet
   e. Koch Snowflakes

   Note that your program ***need not have*** "New Point," "Kill Point" and "Zoom Out" buttons. However, your program ***should*** allow the user to drag the vertices of the shapes to different locations.

EUCLID AND THE GCD

Definition of GCD

By definition, the *Greatest Common Divisor* (GCD) of two positive integers is the largest integer that divides both integers exactly.

Brute Force (Slow) Method for Computing the GCD of Two Integers

You have already developed an algorithm for finding the GCD of two integers. Use the provided space to write a pseudo-code description of your algorithm. (See "Fraction Calculator" in unit 2.)

Description of Euclid's (Fast) Method for Computing the GCD of Two Integers

Background

More than 2000 years ago, Euclid published an algorithm for finding the GCD of two numbers. His version was strictly geometric since algebra had not been invented yet, but the algebraic version is described below.

Take any two positive integers a and b, with b smaller than a (i.e. $b < a$).

Euclid noted that there are integers *r* (the remainder) and *q* (the quotient) such that $a = qb + r$:
If b is divided into a, q is the quotient and r is the remainder.
(For example, if $a = 120$, $b = 25$, then $120 = 4(25) + 20$, which means that $q = 4$ and $r = 20$.)

Any common factor, *N*, of *b* and *r* divides *a* exactly:
If N divides b, it also divides qb. Since N divides r, it must also divide the sum, $qb + r$, which is a of course.
(Continuing the above example, $N = 5$ divides both $b = 25$ and $r = 20$. Therefore, $N = 5$ must also divide $qb + r = 4(25) + 20 = 120$ since $N = 5$ is a common factor of 4(25) and 20.)

Any common factor, *M*, of *a* and *b* divides *r* exactly:

Since $r = a - qb$, $\frac{r}{M} = \frac{a}{M} - \frac{qb}{M}$. Since $\frac{a}{M}$ and $\frac{qb}{M}$ are both integers, then their difference, $\frac{r}{M}$, must also be an integer.
Therefore, M divides r.
(M=5 divides both $a = 120$ and b =25. Therefore it must also divide $a - qb = 120 - 4(25) = 20$)

It follows that the largest *N* must equal the largest *M*. In other words, gcd(*a*,*b*) = gcd(*b*,*r*). Since b is less than a and r is less than b, we can repeat these steps substituting b for a and r for b until r becomes 0. The final step has $a = qb + 0$ and b is the desired GCD.

Summary

The Euclid algorithm can be expressed concisely by the following recursive formula:

gcd(*N*, *M*) = gcd(*M*, *N* mod *M*), where $M < N$.

Note: Please recall that N mod M means the remainder obtained when M is divided by N.

Example

Here is an example of Euclid's algorithm in action.

Find the GCD of 2322 and 654.

gcd(2322, 654) = gcd(654, 2322 mod 654) = gcd(654, 360)
gcd(654, 360) = gcd(360, 654 mod 360) = gcd(360, 294)
gcd(360, 294) = gcd(294, 360 mod 294) = gcd(294, 66)
gcd(294, 66) = gcd(66, 294 mod 66) = gcd(66, 30)
gcd(66, 30) = gcd(30, 66 mod 30) = gcd(30, 6)
gcd(30, 6) = gcd(6, 30 mod 6) = gcd(6, 0)
gcd(6, 0) = 6

Therefore, gcd(2322,654) = 6.

Your Task

1. Use Euclid's method to calculate gcd(4896, 432)

2. Use the provided space to write a pseudo-code description of the Euclid GCD algorithm. When you are finished, show the pseudo-code to me. Once I approve of your pseudo-code, you will write a VB function procedure that uses Euclid's algorithm to calculate the GCD of two numbers.

Test out the Euclid Algorithm

In the folder **I:\Out\Nolfi\Ics3mo\Euclid's GCD Algorithm** you will find two implementations of Euclid's algorithm. One of them is just a straight implementation of the algorithm. The other compares Euclid's algorithm to its slower counterpart from unit 1. Test both programs thoroughly. List your observations below.

Learning about Arrays and Nested Loops through the "Generating Random Integers without Repetition" Problem

Purpose: To learn about ***arrays*** and ***nested loops***

Motivation: The problem of generating random numbers without repetition

Statement of the Problem

Suppose that you wanted to write a program to generate random numbers for a lottery such as **Lotto 6/49®**. It is simple enough to write a **"For … Next"** loop that generates six random integers, but how would you prevent the computer from generating the ***same*** random number two or more times?

A Solution

Consider the following example of generating six random integers between 1 and 49 without repetition. A new random integer is generated with each iteration of the main loop. Whenever a previously generated integer appears, a new one must be generated to replace it. Pseudo-code for this algorithm is given below.

3	3	3	3	3	3	3	3	3
	17	17	17	17	17	17	17	17
		3	12	12	12	12	12	12
				14	14	14	14	14
					17	43	43	43
							14	45

A "3" has already been generated so a new random integer needs to be generated to replace it.

A "17" has already been generated so a new random integer needs to be generated to replace it.

A "14" has already been generated so a new random integer needs to be generated to replace it.

```
For I = 1 To 6
   Do
      Generate random integer
   Loop Until random integer has not already been generated
   Set random integer I to the new one just generated
Next I
```

Why Arrays are Necessary to implement the above Algorithm

An ***array*** is a structure that allows you to use ***a single name*** to refer to a group of two or more variables. To distinguish one variable in the group from another, a number, called the ***index*** or ***subscript***, is used. Arrays help you to create smaller and simpler code in many situations, because you can set up loops that deal efficiently with any number of cases.

Carefully study the following code. You can find a copy in the folder **I:\Out\Nolfi\Ics3mo\Space Versus Time\Lotto 649.**

```
  Private Sub cmdQuickPick_Click()
      Dim QuickPick(1 To 6) As Byte, RandomPick As
Byte

      Dim I As Byte, J As Byte, Repetition As Boolean
      For I = 1 To 6
          Do

              RandomPick = Int(Rnd * 49 + 1)

              Repetition = False
              For J = 1 To I - 1

                  If RandomPick = QuickPick(J) Then

                      Repetition = True

                      Exit For

                  End If

              Next J
          Loop Until Repetition = False
          QuickPick(I) = RandomPick
      Next I
```

"QuickPick" Array

Index	Data
1	3
2	17
3	12
4	14
5	43
6	25

QuickPick(1)
QuickPick(2)
QuickPick(3)
QuickPick(4)
QuickPick(5)
QuickPick(6)

Notice that each ***element*** (member) of the array has the ***same name*** but a ***different index*** (subscript).

```
    For I = 0 To 5
        lblQuickPick(I).Caption = QuickPick(I + 1)
    Next I
End Sub
```

General Facts about Arrays

- All the elements in an array usually have the same data type.
- Of course, when the data type is **Variant**, the individual elements may contain different kinds of data (objects, strings, numbers and so on). You can declare an array of any of the fundamental data types, including user-defined types and object variables.
- Because Visual Basic allocates space for each index number, avoid declaring an array larger than necessary.
- Arrays have both ***upper*** and ***lower*** bounds and the elements of the array are contiguous within those bounds.
- In Visual Basic, there are two types of arrays. ***Fixed-size arrays*** always remain the same size and ***dynamic arrays*** can be resized at run-time. Dynamic arrays will be discussed later.

Declaring Fixed-Size Arrays

There are three ways to declare a fixed-size array, depending on the scope you want the array to have:

- To create a ***public array***, use the **Public** statement in the "declarations" section of a module to declare the array.
- To create a ***module-level array***, use the **Private** or **Dim** statement in the "declarations" section of a module to declare the array.
- To create a ***local array***, use the **Private** or **Dim** statement in a procedure to declare the array.

Setting Upper and Lower Bounds

When declaring an array, follow the array name by the ***upper bound*** in parentheses. The upper bound cannot exceed the range of a Long data type (–2,147,483,648 to 2,147,483,647). For example, these array declarations can appear in the "declarations" section of a module:

```
Dim Counter(14) As Integer  '15 elements with indices ranging from 0 to 14.
Dim Sum(20) As Double       '21 elements with indices ranging from 0 to 19.
```

To create a public array, you simply use **Public** in place of **Dim** or **Private**.

```
Public Counter(14) As Integer
Public Sum(20) As Double
```

The first declaration creates an array with 15 elements, with subscripts running from 0 to 14. The second creates an array with 21 elements, with subscripts ranging from 0 to 20. ***The default lower bound is 0***.
To specify a lower bound, provide it explicitly (as a **Long** data type) using the **"To"** keyword:

```
Dim Counter(1 To 15) As Integer
Dim Sum(100 To 120) As String
```

Working with Arrays (Array Exercises)

1. Given the array **Num** with the values shown, fill in the remaining *parallel arrays* based on the code below. The arrays have been appropriately declared.

Array Index	Num	NumDoubled	NumMod2	Result	Answer	Total	Num1	Num2	Num3	Num4
1	25									
2	13									
3	34									
4	16									
5	9									

```
For M = 1 To 5
    NumDoubled(M) = 2* Num(M)
Next M
```

```
For Y = 1 To 5
    NumMod2(Y) = Num(Y) Mod 2
Next Y
```

```
For Z = 1 To 4 'be careful here
    Result(Z) = 3*Num(Z) – 1
Next Z
```

```
For P = 1 To 5
    Answer(P) = Num(P) + Num(P)
Next P
```

```
Sum = 0
For X = 1 to 5
    Sum = Sum + Num(X)
    Total (X) = Sum
Next X
```

```
For W = 1 To 4 'be careful here
    Num1(W) = Num(W+1)
Next W
```

```
For R = 2 To 5
    Num2(R) = Num(R-1)
Next R
```

```
For B = 1 To 4
    Num3(B) = Num(B) + Num(B+1)
Next B
```

```
M = 1
For D = 1 To 5
    If Num(D) > 15 Then
       Num4(M) = Num(D)
       M = M + 1
    End If
Next D
```

2. Write simple **For…Next** loops to do the following to the array **Num** in question **1**.

a) Add up the numbers and show the answer in a label box.

b) Find and show the largest number in a label box.

c) Count how many even numbers there are and show the answer in a label box.

d) Add 1 to all the odd numbers in the array.

e) Copy the numbers to a new array in reverse order.

3. For each loop below, draw a diagram of the **new** array formed and show any output produced. Use the original **Age** array for each question.

Index	Age
1	43
2	64
3	25
4	78
5	19

```
For Cell = 1 To 5
    Age(Cell) = Age(Cell)*2
Next Cell
```

```
For X = 1 To 5
    Age (X) = Age(6-X)
Next X
```

```
X=1
Do While X < 6
    Age (X) = Age(X)+5
Loop
```

```
X = 1
Answer = Age(X) Mod 2
Do While Answer <> 0
    Print "Odd";AGE(X)
    X = X + 1
    Answer = Age(X) Mod 2
Loop
```

```
X = 1
Do
    If Age (X+1) > Age (X) Then
       Age(X+1) = Age(X)
    End If
    X = X + 1
Loop Until X > 4
```

```
X = 4
Do
    Num(X)= Age(X) + Age(X+1)
    X = X – 1
    Print X; Num(X)
Loop While X > 1
```

```
X = 1
Do While X < 5
    Num(X+1) = Age(6 - X)
    X = X + 1
Loop
```

```
X = 1
Do
    Age(X)= Age(X) + Age(6-X)
    X = X + 1
Loop While X < 5
```

```
X = 2
Do
    Age(X) = Age(X) - Age(X - 1)
    X = X + 1
Loop Until X = 5
```

4. Using the array **Age** shown in question 3, trace the execution of (i.e. create a memory map for) the following code segment. In addition, state the purpose of the code segment.

```
For A = 1 To 5
    Biggest = 0
    For B = 1 to 5
       If Age(B) > Biggest Then
          Biggest = Age(B)
          Pos = B
       End If
    Next B
    Print Age(Pos)
    Age(Pos) = 0
Next A
```

5. Write a VB program that can perform each of the following functions.

a) Allow the user to enter a set of marks.

b) Find and display the *average*, *median* or *mode* of the entered marks.

c) Raise or lower one or more marks by a specified percentage.

d) Display a list of the failing marks.

e) Display a list of the passing marks.

Space versus Time: The Eternal Conflict in Computer Science

Background

The two most important resources that a computer uses are ***main memory*** (RAM) and ***processor time*** (CPU time). Every competent programmer wishes to write programs that use ***as little memory*** and ***as little processor time as possible***. In other words, software developers want their programs to be ***fast*** and ***small***. Unfortunately, there is a strong tendency for these two resources to offset each other. Reducing the amount of memory that a program uses ***tends*** to make it use more processor time (i.e. run more slowly). Decreasing the amount of processor time (i.e. increasing the speed) required by an algorithm tends to increase the amount of memory needed.

Increase Processor Time Required
OR
Increase Memory Required

Decrease Memory Required
OR
Decrease Processor Time Required

Balancing memory requirements and processor time can be a very tricky business!

A Problem that Illustrates the Trade-off between Space and Time

If *N* represents any positive integer, generate *N* random integers ***without repetition***.

Two Different Solutions

Solution 1

```
'Solution 1: Generate random integers without repetition.
Option Explicit
Dim RandomNum() As Integer
Private Sub cmdGenerateRandomNums_Click()
   lblRandomNums.Caption = ""
   Dim NumRandomNums As Integer, I As Integer
   Dim J As Integer, NumToChooseFrom As Integer
   Dim Repetition As Boolean, RandomNumList As String
   NumRandomNums = Val(txtNumsToChoose.Text)
   NumToChooseFrom = Val(txtNumToChooseFrom.Text)
   ReDim RandomNum(1 To NumRandomNums)
   lblRandomNums.Caption = ""
   RandomNumList = ""
   For I = 1 To NumRandomNums
       Do
           Repetition = False
           RandomNum(I) = Int(Rnd * NumToChooseFrom + 1)
           For J = 1 To I - 1
               If RandomNum(I) = RandomNum(J) Then
                   Repetition = True
                   Exit For
               End If
           Next J
       Loop Until Not Repetition
       RandomNumList = RandomNumList & Str(RandomNum(I))
   Next I
   lblRandomNums.Caption = RandomNumList
End Sub
```

Questions

1. Briefly explain the algorithm used in solution 1.

2. You will find the source code for solution 1 in the folder **I:\Out\Nolfi\Space Versus Time**. Run the program several times using different values of "NumRandomNums" and "NumToChooseFrom." Try values of NumRandomNums as large as 5000 and NumToChooseFrom as large as 10000. What do you observe?

Solution 2

```
'Solution 2: Generate random integers without repetition.
Option Explicit

Dim RandomNum() As Integer
Dim AlreadyUsed() As Boolean
Private Sub cmdGenerateRandomNums_Click()
   lblRandomNums.Caption = ""
   Dim NumRandomNums As Integer, I As Integer
   Dim NumToChooseFrom As Integer
   Dim RandomNumList As String
   NumRandomNums = Val(txtNumsToChoose.Text)
   NumToChooseFrom = Val(txtNumToChooseFrom.Text)
   ReDim RandomNum(1 To NumRandomNums)
   ReDim AlreadyUsed (1 To NumToChooseFrom)
   RandomNumList = ""
   For I = 1 To NumRandomNums
       Do
           RandomNum(I) = Int(Rnd * NumToChooseFrom + 1)
       Loop Until Not AlreadyUsed (RandomNum(I))
       AlreadyUsed (RandomNum(I)) = True
       RandomNumList = RandomNumList & Str(RandomNum(I))
   Next I
   lblRandomNums.Caption = RandomNumList
End Sub
```

Questions

1. Briefly explain the algorithm used in solution 2.

2. You will find the source code for solution 2 in the folder **I:\Out\Nolfi\Ics3m0\Space Versus Time**. Run the program several times using different values of "NumRandomNums" and "NumToChooseFrom." Try values of NumRandomNums as large as 5000 and NumToChooseFrom as large as 10000. What do you observe this time? Compare your observations to those for solution 1.

More Important Questions

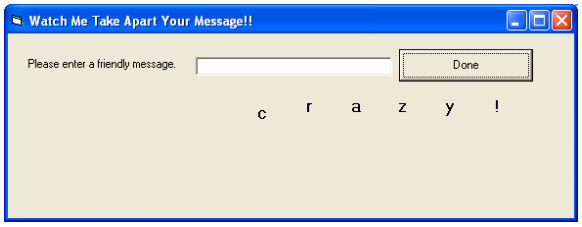
1. Which solution is faster? Which uses less memory? Explain.

2. Which solution would you choose?

3. What is the purpose of the **ReDim** keyword used in both solutions?

Introduction to Substrings, Control Arrays and Translating Objects

```
' I have designed this program to illustrate concepts that are
' new to most of the students in this course.  The new concepts
' are listed below.
' FINDING SUBSTRINGS OF STRINGS
' This program illustrates how to scan a string character-by-
' character by using a "For...Next" loop and the "Mid" intrinsic
' function.
' USING CONTROL ARRAYS TO GROUP CONTROLS (Objects)
' TRANSLATING OBJECTS

Dim NumSlides As Byte
Const InitialTop = 960

Private Sub cmdDone_Click()
    Dim Position As Integer, Length As Integer
    Dim Message As String
    Message = Trim(txtMessage.Text)
    Length = Len(Message)
    txtMessage.Text = ""
    'Scan the entered string character by character. Fill the
    'control array "lblCharacter" with the individual
    'characters found in "Message."
    For Position = 1 To Length
        lblCharacter(Position - 1).Caption = Mid(Message, _
                                                Position, 1)
    Next Position
    'This initializes the chain reaction of the enabling and
    'disabling of the timers.
    NumSlides = 0
    tmrTranslateCharacter(0).Enabled = True
End Sub

'This sub procedure causes the element "Index" of the control
'array "lblCharacter" to move 100 units down the form.  After 10
'"slides" down the form, the timer for element "Index" is
'disabled and the timer for element "Index+1" is enabled.
Private Sub tmrTranslateCharacter_Timer(Index As Integer)
    NumSlides = NumSlides + 1
    lblCharacter(Index).Top = lblCharacter(Index).Top + 100
    If NumSlides = 10 Then
        NumSlides = 0
        lblCharacter(Index).Caption = ""
        lblCharacter(Index).Top = InitialTop
        tmrTranslateCharacter(Index).Enabled = False
        If Index < 11 Then
            tmrTranslateCharacter(Index + 1).Enabled = True
        End If
    End If
End Sub
```



Answer each of the following questions. You may need to consult the MSDN help files or even MSDN online (http://msdn.microsoft.com).

1. Define the term “substring.” Explain how it applies to the code shown at the left.

2. Describe the purpose and the *syntax* of the intrinsic functions “Left,” “Right” and “Mid.”

3. What causes the “lblCharacter” label boxes to move down the form? How would you make an object move across a form? Would it be possible to cause an object to move diagonally or along a curve?

4. Explain the concept of a “control array” and describe how programming tasks can be simplified by using control arrays.

```
'This program can be found in I:\Out\Nolfi\Ics3m0\String Examples
Option Explicit
Private Sub cmdGo_Click()
    'Declaration of local variables.
    Dim YourString As String, Character As String, ReversedString As String, DashPositions As String
    Dim SpacedOut As String, YourString2 As String, Name As String, AbbreviatedName As String
    Dim Position As Long, FirstSpace As Long
   'Variable Initializations
    YourString = Trim(txtString.Text)
    YourString2 = Trim(txtString2.Text)
    Name = Trim(txtName.Text)
    ReversedString = ""
    DashPositions = ""
    SpacedOut = ""
    'Scan "YourString," character by character, from right to left.
    'Build "ReversedString," "DashPositions" and "SpacedOut."
    For Position = Len(YourString) To 1 Step -1
        Character = Mid(YourString, Position, 1)
        ReversedString = ReversedString & Character
        If Character = "-" Then
            DashPositions = Str(Position) & "," & DashPositions
        End If
        SpacedOut = Character & " " & SpacedOut
    Next Position
    'Find the position of the first space in "Name." Then set the value of "AbbreviatedName" accordingly.
    Position = 1
    Do While Mid(Name, Position, 1) <> " " And Position <= Len(Name)
        Position = Position + 1
    Loop
    'Remove any extra spaces between the given name and the surname.
    FirstSpace = Position
    Do
        Position = Position + 1
    Loop Until Mid(Name, Position, 1) <> " "
    AbbreviatedName = Mid(Name, 1, FirstSpace) & Mid(Name, Position, 1) & "."
    If DashPositions = "" Then
        DashPositions = "No dashes found."
    Else 'Remove comma at the very end of "DashPositions"
        DashPositions = Mid(DashPositions, 1, Len(DashPositions) - 1)
    End If
    'Output
    lblReversed.Caption = ReversedString
    lblDashes.Caption = DashPositions
    lblSpacedOut.Caption = SpacedOut
    If YourString < YourString2 Then
        lblAlphabetical.Caption = YourString & " " & YourString2
    Else
        lblAlphabetical.Caption = YourString2 & " " & YourString
    End If
    lblName.Caption = AbbreviatedName
End Sub
' Intercept the unloading of the form to prevent the user from accidentally quitting.  This sub procedure is
' invoked (called into action) whenever the "Close" button or the "X" (top right hand corner of form) is
'clicked.  This happens because both actions generate the "Unload" event.
Private Sub Form_Unload(Cancel As Integer)
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you wish to close this program?", _
                      vbYesNo + vbDefaultButton2, "Leaving so soon?")
    If Response = vbYes Then
        End
    Else
        'Set "Cancel" to any non-zero value to cancel the close.
        Cancel = 1
    End If
End Sub
Private Sub cmdQuit_Click()
    Unload Me 'Generate the "Unload" event.
End Sub
```

Why are these string variables initialized to the null (empty) string?

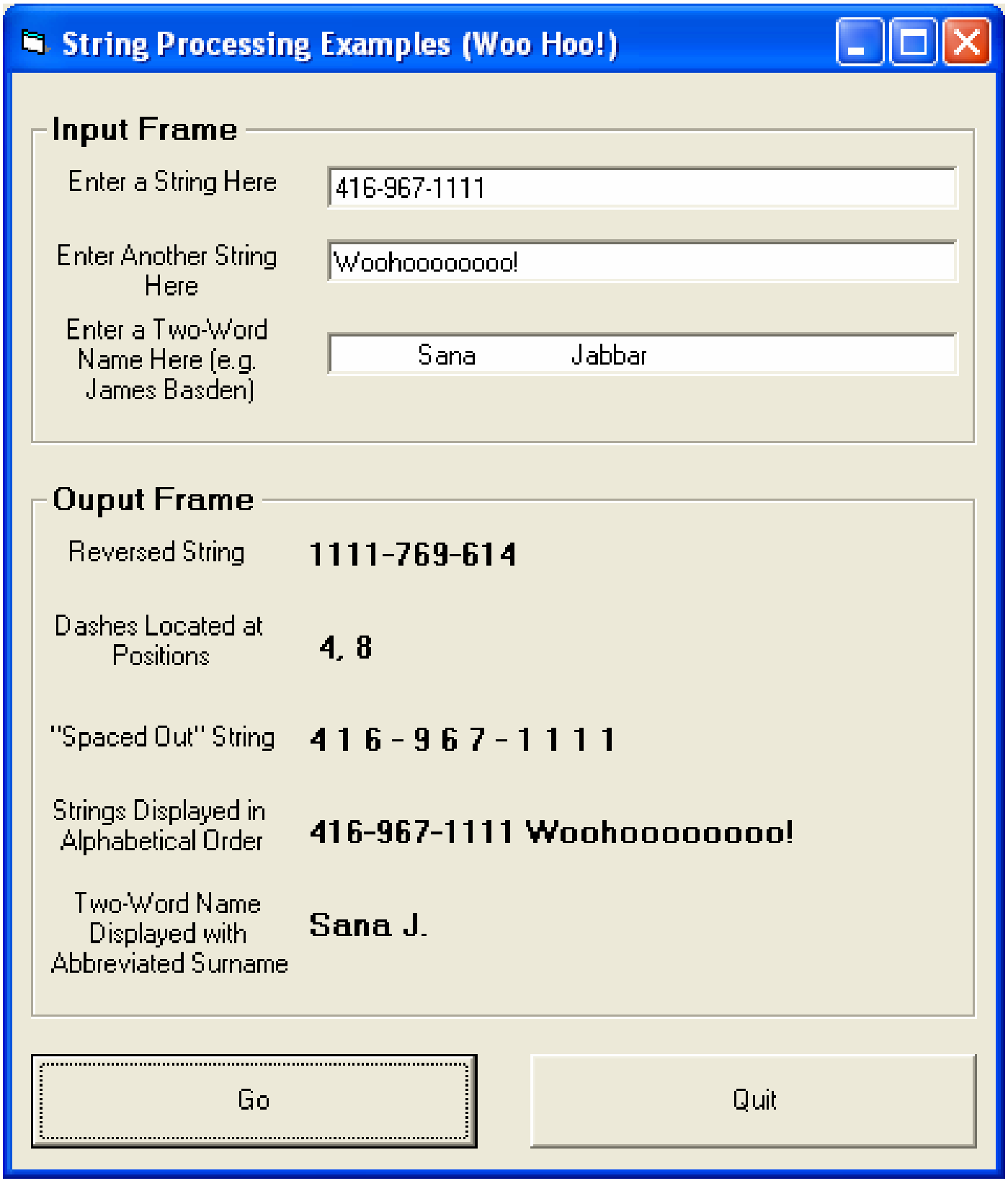
Why does this “For Loop” count down from the length of “YourString” to 1 in steps of –1?

What is the purpose of this condition?

Exercise

Modify this program (the code can be found in the usual folder on the “I” drive) so that a string is displayed that combines the characters in an even position in “YourString”with the characters in an odd position in “YourString2.” For example, the strings “Benjamin” and “Gumbley” would combine to form the string

“Gemjlmyi.”


String Processing Examples (Woo Hoo!)
Input Frame
Enter a String Here
416-967-1111
Enter Another String Here
Woohooooooooo!
Enter a Two-Word Name Here (e.g. James Basden)
Sana Jabbar
Ouput Frame
Reversed String
1111-769-614
Dashes Located at Positions
4, 8
"Spaced Out" String
4 1 6 - 9 6 7 - 1 1 1 1
Strings Displayed in Alphabetical Order
416-967-1111 Woohooooooooo!
Two-Word Name Displayed with Abbreviated Surname
Sana J.
Go
Quit

Exercises

Variable Names	Name	AccountNumber	Word	Variable Values	“Fabulous Fabrizio”	“21574365”	“HELP!”

1. Using the above ***string variables***, show the result of each of the following code segments. Note that “A” is a ***string variable*** while B, C, P and X are ***integer variables***.

```
A = Left(Name, 4)
Print A
```

```
A = Right(Word, 3)
Print A
```

```
A = Mid(AccountNumber, 3, 4)
Print A
```

```
A = Word & Right(Name, 7)
Print A
```

```
B = Len(Name)*Len(Word)
Print B
```

```
B = Asc(Mid(AccountNumber, 7, 1)) – 48
C = Asc(Mid(AccountNumber, 3, 1)) - 48
Print B + C
```

```
For X = 1 To Len(Word)
    A = Mid(Word, X, 1)
    Print A
Next X
```

```
For X = Len(Name) To 1 Step -2
    A = Mid(Name, X, 1)
    Print A;
Next X
Print 'Cursor Return
```

```
For X = 1 To Len(Word)
    A = Mid(Word, X, 1) & "*"
    Print A;
Next X
Print
```

```
For X = 1 To 4 step 2
    A = Mid(Word, X, 2)
    Print A
Next X
```

```
P = 1
Do While Mid(Name, P,1) <> " "
    P = P + 1
Loop
Print Mid(Name,13-P,P+1)
```

```
P = 1
Do While Mid(Name, P, 1) <> " "
    P = P + 1
Loop
Print Mid(Name, P + 1, P)
```

2. The string array **Book** has been loaded as shown. What is the output for each of these loops?

Book	
0	Math
1	Hist
2	Geog
3	Engl

```
a) For X = 0 To 3
       For M = 1 To 4
          A = Mid(Book(X), M, 1)
       Next M
       Print A
   Next X
```

```
b) For M = 1 To 4
       A = Mid(Book(M-1), M, 1)
       Print A
   Next M
```

3. Write code segments to perform the following tasks.
   a. Enter a word and display its letters in reverse order. (E.g. **“System”** would become **“metsyS.”**)
   b. Enter a phone number and then display the positions of the **“-”** symbol. (E.g. “905-826-1195 would display “4” and “8.”)
   c. Enter two words and display them with the longer WORD first.
   d. Enter a word and then display it with a space between each pair of consecutive letters. (E.g. **“Visual”** would become **“V i s u a l .”**)
   e. Enter two words and display them in ascending alphabetical order. The words can be in lower or upper case, so be careful in your testing.
   f. Enter a word and then display a solid “square” of “X’s” with dimensions being the size of the WORD. (E.g. the word “BIG” would produce a 3x3 square as shown below.)

      XXX
      XXX
      XXX

   g. Enter a two-word name and then display it in the form *given name* followed by the first letter of the *surname*. (E.g. **“Ashley Langlois”** would be displayed as **“Ashley L.”**

Character Sets and String Manipulation Functions

ANSI, DBCS, and Unicode: Definitions

Visual Basic uses **Unicode** to store and manipulate strings. Unicode is a character set in which 2 bytes are used to represent each character. Some other programs, such as the Windows 95 API, use ANSI (American National Standards Institute) or DBCS to store and manipulate strings. When you move strings outside of Visual Basic, you may encounter differences between Unicode and ANSI/DBCS.

This table shows the ANSI, DBCS and Unicode character sets in different environments.

Environment	*Character Set(s) Used*
Visual Basic	Unicode
32-bit object libraries	Unicode
16-bit object libraries	ANSI and DBCS
Windows NT/2000/XP API	Unicode
Automation in Windows NT/2000/XP	Unicode
Windows 95/98/Me API	ANSI and DBCS
Automation in Windows 95/98/Me	Unicode

ANSI (American National Standards Institute)

ANSI is the most popular character standard used by personal computers. Because the ANSI standard uses only a single byte to represent each character, it is limited to a maximum of 256 character and punctuation codes. Although this is adequate for English, it does not fully support many other languages. Note that originally, ANSI was called **ASCII** (American Standard Code for Information Interchange). The ASCII standard uses ***seven bits*** to represent each character, allowing for a maximum of 128 (2^7) characters. The ANSI character set, which uses ***eight bits*** (one byte), includes the ASCII character set (characters 0 to 127) plus an additional 128 characters (characters 128 to 255).

DBCS (Double-Byte Character System)

DBCS is used in Microsoft Windows systems that are distributed in most parts of Asia. It provides support for many different East Asian language alphabets, such as Chinese, Japanese and Korean. DBCS uses the numbers 0 – 127 to represent the ASCII character set. Some numbers greater than 127 function as *lead-byte characters*, which are not really characters but simply indicators that the next value is a character from a non-Latin character set. In DBCS, ASCII characters are only 1 byte in length, whereas Japanese, Korean and other East Asian characters are 2 bytes in length.

Unicode

Unicode is a character-encoding scheme that uses **2 bytes** (16 bits) for ***every*** character. The International Standards Organization (ISO) defines a number in the range of 0 to 65,535 ($2^{16} – 1$) for just about every character and symbol in every language (plus some empty spaces for future growth). On all 32-bit versions of Windows, Unicode is used by the Component Object Model (COM), the basis for OLE and ActiveX technologies. Unicode is fully supported by Windows NT/2000/XP. **Although both Unicode and DBCS have double-byte characters, the schemes are entirely different.** (Visit www.unicode.org to find out more.)

Example: Character codes for "A" in ANSI, Unicode, and DBCS

Note: In Visual Basic, the code "&H" is used to indicate that the number that follows is in hexadecimal form.

Character	Description	Hexadecimal Representation		Binary Representation	
		Byte 1	Byte 2	Byte 1	Byte 2
A	**ANSI** Character "A"	&H41		01000001	
A	**Unicode** Character "A"	&H41	&H00	01000001	00000000
Ａ	**DBCS** Japanese Wide-Width "Ａ"	&H82	&H60	10000010	01100000
Ａ	**Unicode** Wide-Width "Ａ"	&H21	&HFF	00100001	11111111

Issues Specific to the Double-Byte Character Set (DBCS)

DBCS is a different character set from Unicode. Because Visual Basic represents all strings internally in Unicode format, both ANSI characters and DBCS characters are converted to Unicode and Unicode characters are converted to ANSI characters or DBCS characters automatically whenever the conversion is needed. You can also convert between Unicode and ANSI/DBCS characters manually. For more information about conversion among different character sets, see "DBCS String Manipulation Functions" below.

When developing a DBCS-enabled application with Visual Basic, you should consider:

- Differences among Unicode, ANSI, and DBCS.
- DBCS sort orders and string comparison.
- DBCS string manipulation functions.
- DBCS string conversion.
- How to display and print fonts correctly in a DBCS environment.
- How to process files that include double-byte characters.
- DBCS identifiers.
- DBCS-enabled events.
- How to call Windows APIs.

Tip

Developing a DBCS-enabled application is good practice, whether or not the application is run in a locale where DBCS is used. This approach will help you develop a flexible, portable, and truly international application. None of the DBCS-enabling features in Visual Basic will interfere with the behaviour of your application in environments using exclusively single-byte character sets (SBCS). Also, the size of your application will not increase because both DBCS and SBCS use Unicode internally.

For More Information

For limitations on using DBCS for access and shortcut keys, see "Designing an International-Aware User Interface" in the MSDN collection.

DBCS String Manipulation Functions

Although a double-byte character consists of a lead byte and a trail byte and requires two consecutive storage bytes, it must be treated as a single unit in any operation involving characters and strings. Several string manipulation functions properly handle all strings, including DBCS characters, on a character basis. These functions have an ANSI/DBCS version and a binary version and/or Unicode version, as shown in the following table. Use the appropriate functions, depending on the purpose of string manipulation. The "B" versions of the functions in the following table are intended especially for use with strings of binary data. The "W" versions are intended for use with Unicode strings.

Function	Description	Function	Description
Asc	Returns the ANSI or DBCS character code for the first character of a string.	InStr	Returns the first occurrence of one string within another.
AscB	Returns the value of the first byte in the given string containing binary data.	InStrB	Returns the first occurrence of a byte in a binary string.
AscW	Returns the Unicode character code for the first character of a string.	Left, Right	Returns a specified number of characters from the right or left sides of a string.
Chr	Returns a string containing a specific ANSI or DBCS character code.	LeftB, RightB	Returns a specified number of bytes from the left or right side of a binary string.
ChrB	Returns a binary string containing a specific byte.	Len	Returns the length of the string in number of characters.
ChrW	Returns a string containing a specific Unicode character code.	LenB	Returns the length of the string in number of bytes.
Input	Returns a specified number of ANSI or DBCS characters from a file.	InputB	Returns a specified number of bytes from a file.
Mid	Returns a specified number of characters from a string.	MidB	Returns the specified number of bytes from a binary string.

The functions without a “B” or “W” in this table correctly handle DBCS and ANSI characters. In addition to the functions above, the **String** function handles DBCS characters. This means that all these functions consider a DBCS character as one character even if that character consists of 2 bytes.

The behaviour of these functions is different when they are handling SBCS and DBCS characters. For instance, the Mid function is used in Visual Basic to return a specified number of characters from a string. In locales using DBCS, the number of characters and the number of bytes are not necessarily the same. Mid would only return the number of characters, not bytes.

In most cases, use the character-based functions when you handle string data because these functions can properly handle ANSI strings, DBCS strings and Unicode strings.

The byte-based string manipulation functions, such as LenB and LeftB, are provided to handle the string data as binary data. When you store the characters to a **String** variable or get the characters from a **String** variable, Visual Basic automatically converts between Unicode and ANSI characters. When you handle the binary data, use the Byte array instead of the String variable and the byte-based string manipulation functions.

The ANSI Character Set

(Characters 0 – 127 Originally Called “ASCII” Character Set)

0	null char	24	ctrl-X	48	0	72	H	96	`	120	x	144	€	168	¨	192	À	216	Ø	240	ð
1	ctrl-A	25	ctrl-Y	49	1	73	I	97	a	121	y	145	€	169	©	193	Á	217	Ù	241	ñ
2	ctrl-B	26	ctrl-Z	50	2	74	J	98	b	122	z	146	€	170	ª	194	Â	218	Ú	242	ò
3	ctrl-C	27	ESC	51	3	75	K	99	c	123	{	147	€	171	«	195	Ã	219	Û	243	ó
4	ctrl-D	28	□	52	4	76	L	100	d	124	|	148	€	172	¬	196	Ä	220	Ü	244	ô
5	ctrl-E	29	□	53	5	77	M	101	e	125	}	149	€	173	-	197	Å	221	Ý	245	õ
6	ctrl-F	30	□	54	6	78	N	102	f	126	~	150	€	174	®	198	Æ	222	Þ	246	ö
7	ctrl-G	31	□	55	7	79	O	103	g	127	□	151	€	175	¯	199	Ç	223	ß	247	÷
8	ctrl-H (backspace)	32	space	56	8	80	P	104	h	128	€	152	€	176	°	200	È	224	à	248	ø
9	ctrl-I (TAB)	33	!	57	9	81	Q	105	i	129	€	153	€	177	±	201	É	225	á	249	ù
10	ctrl-J (linefeed)	34	"	58	:	82	R	106	j	130	€	154	€	178	²	202	Ê	226	â	250	ú
11	ctrl-K	35	#	59	;	83	S	107	k	131	€	155	€	179	³	203	Ë	227	ã	251	û
12	ctrl-L	36	$	60	<	84	T	108	l	132	€	156	€	180	´	204	Ì	228	ä	252	ü
13	ctrl-M (ENTER)	37	%	61	=	85	U	109	m	133	€	157	€	181	µ	205	Í	229	å	253	ý
14	ctrl-N	38	&	62	>	86	V	110	n	134	€	158	€	182	¶	206	Î	230	æ	254	þ
15	ctrl-O	39	'	63	?	87	W	111	o	135	€	159	€	183	·	207	Ï	231	ç	255	ÿ
16	ctrl-P	40	(	64	@	88	X	112	p	136	€	160	space	184	¸	208	Ð	232	è		
17	ctrl-Q	41	)	65	A	89	Y	113	q	137	€	161	¡	185	¹	209	Ñ	233	é		
18	ctrl-R	42	*	66	B	90	Z	114	r	138	€	162	¢	186	º	210	Ò	234	ê		
19	ctrl-S	43	+	67	C	91	[	115	s	139	€	163	£	187	»	211	Ó	235	ë		
20	ctrl-T	44	,	68	D	92	\	116	t	140	€	164	¤	188	¼	212	Ô	236	ì		
21	ctrl-U	45	-	69	E	93	]	117	u	141	€	165	¥	189	½	213	Õ	237	í		
22	ctrl-V	46	.	70	F	94	^	118	v	142	€	166	¦	190	¾	214	Ö	238	î		
23	ctrl-W	47	/	71	G	95	_	119	w	143	€	167	§	191	¿	215	×	239	ï		

€, □ These characters are not supported by Microsoft Windows.

Key Code Constants in Visual Basic

Constant	Value	Description	Constant	Value	Description	Constant	Value	Description
vbKeyA	65	A key	**vbKey0**	48	0 key	**vbKeyF1**	0x70	F1 key
vbKeyB	66	B key	**vbKey1**	49	1 key	**vbKeyF2**	0x71	F2 key
vbKeyC	67	C key	**vbKey2**	50	2 key	**vbKeyF3**	0x72	F3 key
vbKeyD	68	D key	**vbKey3**	51	3 key	**vbKeyF4**	0x73	F4 key
vbKeyE	69	E key	**vbKey4**	52	4 key	**vbKeyF5**	0x74	F5 key
vbKeyF	70	F key	**vbKey5**	53	5 key	**vbKeyF6**	0x75	F6 key
vbKeyG	71	G key	**vbKey6**	54	6 key	**vbKeyF7**	0x76	F7 key
vbKeyH	72	H key	**vbKey7**	55	7 key	**vbKeyF8**	0x77	F8 key
vbKeyI	73	I key	**vbKey8**	56	8 key	**vbKeyF9**	0x78	F9 key
vbKeyJ	74	J key	**vbKey9**	57	9 key	**vbKeyF10**	0x79	F10 key
vbKeyK	75	K key	**vbKeyLButton**	0x1	Left mouse	**vbKeyF11**	0x7A	F11 key
vbKeyL	76	L key	**vbKeyRButton**	0x2	Right mouse	**vbKeyF12**	0x7B	F12 key
vbKeyM	77	M key	**vbKeyCancel**	0x3	CANCEL key	**vbKeyF13**	0x7C	F13 key
vbKeyN	78	N key	**vbKeyMButton**	0x4	Middle mouse	**vbKeyF14**	0x7D	F14 key
vbKeyO	79	O key	**vbKeyBack**	0x8	BACKSPACE	**vbKeyF15**	0x7E	F15 key
vbKeyP	80	P key	**vbKeyTab**	0x9	TAB key	**vbKeyF16**	0x7F	F16 key
vbKeyQ	81	Q key	**vbKeyClear**	0xC	CLEAR key	**vbKeyEnd**	0x23	END key
vbKeyR	82	R key	**vbKeyReturn**	0xD	ENTER key	**vbKeyHome**	0x24	HOME key
vbKeyS	83	S key	**vbKeyShift**	0x10	SHIFT key	**vbKeyLeft**	0x25	LEFT
vbKeyT	84	T key	**vbKeyControl**	0x11	CTRL key	**vbKeyUp**	0x26	UP
vbKeyU	85	U key	**vbKeyMenu**	0x12	MENU key	**vbKeyRight**	0x27	RIGHT
vbKeyV	86	V key	**vbKeyPause**	0x13	PAUSE key	**vbKeyDown**	0x28	DOWN
vbKeyW	87	W key	**vbKeyCapital**	0x14	CAPS LOCK	**vbKeySelect**	0x29	SELECT
vbKeyX	88	X key	**vbKeyEscape**	0x1B	ESC key	**vbKeyPrint**	0x2A	PRT SCR
vbKeyY	89	Y key	**vbKeySpace**	0x20	SPACEBAR	**vbKeyExecute**	0x2B	EXECUTE
vbKeyZ	90	Z key	**vbKeyPageUp**	0x21	PAGE Up	**vbKeySnapshot**	0x2C	SNAPSHOT
			vbKeyPageDown	0x22	PAGE Down	**vbKeyInsert**	0x2D	INSERT
						vbKeyDelete	0x2E	DELETE
						vbKeyHelp	0x2F	HELP key
						vbKeyNumlock	0x90	Num Lock

Constant	Value	Description
vbKeyNumpad0	0x60	0 key
vbKeyNumpad1	0x61	1 key
vbKeyNumpad2	0x62	2 key
vbKeyNumpad3	0x63	3 key
vbKeyNumpad4	0x64	4 key
vbKeyNumpad5	0x65	5 key
vbKeyNumpad6	0x66	6 key
vbKeyNumpad7	0x67	7 key
vbKeyNumpad8	0x68	8 key
vbKeyNumpad9	0x69	9 key
vbKeyMultiply	0x6A	MULT. SIGN (*)
vbKeyAdd	0x6B	PLUS SIGN (+)
vbKeySeparator	0x6C	ENTER key
vbKeySubtract	0x6D	MINUS SIGN
vbKeyDecimal	0x6E	DECIMAL POINT
vbKeyDivide	0x6F	DIVISION SIGN (/)

This table lists the VB key code constants for the numeric keypad. Notice that the constant names are different from the names of corresponding keys elsewhere on the keyboard.

e.g. vbKeyReturn → ENTER key near SHIFT key

vbKeySeparator → ENTER key on numeric keypad

Notice that some of the key code values are written as ordinary ***decimal*** (base 10) values while others are written as ***hexadecimal*** (base 16) values. The hexadecimal values are preceded by the "0x" prefix. The "0x" prefix is used in C/C++ to denote hexadecimal numbers. Recall that "&H" is the Visual Basic notation for hexadecimal numbers.

Details on the hexadecimal system will be given in class.

Exercises

1. Why it is better to use VB key code constant names instead of the actual numerical values? (For example, why is it better to use the key code constant name **"vbKeyBack"** instead of the actual ANSI code "8?")

2. Briefly describe the similarities and differences among the character standards ANSI, DBCS and Unicode.

3. Explain why using an *n-bit* code allows for the representation of 2^n different characters. How many characters can be represented using
   a. an 8-bit code?
   b. a 16-bit code?
   c. a 32-bit code?

4. Colours are also represented as bit patterns. Explain why a 24-bit colour code allows for the representation of over sixteen million colours.

Credit Card Validation Assignment

Introduction

Assignment created by Mr. Dobias

All major credit cards contain an embedded pattern of digits. These digits are called "check digits," as they provide an easy way to check if any particular credit card number is numerically valid. In this assignment, you will design and write a program that determines whether a credit card number is numerically (structurally) valid.

Rules for Credit Card Number Validity:

1. *Length and Prefix*

Credit Card Type	Valid Length	Valid Prefix
Visa	**13 or 16**	**4**
Master Card	**16**	**51-55**
Amex	**15**	**34 or 37**
Discover	**16**	**6011**

All valid credit card numbers must have their respective length and prefix.

2. *Internal Numeric Formula*

Further, a credit card number must be validated as follows:

- Add all digits from right to left.
- Alternate "check" digits must be adjusted in the following way:
  - Multiply each alternate digit by 2.
  - If the product is more than a single digit (i.e. greater than 9), add the two digits to obtain a single digit.
- The sum of all digits mod 10 must be equal to 0 (i.e. when the sum must be divisible by 10).

Initially, this may seem difficult, but the method is quite simple.

Example

Suppose you are testing the following Visa number: 4947152680730

1. *Length and Prefix*

Clearly the number has the correct prefix (4) and a correct length (13).

2. *Internal Numeric Formula*

We will add the digits from right to left (0,3,7,… to …7,4,9,4), multiplying and adjusting alternate "check" digits:

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th
0	3*2 =**6**	**7**	0*2 =**0**	**8**	6*2 =12 1+2 =**3**	**2**	5*2 =10 1+0 =**1**	**1**	7*2 =14 1+4 =5	**4**	9*2 =18 1+8 =**9**	**4**

Therefore we have: Sum = 0+6+7+0+8+3+2+1+1+5+4+9+4 = 50

Now we must determine if the sum is divisible by 10 (sum mod 10 is equal to 0).

50 mod 10 = 0.

Therefore, the credit card number is valid, since it meets all conditions.

Program Plan

You may, if you wish, plan your program as follows:

Step 1: Capture User Input

- What is the credit card type?
- What is the credit card number?

Step 2: Determine Number Validity

- Check length.
- Check prefix.
- Calculate and check the sum with the internal numeric formula (see above).

Step 3: Display the Result

- Output a message about credit card validity ("Credit Card Valid" or "Credit Card Invalid").
- Allow the user to test another number.

Additional Notes

- Please note that this method does not determine if a number is real, only valid.
- You are expected to follow standard programming naming conventions (variable names, indentation, code documentation).
- Think carefully of which data types you will use.
- The input for the program is a credit card type and a credit card number. The output should be a determination (true or false) of credit card's numerical validity.
- HINT: think before you start programming. Do simple things first.
- You are also expected to provide a set of test cases (sample inputs with corresponding sample outputs) for your program that shows program correctness (i.e. that your program produced correct outputs for all inputs).
- If the user input is anything but a valid number, your program should consider the number invalid.
- If you're stuck, make use of online and offline documentation and resources.

Additional Challenge for Extra Credit

Include a feature that allows the user to ***generate*** valid credit card numbers.

Remember…

"Remember, whether you say you ***can*** do something or you say you ***can't***, you're right."

Anthony Robbins

Practice Exercises

1. Working in groups of 2-3, determine which of these credit card numbers are valid based on the prefix, length, and internal numeric formula:

a) Is VISA number 4947152680730 valid?
Check Prefix: 4 (correct)
Check Length: 13 (correct)
Check Internal Numeric Formula: Sum = 6+7+8+3+2+1+1+5+4+9+4 = 50.
50 mod 10 = 0 (correct)
This number is VALID.

b) Is DISCOVER number 601195145328714 valid?
Check Prefix:
Check Length:
Check Internal Numeric Formula:
This number is

c) Is MASTERCARD number 5358390378156038 valid?
Check Prefix:
Check Length:
Check Internal Numeric Formula:
This number is

d) Is AMEX number 375627815798423 valid?
Check Prefix:
Check Length:
Check Internal Numeric Formula:
This number is

2. In your groups, change the invalid numbers above (by changing their digit or digits) into valid ones. How many ways are there to do this?

3. Individually, come up with your own valid number (from scratch).

Evaluation Guide for Credit Card Validator Program

Categories	Criteria	Descriptors					Level	Average
		Level 4	Level 3	Level 2	Level 1	Level 0		
Knowledge and Understanding (KU)	**Understanding of Programming Concepts**	Extensive	Good	Moderate	Minimal	Insufficient		
	Understanding of the Problem	Extensive	Good	Moderate	Minimal	Insufficient		
Application (APP)	**Correctness** To what degree is the output correct?	Very High	High	Moderate	Minimal	Insufficient		
	Run-time Error Handling How stable is the software?	Highly Stable	Stable	Moderately Stable	Somewhat Unstable	Very Unstable		
	Declaration of Variables To what degree are the variables declared with appropriate data types?	Very High	High	Moderate	Minimal	Insufficient		
	Unnecessary Duplication of Code To what degree has the student avoided unnecessary duplication of code?	Very High	High	Moderate	Minimal	Insufficient		
	Debugging To what degree has the student employed a logical, thorough and organized debugging method?	Very High	High	Moderate	Minimal	Insufficient		
Thinking, Inquiry and Problem Solving (TIPS)	**Algorithm Design and Selection** To what degree has the student used approaches such as solving a specific example of the problem to gain insight into the problem that needs to be solved?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Design and Select Algorithms Independently To what degree has the student been able to design and select algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Ability to Implement Algorithms Independently To what degree is the student able to implement chosen algorithms without assistance?	Very High	High	Moderate	Minimal	Insufficient		
	Efficiency of Algorithms and Implementation To what degree does the algorithm use resources (memory, processor time, etc) efficiently?	Very High	High	Moderate	Minimal	Insufficient		
Communication (COM)	**Indentation of Code** **Insertion of Blank Lines in Strategic Places** (to make code easier to read)	Very Few or no Errors	A Few Minor Errors	Moderate Number of Errors	Large Number of Errors	Very Large Number of Errors		
	Comments • Effectiveness of explaining abstruse (difficult-to-understand) code • Effectiveness of introducing major blocks of code • Avoidance of comments for self-explanatory code	Very High	High	Moderate	Minimal	Insufficient		
	Descriptiveness of Identifier Names Variables, Constants, Objects, Functions, Subs, etc **Inclusion of Property Names with Object Names** (e.g. 'txtName.Text' instead of 'txtName' alone) **Clarity of Code** How easy is it to understand, modify and debug the code? **Adherence to Naming Conventions** (e.g. use "txt" for text boxes, "lbl" for labels, etc.)	Masterful	Good	Adequate	Passable	Insufficient		
	User Interface To what degree is the user interface well designed, logical, attractive and user-friendly?	Very High	High	Moderate	Minimal	Insufficient		

Notes on Debugging to Help you with your Credit Card Validator Program

Use "Print" statements, breakpoints and the "Debug" menu in VB when you need to verify whether your program is working correctly!

Example 1

```
Option Explicit                                                    'Line 1

Private Sub Command1_Click()

    Dim Number As Integer, Sum As Integer, X As Integer            'Line 2

    Number = Val(Text1.Text)                                       'Line 3
    Sum = 0                                                        'Line 4

    'Adds up numbers 1 to number-1
    For X = 1 To Number - 1                                        'Line 5
        Sum = Sum + X                                              'Line 6
        'Printing loop counter and the running sum
        Print X, Sum                                               'Line 7
    Next X                                                         'Line 8

    Text1.Text = CStr(Sum)                                         'Line 9

End Sub
```

Example 2

```
Private Sub Command1_Click()                                       'Line 10

    Dim ccNum As String, Message As String                         'Line 11
    Dim Sum As Integer, X As Integer                               'Line 12

    Sum = 0                                                        'Line 13
    ccNum = Trim(Text1.Text)                                             'Line 14

    'Calculates sum of all the digits
    For X = 1 To Len(ccNum)                                        'Line 15
        Sum = Sum + Val(Mid(ccNum, X, 1))                          'Line 16
        'Printing loop counter and the running sum
        Print X, Sum                                               'Line 17
    Next X                                                         'Line 18

    Message = "The sum of the digits is " & CStr(Sum) & "."        'Line 19
    MsgBox Message                                                 'Line 20

End Sub                                                            'Line 21
```

Questions

1. What is the purpose of the **`Option Explicit`** statement on line 1? How does it help you to reduce the amount of time needed to debug your programs?
2. On line 3, the "Val" function is used to convert the string value stored in the "Text" property of the text box to a numeric value. If you omit the "Val" function, will the code still work? Why is it a bad idea to leave out the "Val" function.
3. In line 9, the "CStr" function is used to convert the numeric value stored in the "Sum" variable to a string value. If you omit the "CStr" function, will the code still work? Why is it a bad idea to leave out the "CStr" function.
4. Will the code on line 16 still work if the "Val" function is omitted? Why is it a bad idea to omit the "Val" function?
5. Explain why the use of "Print" statements can be very helpful in the debugging process.
6. What other methods of debugging have you already used in the development of VB programs?
7. Explore the "Debug" menu in VB. Explain how all the options in this menu ("Step Into", "Step Out," "Step Over," "Run to Cursor," "Add Watch," "Edit Watch," "Quick Watch," "Toggle Breakpoint," "Clear All Breakpoints," "Set Next Statement," "Show Next Statement") can help you to debug programs.

Assignment on Two-Dimensional Arrays (Optional Topic)

Data Encryption using the Vigenère Cipher

Throughout history, espionage has always been widely practiced by governments and other organizations. To aid the cause of spies, the military and other groups, algorithms like the Vigenère cipher have been, and are continuing to be created. The Vigenère cipher is a data encryption algorithm that was published in 1586 by Blaise de Vigenère from the court of Henry III of France. It was created during a very turbulent period in European history to address the phenomenon of a rapidly growing number of new European nations. Many European governments felt that it was necessary to use methods of enciphering text to prevent the myriad new nations from obtaining sensitive information. Used for almost 300 years and considered virtually unbreakable, the Vigenère cipher was finally broken, in 1863, by a Prussian major named Kasiski. Since the Vigenère cipher was used by the Confederate army ("The South") during the American Civil War, Kasiski's discovery ultimately helped to give the upper hand to the Union army ("The North").

The Vigenère cipher is an example of a *Polyalphabetic Substitution Cipher*. It is a substitution cipher because letters in the original text are replaced by other letters. It is called polyalphabetic because it involves several *Caesar Shifts*. A Caesar shift is a substitution cipher that involves a simple transposition of letters. For instance, in a Caesar shift of three characters, an "A" would be replaced by a "D," a "B" would be replaced by an "E," a "C" would be replaced by an "F" and so on. Caesar shifts are not very useful ciphers because they are extremely easy to break. However, the Vigenère, being polyalphabetic, is much more difficult to break. It consists of multiple Caesar shifts. Notice that the first row of the table shown below is a Caesar shift of zero characters, the second row is a Caesar shift of one character, the third row is a Caesar shift of two characters and so on.

		Plaintext																								
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
KEYWORD	**A**	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	B	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
	C	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	D	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
	E	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	F	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	G	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	H	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	I	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	J	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	K	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	L	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	M	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	O	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
	P	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
	Q	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
	R	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
	S	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
	T	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	U	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	V	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	W	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	X	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	Y	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
	Z	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

Example

To use the Vigenère cipher to encode a message, a secret *keyword* is needed. To ensure secrecy, the keyword should be known only to the sender and the recipient of the message.

Suppose that the keyword is **JAMES** and that the message is **"FABRIZIO USES FORGED NOTES."** The keyword should be placed below the *plaintext* as many times as necessary (as shown below). Then the encoded message is generated by replacing each character in the *plaintext* with the character found at the intersection of the *plaintext* letter column and the *keyword* letter row (in the above table). Complete the third row below to generate the enciphered message (*ciphertext*).

Plaintext	F	A	B	R	I	Z	I	O		U	S	E	S		F	O	R	G	E	D		N	O	T	E	S
Keyword	J	A	M	E	S	J	A	M		E	S	J	A		M	E	S	J	A	M		E	S	J	A	M
Ciphertext	**W**	**A**	**P**																							

Write a program to implement the Vigenère cipher. Your program should be able to decipher (decode, decrypt) as well as encipher (encode, encrypt). The **plaintext** processed by your program should be read (input) from a data file and the **ciphertext** output by your program should be written to (stored in) a data file. There are several security issues to consider when developing this program. As the development proceeds, you will become aware of them.

Questions

1. Define the terms *cipher, encipher, decipher, cryptography, encrypt, decrypt, encode, decode, keyword, Caesar shift, polyalphabetic, plaintext, ciphertext.*

2. When was the Vigenère cipher broken and by whom? Which army fell partly because of the other side having learned how to "crack the code?"

3. Is the Vigenère cipher suitable for modern data encryption applications?

4. Do you think that it is possible to create an unbreakable data encryption algorithm?

5. Do some research to find out the names of the encryption algorithms used nowadays for making secure transactions on the Internet.

Using Visual Basic to Produce String Art

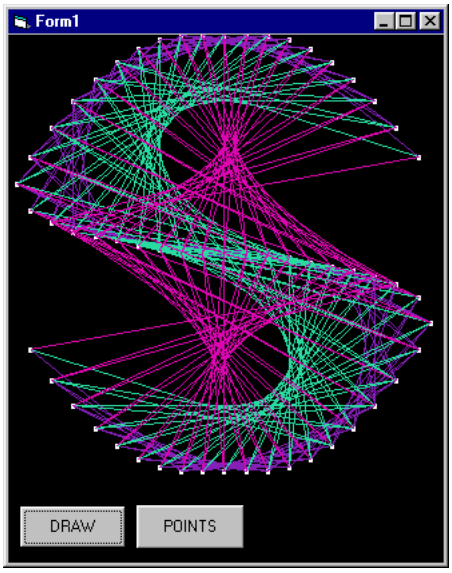
The String Art Algorithm

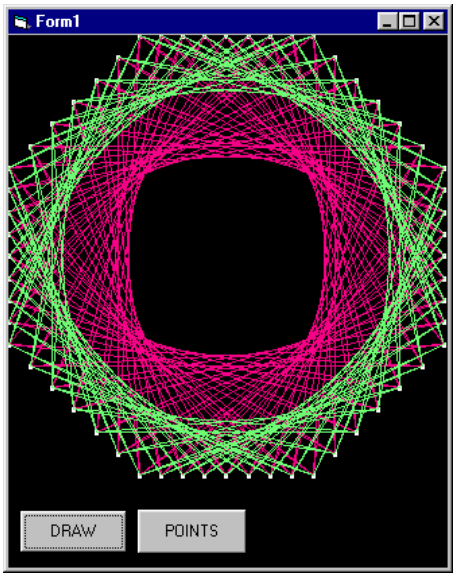
A set of N points is read in from a data file (or are defined from code) and connected according to the following ***algorithm***. Note that the following ***IS NOT* Visual Basic code!** It is ***pseudo-code***! Your job is to ***translate*** the pseudo-code into VB!

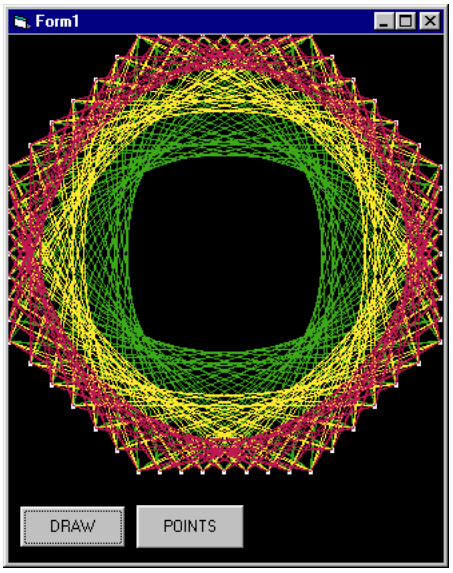
```
'Initialize the values of A and B
Set A=1 and B=some value between 1 & N
loop
   join point A to point B
   add 1 to A
   join point B to point A
   add 1 to B
   if B > N
      set B=1
until A = N
```

By changing the initial value of B (just before the loop) a different pattern can be produced.

Examples of String Art







Exercises

6. How many points are used in string art example 1?

7. How many points are used in string art example 2?

8. How many points are used in string art example 3?

9. Explain the string art algorithm in plain English.

10. Write a VB program that can produce any string art given "N" points and an initial value of "B." Include a feature that allows the user to change the initial value of "B" and to select the colours used. Allow the user to select up to three colours.

Fractals

Fractal Geometry

Fractal geometry is the branch of mathematics that deals with producing extremely irregular curves or shapes for which any suitably chosen part is similar in shape to a given larger or smaller part when magnified or reduced to the same size (this property of fractals is known as ***self-similarity***.). A "picture" or "image" produced by a fractal geometry algorithm is usually called a ***fractal***. Fractal geometry is closely related to a branch of mathematics known as ***chaos theory***.

The Chaos Game

To gain a basic understanding of fractals, it is helpful to play a game called the ***chaos*** game. The game proceeds in its simplest form as follows. Place three dots at the vertices of any triangle. Colour the top vertex red, the lower left green and the lower right blue. Then take a die and colour two faces red, two green and two blue.

To play the game, you need a ***seed***, an arbitrary starting point in the plane. Starting with this point, the algorithm begins with a roll of the die. Then, depending upon which colour comes up, plot a point halfway between the seed and the appropriate coloured vertex. Repeat this process using the terminal point of the previous move as the seed for the next. ***To obtain the best possible results, do not plot the first 15 (or so) points generated by this algorithm! Only begin plotting after the first 15 points have been generated!***

For example, *Figure 1* shows the moves associated with rolling red, green, blue and blue in order.

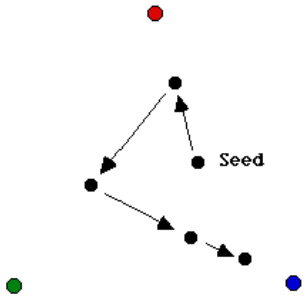
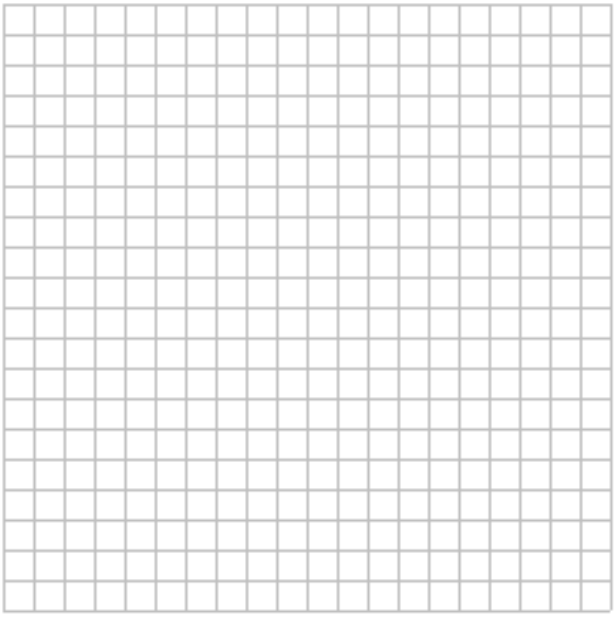


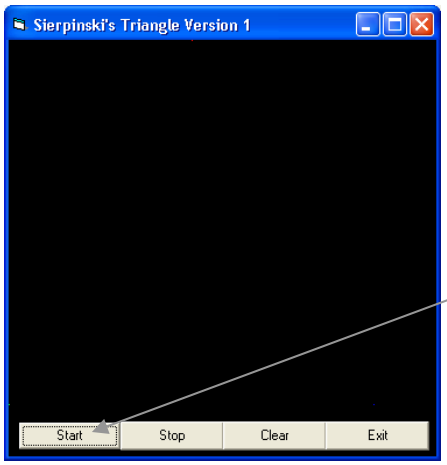
Figure1 Playing the chaos game with rolls of red, green, blue, blue.



People who have never played this game are always surprised and amazed at the result! Most expect the algorithm to yield a blur of points in the middle of the triangle. Some expect the moving point to fill the whole triangle. Surprisingly, however, the result is anything but a random mess. The resulting picture is one of the most famous of all fractals, the *Sierpinski triangle*.

The Sierpinski Triangle

Now to see the *Chaos Game* in action, run the program "Sierpinski's Triangle.vbp" in the folder **"I:\Out\Nolfi\Drawing, Graphics, Game Program Examples\Sierpinski's Triangle V1 and V2."**



You must be patient once you click on the "Start" button!

It takes a few minutes for this program to generate Sierpinski's triangle. However, it will be well worth the wait! You ***will*** be amazed by the figure generated by this seemingly random and chaotic algorithm!

Assignment (To be handed in)

3. Use the Internet (or whatever other resources that you wish to use) to find algorithms that produce the following fractals:

g. Sierpinski Triangle (this one is easy because I have already given it to you)
h. Sierpinski Pentagon
i. Sierpinski Hexagon
j. Sierpinski Carpet
k. Koch Snowflakes

Then create a word processor document that gives a brief outline of each algorithm. Include diagrams to supplement the description of each algorithm.

4. Using a Web browser, load the Java applet with URL http://math.bu.edu/DYSYS/applets/fractalina.html. Experiment with this applet for a few minutes to familiarize yourself with its various features. Then write a Visual Basic program that is similar to the "Fractalina" applet. Your program must be able to generate the following fractals:

f. Sierpinski Triangle
g. Sierpinski Pentagon
h. Sierpinski Hexagon
i. Sierpinski Carpet
j. Koch Snowflakes

Note that your program ***need not have*** "New Point," "Kill Point" and "Zoom Out" buttons. However, your program ***should*** allow the user to drag the vertices of the shapes to different locations.