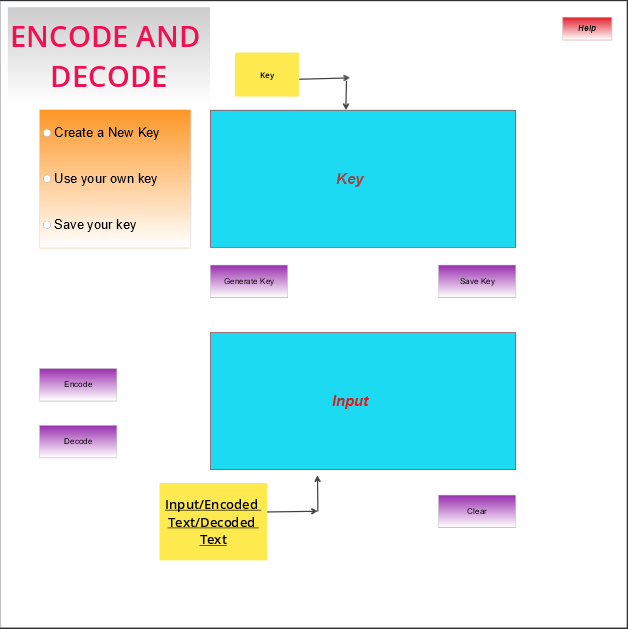
# INTRODUCTION

You must create software that will encrypt strings by the encryption rules ticket you created beforehand. This software will be able to decrypt a text after loading a rule or encrypting it from a new set of rules or with the one saved in database. If it is a new encryption key, the user will have the choice of selecting the type of database he want to it (SQL, ACCESS or TEXT FILE).

# INTERFACE:

* Help button that reads from a text file. Explain how the application works.
* For login, don’t use 2D array.
* User should have the choice of database when saving an encryption key.
* Form for Login, Encoder and Help (eg., frmLogin, frmEncoder, frmHelp).

# SKETCH OF USER INTERFACE:



# CODING SPECIFICATIONS:

The original key:

Àc0Â)nâÉvéÈ1hÊ$ëwTî2ïÔ3ûx}ÿÇk4r GÛAp5OKËW>B%Ct&ZQèD6EqUôF@HàI^J+çL7YMzVÜjN8PêR=SX-\*b/l<g{Ï9[eŸ].i,a;u!my#oÎ?(düs\_f

The base chain creates 4 unique random chains that will be our keys. 2 of them will be inverted on the display and during decryption will have to be inverted again to have the correct result.

The original text will be treated as follows: depending on whether the line is even or not, it will be inverted and will have its 2 dedicated keys, then depending on the position of the character, its new form will be defined whether or not even with the 2 keys.

Add a dummy character to each character, moreover characters that are not in the base string will remain as they are but accompanied by the dummy character.

# REQUIRED:

Inheritance, overloading, encapsulation, objects, reader/builder, re-use of objects as much as possible. Reduce the code in the click event of the forms.

Database connection: SQL, Microsoft Access, Text Files (stream reader/writer).

# CODING STANDARDS:

* Camel casing
* Maintain code to minimum on Click event of the forms
* Separate the functions per class
* Meaningful declaration of variables
* Prefix classes with c
* Prefix buttons with btn
* Prefix labels with lbl
* Prefix forms with frm
* Prefix radio buttons with rad
* Prefix richtextbox with rtxt
* Prefix combobox with cmb
* Comments for each method and variables

# CODING:

**Building the keys:**

* Send the original string in a list.
* Copy the original list in 4 different lists
* Sort the list randomly
* OnCLick event of new keys button: - Lists are sorted in an inverted manner.
* Concatenate each letter of each list into a variable with a line break in between each list
* Display the variable in the appropriate textbox.

# ENCODING:

1. On click of “Encode” button, the key from Top rich textbox is verified.
2. If length of key is 459, then this key is split on each Environment.New Line (\r\n) and sent into an array[].
3. All the 4 lists are cleared at this point.
4. All the 4 lists are filled with corresponding index values of the key from the array[].
5. List 2 and List 4 are reversed.
6. The input string is then split and sent into words Array[].
7. Then we loop through the words array and add the letters to the encoded string based on the line numbers and position of letters inside each line.

***If line number is EVEN:***

**If position is even:**

* We generate a random number from the count of letters in lstOriginal.
* This generated random number is stored in “theRnd”
* Now, we create a string variable **EncodedWord**.
* From the lstOriginal, the letter at “theRnd” position is copied into the **encodedWord.**
* The character at index I is copied into another variable named “theLetter”.
* If lstOriginal contains theLetter, we copy the index in another variable (ind).
* We check the value of ind variable inside List 2 and copy that value inside **encodedWord.**

**If position is odd:**

* We generate a random number from the count of letters in lstOriginal.
* The generated random number is stored in aRnd variable.
* Now, the letter at “aRnd” position in lstOriginal is copied into **encodedWord**.
* The character at index I is copied into another variable “aLetter”.
* If lstOriginal contains aLetter, we copy the index in another variable (ins).
* We check the value of ins variable inside List 2 and copy that value inside **encodedWord.**

***If line number is ODD:***

**If position is even:**

* We generate a random number from the count of letters in lstOriginal
* We check the value of lstOriginal at the index of the random number that we generated earlier.
* Then we loop through the letters in the line and store the letter inside theLetter variable.
* If lstOriginal contains theLetter, we copy the index of that value and store it in ind variable.
* We copy the value of this index at List 3 and the corresponding value is stored inside **encodedWord**.
* If lstOriginal does not contain theLetter, we copy theLetter in **encodedWord.**

**If position is odd:**

* We create a random number from the lstOriginal and store inside aRnd.
* We store the value of lstOriginal at the index equal to aRnd inside **encodedWord**.
* We then loop through the letters and store each odd position letter in letters.
* If lstOriginal contains aLetter(i.e. the first character of the loop), we take its index and store it inside ind variable.
* We then check the value of List 4 at this ind index and copy this value inside **encodedWord**.
* If lstOriginal does not contain this value, we simply put this first looped value inside **encodedWord**.

# DECODING:

***If line number is EVEN:***

**If position is even:**

* We generate a random number from the count of letters in lstOriginal.
* This generated random number is stored in “theRnd”
* Now, we create a string variable **decodedWord**.
* From the lstOriginal, the letter at “theRnd” position is copied into the **decodedWord.**
* The character at index I is copied into another variable named “theLetter”.
* If lstOriginal contains theLetter, we copy the index in another variable (ind).
* We check the value of ind variable inside List 2 and copy that value inside **decodedWord.**

**If position is odd:**

* We generate a random number from the count of letters in lstOriginal.
* The generated random number is stored in aRnd variable.
* Now, the letter at “aRnd” position in lstOriginal is copied into **decodedWord**.
* The character at index I is copied into another variable “aLetter”.
* If lstOriginal contains aLetter, we copy the index in another variable (ins).
* We check the value of ins variable inside List 2 and copy that value inside **decodedWord.**

***If line number is ODD:***

**If position is even:**

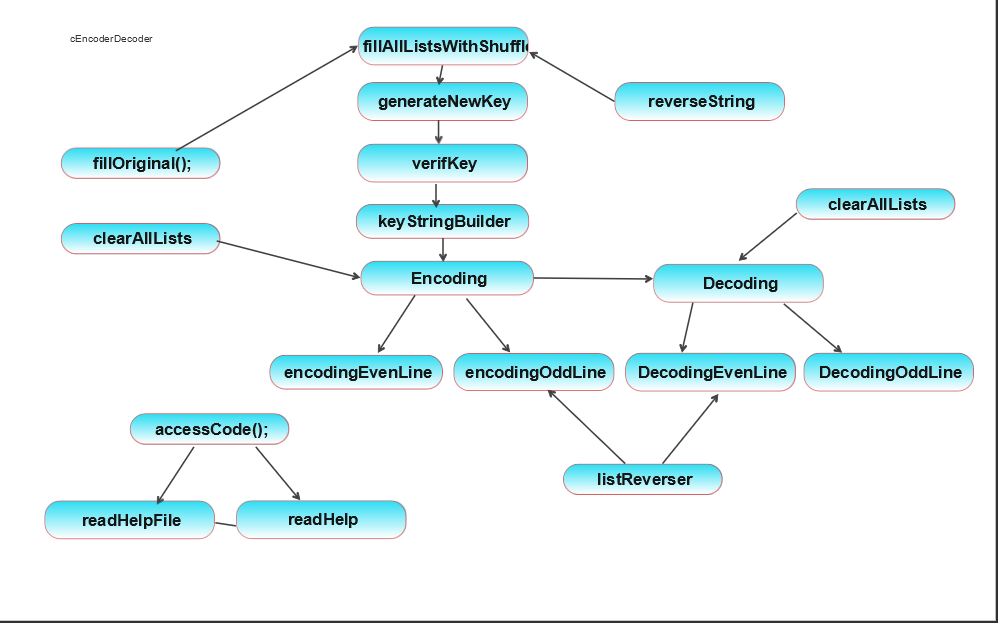
* We generate a random number from the count of letters in lstOriginal
* We check the value of lstOriginal at the index of the random number that we generated earlier.
* Then we loop through the letters in the line and store the letter inside theLetter variable.
* If lstOriginal contains theLetter, we copy the index of that value and store it in ind variable.
* We copy the value of this index at List 3 and the corresponding value is stored inside **encodedWord**.
* If lstOriginal does not contain theLetter, we copy theLetter in **decodedWord.**

**If position is odd:**

* We create a random number from the lstOriginal and store inside aRnd.
* We store the value of lstOriginal at the index equal to aRnd inside **decodedWord**.
* We then loop through the letters and store each odd position letter in letters.
* If lstOriginal contains aLetter(i.e. the first character of the loop), we take its index and store it inside ind variable.
* We then check the value of List 4 at this ind index and copy this value inside **decodedWord**.

If lstOriginal does not contain this value, we simply put this first looped value inside **decodedWord**.

**CLASSES USED:**



# DEVELOPMENT PROCESS

* Sketch the database: Please find the sketch in SKETCH OF USER INTERFACE section.
* Write down the list of methods we will need : Methods: accessCode(),fillOriginal(),clearAllLists(),listReverser(),keyStringBuilder(),Encoding(),encodingEvenLine(),encodingOddLine(),reverseString(),Decoding(),DecodingEvenLine(),DecodingOddLine(),generateNewKey(),fillAllListsWithShuffle(),verifKey(),readHelpFile(),readHelp()].
* Make a diagram of all classes you are planning on using
* Sketch the interface
* Creation of solution
* (Share Version) – github
* Create interface and all the classes
* Create the login form
* Test and adjust all functionalities

# CODING THE LOGIN FORM

* We create a method named “accessCode” to generate a string of 4 random numbers.
* We create an input for the user
* We check if the user input is same as the code generated by the 4 random numbers.
* If it is equal, user is able to login,
* When user logins, we hide the login form
* If user input not equal to generated code, user sees a message “try again”.

# CODING THE HELP FORM

* Create a button named “Help”.
* On the click of the button, a form will open will the help text.
* The text in the help form comes from a file named help.txt.
* Create a class cHelp that extends the cConnect class.
* The cConnect class contains the readHelpFile Function and the string path.
* The readHelpFile function is called in the cHelp class.
* The readHelpFile function inherits ReadAllText function and reads all the text in the Help.txt File

# FINAL PHASE:

* In the final phase of the application development process, the application is tested and put into production.