**Jacob Billington**

**Frequency Analysis Project**

**Technical Solution**

Contents

[3.1 An overview guide of code modules 3](#_Toc506881044)

[3.2 File structures and path names with running instructions 4](#_Toc506881045)

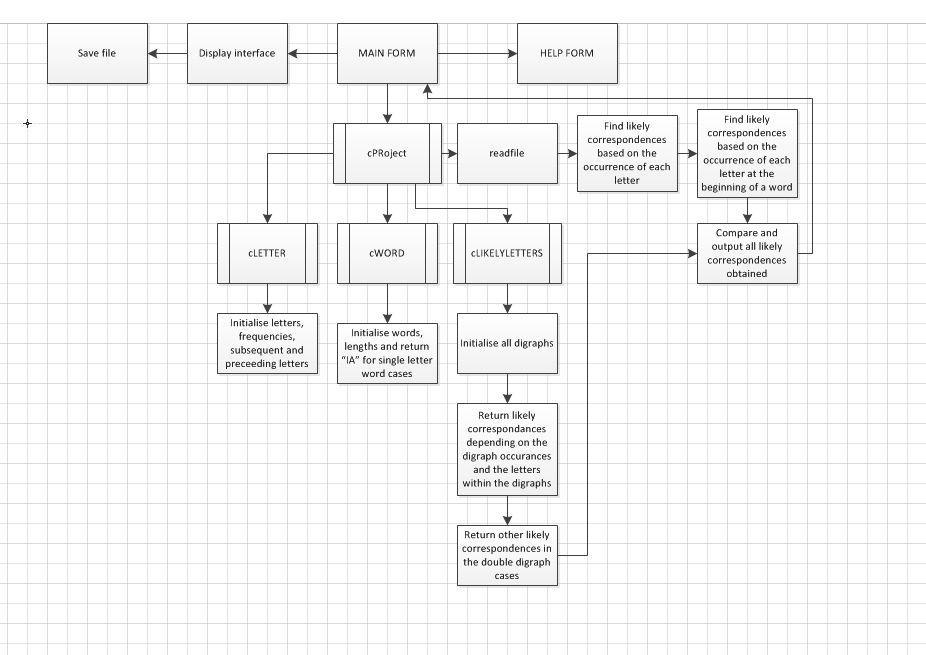
[3.3 Explanations of particularly difficult code sections 5](#_Toc506881046)

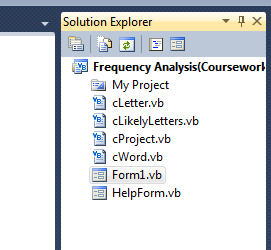
[3.4 Program listings 6](#_Toc506881047)

# 3.1 An overview guide of code modules

|  |  |
| --- | --- |
| Inputs | Processes |
| Ciphertext  File names  Commands  Characters | Read from file  Write to file  Split text into words  Split text into digraphs  Calculate frequency of letter occurrence  Sort letters  Find most likely corresponding letters  Allow user to change characters in the text for actual letters |
| Storage | Outputs |
| CipherText Folder   * Stores text files containing unprocressed cipher text * Also contrains text files of ciphers which have been only partially decrypted   Plaintext Folder   * Stores text files containing the plaintext / decrypted cipher outputted by the program | Most likely characters of plaintext  Ciphertext |

This flowchart outlines briefly the modules / classes within the program and also a brief explanation of the stages that each class runs. The connectors show how data is moved between classes and subprocedures to provide the user with appropriate and relevant information or other procedures with data required to calculate the likely correspondances for the ciphertext given to the system.



There are six different self-written and identified classes that are used in this program, as is shown by the screen capture of the IDE (given left).

The cLetter Class:

This class contains the properties required by the rest of the programming pertainting to every letter found in the cipher provided. The target/ principle letter is stored along with the letters before and after it (all stored as characters). The frequency (both as integer and percentage) occurrence of the letter in the cipher is also saved. The methods of this class include getters and setters (polymorphic procedures) that allow other classes to find and set data pertaining to certain letters within the cipher.

Public Class cLetter (class definition):

Properties:

* Private pPrincipleLetter As Char
* Private pIsVowell As Boolean
* Private pLetterAfter As Char
* Private pLetterBefore As Char
* Private pFrequencyINT As Integer
* Private pFrequencyPRCNT As Single

Methods:

* Public Sub New()
* Public Function PrincipleLetterInitialisation() As Char
* Public Sub PrincipleLetterInitialisation(ByVal NewLetter As Char)
* Public Function SubsequentLetter() As Char
* Public Sub SubsequentLetter(ByVal NewLetter As Char)
* Public Function PreceedingLetter() As Char
* Public Sub PreceedingLetter(ByVal NewLetter As Char)
* Public Sub SetFrequencyINT(ByVal CalculatedFrequency As Integer)
* Public Sub SetFrequencyPRCNT(ByVal CalculatedFrequency As Single)

The cWord Class:

This class contains properties assigned to words once they have been split up. These properties include: the word itself (written as a string), the index of the first letter of the word (in regards to the element index of the MyLetters array) and the length of the word in regards to how many letters the word contains. Its methods include setters to initialise the words and other properties. Along with a function which returns the only values that English one letter words can represent (of course only run in the event that a word is only a single letter long).

Public Class cWord (class definition):

Properties:

* Private pWordStart As Integer
* Private pLength As Integer
* Private pWord As String

Methods:

* Public Sub New()
* Public Sub WordInitialisation(ByVal FirstLetterIndex As Integer, ByVal NewWord As String)
* Public Function GetLetters(ByVal position As Integer)
* Public Function GetLetters(ByVal position As Integer, ByVal NumberOfLetters As Integer)
* Public Function IfSingleLetters() As String

The cLikelyLetters Class:

This class contains all the code pertaining to the use and processing of digrpahs found in the code. A digraph is a pair of letters that lie next to eachother in the text without a space or any other character between them. Digraphs are integral for calculating possible letters that can be represented but they have unique rules and values so are better handled outside of the main project. Its properties include structured digraph arrays (that contain all data pertaining to digrpahs, i.e. their frequency, the digraph itself, the position of the digraph in the text and whether or not it has already been processed. It methods include taking the 30 most common digrpahs from the text and storing multiple likely digraphs that could represent the cipher digraph. There is also proceudres that handle digrpahs containing only a single letter repeated as these “double digraphs” (as they are reffered to in the program) require a different approach to decode. When finished being processed the most likely values for each letter present in each digraph is outputted.

Public Class cLikelyLetters (class definition)

Variable Structures:

* Structure LetterWeight
  + Dim PlaintextLetter As Char
  + Dim NumberOfTestsPassed As Integer
  + Dim PercentageLikelihood As Single
  + Dim FOUND As Boolean
* Structure Digraph
  + Dim LetterPair As String
  + Dim Occurance As Integer
  + Dim LikelyCorrespondance As String
  + Dim DoublePos As Integer
  + Dim Processed As Boolean

Properties:

* Private AllLetters(26) As LetterWeight
* Private CommonDigraphs(30) As Digraph
* Private Correspondances() As String
* Private NumberOfTests As Integer
* Const Alphabet As String = "ABCDEFGHIJKL MNOPQRSTUVWXYZ"
* Const AlphabetInFreqOrder As String = "ETAOINSRHDLUCMFYWGPBVKXQJZ"
* Const AlphabetInBeginningFreqOrder As String = "TAOISWCBPHFMDRELNGUVYJKQXZ"
* Const MostCommonDigrams As String = "THHEINERANREEDONESSTENATTONTHANDOUEANGASORTIISETITARTESEHIOF"
* Const MostCommonDoubleDigraphs As String = "SSEETTFFLLMMOO"

Methods:

* Public Sub AddlikelihoodOfLetters(ByVal NewLetter As Char)
* Public Sub CompareDigraphs(ByVal LoneDigraph As String, ByVal PositionSorted As Integer)
* Public Function DigraphsAndLetters(ByVal IndivLetter As Char)
* Public Sub DoubleDigraphs(ByVal principleLetter As Char)

The HelpForm Class:

While the HelpForm is in fact a secondary form it is still a class containg properties and methods so is included here. This class contains minimal code or object upon the form, the purpose of this form is to allow the user to have a step by step guide as to how to decrypt a cipher in this method and general guide that can be applied to any cipher that uses monoalphabetic cubstitution. There are three buttons that a user can select which will display text relating to whatever stage of the decryption that the user wishes to learn about. This is displayed in a large rich text box. The form is shown when a “TUTORIAL” button is pressed by the user on the main form (form1).

Public Class HelpForm (class definition):

Properties:

* Private Const BEGIN As String
* Private Const GETTINGSTARTED As String
* Private Const PROGRESSING As String
* Private Const FINISH As String

Methods:

* Private Sub HelpForm\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
* Private Sub GettingStartedButton\_Click(sender As System.Object, e As System.EventArgs) Handles GettingStartedButton.Click
* Private Sub ProgressingButton\_Click(sender As System.Object, e As System.EventArgs) Handles ProgressingButton.Click
* Private Sub FinalisingButton\_Click(sender As System.Object, e As System.EventArgs) Handles FinalisingButton.Click

The Form 1 Class:

This form contains the main user interface and allows the user to see certain pieces of information and allows the user to enter data for processing based on the data provided. Instead of using multiple forms for different stages the program may be in, this form “hides” and “shows” various objects based on what data needs to be entered and whate the user requires in order to enter said data. For instance, upon the loading of the form (this occurs on program startup because Form1 is the designated startup form) the user is shown a text box and a button with the instruction to enter the file name belonging to the cipher that the user wishes to process, this is the first required piece of information, after this the text box, button and label are hidden and replaced with the objects required for decryption (cipher text box, combo boxes, list box and so on). Some event handling involves reseting all changes the user has made, changing the interface to allow the user to save the cipher into a text file, changing the letter under scrutiny and selecting the letter that will replace it. This form requires the “likelyLetters” string that has been calculated by various other classes and collated in the cProject. It uses this to find the most likely letters (characters to display in the CorrespondingPlaintextComboBox) that are related to the previously selected target character (the selected item in the ReplaceLetterComboBox). Using the event handlers the form allows the user to input only a select range of characters from the comboboxes (for defensive programming / error prevention) to select and change any particular letter from the textbox, to whatever letter provided that the user deems to fit best. The form also allows the user to step back to negate their own last action. Along with resetting all of their changes made to the original file contents. Furthermore there is a listbox that outlines all of the changes that the user has made in clear English. Beyond this the user can also save the file and open the HelpForm and there is a recursive subroutine which manages to eliminate most improbable cases where the “CorrespondingPlaintextComboBox” may not include the true letter that the cipher character represents.

Public Class Form1 (class definition):

Properties:

* Private WhatIsBeingManipulated As Boolean
* Private BASEPROJECTINSTANCE As cProject
* Private TextIncomplete As String
* Private Lettersfound As String

Methods:

* Private Sub Form1\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
* Private Function FileExists(ByVal CipherORPlainText As Boolean) As Boolean
* Private Sub ReadFileButton\_Click(sender As System.Object, e As System.EventArgs) Handles ReadFileButton.Click
* Private Sub IsCipherButton\_Click(sender As System.Object, e As System.EventArgs)
* Private Sub IsPlaintextButton\_Click(sender As System.Object, e As System.EventArgs)
* Private Sub ReplaceLetterComboBox\_SelectedIndexChanged(sender As System.Object, e As System.EventArgs) Handles ReplaceLetterComboBox.SelectedIndexChanged
* Private Sub ResetButton\_Click(sender As System.Object, e As System.EventArgs) Handles ResetButton.Click
* Private Sub CorrespondingPlaintextComboBox\_SelectedIndexChanged(sender As System.Object, e As System.EventArgs) Handles CorrespondingPlaintextComboBox.SelectedIndexChanged
* Private Sub ChangeCorrespondanceNearlyDone()
* Private Sub UnfoundLettersRECURSIVE(ByVal letterUNFound As Char, ByVal AllUnfoundLetters As String, ByVal FoundLetters As String)
* Private Sub BackButton\_Click(sender As System.Object, e As System.EventArgs) Handles BackButton.Click
* Private Sub TutorialButton\_Click(sender As System.Object, e As System.EventArgs) Handles TutorialButton.Click
* Private Sub SaveButton\_Click(sender As System.Object, e As System.EventArgs) Handles SaveButton.Click
* Private Sub SaveNameFileButton\_Click(sender As System.Object, e As System.EventArgs) Handles SaveNameFileButton.Click

The cProject Class:

This class contains all of the processes required to find the “likely correspondence” for each letter in the cipher text. To begin the class opens the file once the user has inputed a valid file name, this is done in two parts: the assembly of the entire file directory (given the user only inputs the name of the file) which will be checked to see if the file does indeed exist, then the opening of the file with the provided directory for read access (i.e. the program can read from the file but not write to it). The entire contents of the file are saved into a string and outputted after which the file is closed. Each seperate character in the file is also saved in its own array which contains the letter itself and the letters before and after it. This array is sorted and the frequency that each letter appears in the provided cipher is stored in that same array. Another array contains each of the individual letters (only the letters of the alphabet if, for example, all are used in the cipher) they contain the letter and information about that letter including their frequency of appearance and the likelyletters that have been calculated and attributed to it. The likely letters are first caluculated by comparing the frequency of the letter occurring in the cipher with the relative frequency of letters occurring in the English languae. The top 5 most appropriate letters are saved for each cipher character. After this the entire cipher is split into words and compared (with any one letter words having a permanent “LikelyLetters” variable containing only “IA”). The frequency of a letter in the cipher occurring at the beginning of a word is compared with the relative frequencies of letters at the beginnings of words. More possible likely letters are added based on this comparion. After this the cipher is split into different digraphs/ digrams and saved into an array, the top 30 most commonly occurring are found and passed into the cLikelyLetters class for processing. This should return a range of likely letters that have been calculated through the various digraph frequency rules. All of the likely letters obtained from all methods are compared, if there are any repeats those letters are seen to be more likely, given they have passed multiple tests, and put at the top of the likely letters string so that they will be displayed first on the screen. As well as repositioning repeated letters, the repeated occurences of those letters are removed leaving only one letter. This creates a string of unique letters in decreasing order of likelihood for easy manipulation once it has been displayed on the form.

Public Class cProject (class definition)

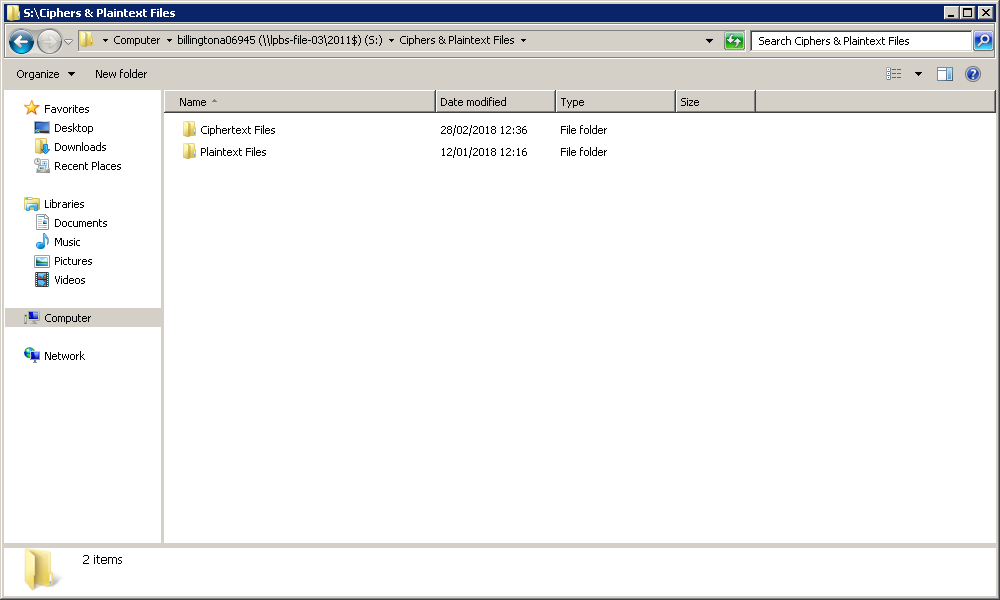
Properties:

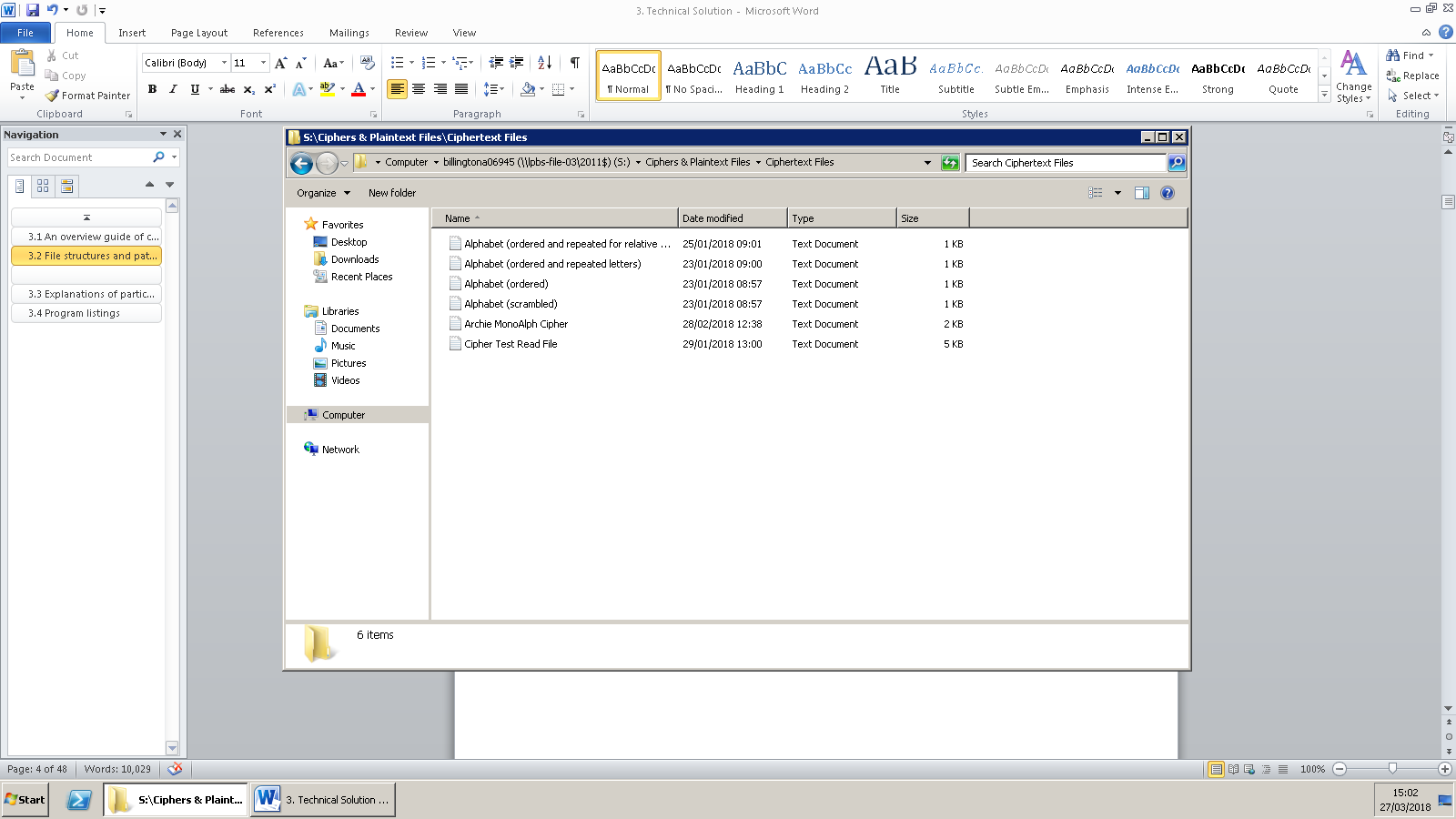
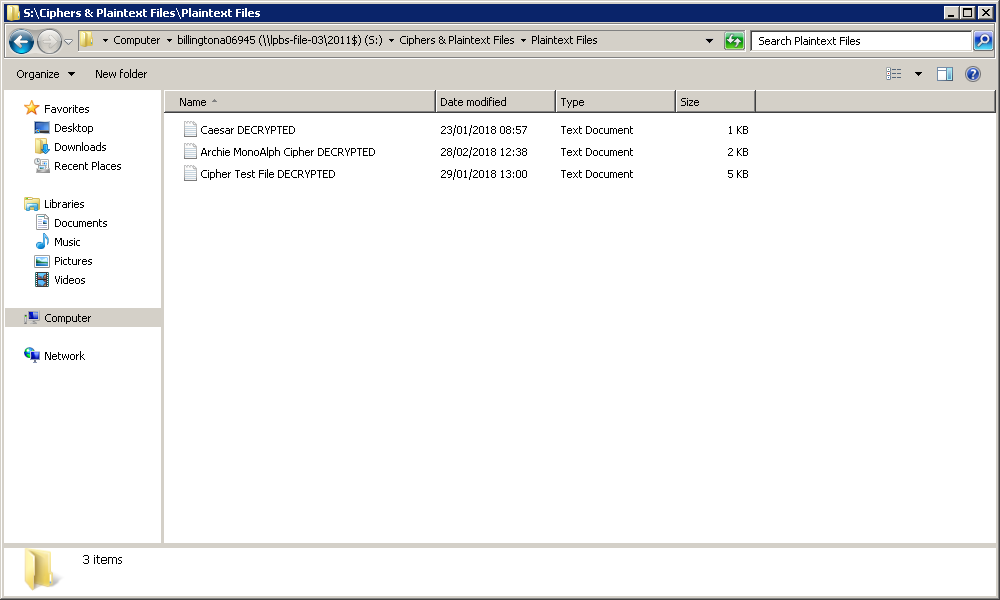
* Private MyLetters() As cLetter
* Private MyLettersSORTED() As cLetter
* Private IndividualLetters() As AllIndividiualLetters
* Private MyWords() As cWord
* Private LetterBreakdown As cLikelyLetters
* Private NumberOfLetters As Integer
* Private NumberOfWords As Integer
* Private Digraphs() As String
* Private DigraphOccurances() As Integer
* Const FilePath As String = "s:\\Computing\Coursework\Ciphers & Plaintext Files"
* Const CipherFolderName As String = "\Ciphertext Files\"
* Const PlaintextFolderName As String = "\Plaintext Files\"
* Const AlphabetInFreqOrder As String = "ETAOINSRHDLUCMFYWGPBVKXQJZ"
* Const AlphabetInBeginningFreqOrder As String = "TAOISWCBPHFMDRELNGUVYJKQXZ"

Methods:

* Public Sub New()
* Public Function GetIndividualLetters() As String
* Public Function GetIndivCorrespondance(ByVal ComboLetter As Char) As String
* Public Function GetNumberOfLetters() As Integer
* Public Sub AddLetter(ByRef SubjectArray() As cLetter)
* Public Sub MergeSort(ByVal FrontPointer As Integer, ByVal EndPointer As Integer, ByVal LettersOrFrequency As Integer)Private Sub RadixSortLetters()
* Private Sub MergeLetters(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)
* Private Sub MergeFrequencies(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)
* Private Sub MergeBeginningFrequencies(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)
* Private Sub MergeDigraphs(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)
* Private Function BinarySearchFrequency(ByVal Target As String, ByVal Lowest As Integer, ByVal highest As Integer) As Integer
* Private Sub CalculateLetterFrequencyINT(ByVal LetterBeingUsed As Char)
* Private Function CalculateLetterFrequencyPRCNT(ByVal FrequencyOfLetterINT As Integer) As Single
* Public Sub FinalisingInitialisationSAVEFREQ()
* Public Function GetFileName() As String
* Private Function GetFileNameExtended(ByVal CipherORPlainText As Boolean) As String
* Public Function ReadFile() As String
* Public Sub MostLikelyBySingleFrequency()
* Public Sub RemoveNullSpacesInIndivLetters()
* Public Sub SplitIntoWords()
* Public Sub SplitIntoDigraphs()
* Public Sub FirstLetterRules()
* Public Sub AnyRepeats()

# 3.2 File structures and path names with running instructions

There are two folders required for the program to run. They are placed in a single umbrella folder called “Ciphers & Plaintext Files”, which in turn is placed on the hard drive of the users machine. The two folders that are found within the “Ciphers & Plaintext Files” folder are called “Ciphertext Files” and “Plaintext Files” respectively.

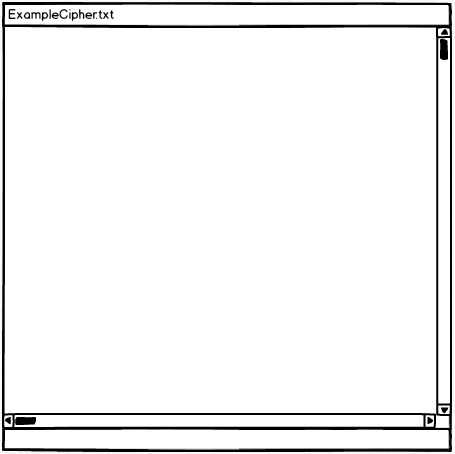
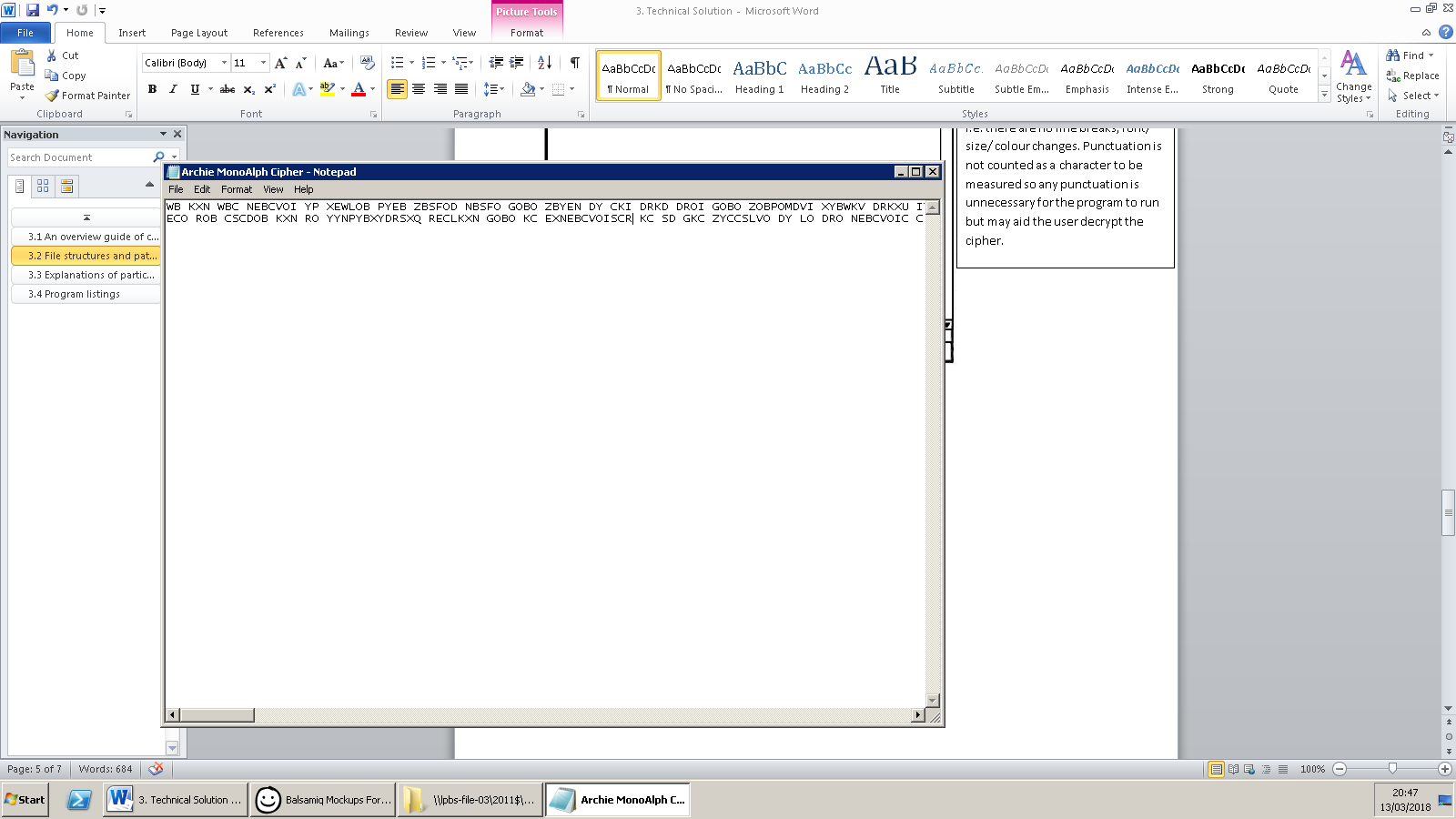


This shows the layout of the folder “Plaintext Files” in which are multiple text files which have been saved by the user through completinon of the deciphering program. Saved as a text file with only characters shown in the text box, and no additional formatting.

This shows the layout of the folder “Ciphertext Files” in which are multiple text files containing a cipher with no extra formatting. Only text files are used by the program so only text files are needed to be stored here. Below are some examples of files used in the creation and testing of the program

This shows the structure of the folder “Ciphers & Plaintext Files” in which is only the two other folders “Ciphertext Files” and “Plaintext Files”

|  |  |
| --- | --- |
| File | Purpose and Contents |
| ExampleCipher.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Ciphertext Files\ExampleCipher.txt | A text document containing a mono-alphabetic cipher. Used as a worked example cipher ready for decryption by the user so they may implement the skills step by step as guided by the program. |
| TutorialCipher.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Ciphertext Files\TutorialCipher.txt | A text document containing a mono-alphabetic substitution cipher. This cipher has the ideal frequencies of letter and digraph occurrence so that the user can work through it using the help information quickly and easily so that the user can understand the concept of frequency analysis and how the program works. |
| ExampleCipherSOLVED.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Plaintext Files\ExampleCipherSolved.txt | A text document which contains the English plaintext calculated from the file “ExampleCipher.txt”. It is outputted directly by the program after the “ExampleCipher” has been solved. |
| TutorialCipherSOLVED.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Plaintext Files\ExampleCipher.txt | A text document which contains the English plaintext calculated from the file “TutorialCipher.txt”. It is outputted directly by the program after the “TutorialCipher” has been solved. |
| EncryptedCipher1.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Ciphertext Files\ExampleCipher.txt | A text document containing a cipher that a user can create / obtain for them to work through themselves within the program. Must be stored in the same folder as all other cipher files.  This file name is an example and the user has the option to name it anything they choose. |
| EncryptedCipher1SOLVED.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Plaintext Files\ExampleCipher.txt | A text document containing the English plaintext that the user and the program have calculated and then changed and stored based on the cipher in the file “EncryptedCipher1.txt”. This file name is an example and the user has the option to name it anything they choose. |
| UnfinishedDecryptionOfEncryptedCipher1.txt  C:\Users\ME\Documents\Ciphers & Plaintext Files\Ciphertext Files\ExampleCipher.txt | This is a text document containing the partially decrypted cipher of “EncryptedCipher1.txt” the user has the option to save their progress at any point during their decryption to come back to at a later time. The program will calculate the letters that have been changed by the user from the original file as likely values for the plaintext regardless so all the same values and options will be identical to those the program supplied during the encryption of the initial file.  This file name is an example and the user has the option to name it anything they choose. |

This mock-up file to the left in approximation of what a particular file may look like. This is included in this section to point out that there is no formatting on the text in this file. I.e. there are no line breaks, font/ size/ colour changes. Punctuation is not counted as a character to be measured so any punctuation is unnecessary for the program to run but may aid the user decrypt the cipher.

This actual example of the same file shows that there is no break line where the text will continue on the next line when the length of the text exceeds the length of the form. By using a return line the program counts this as an extra character. For absolute accuracy the return lines are omitted in the file. This means that the only text that has to be entered is the text waiting to be processed by the program.

These files are located in one of two folders (“Plaintext Files” & “Ciphertext Files”) which sort the finished saved files from the yet to be processed plaintext files. All files in the plaintext folder are only written to, not opened by the program. The majority of the cipher text files are only opened and not written to, except in the case of saving an unfinished cipher file to the cipher folder. This method of using two folders allows greater organisation in anticipation of the user’s ease. After high usage many files may be created and two folders allows the increased organisation without the user needing to change the file paths in the code.

WB KXN WBC NEBCVOI YP XEWLOB PYEB ZBSFOD NBSFO GOBO……… ZBYEN DY CKI DRKD DROI GOBO ZOBPOMDVI XYBWKV DRKXU IYE….. TECD NSNXD RYVN GSDR CEMR XYXCOXCO WB NEBCVOI GKC DRO….

# 

# 3.3 Explanations of particularly difficult code sections

|  |  |  |
| --- | --- | --- |
| Algorithm | Pseudo Code | Actual Code |
| Radix SortLetters  This algorithm is a highly efficient way of sorting the large array of letters that can sometimes be up to and in excess of 5000 elements (for every character in the text). Radix sorts group the data in terms of their most significant digit and sort through the groups in regards to their second most significant digit. In this program the data being sorted is in the form of an array of characters. To make these into integer form to sort them in this manner I have used the function (Asc()) to provide the comparison with the ASCII values for each character. | FOR counter = 31 to -1 STEP -1  IndexJ = 0  FOR IndexI = 0 to NumberOfLetters -1  Move = leftshift(Letters(i))  IF WHEN counter = 0, NOT move = true THEN  Letters(IndexI – IndexJ) = Letters(indexI)  ELSE  Temp(indexJ) = Letters(indexI)  IndexJ = IndexJ + 1  END IF  NEXT  Letters = Temp  NEXT | Dim i As Integer, j As Integer  Dim temp As cLetter() = New cLetter(MyLettersSORTED.Length - 1) {}  For shift = 31 To -1 + 1 Step -1  j = 0  For i = 0 To MyLettersSORTED.Length - 1  Dim move As Boolean = (Asc(MyLettersSORTED(i).PrincipleLetterInitialisation()) << shift) >= 0  If If(shift = 0, Not move, move) Then MyLettersSORTED(i - j) = MyLettersSORTED(i)  Else  temp(j) = MyLettersSORTED(i)  j += 1  End If  Next Array.Copy(temp, 0, MyLettersSORTED, MyLettersSORTED.Length - j, j)  Next |
| Merge Sort  This sort uses abstraction to remove many comparisons which make bubble sorts so inefficient. A merge sort accomplishes this by splitting the data set into groups of 1. Then comparing each of those “groups” with its neighbouring groups and assembling those groups in the correct order so that the data is assembled in the sorted order with significantly less iterations and runs than a less efficient bubble sort. | FOR counter = 0 to 27  Temp(counter) = Array(counter)  NEXT  Left = start  Right = Middle + 1  Main = start  WHILE left <= middle AND right <= End  IF temp(left) >= temp(right) THEN  Array(main) = Temp(left)  ELSE  Array(main) = Temp(right)  right = right + 1  END IF  Main = Main + 1  REPEAT  IF left <= middle THEN  FOR Counter = 1 to middle – left + 1  Array(main +Counter– 1) = temp(left + counter -1)  REPEAT  ELSE  FOR Counter = 1 to end – right +1  Array(main +Counter– 1) = temp(right + counter -1)  REPEAT  END IF | Dim TempArray(28) As AllIndividiualLetters  Dim LeftCounter As Integer  Dim RightCounter As Integer  Dim MainCounter As Integer  TempArray(28) = New AllIndividiualLetters  For looper = 0 To 27  TempArray(looper) = IndividualLetters(looper)  Next  LeftCounter = StartPoint  RightCounter = CentrePointer + 1  MainCounter = StartPoint  Do While (LeftCounter <= CentrePointer) And (RightCounter <= EndPoint)  If (TempArray(LeftCounter).FrequencyINT >= TempArray(RightCounter).FrequencyINT) Then  IndividualLetters(MainCounter) = TempArray(LeftCounter)  LeftCounter += 1  Else  IndividualLetters(MainCounter) = TempArray(RightCounter)  RightCounter += 1  End If  MainCounter += 1  Loop  If LeftCounter <= CentrePointer Then  For looper = 1 To CentrePointer - LeftCounter + 1  IndividualLetters(MainCounter + looper - 1) = TempArray(LeftCounter + looper - 1)  Next  Else  For looper = 1 To EndPoint - RightCounter + 1  IndividualLetters(MainCounter + looper - 1) = TempArray(RightCounter + looper - 1)  Next  End If |
| Binary SearchFrequencies  Efficiently searches through the large MyLettersSORTED array to find relevant data quickly. The algorithm works by inspecting the middle item of the array first, then if the value being searched for is less (where it will inspect the central value of the lower quartile) or greater where it will inspect the central value of the upper quartile) and continue until found. Much more efficient (in regards to time complexity and memory usage than a linear search). | IF low > high THEN  RETURN 0  ELSE  Middle = (low + high) INTDIV 2  IF Target = Array(middle) THEN  RETURN middle  ELSE  IF target < array THEN  CALLSELF(Target, low, middle -1)  ELSE  CALLSELF(Target, middle +1, high)  END IF  END IF  END IF | Dim Middle As Integer  If Lowest > highest Then  Return 0  Else  Middle = ((Lowest + highest) \ 2)  If Target = MyLettersSORTED(Middle).PrincipleLetterInitialisation() Then  Return Middle  Else  If Target < MyLettersSORTED(Middle).PrincipleLetterInitialisation() Then  BinarySearchFrequency(Target, Lowest, Middle - 1)  Else  BinarySearchFrequency(Target, Middle + 1, highest)  End If  End If  End If |
| UnfoundLettersRecursive  This algorithm uses recursion to find all the letters which have yet to be entered by the user (using a variable containing all of the found characters) it then returns the unfound letters so that all items in the correspondingplaintextcombobox to the unfound letters regardless of the values of the other combo box selected. | FOR Counter1 = 1 to LetterF.LENGTH  FOR Counter2 = 1 to LetterU.LENGTH  IF MID(letterU, counter2, 1) = Target THEN  TEMP = letterU  LetterU = MID (temp, 1, counter 2 -1)  LetterU &= MID (temp, Counter2 +1)  LetterF = MID (letterF, 2)  END IF  REPEAT  REPEAT  IF LetterF.length <> 0 then  CALLSELF(MID(LetterF,1,1), LetterU, letterF)  END IF  IF LetterU.LENGTH = 26 – LetterF.LENGTH THEN  OuputString = LetterU  END IF | For FoundLooper = 1 To FoundLetters.Length  For UnfoundLooper = 1 To AllUnfoundLetters.Length  If Mid(AllUnfoundLetters, UnfoundLooper, 1) = letterUNFound Then TempUnfoundLetters = AllUnfoundLetters AllUnfoundLetters = Mid(TempUnfoundLetters, 1, UnfoundLooper - 1)  AllUnfoundLetters &= Mid(TempUnfoundLetters, UnfoundLooper + 1)  FoundLetters = Mid(FoundLetters, 2)  End If  Next  Next  If FoundLetters.Length <> 0 Then UnfoundLettersRECURSIVE(Mid(FoundLetters, 1, 1), AllUnfoundLetters, FoundLetters)  End If  If AllUnfoundLetters.Length = 26 - Lettersfound.Length Then  LettersUNfound = AllUnfoundLetters  End If |
| Save File  After pressing the save button the screen will change to the start-up screen where the user has a text box to enter the file name and a button to enter that name. Also a combo box containing whether or not the cipher has been completely deciphered or not. Then saves the contents of the ciphertext box in the corresponding file. | IF combobox.SELECTEDITEM = TRUE then  GET CTEXT FILEDIRECTORY  ELSE  GET PTEXT FILEDIRECTORY  END IF  GET FULLFILEPATH  FILEOPEN(FULLFILEPATH, OUTPUTACCESS)  WRITE(Textbox.TEXT)  FILECLOSE | FinishedComboBox.Show()  If FinishedComboBox.SelectedItem = "Finished" Then  FileDirectory = FilePath  FileDirectory &= CipherFolderName  Else  FileDirectory &= PlaintextFolderName  End If  FileDirectory &= BASEPROJECTINSTANCE.GetFileName()  FileOpen(2, FileDirectory, OpenMode.Output)  Print(CipherTextBox.Text)  FileClose(2) |
| ReadFile  This function opens the cipher file for read access. Saves the contents of said file to a string type variable. After the contents are saved the file is closed however the variables are not yet initialised correctly. The filecontents variable is then used to add data to the MyLetters array. This array is structured such that each indexed element has variables that store every character in the file, ever letter before and after each letter as well. This may seem excessive but it makes for easier access later in the program. | FILEOPEN(FULLFILEPATH, INPUTACCESS)  DO UNTIL EOF  wholeFile = GetFIleContents  REPEAT  FILECLOSE  FOR counter = 0 to NumberOFletters -1  IF counter <> 0 THEN  AddLetter(Array)  END IF  Array(looper) = wholefile(looper)  IF counter <> 0 THEN  Array(counter).PLETTER = wholefile(looper – 1)  END IF  IF counter <> 0 THEN  Array(counter).SLETTER = wholefile(looper + 1)  END IF  REPEAT  ArraySorted = array  RETURN wholeFile | Dim WholeFile As String  Dim LetterCounter As Integer  LetterCounter = 0  FileOpen(1, GetFileNameExtended(True), OpenMode.Input)  Do Until EOF(1)  WholeFile &= LineInput(1)  Loop  FileClose(1)  WholeFile = UCase(WholeFile)  For looper = 0 To WholeFile.Length - 1  If looper <> 0 Then  AddLetter(MyLetters)  End If MyLetters(looper).PrincipleLetterInitialisation(Mid(WholeFile, looper + 1, 1))  If looper <> 0 Then MyLetters(looper).PreceedingLetter(MyLetters(looper - 1).PrincipleLetterInitialisation())  End If  If looper <> 0 Then  MyLetters(looper -1).SubsequentLetter(MyLetters(looper).PrincipleLetterInitialisation())  End If  Next  NumberOfLetters = WholeFile.Length  MyLettersSORTED(0) = MyLetters(0)  For Looper = 1 To NumberOfLetters - 1 AddLetter(MyLettersSORTED) MyLettersSORTED(Looper) = MyLetters(Looper)  Next  Return WholeFile |
| FileExists  A procedure that uses defensive programming to see if a file exists using the built in function [ File.exists( ) ]. If so returns true so that the file can be used. Else the user must entire the name of an existing file until one the program accepts it. | IF cORp = TRUE THEN  FileDirectory &= CName  ELSE  FileDirectory &= PName  END IF  FileDirectory &= FILENAME  IF File.Exists(filedirectory) = TRUE THEN  RETURN TRUE  END IF | Dim FileDirectory As String  FileDirectory = FilePath  If CipherORPlainText = True Then  FileDirectory &= CipherFolderName  Else  FileDirectory &= PlaintextFolderName  End If  FileDirectory &= BASEPROJECTINSTANCE.GetFileName()  If IO.File.Exists(FileDirectory) = True Then  Return True  End If |
| BackButton\_Click Removes last value from ChangedLetterListBox and returns all occurrences of the changed letter in the text to its original letter, effectively reversing the change last made to that letter in the text. The issue with this button is that it does not always step back one place in the event that it is used after the same letter has been changed multiple times. Essentially resets the letter to its original cipher value. Defensive programming is used such that the button does nothing when pressed with nothing changed yet. | IF ListBox.item(0).text <> NOTHING THEN  ListBox.removeLast  Last = 1  FOR counter = Last to Numberofletters  IF RLcombobox.selected = MID(text,counter, 1) THEN  Textbox.select(counter – 1,1)  TextBox.selected = RLcombobox.selected  END IF  REPEAT  Last = 1  FOR counter = last to numberofletters  IF RLcombobox.selected = mid(text, counter, 1) THEN  Textbox.select(counter – 1,1)  Textbox.selectedcolour = DARKRED  END IF  REPEAT  END IF | Dim PositionLast As Integer  Dim PartialCompletedText As String  Dim TempText As String  Dim TempAlteredText As String  If ChangedLettersListBox.Items(0).text <> Nothing Then  ChangedLettersListBox.Items.RemoveAt(ChangedLettersListBox.Items.Count - 1)  PositionLast = 1  For looper = PositionLast To CipherTextBox.Text.Length  If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then  CipherTextBox.Select(looper - 1, 1)  CipherTextBox.SelectedText = ReplaceLetterComboBox.SelectedItem  End If  Next  PositionLast = 1  For looper = PositionLast To CipherTextBox.Text.Length  If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then  CipherTextBox.Select(looper - 1, 1)  CipherTextBox.SelectionColor = Color.DarkRed  End If  Next  End If |
| CorrespondingPlaintext  ComboBox\_SelectedIndexChanged  Finds all occurrences of a single letter within the cipher text and replaces it with the changed letter. Rewrites the new changed cipher to the screen. Also selects all letters that have been changed to font colour blue and the rest of the letters to font colour dark red. If more letters are changed then all the letters that differ to the original cipher are written in blue. | Textbox.FONTCOLOUR = DARKRED  Last = 1  FOR counter = Last to Numberofletters  IF RLcombobox.selected = MID(text,counter, 1) THEN  Textbox.select(counter – 1,1)  TextBox.selected = CPTcombobox.selected  END IF  REPEAT  Last = 1  FOR counter = last to numberofletters  IF RLcombobox.selected = mid(text, counter, 1) THEN  Textbox.select(counter – 1,1)  Textbox.selectedcolour = BLUE  END IF  REPEAT  IF CPTcombobox.selected IS NOT IN LetterF THEN  LetterF &= CPTcombobox.selected  END IF  Listbox.add(RLcombobox.selected & “changed to:” & CPT.selected)  IF LetterF.length >= SingnificanceConstant THEN  CALL(UnfondLettersRECUSRIVE(“A”, “ABCDEFGHIJKLMNOPQRSTUVWXYZ”, LetterF)  CALL(ChangeCorrespondance)  END IF | CipherTextBox.ForeColor = Color.DarkRed  PositionLast = 1  For looper = PositionLast To CipherTextBox.Text.Length  If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then CipherTextBox.Select(looper - 1, 1) CipherTextBox.SelectedText = CorrespondingPlaintextComboBox.SelectedItem  End If  Next  PositionLast = 1  For looper = PositionLast To CipherTextBox.Text.Length  If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then CipherTextBox.Select(looper - 1, 1) CipherTextBox.SelectionColor = Color.Blue  End If  Next  If InStr(LettersFound, CorrespondingPlaintextComboBox.SelectedItem) = 0 Then  LettersFound &= CorrespondingPlaintextComboBox.SelectedItem  End If  ChangedLettersListBox.Items.Add(ReplaceLetterComboBox.SelectedItem & " changed to: " & CorrespondingPlaintextComboBox.SelectedItem)  If Lettersfound.Length >= HowManyLettersBeFoundFirst Then UnfoundLettersRECURSIVE("A"c, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", Lettersfound) ChangeCorrespondanceNearlyDone()  End If |
| DoubleDigraphs  Only few letters actually occur consecutively in English such as “TT”, “SS” and so on and so forth. This section of code cycles through all of the digraphs and sees which of them have the same letters twice repeated. These digraphs will be counted to find how often they occur. Once this is done they will be compared to a constant containing the most common double digraphs. Using this constant this algorithm calculates the most likely values that could represent the actual letter in the digraph and then sends that to the likelycorrespondance variable saved within the class. | Digraph = Letter1 + letter2  Pos = 0  FOR counter – 0 to Array.length – 1  IF MID(array(counter), 1,1) = MID(array(counter),2,1) THEN  Pos = pos + 1  Array(counter).pos = pos  END IF  REPEAT  FOR counter = 0 to array.length -1  IF Digraph = array(counter) THEN  Array(counter).LC = MID(DigraphCONST, Array(counter).pos \* 2, 2)  IF Array(counter).pos <> 1) OR array(counter).pos <> DigrpahCONST.length / 2 THEN  Array(counter).LC &= MID(digraphCONST, array(looper).pos \* 2) +2, 2)  Array(counter).LC &= MID(DigraphCONST, array(looper).pos \*2) -2, 2)  END IF  IF array(counter).pos = 1) THEN  array(counter).LC &= MID(digraphCONST, array(counter).pos \*2) + 2,2)  Array(counter).LC &= MID(digraphCONST,  array(looper).pos \*2) -2, 2)  END IF  IF array(counter).pos = len(digraphCONST) / 2 THEN  array(counter).LC &= MID(digraphCONST, array(counter).pos \*2) + 2,2)  Array(counter).LC &= MID(digraphCONST,  array(looper).pos \*2) -2, 2)  END IF  END IF  REPEAT | Dim cipherdigraph As String  Dim Position As Integer  cipherdigraph = principleLetter & principleLetter  Position = 0  For looper = 0 To CommonDigraphs.Length - 1  If Mid(CommonDigraphs(looper).LetterPair, 1, 1) = Mid(CommonDigraphs(looper).LetterPair, 2, 1) Then  Position += 1 CommonDigraphs(looper).DoublePos = Position  End If  Next  For looper = 0 To CommonDigraphs.Length - 1  If cipherdigraph = CommonDigraphs(looper).LetterPair Then CommonDigraphs(looper).LikelyCorrespondance = Mid(MostCommonDoubleDigraphs, CommonDigraphs(looper).DoublePos \* 2, 2)  If (CommonDigraphs(looper).DoublePos <> 1) Or (CommonDigraphs(looper).DoublePos <> (Len(MostCommonDoubleDigraphs) / 2)) Then CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2) CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)  End If  If CommonDigraphs(looper).DoublePos = 1 Then CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2) CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)  End If  If CommonDigraphs(looper).DoublePos = Len(MostCommonDoubleDigraphs) / 2 Then CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2) CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)  End If  End If  Next |

# 

# 3.4 Program listings

Public Class Form1

Private WhatIsBeingManipulated As Boolean

Private BASEPROJECTINSTANCE As cProject

Private TextIncomplete As String

Private Lettersfound As String

Private LettersUNfound As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Const HowManyLettersBeFoundFirst As Integer = 8

Const FilePath As String = "s:\\Ciphers & Plaintext Files"

Const CipherFolderName As String = "\Ciphertext Files\"

Const PlaintextFolderName As String = "\Plaintext Files\"

Private Sub Form1\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

'upon load hide all unnecessary objects on the form

'Show objects required for the user to proceed

CipherTextBox.Hide()

PickLetterLabel.Hide()

ReplaceLetterComboBox.Hide()

CorrespondingPlaintextComboBox.Hide()

ResetButton.Hide()

ChangedLettersListBox.Hide()

ChangedLetterListBoxLabel.Hide()

BackButton.Hide()

TutorialButton.Hide()

SaveButton.Hide()

SaveNameFileButton.Hide()

FinishedComboBox.Hide()

ReadFileButton.Show()

FileNameTextBox.Show()

FileNameInstructionLabel.Show()

FileNameTextBox.Select()

BASEPROJECTINSTANCE = New cProject

End Sub

Private Function FileExists(ByVal CipherORPlainText As Boolean) As Boolean

Dim FileDirectory As String

FileDirectory = FilePath

'READWRITE = TRUE ==== CIPHERTEXT FILES

'READWRITE = FALSE ==== PLAINTEXT FILES

'assembles the file path based on whethere the user wishes to process a file from the cipher text folder of the plaintext folder

If CipherORPlainText = True Then

FileDirectory &= CipherFolderName

Else

FileDirectory &= PlaintextFolderName

End If

FileDirectory &= BASEPROJECTINSTANCE.GetFileName()

'checks to see if the file exists before continuing

If IO.File.Exists(FileDirectory) = True Then

Return True

End If

End Function

Private Sub ReadFileButton\_Click(sender As System.Object, e As System.EventArgs) Handles ReadFileButton.Click

Dim AllLetters As String

Dim numberofletters As Integer

'if the file does exist then run the main bulk of code:

If FileExists(True) = True Then

'the text is set to the text read file from the file in cProject and then returned

TextIncomplete = BASEPROJECTINSTANCE.ReadFile()

'changes interface

'hides file related objects

ReadFileButton.Hide()

FileNameInstructionLabel.Hide()

FileNameTextBox.Hide()

'set text in textbox equal to the returned text

CipherTextBox.Text = TextIncomplete

'obtains number of letters

numberofletters = BASEPROJECTINSTANCE.GetNumberOfLetters

'========================================

'UNSURE

'true means sort by letters

'MERGESORT TOO SLOW

'Radix sort used

'sort letters array

BASEPROJECTINSTANCE.MergeSort(0, numberofletters - 1, 0)

'=======================================

'MERGESORT TAKES TOO LONG ^^^^

'CAN SHOW THE TEXT ON SCREE OR SOME KIND OF LOADING SCREEN OR ESTIMATED TIME WHILE MERGESORT TAKES PLACE

'========================================

'Finish initialisation

'Finish initialising the program by adding data to varaibles and finding the frequency of letter occurence

BASEPROJECTINSTANCE.FinalisingInitialisationSAVEFREQ()

'======================================

AllLetters = BASEPROJECTINSTANCE.GetIndividualLetters()

'GET LETTERS FOR COMBO BOX

'adds items to the combo box based on the data provided by allLetters string

'Adds each item seperately so that they only contain characters

For looper = 0 To Len(AllLetters) - 1

ReplaceLetterComboBox.Items.Add(Mid(AllLetters, looper + 1, 1))

Next

'SHOW INTERFACE

'display all decryption related objects on the screen

'show buttons for saving, tutorial forms, text boxes and comboboxes

CorrespondingPlaintextComboBox.Show()

ReplaceLetterComboBox.Show()

CipherTextBox.Show()

ResetButton.Show()

ChangedLettersListBox.Show()

ChangedLetterListBoxLabel.Show()

BackButton.Show()

SaveButton.Show()

TutorialButton.Show()

'finish calculating the most common plaintext letters each letter of the cipher alphabet can represent

BASEPROJECTINSTANCE.MostLikelyBySingleFrequency()

'call project and split into words

BASEPROJECTINSTANCE.SplitIntoWords()

'undergo first letter, beginning letter and digraph rules

BASEPROJECTINSTANCE.FirstLetterRules()

BASEPROJECTINSTANCE.SplitIntoDigraphs()

BASEPROJECTINSTANCE.AnyRepeats()

Else

'skips all code and returns a message box if the file is incorrect

MsgBox("FILE NAME INVALID: Enter new file name")

End If

End Sub

Private Sub IsCipherButton\_Click(sender As System.Object, e As System.EventArgs)

ReadFileButton.Show()

FileNameTextBox.Show()

FileNameInstructionLabel.Show()

FileNameTextBox.Select()

'READWRITE = TRUE ==== CIPHERTEXT FILES

'READWRITE = FALSE ==== PLAINTEXT FILES

End Sub

Private Sub IsPlaintextButton\_Click(sender As System.Object, e As System.EventArgs)

ReadFileButton.Show()

FileNameTextBox.Show()

FileNameInstructionLabel.Show()

'READWRITE = TRUE ==== CIPHERTEXT FILES

'READWRITE = FALSE ==== PLAINTEXT FILES

End Sub

Private Sub ReplaceLetterComboBox\_SelectedIndexChanged(sender As System.Object, e As System.EventArgs) Handles ReplaceLetterComboBox.SelectedIndexChanged

Dim NewLetters As String

'event of changing letters in the ReplaceLetterComboBox

'Refers to the letter in the cipher text that the user wishes to change

'Clear all items currently in the CorrespondingPlaintextComboBox

CorrespondingPlaintextComboBox.Items.Clear()

CorrespondingPlaintextComboBox.Text = ""

'If the letters unfound is not full (i.e. does not contain the entire alphabet)

If LettersUNfound.Length <> 26 Then

'call recursive algorithm sequence

ChangeCorrespondanceNearlyDone()

Else

'the letters that correspond to the letter just selected are collected by the cProject instance

'new letters for the correspondingPlaintext ComboBox are saved into a local variable

NewLetters = BASEPROJECTINSTANCE.GetIndivCorrespondance(ReplaceLetterComboBox.SelectedItem)

'add each character within the NewLetters string into seperate items in the corresponding plaintext combobox

For looper = Len(NewLetters) To 1 Step -1

CorrespondingPlaintextComboBox.Items.Add(Mid(NewLetters, looper, 1))

Next

End If

End Sub

Private Sub ResetButton\_Click(sender As System.Object, e As System.EventArgs) Handles ResetButton.Click

'upon click of the ResetButton:

'The ciphertextbox text is equated back to the original text provided upon readfile procedure

CipherTextBox.Text = TextIncomplete

'the list box of changes is cleared

ChangedLettersListBox.Items.Clear()

'LettersFound is emptied

Lettersfound = Nothing

'all colour changes in the text box are negated and set to black

CipherTextBox.ForeColor = Color.Black

'all user changes are effectively replaced

End Sub

Private Sub CorrespondingPlaintextComboBox\_SelectedIndexChanged(sender As System.Object, e As System.EventArgs) Handles CorrespondingPlaintextComboBox.SelectedIndexChanged

Dim PositionLast As Integer

Dim PartialCompletedText As String

Dim TempText As String

Dim TempAlteredText As String

''PartialCompletedText = CipherTextBox.Text

''TempAlteredText = TextIncomplete

''PositionLast = 1

'PartialCompletedText = CipherTextBox.Text

'TempAlteredText = TextIncomplete

'PositionLast = 1

CipherTextBox.ForeColor = Color.DarkRed

'For looper = PositionLast To CipherTextBox.Text.Length

' If ReplaceLetterComboBox.SelectedItem() = Mid(CipherTextBox.Text, looper, 1) Then

' PartialCompletedText &= Mid(TempText, 1, looper - 1)

' PartialCompletedText &= CorrespondingPlaintextComboBox.SelectedItem

' TempText = Mid(TempText, looper + 1)

' TempAlteredText = Mid(TempAlteredText, looper + 1)

' 'PartialCompletedText.Replace(ReplaceLetterComboBox.SelectedItem, CorrespondingPlaintextComboBox.SelectedItem)

' End If

'Next

'CipherTextBox.Text = PartialCompletedText

PositionLast = 1

For looper = PositionLast To CipherTextBox.Text.Length

'if a letter targeted in the ciphertextbox is equal to the character selected by the user in the ReplaceLEtter combo box

'Then the letter targeted in the ciphertext box is selected and the text is changed to the letter selected in the correspondingplaintext combo box

If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then

CipherTextBox.Select(looper - 1, 1)

CipherTextBox.SelectedText = CorrespondingPlaintextComboBox.SelectedItem

End If

Next

PositionLast = 1

For looper = PositionLast To CipherTextBox.Text.Length

'if a letter targeted in the ciphertextbox is equal to the character selected by the user in the ReplaceLEtter combo box

'Then the letter targetted in the cipher text box is selected then the colour of the selection made is changed to blue (font)

If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then

CipherTextBox.Select(looper - 1, 1)

CipherTextBox.SelectionColor = Color.Blue

End If

Next

'if the letter changed has not yet been changed then add that letter to the lettersFound string

If InStr(LettersFound, CorrespondingPlaintextComboBox.SelectedItem) = 0 Then

LettersFound &= CorrespondingPlaintextComboBox.SelectedItem

End If

'add an item to the changedLettersListBox

'Item added shows the last process that the user has done/ what the user has done

ChangedLettersListBox.Items.Add(ReplaceLetterComboBox.SelectedItem & " changed to: " & CorrespondingPlaintextComboBox.SelectedItem)

If Lettersfound.Length >= HowManyLettersBeFoundFirst Then

UnfoundLettersRECURSIVE("A"c, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", Lettersfound)

'begin recursive letters found sequence in the case that the letters found length is not equal to the howmanylettersbefoundfirst constant

ChangeCorrespondanceNearlyDone()

End If

End Sub

Private Sub ChangeCorrespondanceNearlyDone()

For looper = 1 To LettersUNfound.Length

'add all of the letters that have yet to be found into the corresponding plaintext combo box

'this is done to eliminate any improbable anomalies

CorrespondingPlaintextComboBox.Items.Add(Mid(LettersUNfound, looper, 1))

Next

End Sub

Private Sub UnfoundLettersRECURSIVE(ByVal letterUNFound As Char, ByVal AllUnfoundLetters As String, ByVal FoundLetters As String)

Dim TempUnfoundLetters As String

'This algorithm uses recursion to find all the letters which have yet to be entered by the user (using a variable containing all of the found characters)

For FoundLooper = 1 To FoundLetters.Length

For UnfoundLooper = 1 To AllUnfoundLetters.Length

If Mid(AllUnfoundLetters, UnfoundLooper, 1) = letterUNFound Then

TempUnfoundLetters = AllUnfoundLetters

AllUnfoundLetters = Mid(TempUnfoundLetters, 1, UnfoundLooper - 1)

AllUnfoundLetters &= Mid(TempUnfoundLetters, UnfoundLooper + 1)

FoundLetters = Mid(FoundLetters, 2)

'remove found letter from alphabet

End If

Next

Next

If FoundLetters.Length <> 0 Then

'recursively call the procedure

'undergo again with new parameters

UnfoundLettersRECURSIVE(Mid(FoundLetters, 1, 1), AllUnfoundLetters, FoundLetters)

End If

'final case if statement

'in this case save the final letters and end

If AllUnfoundLetters.Length = 26 - Lettersfound.Length Then

LettersUNfound = AllUnfoundLetters

End If

End Sub

Private Sub BackButton\_Click(sender As System.Object, e As System.EventArgs) Handles BackButton.Click

Dim PositionLast As Integer

Dim PartialCompletedText As String

Dim TempText As String

Dim TempAlteredText As String

If ChangedLettersListBox.Items(0).text <> Nothing Then

'remove last item in list box

'removes the notage of the last process done

ChangedLettersListBox.Items.RemoveAt(ChangedLettersListBox.Items.Count - 1)

'carries out the same procedure as altering letters in the code. the comparison however in this case is changing all values back to their original by assinging the value selected in the replaceletter combo box

'as opposed to the correspondingletter combo box letter being used

PositionLast = 1

For looper = PositionLast To CipherTextBox.Text.Length

'if the letter selected by the replace letter combobox is the target of the letter selected from the original text then

'change the letter back to its original value

If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then

CipherTextBox.Select(looper - 1, 1)

CipherTextBox.SelectedText = ReplaceLetterComboBox.SelectedItem

End If

Next

PositionLast = 1

For looper = PositionLast To CipherTextBox.Text.Length

'if a letter targeted in the ciphertextbox is equal to the character selected by the user in the ReplaceLEtter combo box

'Change the colour back to dark red so it suits the rest of the colour of unchanged letters

If ReplaceLetterComboBox.SelectedItem() = Mid(TextIncomplete, looper, 1) Then

CipherTextBox.Select(looper - 1, 1)

CipherTextBox.SelectionColor = Color.DarkRed

End If

Next

End If

'effectively disregards the last process the user has done

End Sub

Private Sub TutorialButton\_Click(sender As System.Object, e As System.EventArgs) Handles TutorialButton.Click

HelpForm.Show()

'show the form explaining how a user

End Sub

Private Sub SaveButton\_Click(sender As System.Object, e As System.EventArgs) Handles SaveButton.Click

'hide all shown object

'hides all objects pertaining to breaking the cipher

CipherTextBox.Hide()

PickLetterLabel.Hide()

ReplaceLetterComboBox.Hide()

CorrespondingPlaintextComboBox.Hide()

ResetButton.Hide()

ChangedLettersListBox.Hide()

ChangedLetterListBoxLabel.Hide()

BackButton.Hide()

TutorialButton.Hide()

SaveButton.Hide()

FinishedComboBox.Hide()

'shows object involving saving the file

'allows user to enter the file data for saving

SaveNameFileButton.SetBounds(ReadFileButton.Location.X, ReadFileButton.Location.Y, ReadFileButton.Width, ReadFileButton.Height)

SaveNameFileButton.Show()

FileNameTextBox.Show()

FileNameInstructionLabel.Show()

End Sub

Private Sub SaveNameFileButton\_Click(sender As System.Object, e As System.EventArgs) Handles SaveNameFileButton.Click

'show combo box to show if the file is completed (saving as plaintext) or unfinished (saving as cipher)

FinishedComboBox.Show()

Dim FileDirectory As String

'sets the constant filepath equal to the filedirectory

FileDirectory = FilePath

If FinishedComboBox.SelectedItem = "Finished" Then

'READWRITE = TRUE ==== CIPHERTEXT FILES

'READWRITE = FALSE ==== PLAINTEXT FILES

'set the file directory to include the plaintext folder name at the end of the file path

FileDirectory &= PlaintextFolderName

Else

'set the file directory to include the ciphertext folder name at the end of the file path

FileDirectory &= CipherFolderName

End If

'get the file name

FileDirectory &= BASEPROJECTINSTANCE.GetFileName()

'open file given by file path

FileOpen(2, FileDirectory, OpenMode.Output)

'put contents of the ciphertextbox into the file

Print(CipherTextBox.Text)

'close the file

FileClose(2)

End Sub

End Class

Public Class HelpForm

Private Const BEGIN As String = "Welcome to the frequency analysis tutorial. Click any of the adjacent buttons to get information on that stage of the decryption"

Private Const GETTINGSTARTED As String = "Generally we can equate the values ofech letter in the cipher text to English plaintext based on the frequencies of their occurance in the cipher a the relative frequencies of English characters in the most common words of the English language. Despite this information it is naive to assume that the most frequently occuring letter in the English language is the actual/ original letter represented by the most frequently occuring letter in the ciphertext. However we can safely assume that in a large text the three most commonly occuring letters are equivalent to one of the three most commonly occuring letters in English. Beginning the deciphering process by finding the first letter is generally considered the most difficult part of the process. Rest assuared it will get easier as you progress and find more letters. So to begin: if the ciphertext contains at least one single letter word then you are in luck and this step is made very easy. You can simply equate that letter to either I or A because those are the only two letters in the English language that are also written as single letter words. If luck has forsaken you in this matter than you can search through the letters in the cipher to find which have a likely correspondance as given by the program of E,T and A. The other form of frequency analysis which is integral to this stage, we count how often the letter begins a word. The letter E can also be seen before virtually any other letter but T is rarely seen before many other consonants. By using this we can see which of the 3 letters of the ciphertext you have e picked stick out from the other two in terms of where they appear in a word, the one that sticks out most should follow the rules of a consonant and that letter should represent T. By trial and error with the other two letters you can determine what fits best. Use the found T as a guideline."

Private Const PROGRESSING As String = "This step revolves around tricks-of-the-trade so to speak but also no small amount of guesswork. If you have found you can find the letter H pretty easily. The letter H often apears before the letter E in many of the most common words. So now hopefully you have found T, H, E and A. In the event you have yet to find T you can just find a common word where the second and third letters are HE with the first letter being most likely T. After we have found a few letters systematically we can then use those as guidelines to help us find other letters. In a process that seasoned cryptographers call Guessing. We may have words where only one or two letters are unfound. The idea is stick with these words and guess those letters which may be in there. For example we can scour the cipher to find a commonly appearing 3 letter word beginning with A. this of course will represent AND (we hope) and we can equate both the two unknowns at the end of the word to N and D respectively. After we have established these letters progress becomes very simple. Once you have found a few letters more word become gradually filled and less unknowns are present in them. However a word of warning, dont get too clicker happy. There a many times that a word with just a single unknown can represent many different words ie MADE and FADE or LAND, BAND or SAND for example. This program aims to conquer this by only allowing the user (thats you by the way) to enter most likely letters for the letter of ciphertext selected. This may mean however that some letters cannot be completed in the early stages of the code. Fret not because this is an unlikely event and usually only occurs when names are used with letters which do not follow the common letter behaviours. Also after a fair amount of letters have been found this disables itself and the remaining leters can be selected."

Private Const FINISH As String = "Well done you've come quite far and you are thusly rewarded by a respite for your hardworking brain. This section can be completed by a toddler. No offence meant given that you are reading the help section and are possible quite stuck. We can infer most of the remaining letters by looking at the context of what we have already found of the code. Lets prethnd fir a momynt that we have a sityatiom where the majority of letters of a word are already found. We can infer these entire words by the letters around them and the preceeding words and the context it is in. The observant ones among you will have noticed that a few lines back there were a few words in the text box that seemed to be spelt horribly, dont worry it was intentional, and I would bet that you understood what I meant perfectly, if not you may need more help in this stage than I can give. But if not then well done it seems you are already profficient in this process and possess the skills required to breeze through this stage. Simply put the letters in words where your mind cannot think of a logical aternative. I may have played this lightly by in actuality many people can trip up over this stage because they overthink this stage. This comes down to a simple analogy: humour me for a second and pretend you hear hoofbeats over the horizon, in reality it could be horses making these noises or perhaps a herd of zebra are making this happen. Simply if you hear hoofbeats then assume that horses are making them, no need to jump to wild conclusions just because they may be possible. Follow your instincts when you only have few letters left they're probably right, and if not the program should catch you out from making little mistakes. This is the time to be clicker happy, if you see a word in your head then put it in. Just remember not to second guess yourself. In other words make sure you're always thinking about not overthinking."

Private Sub HelpForm\_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load

'show the different buttons required for the user to select what they wish to know about

'three buttons which lead to changing the text written in the text box

GettingStartedButton.Show()

ProgressingButton.Show()

FinalisingButton.Show()

'shows instruction of what buttons to press

InstructionsTextBox.Text = BEGIN

End Sub

Private Sub GettingStartedButton\_Click(sender As System.Object, e As System.EventArgs) Handles GettingStartedButton.Click

'shows instructions on how the user may select the first few letters in the text box when this button is selected

InstructionsTextBox.Text = GETTINGSTARTED

End Sub

Private Sub ProgressingButton\_Click(sender As System.Object, e As System.EventArgs) Handles ProgressingButton.Click

'shows instructions on how the user may find few letters after the first have been found

'instructions shown in the text box when this button is selected

InstructionsTextBox.Text = PROGRESSING

End Sub

Private Sub FinalisingButton\_Click(sender As System.Object, e As System.EventArgs) Handles FinalisingButton.Click

'shows instructions on how the user may select the last few letters in the text box when this button is selected

InstructionsTextBox.Text = FINISH

End Sub

End Class

Public Class cProject

''NOTES:

' MERGE SORT UNTESTED (CALLED IN READFILE)

' READFILE UNTESED (NOT WORKING, MYLETTERS CONTAINS ONLY EMPTY ELEMENTS FOR EVERY EMPTY LINE)

' 2 MIN RUNTIME

' UPPER CASE LETTERS GIVE THEIR FREQUENCIES FOR INDIV LETTERS AND THE FREQUENCY OF THE CORRESPONDING LOWER CASE LETTERS ARE LOST

'=========================================================================

'=====================================================================

'============================================================================

'================================================================================

Structure AllIndividiualLetters

Dim Letter As Char

Dim PositionInMyLetters As Integer

Dim FrequencyPRCNT As Single

Dim FrequencyINT As Integer

'Stores frequency of each letter occuring at the beginning of a word

Dim BeginningFrequencyINT As Integer

'Stores all likely letters in order of likelihood

Dim LikelyLetters As String

End Structure

Private MyLetters() As cLetter 'stores all of the letters in the cipher text (with additional information about each)

Private MyLettersSORTED() As cLetter 'Stores all of the letters in the ciphertext with the same information as the corresponding array only sorted

Private IndividualLetters() As AllIndividiualLetters 'stores information about every unique letter in the ciphertext

Private MyWords() As cWord 'Stores every word and information about them

Private LetterBreakdown As cLikelyLetters 'instance variable which reference the object cLikelyLetters (allows user to use the various functions and obtain various information)

Private NumberOfLetters As Integer 'stores number of letters

Private NumberOfWords As Integer 'stores total number of words

Private Digraphs() As String 'stores digraphs (pairs of two consecutive characters without spaces)

Private DigraphOccurances() As Integer 'stores the occurences of each digraph

Const FilePath As String = "S:\\Ciphers & Plaintext Files" 'Stores file path to the 2 seperate folders

Const CipherFolderName As String = "\Ciphertext Files\" 'stores the folder path for ciphers

Const PlaintextFolderName As String = "\Plaintext Files\" 'stores the folder path for plaintext

Const AlphabetInFreqOrder As String = "ETAOINSRHDLUCMFYWGPBVKXQJZ" 'stores the alphabet in order of most frequent based on use in english

Const AlphabetInBeginningFreqOrder As String = "TAOISWCBPHFMDRELNGUVYJKQXZ" 'stores the alphabet in order of how frequent each letter appears at the beginning of a word

Const TextFileSuffix As String = ".txt" 'stores the file suffix of text file (only file type used by this program)

Public Sub New()

'initialises variables

'creates new instances

'Class constructor

ReDim Preserve MyLetters(0)

ReDim Preserve MyLettersSORTED(0)

ReDim Preserve IndividualLetters(27)

ReDim Preserve MyWords(0)

MyLetters(0) = New cLetter

MyLettersSORTED(0) = New cLetter

'===================================

'Instance of object issue

MyWords(0) = New cWord

'===================================

NumberOfLetters = 0

End Sub

Public Function GetIndividualLetters() As String

Dim AllLetters As String

For looper = 0 To IndividualLetters.Length - 1

AllLetters &= IndividualLetters(looper).Letter

Next

'returns all the used individual letters in a single string

'may not contain the entire alphabet because the cipher may not use certain letters

Return AllLetters

End Function

Public Function GetIndivCorrespondance(ByVal ComboLetter As Char) As String

'returns the likely letters

'corresponding to the letter set by the combobox on the form

'(must be public, called by form)

For looper = 0 To IndividualLetters.Length - 1

If IndividualLetters(looper).Letter = ComboLetter Then

Return IndividualLetters(looper).LikelyLetters

End If

Next

End Function

Public Function GetNumberOfLetters() As Integer

Return NumberOfLetters

End Function

Public Sub AddLetter(ByRef SubjectArray() As cLetter)

'Function which adds a new letter to the array of letters and increases the size of array. Returns new letters for ease of manipulating that new letter

'does not assign any values to a letter

Dim arraysize As Integer

arraysize = SubjectArray.Length

ReDim Preserve SubjectArray(arraysize)

SubjectArray(arraysize) = New cLetter

'ReDim Preserve MyLetters(MyLetters.Length + 1)

'MyLetters(MyLetters.Length) = TheNewLetter

'NumberOfLetters = MyLetters.Length

End Sub

''-------------------------------------------------------------------------

''BUBBLE SORT ALGORITHM (inefficient) UNTESTED

'Public Function SortLettersBUBBLE() As cLetter

' Dim TempLetter As cLetter

' For OutLooper = 1 To NumberOfLetters - 1

' For Inlooper = 1 To NumberOfLetters - 1

' If MyLettersSORTED(Inlooper).PrincipleLetterInitialisation > MyLettersSORTED(Inlooper + 1).PrincipleLetterInitialisation Then

' TempLetter = MyLettersSORTED(Inlooper)

' MyLettersSORTED(Inlooper) = MyLettersSORTED(Inlooper + 1)

' MyLettersSORTED(Inlooper + 1) = TempLetter

' End If

' Next

' Next

'End Function

'-------------------------------------------------------------------------

'MERGE SORT ALGORITHM (to replace bubble sort) UNTESTED

Public Sub MergeSort(ByVal FrontPointer As Integer, ByVal EndPointer As Integer, ByVal LettersOrFrequency As Integer)

Dim CentrePointer As Integer

If EndPointer > FrontPointer Then

'Split the list/array in half and split each of those in half until each list consists of only a single item

'Integer division to make sure that centre is always an integer

CentrePointer = (FrontPointer + EndPointer) \ 2

MergeSort(FrontPointer, CentrePointer, LettersOrFrequency)

MergeSort(CentrePointer + 1, EndPointer, LettersOrFrequency)

'Merge lists (Go to merge Subprocedure)

Select Case LettersOrFrequency

Case 0 'sort letters

'MergeLetters(FrontPointer, CentrePointer, EndPointer)

RadixSortLetters()

Case 1

'sort indiv letters by frequency

MergeFrequencies(FrontPointer, CentrePointer, EndPointer)

Case 2

'sort indiv letters by beginning letter freq

MergeBeginningFrequencies(FrontPointer, CentrePointer, EndPointer)

Case 3

'Sort Digraphs by occurances in text

MergeDigraphs(FrontPointer, CentrePointer, EndPointer)

Case 4

'Sort words in order of occurance

End Select

End If

End Sub

Private Sub RadixSortLetters()

Dim i As Integer, j As Integer 'dimensions two integer variables used to reperesent indexes for each array

Dim temp As cLetter() = New cLetter(MyLettersSORTED.Length - 1) {} 'creates an array the same type and with the same contents as the (as of yet) unsorted letters array

'radix sort

For shift = 31 To -1 + 1 Step -1

j = 0

'shifts letters and groups them based on base / unit

'usually done with integers

'ascii values of each character is used to keep the sort radial / numerical

For i = 0 To MyLettersSORTED.Length - 1

Dim move As Boolean = (Asc(MyLettersSORTED(i).PrincipleLetterInitialisation()) << shift) >= 0

If If(shift = 0, Not move, move) Then

MyLettersSORTED(i - j) = MyLettersSORTED(i)

Else

temp(j) = MyLettersSORTED(i)

j += 1

End If

Next

'save the array by copying the contents of the local temp array into the mylettersSORTED array

Array.Copy(temp, 0, MyLettersSORTED, MyLettersSORTED.Length - j, j)

Next

End Sub

Private Sub MergeLetters(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)

Dim TempArray(0) As cLetter

Dim LeftCounter As Integer

Dim RightCounter As Integer

Dim MainCounter As Integer

' Copy array into temporary array

TempArray(0) = New cLetter

'========================================

'looper lowerbound changed from 1 to 0 TEMP FIX?

'=======================================

'========================================

'looper upperbound changed from numberofletter to numberofletter-1 TEMP FIX?

'=======================================

For looper = 0 To NumberOfLetters - 1

AddLetter(TempArray)

TempArray(looper) = MyLettersSORTED(looper)

Next

LeftCounter = StartPoint

RightCounter = CentrePointer + 1

MainCounter = StartPoint

Do While (LeftCounter <= CentrePointer) And (RightCounter <= EndPoint)

If (TempArray(LeftCounter).PrincipleLetterInitialisation <= TempArray(RightCounter).PrincipleLetterInitialisation) Then

'When smallest value in first half

MyLettersSORTED(MainCounter) = TempArray(LeftCounter)

LeftCounter += 1

Else

'When smallest value in second half

MyLettersSORTED(MainCounter) = TempArray(RightCounter)

RightCounter += 1

End If

MainCounter += 1

Loop

'Copy data from end of list

If LeftCounter <= CentrePointer Then

'copy first half

For looper = 1 To CentrePointer - LeftCounter + 1

MyLettersSORTED(MainCounter + looper - 1) = TempArray(LeftCounter + looper - 1)

Next

Else

'Copy second half

For looper = 1 To EndPoint - RightCounter + 1

MyLettersSORTED(MainCounter + looper - 1) = TempArray(RightCounter + looper - 1)

Next

End If

End Sub

Private Sub MergeFrequencies(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)

Dim TempArray(28) As AllIndividiualLetters

Dim LeftCounter As Integer

Dim RightCounter As Integer

Dim MainCounter As Integer

' Copy array into temporary array

TempArray(28) = New AllIndividiualLetters

'========================================

'looper lowerbound changed from 1 to 0 TEMP FIX?

'=======================================

'========================================

'looper upperbound changed from numberofletter to numberofletter-1 TEMP FIX?

'=======================================

For looper = 0 To 27

TempArray(looper) = IndividualLetters(looper)

Next

LeftCounter = StartPoint

RightCounter = CentrePointer + 1

MainCounter = StartPoint

Do While (LeftCounter <= CentrePointer) And (RightCounter <= EndPoint)

If (TempArray(LeftCounter).FrequencyINT >= TempArray(RightCounter).FrequencyINT) Then

'When smallest value in first half

IndividualLetters(MainCounter) = TempArray(LeftCounter)

LeftCounter += 1

Else

'When smallest value in second half

IndividualLetters(MainCounter) = TempArray(RightCounter)

RightCounter += 1

End If

MainCounter += 1

Loop

'Copy data from end of list

If LeftCounter <= CentrePointer Then

'copy first half

For looper = 1 To CentrePointer - LeftCounter + 1

IndividualLetters(MainCounter + looper - 1) = TempArray(LeftCounter + looper - 1)

Next

Else

'Copy second half

For looper = 1 To EndPoint - RightCounter + 1

IndividualLetters(MainCounter + looper - 1) = TempArray(RightCounter + looper - 1)

Next

End If

End Sub

Private Sub MergeBeginningFrequencies(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)

Dim TempArray(IndividualLetters.Length) As AllIndividiualLetters

Dim LeftCounter As Integer

Dim RightCounter As Integer

Dim MainCounter As Integer

' Copy array into temporary array

TempArray(IndividualLetters.Length) = New AllIndividiualLetters

'========================================

'looper lowerbound changed from 1 to 0 TEMP FIX?

'=======================================

'========================================

'looper upperbound changed from numberofletter to numberofletter-1 TEMP FIX?

'=======================================

For looper = 0 To IndividualLetters.Length - 1

TempArray(looper) = IndividualLetters(looper)

Next

LeftCounter = StartPoint

RightCounter = CentrePointer + 1

MainCounter = StartPoint

Do While (LeftCounter <= CentrePointer) And (RightCounter <= EndPoint)

If (TempArray(LeftCounter).BeginningFrequencyINT >= TempArray(RightCounter).BeginningFrequencyINT) Then

'When smallest value in first half

IndividualLetters(MainCounter) = TempArray(LeftCounter)

LeftCounter += 1

Else

'When smallest value in second half

IndividualLetters(MainCounter) = TempArray(RightCounter)

RightCounter += 1

End If

MainCounter += 1

Loop

'Copy data from end of list

If LeftCounter <= CentrePointer Then

'copy first half

For looper = 1 To CentrePointer - LeftCounter + 1

IndividualLetters(MainCounter + looper - 1) = TempArray(LeftCounter + looper - 1)

Next

Else

'Copy second half

For looper = 1 To EndPoint - RightCounter + 1

IndividualLetters(MainCounter + looper - 1) = TempArray(RightCounter + looper - 1)

Next

End If

End Sub

Private Sub MergeDigraphs(ByVal StartPoint As Integer, ByVal CentrePointer As Integer, ByVal EndPoint As Integer)

Dim TempArray(Digraphs.Length) As String

Dim TempNumbers(DigraphOccurances.Length) As Integer

Dim LeftCounter As Integer

Dim RightCounter As Integer

Dim MainCounter As Integer

' Copy array into temporary array

'========================================

'looper lowerbound changed from 1 to 0 TEMP FIX?

'=======================================

'========================================

'looper upperbound changed from numberofletter to numberofletter-1 TEMP FIX?

'=======================================

For looper = 0 To 29

TempArray(looper) = Digraphs(looper)

Next

LeftCounter = StartPoint

RightCounter = CentrePointer + 1

MainCounter = StartPoint

Do While (LeftCounter <= CentrePointer) And (RightCounter <= EndPoint)

If (DigraphOccurances(LeftCounter) >= DigraphOccurances(RightCounter)) Then

'When smallest value in first half

Digraphs(MainCounter) = TempArray(LeftCounter)

DigraphOccurances(MainCounter) = TempNumbers(LeftCounter)

LeftCounter += 1

Else

'When smallest value in second half

Digraphs(MainCounter) = TempArray(RightCounter)

DigraphOccurances(MainCounter) = TempNumbers(LeftCounter)

RightCounter += 1

End If

MainCounter += 1

Loop

'Copy data from end of list

If LeftCounter <= CentrePointer Then

'copy first half

For looper = 1 To CentrePointer - LeftCounter + 1

Digraphs(MainCounter + looper - 1) = TempArray(LeftCounter + looper - 1)

DigraphOccurances(MainCounter + looper - 1) = TempNumbers(LeftCounter + looper - 1)

Next

Else

'Copy second half

For looper = 1 To EndPoint - RightCounter + 1

Digraphs(MainCounter + looper - 1) = TempArray(RightCounter + looper - 1)

DigraphOccurances(MainCounter + looper - 1) = TempNumbers(RightCounter + looper - 1)

Next

End If

End Sub

Private Function BinarySearchFrequency(ByVal Target As String, ByVal Lowest As Integer, ByVal highest As Integer) As Integer

Dim Middle As Integer

If Lowest > highest Then

Return 0

Else

Middle = ((Lowest + highest) \ 2)

If Target = MyLettersSORTED(Middle).PrincipleLetterInitialisation() Then

Return Middle

Else

If Target < MyLettersSORTED(Middle).PrincipleLetterInitialisation() Then

BinarySearchFrequency(Target, Lowest, Middle - 1)

Else

BinarySearchFrequency(Target, Middle + 1, highest)

End If

End If

End If

End Function

Private Sub CalculateLetterFrequencyINT(ByVal LetterBeingUsed As Char)

Dim WasLetterFound As Boolean

Dim IndexOfLetter As Integer

Dim LoopCounter As Integer

For looper = 0 To NumberOfLetters - 1

WasLetterFound = False

If UCase(MyLettersSORTED(looper).PrincipleLetterInitialisation()) = LetterBeingUsed Then

'Saves the position of first letter

IndexOfLetter = looper

'Loop escape (as soon as the letter is found exit array)

looper = NumberOfLetters - 1

WasLetterFound = True

End If

Next

If WasLetterFound = True Then

LoopCounter = IndexOfLetter

Do While (UCase(MyLettersSORTED(LoopCounter).PrincipleLetterInitialisation()) = LetterBeingUsed) And (LoopCounter <> NumberOfLetters - 1)

'Counts the number of same letters are in the array

LoopCounter += 1

Loop

LoopCounter -= 1

'Do Until (UCase(MyLettersSORTED(LoopCounter + 1).PrincipleLetterInitialisation()) <> LetterBeingUsed And (LoopCounter <> NumberOfLetters - 1))

' 'Counts the number of same letters are in the array

' LoopCounter += 1

'Loop

'frequency = position of final like letter - position of first

For looper = 0 To 27

If LetterBeingUsed = " " Then

IndividualLetters(0).FrequencyINT = ((LoopCounter - IndexOfLetter) + 1)

IndividualLetters(0).PositionInMyLetters = IndexOfLetter

IndividualLetters(0).FrequencyPRCNT = CalculateLetterFrequencyPRCNT(IndividualLetters(0).FrequencyINT)

IndividualLetters(0).Letter = LetterBeingUsed

End If

'asigns values to the integral sections of the IndividualLetters strucutred array

'integral to the program. assigns the relative values corresponding to every letter found in the text (cipher letters)

If Chr(64 + looper) = UCase(LetterBeingUsed) Then

IndividualLetters(looper).FrequencyINT = ((LoopCounter - IndexOfLetter) + 1)

IndividualLetters(looper).PositionInMyLetters = IndexOfLetter

IndividualLetters(looper).FrequencyPRCNT = CalculateLetterFrequencyPRCNT(IndividualLetters(looper).FrequencyINT)

IndividualLetters(looper).Letter = LetterBeingUsed

End If

Next

Else

IndexOfLetter = Nothing

LoopCounter = 0

End If

End Sub

Private Function CalculateLetterFrequencyPRCNT(ByVal FrequencyOfLetterINT As Integer) As Single

'Shows the percentage coverage that the letter has in regards to the total cipher text

Return ((FrequencyOfLetterINT / NumberOfLetters) \* 100)

End Function

Public Sub FinalisingInitialisationSAVEFREQ()

Dim Letter As Char

Dim OutLooper As Integer

For looper = 0 To 26

If looper = 0 Then

Letter = " "

Else

Letter = Chr(64 + looper)

End If

CalculateLetterFrequencyINT(Letter)

Next

'======================================

'======================================

'if letter of array = letter of indivletters then save frequencies

'nested iteration inefficient but compares all possibl values with other values

For OutLooper = 0 To 26

For InLooper = 0 To NumberOfLetters - 1

If UCase(MyLettersSORTED(InLooper).PrincipleLetterInitialisation()) = IndividualLetters(OutLooper).Letter Then

MyLettersSORTED(InLooper).SetFrequencyINT(IndividualLetters(OutLooper).FrequencyINT)

MyLettersSORTED(InLooper).SetFrequencyPRCNT(IndividualLetters(OutLooper).FrequencyPRCNT)

End If

If UCase(MyLetters(InLooper).PrincipleLetterInitialisation()) = IndividualLetters(OutLooper).Letter Then

MyLetters(InLooper).SetFrequencyINT(IndividualLetters(OutLooper).FrequencyINT)

MyLetters(InLooper).SetFrequencyPRCNT(IndividualLetters(OutLooper).FrequencyPRCNT)

End If

Next

Next

End Sub

'GETTERS Function

Public Function GetFileName() As String

Dim CipherTextDocumentName As String

'finds the filename by adding the suffix of the file on the end of the entered name so that the file is given its readable format

CipherTextDocumentName = (Form1.FileNameTextBox.Text & TextFileSuffix)

Return CipherTextDocumentName

End Function

Private Function GetFileNameExtended(ByVal CipherORPlainText As Boolean) As String

Dim FileDirectory As String

FileDirectory = FilePath

'CipherofPlainText is a boolean variable that is true when the required file is in the Cipher folder

'False when the requried file is in the Plaintext folder

If CipherORPlainText = True Then

'concatonates the name of the ciphertext folder on to the already existing file directory to give the full directory

FileDirectory &= CipherFolderName

Else

'concatonates the name of the plaintext folder on to the already existing file directory to give the full directory

FileDirectory &= PlaintextFolderName

End If

'add the name of the file and the text file suffix (which will be returned as a single string after the called function

FileDirectory &= GetFileName()

'Return the full file directory

Return FileDirectory

End Function

'The following file is a public function because it acts like a getter by returning the contents of the file to the form1 class

Public Function ReadFile() As String

Dim WholeFile As String

Dim LetterCounter As Integer

LetterCounter = 0

'COULD ALSO BE WRITTEN AS ALL READING FROM FILE IS TRUE (FOR CIPHERS)

'FileOpen(1, AssembleFilePath(true), OpenMode.Input)

'AND ALL WRITING TO FILES WILL BE ONLY FOR SAVING PLAINTEXTS (FALSE)

FileOpen(1, GetFileNameExtended(True), OpenMode.Input)

'Saves all contents of file into a single string

Do Until EOF(1)

WholeFile &= LineInput(1)

Loop

'Finished using file at this point. Closes file

FileClose(1)

WholeFile = UCase(WholeFile)

'Cycles through every character in the string (containing the contents of the file) and saves each character into the Myletter array

For looper = 0 To WholeFile.Length - 1

If looper <> 0 Then

AddLetter(MyLetters)

End If

MyLetters(looper).PrincipleLetterInitialisation(Mid(WholeFile, looper + 1, 1))

If looper <> 0 Then

MyLetters(looper).PreceedingLetter(MyLetters(looper - 1).PrincipleLetterInitialisation())

End If

If looper <> 0 Then

MyLetters(looper - 1).SubsequentLetter(MyLetters(looper).PrincipleLetterInitialisation())

End If

Next

'Calculate number of letters in document

'===================================================================

'QUICK FIX (Problem: number of letters 2 less than actual array size)

'FIX: Wholefile.length = 1 to end of file

'NumberOfLetters = WholeFile.Length + 1

NumberOfLetters = WholeFile.Length 'currently accurate and working

'====================================================================

'NumberOfLetters = WholeFile.Length - 1

MyLettersSORTED(0) = MyLetters(0)

For Looper = 1 To NumberOfLetters - 1

AddLetter(MyLettersSORTED)

MyLettersSORTED(Looper) = MyLetters(Looper)

Next

'returns whole file to the form (ciphertextbox)

Return WholeFile

'For looper = 0 To NumberOfLetters

' MyLettersSORTED(looper) = SortLettersBUBBLE()

'Next

End Function

'=======================================================================

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''FINDING LIKELY LETTERS'''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'=======================================================================

Public Sub MostLikelyBySingleFrequency()

'(n,n,1) means sort by frequencies

MergeSort(0, 27, 1)

RemoveNullSpacesInIndivLetters()

'For Alphalooper = 0 To 27

' For Indivlooper = 0 To IndividualLetters.Length - 1

' If (IndividualLetters(Indivlooper).Letter = Chr(64 + Alphalooper)) Then

' If Indivlooper \ 5 = 4 Then

' IndividualLetters(Indivlooper).LikelyLetters = Mid(AlphabetInFreqOrder, (((Indivlooper \ 5) \* 5) + 1), 6)

' Else

' IndividualLetters(Indivlooper).LikelyLetters = Mid(AlphabetInFreqOrder, (((Indivlooper \ 5) \* 5) + 1), 5)

' End If

' End If

' Next

'Next

'see which position the letter is in letter frequency

'assign the 5 most likeley letters in regards to frequency by finding the 5 letters nearest to the position of the letter of the same frequency in the constant array of likely letters

'REMINDER: arrays begin with element 0 string begin with char 1

For IndivLooper = 0 To IndividualLetters.Length - 1

If (IndivLooper - 1 > 0) And (IndivLooper + 3 < 27) Then

IndividualLetters(IndivLooper).LikelyLetters = Mid(AlphabetInFreqOrder, IndivLooper - 1, 5)

Else

'Select case to determine procedure in special cases

Select Case IndivLooper

'in some cases assigning the 5 nearest is not the same code as others

'As strings do not "wrap around" there must be code that saves the correct letters

Case 0

'this case saves the first four letters (ommitting the 5th)

IndividualLetters(IndivLooper).LikelyLetters = Mid(AlphabetInFreqOrder, IndivLooper + 1, 4)

Case 1

'this case saves the first five letters (including the 5th)

IndividualLetters(IndivLooper).LikelyLetters = Mid(AlphabetInFreqOrder, IndivLooper, 5)

Case 25

'this case saves the final 5

IndividualLetters(IndivLooper).LikelyLetters = Mid(AlphabetInFreqOrder, IndivLooper - 3, 5)

Case 26

'This case saves the final 4 (ommiting the 5th from last)

IndividualLetters(IndivLooper).LikelyLetters = Mid(AlphabetInFreqOrder, IndivLooper - 3, 4)

End Select

End If

Next

End Sub

Public Sub RemoveNullSpacesInIndivLetters()

Dim NewLength As Integer

NewLength = 27

'removes any spaces in the individual letters array

For looper = 0 To 27

'if there is no letter then remove the item and resize the array

If (IndividualLetters(looper).Letter = Nothing) Or (IndividualLetters(looper).Letter = " "c) Then

NewLength -= 1

For ReArrangeLooper = looper To 26

IndividualLetters(ReArrangeLooper) = IndividualLetters(ReArrangeLooper + 1)

Next

End If

Next

'resize array after all empty items have been moved to the end

ReDim Preserve IndividualLetters(NewLength)

End Sub

Public Sub SplitIntoWords()

Dim WordLength As Integer

Dim TempWord As String

' MyWords(0) = New cWord

'splits the letters into words

For looper = 0 To NumberOfLetters - 1

WordLength += 1

'adds a letter to the tempword string (&=) unless that character is a space (" ")

TempWord &= MyLetters(looper).PrincipleLetterInitialisation

If MyLetters(looper).PrincipleLetterInitialisation() = " " Then

'If the character is a space (" ") then this signifies the end of a word

'therefore the word is saved into the array

NumberOfWords += 1

ReDim Preserve MyWords(NumberOfWords)

'array size is incremented by 1 through redimension

'the value put into my word is not all the word

MyWords(NumberOfWords - 1) = New cWord

TempWord = (Mid(TempWord, 1, Len(TempWord) - 1))

MyWords(NumberOfWords - 1).WordInitialisation((looper - Len(TempWord)), TempWord)

'removes last character because that last character is always a space

TempWord = Nothing

WordLength = 0

End If

'if the number of words is the same as the number of letters (i.e no words more than a single letter in length

If looper = NumberOfLetters - 1 Then

MyWords(NumberOfWords) = New cWord

MyWords(NumberOfWords).WordInitialisation((looper - Len(TempWord)), TempWord)

End If

Next

'if there are any single letter words (must be "a" or an "I" )

'then equate the letter representing that single letter word as having a correspondance of "A" or "I"

For looper = 0 To NumberOfWords - 1

If MyWords(looper).IfSingleLetters() <> "" Then

For Indivlooper = 0 To IndividualLetters.Length - 1

If IndividualLetters(Indivlooper).Letter = MyWords(looper).GetLetters(1) Then

IndividualLetters(Indivlooper).LikelyLetters = MyWords(looper).IfSingleLetters

End If

Next

End If

Next

End Sub

Public Sub SplitIntoDigraphs()

Dim AlreadyFound As Boolean 'boolean varibale that stores if the value has been found already

Dim LastPosition As Integer 'saves the position in the digraph array of the last digraph

Dim tempcorrespondances() As String 'temporary array for possible likely letters

Dim TempLetters As String 'temporary string

'variable initialisation

LetterBreakdown = New cLikelyLetters

ReDim Preserve Digraphs(0)

ReDim Preserve DigraphOccurances(0)

ReDim Preserve tempcorrespondances(27)

For looper = 0 To NumberOfLetters - 1

'Inner loop sees if the tempdigraph array contains any repeated digraphs

'occurance of each letter stored for sorting

If looper + 1 <> NumberOfLetters Then

For digraphlooper = 0 To Digraphs.Length - 1

AlreadyFound = False

'if the digraph consists of the current letter being searched for compounded with its subsequent letter then it is marked as found

'the last value is then removed from the digraph array

If (Digraphs(digraphlooper) = (MyLetters(looper).PrincipleLetterInitialisation & MyLetters(looper + 1).PrincipleLetterInitialisation)) Then

AlreadyFound = True

LastPosition = digraphlooper

digraphlooper = Digraphs.Length - 1

End If

Next

'prevents repeats of the same digraph being in the array

'resulting in an array with unique values in every element

'requred for sorting and finding the 30 most frequent digraphs

If (AlreadyFound = False) And ((MyLetters(looper).PrincipleLetterInitialisation <> " "c) And (MyLetters(looper + 1).PrincipleLetterInitialisation <> " "c)) Then

'add values to arrays and redimension

ReDim Preserve Digraphs(Digraphs.Length)

ReDim Preserve DigraphOccurances(DigraphOccurances.Length)

Digraphs(Digraphs.Length - 1) = MyLetters(looper).PrincipleLetterInitialisation & MyLetters(looper + 1).PrincipleLetterInitialisation

DigraphOccurances(Digraphs.Length - 1) += 1

End If

If AlreadyFound = True Then

DigraphOccurances(LastPosition) += 1

End If

End If

'If (MyLetters(looper).PrincipleLetterInitialisation() = " ") Or (Len(Digraphs(digraphlooper)) <> 2) Then

' Digraphs(looper) = Nothing

'End If

Next

'sort in order of frequency of occurances

MergeSort(0, Digraphs.Length - 1, 3)

'put top 30 most common digraphs into the clikelyletters class for processing

For looper = 0 To 29

If Digraphs(looper) <> Nothing Then

LetterBreakdown.CompareDigraphs(Digraphs(looper), looper)

End If

Next

'Set the letters to digraph calculated letters

For looper = 0 To IndividualLetters.Length - 1

If LetterBreakdown.DigraphsAndLetters(IndividualLetters(looper).Letter) <> Nothing Then

'the values of the temp correspondances are equated to the values returned by the cLikelyLetters class

'uses the digraphs made beforehand

tempcorrespondances(looper) = LetterBreakdown.DigraphsAndLetters(IndividualLetters(looper).Letter)

For CorrospondancesLengthlooper = 1 To tempcorrespondances(looper).Length

For IndividualCorrespondanceLengthLooper = 1 To IndividualLetters(looper).LikelyLetters.Length

If Mid(tempcorrespondances(looper), CorrospondancesLengthlooper, 1) = Mid(IndividualLetters(looper).LikelyLetters, IndividualCorrespondanceLengthLooper, 1) Then

'Sees if the digraph calculated likely letters coincide with the likely letters from other rules

'Places corresponding letters to the last value in the likely letters array

IndividualLetters(looper).LikelyLetters = Mid(IndividualLetters(looper).LikelyLetters, IndividualCorrespondanceLengthLooper, 1) & Mid(IndividualLetters(looper).LikelyLetters, 1, IndividualCorrespondanceLengthLooper - 1) & Mid(IndividualLetters(looper).LikelyLetters, IndividualCorrespondanceLengthLooper + 1, IndividualLetters(looper).LikelyLetters.Length)

End If

Next

Next

End If

Next

'Re arrange so that the most likely letter is the first in the individualletters().likelyletter string

'For indivlooper = 0 To IndividualLetters.Length - 1

' 'add all the letters to the temp letters string

' TempLetters = IndividualLetters(indivlooper).LikelyLetters

' IndividualLetters(indivlooper).LikelyLetters = ""

' For ReshuffleLooper = 0 To TempLetters.Length - 1

' 'IndividualLetters(indivlooper).LikelyLetters = TempLetters

' IndividualLetters(indivlooper).LikelyLetters &= Mid(TempLetters, TempLetters.Length - ReshuffleLooper, 1)

' Next

'Next

End Sub

'--------------------------------------------------------------------------------

'-------------------RULES-------------------------------------------------------

'-------------------------------------------------------------------------------

Public Sub FirstLetterRules()

Dim CurrentLetter As Char 'variable to store the target character

Dim LikelyLetters() As String 'Array which stores the likely letters corresponding to each letter

Dim FinalLikelyLetters() As String 'Array which stores the fully processed and calculated likely letters for each character

'initialise arrays

ReDim Preserve LikelyLetters(IndividualLetters.Length - 1)

ReDim Preserve FinalLikelyLetters(IndividualLetters.Length - 1)

'Find relative occurances of first letters

For WordLooper = 0 To NumberOfWords - 1

'the target letter is sort to the first letter of each word

CurrentLetter = MyWords(WordLooper).GetLetters(1)

For LetterLooper = 0 To IndividualLetters.Length - 1

'searches for the element of the individual letters array which corresponds to the target letter

If IndividualLetters(LetterLooper).Letter = CurrentLetter Then

IndividualLetters(LetterLooper).BeginningFrequencyINT += 1

'increments the varaible storing the beginning frequency corresponding to the target letter by 1

End If

Next

Next

'sort indiv letters in order of beginning letters

'save most likely 5 letters

'compare with already known letters

'elminate assumed letters

MergeSort(0, IndividualLetters.Length - 1, 2)

'For Alphalooper = 0 To 27

' For Indivlooper = 0 To IndividualLetters.Length - 1

' If (IndividualLetters(Indivlooper).Letter = Chr(64 + Alphalooper)) Then

' If Indivlooper \ 5 = 4 Then

' LikelyLetters(Indivlooper) = Mid(AlphabetInBeginningFreqOrder, (((Indivlooper \ 5) \* 5) + 1), 6)

' Else

' LikelyLetters(Indivlooper) = Mid(AlphabetInBeginningFreqOrder, (((Indivlooper \ 5) \* 5) + 1), 5)

' End If

' End If

' Next

'Next

'REMINDER: arrays begin with element 0 string begin with char 1

For IndivLooper = 0 To IndividualLetters.Length - 1

If (IndivLooper - 1 > 0) And (IndivLooper + 3 < 27) Then

IndividualLetters(IndivLooper).LikelyLetters &= Mid(AlphabetInBeginningFreqOrder, IndivLooper - 1, 5)

Else

'Select case to determine procedure in special cases

Select Case IndivLooper

'in some cases assigning the 5 nearest is not the same code as others

'As strings do not "wrap around" there must be code that saves the correct letters

Case 0

IndividualLetters(IndivLooper).LikelyLetters &= Mid(AlphabetInBeginningFreqOrder, IndivLooper + 1, 4)

'this case saves the first four letters (ommitting the 5th)

Case 1

IndividualLetters(IndivLooper).LikelyLetters &= Mid(AlphabetInBeginningFreqOrder, IndivLooper, 5)

'this case saves the first five letters (including the 5th)

Case 25

IndividualLetters(IndivLooper).LikelyLetters &= Mid(AlphabetInBeginningFreqOrder, IndivLooper - 3, 5)

'this case saves the final 5

Case 26

IndividualLetters(IndivLooper).LikelyLetters &= Mid(AlphabetInBeginningFreqOrder, IndivLooper - 3, 4)

'This case saves the final 4 (ommiting the 5th from last)

End Select

End If

Next

'For every letter:

'Compare every letter in both likely letter strings

'Save common letters

For LetterLooper = 0 To IndividualLetters.Length - 1

For IndivLikelyLooper = 1 To Len(IndividualLetters(LetterLooper).LikelyLetters)

For StringLooper = 1 To Len(LikelyLetters(LetterLooper))

If Mid(IndividualLetters(LetterLooper).LikelyLetters, IndivLikelyLooper, 1) = Mid(LikelyLetters(LetterLooper), StringLooper, 1) Then

'if the current letter in question of the saved likelyletters for the current target letter equals the first letter rule calculated

'then save the likely letter

FinalLikelyLetters(LetterLooper) &= Mid(LikelyLetters(LetterLooper), StringLooper, 1)

End If

Next

Next

Next

End Sub

Public Sub AnyRepeats()

Dim RepeatedChar As Char 'this variable stores the character that has been found to have been repeated

Dim TempLikelyLetters As String 'The temporary array that allows the program to overwrite the original

For indivlooper = 0 To IndividualLetters.Length - 1

For CorrespondanceLooper = 1 To IndividualLetters(indivlooper).LikelyLetters.Length

'Search if there are any repeat occurances of a single letter within the same element of the likelyletters array

'ie find any repeatsof the same letter by comparing all letters in the array with eachother

If InStr(IndividualLetters(indivlooper).LikelyLetters, Mid(IndividualLetters(indivlooper).LikelyLetters, CorrespondanceLooper, 1)) > 0 Then

'if there is no repeated occurence of the same letter within a single likelyletter variable corresponding to a single letter then

'the repeated character is saved in a temp variable

RepeatedChar = Mid(IndividualLetters(indivlooper).LikelyLetters, CorrespondanceLooper, 1)

For removeLooper = 1 To IndividualLetters(indivlooper).LikelyLetters.Length

If Mid(IndividualLetters(indivlooper).LikelyLetters, removeLooper, 1) = RepeatedChar Then

'then: for every time that the repeated character is referenced in that same likely letters variable

'remove that repeated letter

'(By saving the likely letters in a temp array but not saving the repeated letter....

TempLikelyLetters = Mid(IndividualLetters(indivlooper).LikelyLetters, 1, removeLooper - 1)

TempLikelyLetters &= Mid(IndividualLetters(indivlooper).LikelyLetters, removeLooper + 1, IndividualLetters(indivlooper).LikelyLetters.Length - removeLooper)

'then using the temp variable to overwrite the likely letters array

IndividualLetters(indivlooper).LikelyLetters = TempLikelyLetters

removeLooper -= 1

End If

Next

'position the repeated character at the front of the string

'during the overwriting code

IndividualLetters(indivlooper).LikelyLetters = RepeatedChar & TempLikelyLetters

End If

Next

For reorderlooper = 1 To IndividualLetters(indivlooper).LikelyLetters.Length

TempLikelyLetters = IndividualLetters(indivlooper).LikelyLetters

''REORDER THE LETTERS LAST FIRST

Next

Next

End Sub

End Class

Public Class cLikelyLetters

Structure LetterWeight

Dim PlaintextLetter As Char

Dim NumberOfTestsPassed As Integer

Dim PercentageLikelihood As Single

Dim FOUND As Boolean

End Structure

Structure Digraph

Dim LetterPair As String

Dim Occurance As Integer

Dim LikelyCorrespondance As String

Dim DoublePos As Integer

Dim Processed As Boolean

End Structure

Private AllLetters(26) As LetterWeight 'array that stores structured values for various data aspects of every letter

Private CommonDigraphs(30) As Digraph 'array which stores various data about the 30 most common digraphs in the cipher

Private Correspondances() As String 'unbounded (as of yet) array which stores the likely correspondances for each letter

Private NumberOfTests As Integer 'number of test each letter has passed as a likely letter

Const Alphabet As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Const AlphabetInFreqOrder As String = "ETAOINSRHDLUCMFYWGPBVKXQJZ"

Const AlphabetInBeginningFreqOrder As String = "TAOISWCBPHFMDRELNGUVYJKQXZ"

Const MostCommonDigrams As String = "THHEINERANREEDONESSTENATTONTHANDOUEANGASORTIISETITARTESEHIOF"

Const MostCommonDoubleDigraphs As String = "SSEETTFFLLMMOO"

Public Sub AddlikelihoodOfLetters(ByVal NewLetter As Char)

'for all letters in the alphabet add how many tests they have past as a coresspondance

For looper = 0 To 25

If AllLetters(looper).PlaintextLetter = NewLetter Then

AllLetters(looper).NumberOfTestsPassed += 1

AllLetters(looper).PercentageLikelihood = (AllLetters(looper).NumberOfTestsPassed / NumberOfTests) \* 100

End If

Next

NumberOfTests += 1

End Sub

Public Sub CompareDigraphs(ByVal LoneDigraph As String, ByVal PositionSorted As Integer)

'position sorted will be equal to a looper when top 30 digraphs are sorted

'Looper that adds a letterpair to an array of digraphs in a sorted position

For looper = 0 To 29

'for first digraph into array

If (PositionSorted = looper) And (looper = 0) Then

CommonDigraphs(looper).LetterPair = LoneDigraph

CommonDigraphs(looper).LikelyCorrespondance = Mid(MostCommonDigrams, looper + 1, 4)

'saves the 2 most common digraphs (highest frequency)

'4 letters for 2 digraphs

End If

'for digraphs sorted in the middle of an array

If (PositionSorted = looper) And (looper <> 0 And looper <> 29) Then

CommonDigraphs(looper).LetterPair = LoneDigraph

CommonDigraphs(looper).LikelyCorrespondance = Mid(MostCommonDigrams, looper, 6)

'saves the 3 digraphs around (1 above, 1 below,1 on) the most likely digrpahs based on frequncy calculated

'6 letters for 3 digraphs

End If

'for digraphs sorted in the last position of array

If (PositionSorted = looper) And (looper = 29) Then

CommonDigraphs(looper).LetterPair = LoneDigraph

CommonDigraphs(looper).LikelyCorrespondance = Mid(MostCommonDigrams, looper - 1, 4)

'saves the 2 digraphs in the end of the constant string (least common of the set saved)

'4 letters for 2 digraphs

End If

'for digraphs with repeated letters

If Mid(CommonDigraphs(looper).LetterPair, 1, 1) = Mid(CommonDigraphs(looper).LetterPair, 2, 1) Then

DoubleDigraphs(Mid(CommonDigraphs(looper).LetterPair, 1, 1))

'double digraphs have other levels of commonimity and frequency

'runs new procedure to calculate said likely letters

End If

Next

End Sub

Public Function DigraphsAndLetters(ByVal IndivLetter As Char)

'checks to see if expected letters correspond to assumed digraphs

'aheadorbehind = true when ahead / false when behind

Dim Letters() As Char

'If AheadOrBehind = True Then

' CipherDigraph = principleLetter & NeighbouringLetter

'End If

'If AheadOrBehind = False Then

' CipherDigraph = NeighbouringLetter & principleLetter

'End If

'For looper = 0 To 29

' If CipherDigraph = CommonDigraphs(looper).LetterPair Then

' 'saves that letters pass another test

' AddlikelihoodOfLetters(Mid(CipherDigraph, 1, 1))

' AddlikelihoodOfLetters(Mid(CipherDigraph, 2, 1))

' End If

'Next

'Saves letters into an array

ReDim Preserve Correspondances(26)

ReDim Preserve Letters(26)

For looper = 0 To 25

Letters(looper) = Chr(looper + 65)

Next

'cycles through all letters

'sees if any other occurances of the letters happens with the same likely letters

'Stores the correspondances in an array with the same index as the letter it corresponds to in the letters array

For DigraphLooper = 0 To CommonDigraphs.Length - 1

For LetterLooper = 0 To Letters.Length - 1

For Bilooper = 1 To 2

'compares all digraphs (and letters within the digraph) with eachother

'makes every single comparison possible

If Letters(LetterLooper) = Mid(CommonDigraphs(DigraphLooper).LetterPair, Bilooper, 1) Then

'if the letter targetted is the letter targretted in the targetted digraph then

' set the likelycorrespondance array element targetted to the the value found in the likelycorrespondance calulated from the digraph

For LikelyLooper = 1 To Len(CommonDigraphs(DigraphLooper).LikelyCorrespondance) / 2

If Bilooper = 1 Then

Correspondances(LetterLooper) = Mid(CommonDigraphs(DigraphLooper).LikelyCorrespondance, (LikelyLooper \* 2) - 1, 1)

End If

If Bilooper = 2 Then

Correspondances(LetterLooper) = Mid(CommonDigraphs(DigraphLooper).LikelyCorrespondance, LikelyLooper \* 2, 1)

End If

Next

End If

Next

Next

Next

For looper = 0 To 25

'return every element in the correspondances array

If IndivLetter = Letters(looper) Then

Return Correspondances(looper)

End If

Next

End Function

Public Sub DoubleDigraphs(ByVal principleLetter As Char)

Dim cipherdigraph As String

Dim Position As Integer

'repeated letters ("TT", "SS", "EE" and so on) are very good indicators of what letters are which

'the premise of this procedure is that there are some letters which do not produce doubles

'and of the ones that do there are some which are more common than others

cipherdigraph = principleLetter & principleLetter

Position = 0

For looper = 0 To CommonDigraphs.Length - 1

If Mid(CommonDigraphs(looper).LetterPair, 1, 1) = Mid(CommonDigraphs(looper).LetterPair, 2, 1) Then

Position += 1

CommonDigraphs(looper).DoublePos = Position

End If

Next

'equates the likely correspondances (only the correspondance related to the digraphs) to the value set by the constants of double digraphs

For looper = 0 To CommonDigraphs.Length - 1

If cipherdigraph = CommonDigraphs(looper).LetterPair Then

'if the targretted digraph is a spotlighted letter repeated then

'save the most wanted double digraphs from the constants into the correspondance

CommonDigraphs(looper).LikelyCorrespondance = Mid(MostCommonDoubleDigraphs, CommonDigraphs(looper).DoublePos \* 2, 2)

'if the targretted digraph is a spotlighted letter repeated then

'save the most wanted double digraphs from the constants into the correspondance

If (CommonDigraphs(looper).DoublePos <> 1) Or (CommonDigraphs(looper).DoublePos <> (Len(MostCommonDoubleDigraphs) / 2)) Then

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2)

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)

End If

If CommonDigraphs(looper).DoublePos = 1 Then

'if the targretted digraph is the first digraph

'save the most wanted double digraphs from the constants into the correspondance

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2)

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)

End If

If CommonDigraphs(looper).DoublePos = Len(MostCommonDoubleDigraphs) / 2 Then

'if the targretted is a central value

'save the most wanted double digraphs from the constants into the correspondance

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) + 2, 2)

CommonDigraphs(looper).LikelyCorrespondance &= Mid(MostCommonDoubleDigraphs, (CommonDigraphs(looper).DoublePos \* 2) - 2, 2)

End If

End If

Next

End Sub

End Class

Public Class cWord

'PROPERTIES:

Private pWordStart As Integer 'start of the word in regards to the index they begin in the et / MyLetters array

Private pLength As Integer 'length of thecword

Private pWord As String 'the actual word in the cipher

'GETTERS & SETTERS (POLYMORPHIC)

Public Sub New()

pWord = ""

pLength = 0

pWordStart = 0

End Sub

'sets value of word

'the lenght of said word

'and the start of the word in regards to the index they begin in the et / MyLetters array

Public Sub WordInitialisation(ByVal FirstLetterIndex As Integer, ByVal NewWord As String)

pLength = Len(NewWord)

pWord = NewWord

pWordStart = FirstLetterIndex

End Sub

'GETTER FUNCTIONS

Public Function GetLetters(ByVal position As Integer)

Return Mid(pWord, position, 1)

End Function

Public Function GetLetters(ByVal position As Integer, ByVal NumberOfLetters As Integer)

Return Mid(pWord, position, NumberOfLetters)

End Function

Public Function IfSingleLetters() As String

'returns the letters that any singe leter word must be equal to

'"I" and "A" are the only english single letter words

If pLength = 1 Then

Return "IA"

End If

End Function

End Class

Public Class cLetter

'PROPERTIES:

Private pPrincipleLetter As Char 'Contains the value of the primary letter being manipulated

'Shared pLetterAfter As cLetter

'Shared pLetterBefore As cLetter

Private pIsVowell As Boolean 'Is it a vowell(True) or consonant(false)

Private pLetterAfter As Char 'Contains value of character after

Private pLetterBefore As Char 'Contains preceeding character

Private pFrequencyINT As Integer 'Number of occurances of principle letter

Private pFrequencyPRCNT As Single 'Percentage coverage of principle letter in entire document

'Private pPossibleLettersORDERED As String 'Contains a string of all letters that the ciphertext can represent in order of decreasing probability

Public Sub New()

pPrincipleLetter = ""

pLetterAfter = ""

pLetterBefore = ""

pFrequencyINT = 0

pFrequencyPRCNT = 0

End Sub

'--------------------------------------------

'GETTERS & SETTERS: (POLYMORPHIC)

'--------------------------------------------

Public Function PrincipleLetterInitialisation() As Char

Return pPrincipleLetter

End Function

Public Sub PrincipleLetterInitialisation(ByVal NewLetter As Char)

pPrincipleLetter = NewLetter

End Sub

Public Function SubsequentLetter() As Char

Return pLetterAfter

End Function

Public Sub SubsequentLetter(ByVal NewLetter As Char)

pLetterAfter = NewLetter

End Sub

Public Function PreceedingLetter() As Char

Return pLetterBefore

End Function

Public Sub PreceedingLetter(ByVal NewLetter As Char)

pLetterBefore = NewLetter

End Sub

'---------------------------------------------------------------------------------

'The following Getters/Setters are not polymorphic because they are not going to be used by the same classes so differently named procedures make for easier readability

'----------------------------------------------------------------------------------

Public Sub SetFrequencyINT(ByVal CalculatedFrequency As Integer)

pFrequencyINT = CalculatedFrequency

End Sub

Public Sub SetFrequencyPRCNT(ByVal CalculatedFrequency As Single)

pFrequencyPRCNT = CalculatedFrequency

End Sub

'RELATIVE FREQUENCIES OF LETTERS:::::(+/- 26 % OF THE RELATIVE FREQUENCIES)

'A = 8.167 +/- 2.12342

'B = 1.492 +/- 0.38792

'C = 2.782 +/- 0.72332

'D = 4.253 +/- 1.10578

'E = 12.702 +/- 3.30252

'F = 2.228 +/- 0.57928

'G = 2.015 +/- 0.5239

'H = 6.094 +/- 1.58444

'I = 6.966 +/- 1.81116

'J = 0.153 +/- 0.03978

'K = 0.772 +/- 0.20072

'L = 4.025 +/- 1.0465

'M = 2.406 +/- 0.62556

'N = 6.749 +/- 1.75474

'O = 7.507 +/- 1.95182

'P = 1.929 +/- 0.50154

'Q = 0.095 +/- 0.0247

'R = 5.987 +/- 1.55662

'S = 6.327 +/- 1.64502

'T = 9.056 +/- 2.35456

'U = 2.758 +/- 0.71708

'V = 0.978 +/- 0.25428

'W = 2.360 +/- 0.6136

'X = 0.150 +/- 0.039

'Y = 1.974 +/- 0.51324

'Z = 0.074 +/- 0.01924

Protected Overrides Sub Finalize()

MyBase.Finalize()

End Sub

End Class