Gareth Jones (JLB) – Computing (SAH)

Computing: Encryption

1. The task

Encode, using transposition, a simple sentence as follows. The plaintext is written out in a grid (row-by-row) but is transmitted column by column. For example, the sentence "THE PIGS FLY SOUTH IN SUMMER" could be written in a 5x6 grid:

T	H	E	*	P
I	G	S	*	F
L	Y	*	S	O
U	T	H	*	I
N	*	S	U	M
M	E	R	*	*

The code would then be TILUNMHGYT*EES*HSR**S*U*PFOIM*

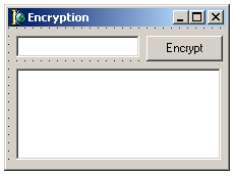
Write a program that takes a short sentence as input and using the above transposition algorithm display the encrypted text.

Convert the code back into plain text.

Extension: Read the message from a file.

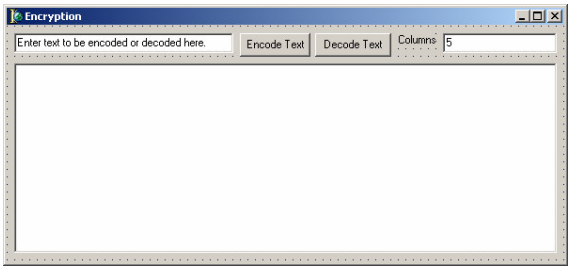
2. Design and development

Initial design:

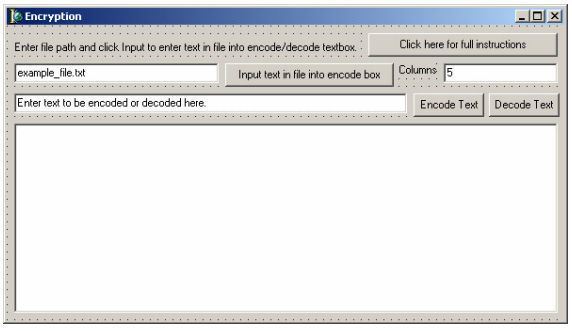


My first design consisted of a memo box, a textbox and a button. When I first approached the problem the entry in this textbox had to be 30 characters long. Clicking the button displayed the encoding grid and the encoded text.

Design development:



To add the ability to decode the text I added another button, and when I added the feature of choosing the number of columns (replacing the constant I had defined in the program by a variable read from a textbox) I added another text box.



If the user did not enter a value in the two textboxes the program crashed, so I added sample text under the "Text" property of the two textboxes with valid text, so that if the user clicked to encode or decode the text it would not crash.

Final design:

In my final design I increased the length of the text entry input display, added a textbox and button to allow text to be inputted from a file and added an instructions button. All the fields have data in them which is valid so if the user does not enter data the program

will not crash. I used labels and text entered into the textboxes to guide the user. The cipher grid and the encoded/decoded text is printed to the memo box.

Pseudo code:

Encode text button

```
Clear.Memobox
Text ← Textentry_textbox
Columns ← Columns_textbox
LengthofText ← Length of 'Text'
if (LengthofText is a multiple of Columns) then
  Rows ← The multiple of Columns that LengthofText is
otherwise
  Rows ← Nearest multiple above the value
  Until LengthofText = Rows * Columns
    Textbox ← Textbox + '*'
  MakeArray(Name:EnteredTextArray,Columns:Columns,Rows:Rows)
    CharacterNumber ← 1
    for i ← 0 to Rows
      for j ← 0 to Columns
      EnteredTextArray(Columns,Rows) ← Text(CharacterNumber)
      CharacterNumber ← CharacterNumber + 1
    for k ← 0 to Rows
      for l ← 0 to Columns
        if EnteredTextArray(Columns,Rows) = ' ' then
          EnteredTextArray(Columns,Rows) ← '*'
    Memobox.'Here is the cipher grid:'
    for m ← 0 to Rows
      Clear.Display
      for n ← 0 to Columns
        Display ← Display + EnteredTextArray(Columns,Rows) + Tab
        Memobox.Show.Display
  MakeArray(Name:CipheredArray,Columns:Rows,Rows:Columns)
    for o ← 1 to Rows
      for p ← 1 to Columns
        CipheredArray(Rows,Columns) ← EnteredTextArray(Columns,Rows)
    for q ← 1 to Columns
      for r ← 1 to Rows
        CipheredText:= CipheredText + Cipher(Rows,Columns)
    Memobox.'Here is the ciphered text'
    Memobox.Show.CipheredText
```

Encode text button

```
Clear.Memobox
Text ← Textentry_textbox
Columns ← Columns_textbox
LengthofText ← Length of 'Text'
if (LengthofText is a multiple of Columns) then
  Rows ← The multiple of Columns that LengthofText is
otherwise
  Rows ← Nearest multiple above the value
  Until LengthofText = Rows * Columns
    Textbox ← Textbox + '*'
  MakeArray(Name:EncodedTextArray,Columns:Rows,Rows:Columns)
  CharacterNumber ← 1
    for i ← 0 to Columns
      for j ← 0 to Rows
        EncodedTextArray(Rows,Columns) ← Text(CharacterNumber)
        CharacterNumber ← CharacterNumber + 1
    Memobox.'Here is the cipher grid:'
    for k ← 0 to Columns
      Clear.Display
      for l ← 0 to Rows
        Display ← Display + EncodedTextArray(Rows,Columns) + Tab
        Memobox.Show.Display
    for m ← 0 to Columns
      for n ← 0 to Rows
        if EncodedTextArray(Rows,Columns) = ' ' then
          EncodedTextArray(Rows,Columns) ← '*'
    for o ← 1 to Columns
      for p ← 1 to Rows
        DecodedText:= DecodedText + EncodedTextArray(Rows,Columns)
```

```
Memobox.'Here is the ciphered text'
Memobox.Show.DecodedText
```

Use text from file button

```
FileToOpen ← FilePath_textbox
Reset.FileToOpen
Buffer ← FileToOpen.Lines
Textentry_textbox ← Buffer
```

Instructions button

```
Memobox.Show.Instructions
```

Data dictionary:

Identifier	**Type**	**Formatting and reasoning**
i *through to* r	Integer	Loop counters
LetterNumber	Integer	Incrementing value within loops
Textbox	String	Characters entered in *txtEntry*
EntryLength	Integer	Number of characters in *Textbox*
PrintOut, CipheredText, RawText	String	Data from arrays, created by loops to be printed to the memo box
ColumnsEntry, Columns	Integer	User entered value, positive whole number of columns. *Columns* calculated for use in loops as dynamic arrays count from 0..*n-1* yet human entry counts from 1..*n*
Rows	Integer	Calculated value for table dimensions, only integer values comprehendible
TextEntry, EncodedText	Dynamic multi-dimensional array	Dimensions of array can only be assigned once *EntryLength* and *Columns* are known (so *Rows* can be calculated)
Cipher		Has dimensions opposite of *TextEntry* into which characters in array can be copied
Tab	Constant	= chr(9) Makes code easier to read as Tab is used regularly
Directory, Buffer	String	Contain data from textfile processes
FileToOpen	TextFile	Points at textfile entered by user in *txtFilePath*

3. Problems

Whilst writing the program I encountered a number of errors.

Using dynamic arrays caused a number of difficulties. By writing the program using a fixed length entry, defining constants under the *const* heading allowed me to gain a simple functioning version before changing to using variables.

I was not aware that in a dynamic array, unlike in a normal array in *Delphi* you can only specify an upper limit. This meant that when I had assigned the values of the loop counters *i ←1 to Columns* as in pseudo code the first row and column of each array was left blank. I resolved this by reassigning *Columns* the value *Columns – 1* and looping from *0* to *Columns*.

Initially I left all my textboxes blank, however if the user clicked a button without completing the necessary fields the program would then crash. I therefore added default

values to all the textboxes, which if left at default values function correctly. This also makes the user interface more intuitive as it is clear what should be typed where.

4. Unit Listing

Here is the source code copied from *Delphi* (*Automatically generated code omitted*):

```
//
// **
// Encode button
// **
//
procedure TfrmMain.btnEncodeClick(Sender: TObject);
var
  EntryLength, i, j, k, l, m, n, o, p, q, r, LetterNumber, Columns, Rows,
ColumnsEntry: Integer;
  Textbox, PrintOut, CipheredText: String;
  TextEntry: array of array of Char;
  Cipher: array of array of Char;
const
  Tab = chr(9);
begin
  mmoDisplay.Clear;
//
// Read values user has entered
//
  Textbox:= txtEntry.Text;
  ColumnsEntry:= StrToInt(txtColumns.Text);
//
// Validate ColumnsEntry
//
  If (ColumnsEntry <= 0) then
    ShowMessage('Invalid number of columns entered')
  else
  Columns:= ColumnsEntry - 1;
  EntryLength:= Length(Textbox);
//
// Calculate value of Rows dependant on entry being a multiple of ColumnsEntered
// (value chosen by user)
//
  if ((EntryLength mod ColumnsEntry) = 0) then
    Rows:= (EntryLength div ColumnsEntry)
  else
    Rows:= (EntryLength div ColumnsEntry) + 1;
//
// Increase string TextBox (entered text) to length required for encryption (i.e. a
// multiple of ColumnsEntry) by adding as many asterisks as necessary.
//
    while (Rows * ColumnsEntry) > (Length(Textbox)) do begin
      Textbox:= Textbox + chr(42);
    end;
//
// Copy characters from text box to an array, called TextEntry
//
    Setlength(TextEntry,ColumnsEntry,Rows);
    LetterNumber:= 1;
      for i:= 0 to Rows do begin
        for j:= 0 to Columns do begin
        TextEntry[j,i]:= Textbox[LetterNumber];
        Inc(LetterNumber)
        end;
      end;
//
// Cycle through array, changing the spaces to asterisks
//
      for k:= 0 to Rows do begin
        for l:= 0 to Columns do begin
          if (TextEntry[l,k] = chr(32)) then
            TextEntry[l,k]:= chr(42)
        end;
      end;
//
// Print the array to a memo box
//
```

```
      mmoDisplay.Lines.Add('Here is the cipher grid:');
      mmoDisplay.Lines.Add('');
      for m:= 0 to Rows do begin
        PrintOut:='';
        for n:= 0 to Columns do begin
        PrintOut:= PrintOut + TextEntry[n,m] + Tab
        end;
        mmoDisplay.Lines.Add(PrintOut);
      end;
//
// Cipher the array (copy to another array column --> row and row --> column
//
    Setlength(Cipher,Rows,ColumnsEntry);
      for o:= 0 to (Rows - 1) do begin
        for p:= 0 to Columns do begin
          Cipher[o,p]:= TextEntry[p,o]
        end;
      end;
//
// Print Cipher to string
//
      for q:= 0 to Columns do begin
        for r:= 0 to (Rows - 1) do begin
          CipheredText:= CipheredText + Cipher[r,q]
        end;
      end;
//
// Print string to textbox
//
      //mmoDisplay.Lines.Add('');
      mmoDisplay.Lines.Add('Here is the ciphered text');
      mmoDisplay.Lines.Add(CipheredText);
end;
//
// **
// Decode button
// **
//
procedure TfrmMain.btnDecodeClick(Sender: TObject);
var
  Textbox, PrintOut, RawText: String;
  ColumnsEntry, Columns, EntryLength, Rows, LetterNumber, i, j, k, l, m, n, q, r:
Integer;
  EncodedText: array of array of Char;
const
  Tab = chr(9);
begin
  mmoDisplay.Clear;
//
// Read values user has entered
//
  Textbox:= txtEntry.Text;
  ColumnsEntry:= StrToInt(txtColumns.Text);
//
// Validate ColumnsEntry
//
  If (ColumnsEntry <= 0) then
    ShowMessage('Invalid number of columns entered')
  else
  Columns:= ColumnsEntry - 1;
  EntryLength:= Length(Textbox);
//
// Calculate value of Rows dependant on entry being a multiple of ColumnsEntered
// (value chosen by user)
//
  if ((EntryLength mod ColumnsEntry) = 0) then
    Rows:= (EntryLength div ColumnsEntry)
  else
    Rows:= (EntryLength div ColumnsEntry) + 1;
//
// Increase string TextBox (entered text) to length required for encryption (i.e. a
// multiple of ColumnsEntry) by adding as many asterisks as necessary.
//
    while (Rows * ColumnsEntry) > (Length(Textbox)) do begin
      Textbox:= Textbox + chr(42);
    end;
//
```

```
// Copy characters from text box to an array, called TextEntry
//
    Setlength(EncodedText,Rows,ColumnsEntry);
    LetterNumber:= 1;
      for i:= 0 to Columns do begin
        for j:= 0 to (Rows - 1) do begin
        EncodedText[j,i]:= Textbox[LetterNumber];
        Inc(LetterNumber)
        end;
      end;
//
// Print the array to a memo box
//
      mmoDisplay.Lines.Add('Here is the cipher grid:');
      mmoDisplay.Lines.Add('');
      for m:= 0 to Columns do begin
        PrintOut:='';
        for n:= 0 to (Rows - 1) do begin
        PrintOut:= PrintOut + EncodedText[n,m] + Tab
        end;
        mmoDisplay.Lines.Add(PrintOut);
      end;
//
// Change the asterisks to spaces
//
      for k:= 0 to Columns do begin
        for l:= 0 to (Rows - 1) do begin
          if (EncodedText[l,k] = chr(42)) then
            EncodedText[l,k]:= chr(32)
        end;
      end;
//
// Print Cipher to string
//
      for q:= 0 to (Rows - 1) do begin
        for r:= 0 to Columns do begin
          RawText:= RawText + EncodedText[q,r]
        end;
      end;
//
// Print string to textbox
//
      //mmoDisplay.Lines.Add('');
      mmoDisplay.Lines.Add('Here is the ciphered text');
      mmoDisplay.Lines.Add(RawText);
end;
//
// **
// Input file code
// **
//
procedure TfrmMain.btnInputFilePathClick(Sender: TObject);
var
  FileToOpen : TextFile;
  Directory,Buffer: string;
begin
  Directory := txtFilePath.Text;
  AssignFile(FileToOpen, Directory);
  Reset(FileToOpen);
  ReadLn(FileToOpen, Buffer);
  txtEntry.Text:= Buffer;
end;
//
// **
// Instructions button code
// **
//
procedure TfrmMain.btnInstructionsClick(Sender: TObject);
begin
  mmoDisplay.Lines.Add('**ENCRYPTION**');
  mmoDisplay.Lines.Add('Enter text to be decoded either manually or select a' +
   ' file to import the data from, entering the full path or just the file '+
   'name if the file is in the same directory as this program. Then enter '+
   'the number of columns to encrypt or decrypt the data with. Click either '+
   'the encrypt or decrypt button then as appropriate.');
end;
```

```
end.
```

5. Testing

Encode button

Aim to test text to encipher of both a multiple and not a multiple of the columns. Test example given in briefing. Test boundary values of ColumnsEntry to check validation works.

Initially when I first tested the program the first row and column of the cipher array seemed to be missing.

Test	Why testing?	Expected outcome	Actual outcome
Check example given in brief, "THE PIGS FLY SOUTH IN SUMMER" with five columns.	Check that brief has been met.	Cipher grid printed to memo box, cipher displayed as in brief. Cipher in grid and in line contains no spaces, replaced by asterisks.	As expected.
No text entered.	Check program does not crash.	Error is generated by execution of array, Clicking "OK" resets program.	As expected. *See figure 1.*
13 Characters "JANE RAN FAST" of text entered, 3 columns selected.	Checking "columns" option functions.	String extended to 15 characters, cipher grid 3 by 5 created, cipher "JEAFTA*NA*NR*S*" created.	As expected. *See figure 2.*
24 Characters, 4 Columns.	Check extra row of asterisks not added.	Encryption works, no extra line of asterisks inserted.	As expected.
The text "THE PIGS FLY SOUTH IN SUMMER" is entered and "0", "-1" and "1" columns are chosen. (3 tests)	Check columns error for "-1," "0," executes correctly for "1".	Error message displayed, 1 column accepted.	As expected. *See figure 3.*

Figure 1

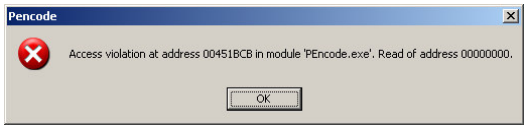


Figure 2

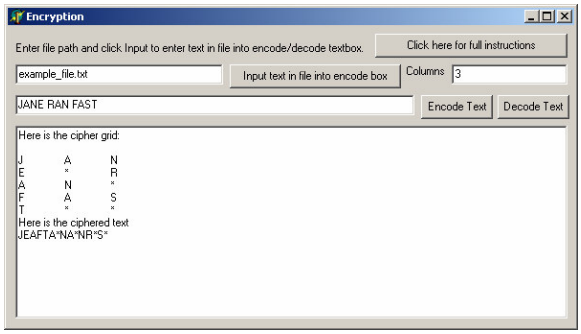
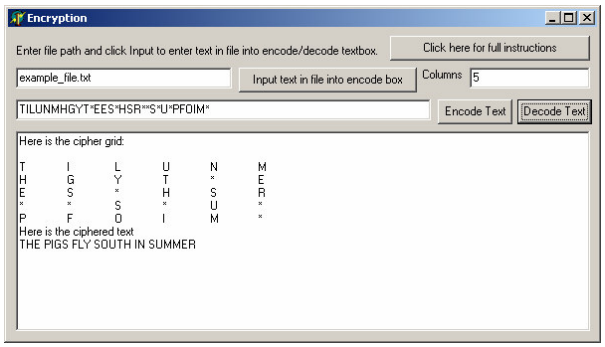


Figure 3

Decode button

Test	**Why testing?**	**Expected outcome**	**Actual outcome**
Check all examples in "*Encode button*" section of testing decode correctly.	Check operation is reversible.	Original input regenerated.	As expected. *See figure 4.*
Check where Columns is not a multiple of EntryLength decryption still functions.	Check decoder can handle these strings.	Generate original input.	As expected.
Check invalid columns entry generates error "-1," "0," "1"	Check columns error for "-1," "0," executes correctly for "1".	Error message displayed, 1 column accepted.	As expected. *See figure 3.*

Figure 4



Use text from file button

Test	**Why testing?**	**Expected outcome**	**Actual outcome**
Check example file inputs data in file to text entry box.	Check default settings function correctly.	Text inputted correctly.	As expected. *See figure 5 (next page).*
Enter file name that does not exist.	Check error message displayed.	Error message – "File not found" displayed.	As expected. *See figure 6.*
Full path of file given.	Check path recognised.	Text inputted correctly.	As expected.
Textbox left blank.	Check error message displayed.	Error message – "I/O error" displayed.	As expected. *See figure 7.*

Figure 6

Figure 7

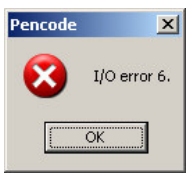
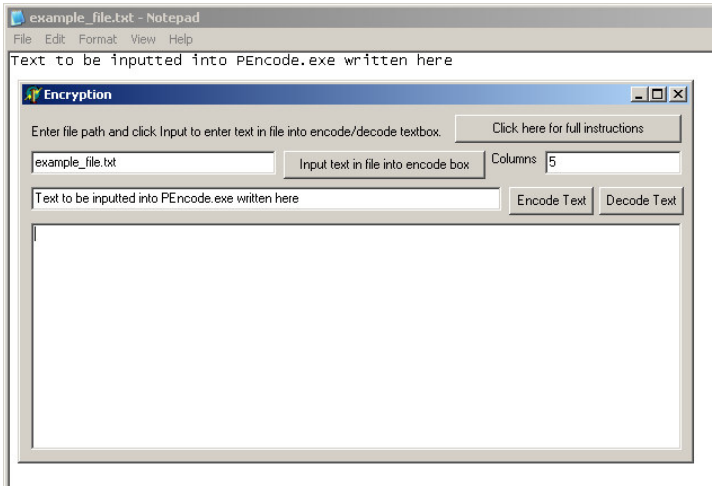


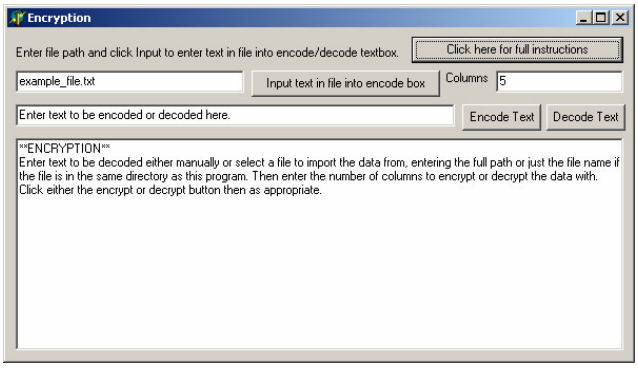
Figure 5



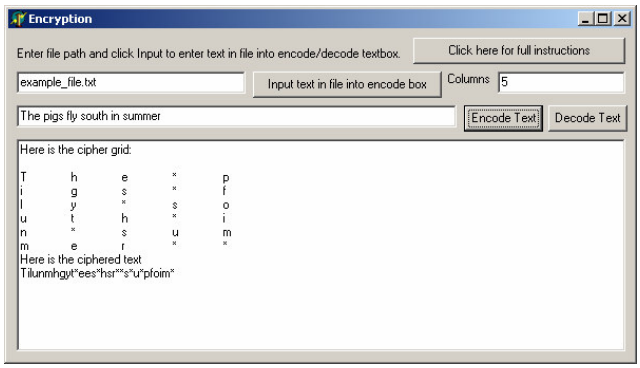
Instructions button

Test	**Why testing?**	**Expected outcome**	**Actual outcome**
Click instructions button.	Check functionality of instructions button.	Instructions printed to memo box.	As expected. *See figure 8.*

Figure 8



6. Sample run



7. Appraisal

I have met the requirements of the task by creating a program that can encode the text "THE PIGS FLY SOUTH IN SUMMER" using the algorithm outlined in the brief. The program I have created can accept any characters, apart from an asterisk, not just uppercase letters.

I chose to complete the extension so my program can import data from a text file, and extended the program using multi-dimensional dynamic arrays to allow for variable length input for both encryption and decryption.

I also added an instructions button and option for the user to specify the columns used to encrypt data. This option was simple to perform as throughout the program I had defined the Columns by a constant, which was simple to change to a variable. This was after feedback on the last programming assignment.

I feel much more confident about arrays, and whilst writing the program tackled some logic errors using the *Delphi* debugging "watch" tool and have spent time experimenting with new features and concepts.

As I review the code I notice that when attacking the decryption task I, having a firmer grasp of how to use dynamic arrays, was able to write the code more efficiently. This required using fewer variables and the code was generally more concise.

I found the task initially very challenging yet by approaching in a variety of smaller tasks, clearly split up by annotation within section 4 (*Unit listing*) found the task much more manageable. Initially I created the program using fixed lengths, defined as constants, which made trouble shooting much easier and the progression to variable length entries much easier than coding from scratch.