Giorgi Solomnishvili; RedID: 822164480

[Encryption/Decryption of a Text]

4/27/2019

Content

(* - Mandatory)

1*. Introduction

I have created a program that asks the user to enter a text and this program will encrypt or decrypt the text, depending on the user's choice. For encryption/Decryption, the program utilizes the Playfair Cipher method with some modifications. Those modifications' are the ideas of the author of this project.

Playfair cipher is an example of symmetric encryption, meaning that the same keyword is used for encryption and decryption. The keyword Is a word (maximal 25 different letters), chosen by the user. The different keywords yield in different cipher. The program creates int type array with 25 elements. ASCII values of the keyword fill the table. The rest of the table is filled with ASCII values of the letters which are not included in the keyword. Therefore, with different keywords, a different combination of letters is generated. There are 26 letters in the English alphabet but the array has room for 25 letters. As a result, one letter always stays hidden. Having different keywords might yield in having different letters hidden. For example, if a keyword is "**ASEMBLY**", the table will look like this: (Solomnishvili, 2019, "Progress Report")

A S E M B
L Y C D F
G H I J K
N 0 P Q R
T U V W X

(I will convert those letters into ASCII code.)

As seen from the table, the letter Z is hidden. After constructing this table, the user will enter the text. Letters from the text will be paired and encrypted based on the following 6 rules: (Solomnishvili, 2019, "Progress Report")

- If paired letters belong to the same column in the table, each of them will be replaced by their southern neighboring letters. For example, **AG** will be replaced by **LO.**
- If paired letters belong to the same row in the table, each of them will be replaced by their right neighboring letters. For example, **AE** will be replaced by **SM.**
- If they are not in the same column or row, they will form a square. For example, if paired letters are **ED,** the square will be **EMCD**. The encrypted text will have **MC** in place of **ED**.
- If one of the paired letters is a dot or comma, their values do not change. They just swap their places. For example: A. will become .A
- if one of them is j, they will swap their places.
- If one of them is space, nothing changes.

This is a mixed C and Assembler project, meaning that main function is written in C and it calls other C functions and Assembly subroutines. The program uses C functions such as fillthetablee, ommitdoubleletter to prepare the table. Moreover, C functions ASCIIvalue and ReverseASCIIvalue are used to convert the text into the array of integers and convert the array of integers into the text. Subroutines, such as findmissingletter and findPosition, is used to find the hidden letter (the one that is not in the table) and to find to-be-ciphered letters position in the table. The last two main subroutines are encryption and decryption. They perform encryption and decryption of the text. In this report, those functions and subroutines will be discussed in details.

2. Pseudocode

Functions in C

The First Function - int fillTheTable(int table[], char keyword[]): This function takes a string and an int array as parameter. The string array is a keyword, entered by the user; int array has 25 elements. The function checks if the keyword has the same letter twice. If this is the case, the function replaces the second letter with a hyphen (rr is replaced by -r). Then the function calls another function - singleLetter (discussed below). This function eliminates all hyphens by shifting the whole string left by one character.
After that, fillTheTable function fills the table with ASCII values of the keyword. The rest of the table is filled with the ASCII values of the rest of the alphabet.

```
DECLARE int I,j,k,l,m,f = 0

FOR I is 0, till I is less than the length of keyword
    FOR j is i+1, till j is less the length of keyword
        IF keyword[i]=keyword[j]
            SET Keyword[i] to '-'
            BREAK
        END IF
    END FOR
END FOR

SET counter to 1

    WHILE counter
        CALL singleletter
        SET singlletter's return value to counter
    END WHILE

FOR i=0, till I is less than length of keyword
    SET table[i] to keyword[i]
END FOR

SET l to i
FOR j is 97, til j is less than 122
    SET f to 0
    FOR k is 0, k is less than i
        IF j is table[k]
            SET f to 1
        END IF
        IF f is 0
```

```
            BREAK
            END IF
    END FOR
IF f is 1
    CONTINUE
END IF
Table[l]=j
INCREMENT l
END FOR
```

The Second Function - int singleletter(char keyword[]):

If a keyword has two or more same letters, fillTHETable function replaces all of them with '-' (except one). The singleletter function looks for '-'s and if it finds them, it shifts keyword to the left by 1 character. If the function does not find any '-', it returns 0.

```
SET counter to 1

FOR I is 0, till less than the length of the keyword
    IF keyword[i] is '-'
            Increment counter
            FOR l=i+1, till l is less than the length of the keyword
                    Keyword[l-1]=keyword[l]
            END FOR
    END IF
END FOR
RETURN counter
```

The Third Function - void asciivalues(char text[], int inttext[]): The function assigns ASCII values of the letters of the text to the elements of intext array.

```
FOR I is 0, till less than the length of the text
    Intext[i]=text[i]
END FOR
```

SUBROUTINES IN ASSEMBLY

The First Subroutine – FindMissingLetter: Table is filled with 25 letters. There 26 letters in the English alphabet. Thus, one of those letters is hidden. This function finds that hidden letter.

The function takes one int array parameter – table (which is already filled by the ASCII values of the alphabet).

Pointer to the first element of the table is moved to the register ebx
Register edx is set to 96
Register ecx is set to 1

Loop1
    If ecx is 0
        Jump after
    SET register ecx to 0
    INCREMENT edx
    SET register esi to -1
LOOP11 (nested loop)
Until esi is less than 25
Move esi-th element from table to eax
If eax I does not equal edx
    Jump to loop11
    Move 1 to ecx
    Jmp loop1

After1

Return whatever is in edx

The Second Subroutine – _Encryption: This function takes two int pointer, int array and char as a parameter. int pointers represent letters of the text which need to be encrypted. The int array is the table. Char parameter – ommittedletter, is a hidden letter (the one which is not included in the table).

We have several possibilities:

1 To-be-encrypted letters contain hidden letter
2 Both of To-be-encrypted letters are hidden letters
3 One of To-be-encrypted letters is not alphabet (it is a dot, comma, …..)
4 To-be-encrypted letters are in the same row of the table
5 To-be-encrypted letters are in the same column of the table
6 To-be-encrypted letters are not in the same row or column

1,2 – To-be-encrypted letters contain hidden letter

I compare To-be-encrypted letter to the ommittedletter.
IF they are the same, I xor this To-be-encrypted with '#'. The result of this operation replaces To-be-encrypted letter.
Moreover, I have two flags. One of them is set to 1 and the other is set to 2.
If the first To-be-encrypted letter is ommittedletter, the first flag is set to 0
If the second To-be-encrypted letter is ommittedletter, the second flag is set to 0

After that, the first and second flags are ored. **IF the result is 0**, then both of To-be-encrypted letters are ommittedonce, they are already encrypted and the function jumps to the end. **If the result is not 3,** then either of them is ommittedletter, the program jumps to the swap part, those two letters swap places. **IF the result is 3,** then either of them is an ommittedletter, none of them is encrypted and the function continues execution.

3 - Both of To-be-encrypted letters are hidden letters

The function checks if one of the to-be-encrypted letters is a dot, comma, etc. If that is the case, the program will jump to the swap.

4 – To--be-encrypted letters are in the same row of the table

The encryption function calls another function called findposition. Findposition finds the index of to-be-encrypted letter. It also determines to which column and row does the to-be-encrypted letters belong.

After that, the program determines which to be encrypted letter is right, meaning whose column number is greater. The other letter is encrypted right away. It gets the value of the (position)th element from the table.

Then the function checks if the letter, who is right, is in the fifth column. If not, the second letter is encrypted in the same way as the first one. If the second letter is in the fifth column, the position is decremented by 5 and the second letter takes the value of (position-5)th element from the table.

5 - To--be-encrypted letters are in the same column of the table

The encryption function calls another function called findposition. Findposition find the index of to-be-encrypted letter. It also determines to which column and row does the to-be-encrypted letters belong.

After that, the program determines which to be encrypted letter is below, meaning whose row number is greater. The other letter is encrypted right away. Its position is incremented by 4 and it is replaced by position+4th element from the table.

Then the function checks if the letter, who is below, is in the fifth row. If not, the second letter is encrypted in the same way as the first one. If the second letter is in the fifth row, the position is decremented by 21 and the second letter takes the value of (position-21)th element from the table.

6 - To-be-encrypted letters are not in the same row or column

In that case, the position of the first letter is computed with the following formula:

$P_1=(r_2-1)+c_1$

P_1 – position of the first encrypted letter.

r_2- number of the row of the second letter

c_1 – the number of the column of the first letter

The position of the second letter is computed with the formula:

$P_2=(r_1-1)+c_2$

P_2 – position of the first encrypted letter.

R_1- number of the row of the second letter

C_2 – the number of the column of the first letter

The Third Subroutine – _Dencryption: This function takes two int pointer, int array and char as a parameter. int pointers represent letters of the text which need to be encrypted. Int array is table. Char parameter – ommittedletter, is a hidden letter (the one which is not included in the table).

This function is like the Encryption function. The only difference is when letters are in the same row or in the same column.

If they are in the same column, their position is decremented by 6. If one of them is in the first row, its position is incremented by 19.

If they are in the same row, their position is decremented by 2. If one of them is in the first column, its position is incremented by 3.

The Fourth Subroutine – _FindPosition: This function takes a char parameter and an int array.

A char parameter is to-be-encrypted letter.
An int array is the table.

The function fetches values from the table and compares with to-be-encrypted letter. The function counts iteration. The number of iteration when the values match is the position of the letter in the table.

To find a number of the row, we divide the position by 5. If we get no remainder, the result of the division is the number of row and column is 5. Otherwise, the result of division +1 is the number of row and (5- (5 times the number of row -position)) is the number of the column.

3*. All source code for each function in C or assembly language

Source Code for C:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int singleletter(char keyword[]){
   int i,j,l;
   int counter=0;
   for(i=0;i<strlen(keyword);++i){
      if(keyword[i]=='-'){
            ++counter;
         for(l=i+1;l<=strlen(keyword);++l){
            keyword[l-1]=keyword[l];
         }
      }
   }
   return counter;
}
int fillTheTable(int table[], char keyword[]){
   int i=0;
   int j,k,l,m;
   int f=0;

   for(i=0;i<strlen(keyword);++i){// checks if one letter is more than once
      for(j=i+1;j<strlen(keyword);++j){
         if(keyword[i]==keyword[j]){
            keyword[i] = '-';
```

```
            break;
          }
        }
    }
    int counter=1;
    while(counter) counter=singleletter(keyword);

    for(i=0;i<strlen(keyword);++i) table[i] = (int)keyword[i];

    l=i;
    for(j=97;j<122;++j){
          f=0;
       for(k=0;k<i;++k){
          if(j==table[k]){
             f=1;
          }
          if(f==1) break;
       }
       if(f==1) continue;

       table[i]=j;
       l++;
       i++;
    }
    return j;
}

void asciivalues(char text[], int inttext[]){
    int i;
    for(i=0;i<strlen(text);++i) inttext[i] = text[i];
}
void reverseAscii(char text[], int inttext[]){
    int i;
    for(i=0;i<strlen(text);++i)
    text[i] = inttext[i];
}
void gotoxy(int x, int y){
    int i=0;
    for(i=0;i<y;++i) printf("\n");
    for(i=0;i<x;++i) printf(" ");
}
void display(){
    int* indexofchoice;
    int num = 1;
    indexofchoice = &num;
    system("cls");
    system("color 0A");
    gotoxy(34,4);
    printf("E N C R Y P T I O N / D E C R Y P T I O N\n");
    gotoxy(44,1);
    printf("%d. ENCRYPTION\n",*indexofchoice);
    gotoxy(44,1);
    printf("%d. DECRYPTION\n",(*indexofchoice)+1);
```

```
    gotoxy(44,1);
    printf("%d. QUIT\n",(*indexofchoice)+2);
    gotoxy(44,1);
    printf("Enter Your Choice: ");
}
int main()
{
    char keyword[100];
    int table[25];
    char text[1000];
    int i,omittedLetter,k;
    char userchar[2];

    display();
    gets(userchar);
    system("cls");


        if(userchar[0]=='3') return 0;

    if(userchar[0]=='1'){
        while(1){
    printf("Enter the Keyword: ");
    gets(keyword);
    printf("Enter the text: ");
    gets(text);
    if(strlen(text)%2!=0) strcat(text," ");
    int inttext[strlen(text)];
    asciivalues(text,inttext);

    fillTheTable(table,keyword);
    omittedLetter=FindMissingLetter(table);

    for(i=0;i<(strlen(text)-1);i=i+2)
    k=Encryption(&inttext[i],&inttext[i+1],table,omittedLetter);

    reverseAscii(text,inttext);
    printf("Encrypted text: %s\n",text);

    printf("Do you want to Encrypt again? (y/n) ");
    gets(userchar);
    if(userchar[0]=='n'||userchar[0]=='N') return main();

    system("cls");}
    }
    else if(userchar[0]=='2'){
        while(1){
    printf("Enter the Keyword: ");
    gets(keyword);
    printf("Enter the text: ");
    gets(text);
    if(strlen(text)%2!=0) strcat(text," ");
    int inttext[strlen(text)];
```

```
        asciivalues(text,inttext);

        fillTheTable(table,keyword);
        omittedLetter=FindMissingLetter(table);

        for(i=0;i<(strlen(text)-1);i=i+2)
        k=Dencryption(&inttext[i],&inttext[i+1],table,omittedLetter);

        reverseAscii(text,inttext);
        printf("Decrypted text: %s\n",text);

        printf("Do you want to Decrypt again? (y/n) ");
        gets(userchar);
        if(userchar[0]=='n'||userchar[0]=='N') return main();

        system("cls");}
        }

        return 0;
    }
```

Source Code For Subroutines:

1 - FindMissingLetter:

```
# this function tells us which letter is missing
.global _FindMissingLetter
_FindMissingLetter:

pushl %ebp
movl %esp, %ebp
subl $12, %esp

movl 8(%ebp), %ebx # table is here

movl $96, %edx #check variable
# movl $0, %esi  # i
movl $25, %eax
movl $1, %ecx # f

loop1:
cmpl $0, %ecx
jz after1
movl $0, %ecx
incl %edx

movl $-1, %esi  # i
loop11:
```

```
incl %esi
cmpl $25, %esi
jz loop1
movl (%ebx,%esi,4), %eax
cmpl %eax, %edx
jnz loop11
movl $1, %ecx
jmp loop1

after11:
after1:
movl %edx, %eax
movl %ebp, %esp
popl %ebp
ret
```

2 – FindPosition

```
#this finds position of the letter in the table
.global _FindPosition
_FindPosition:
# function uses eax, ebx, ecx, esi
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %ebx  #Here is address to table
movl 12(%ebp), %ecx  #Here is character, whose position is up to be determined

movl $-1, %esi #index
LoopForPosition:
incl %esi
movl (%ebx,%esi,4), %eax
cmpl %eax, %ecx
jnz LoopForPosition
incl %esi # here is the position of the letter

#determine number of row
xorl %edx, %edx
movl %esi, %eax
movl $5, %ecx
idivl %ecx #position/5 # result is in eax, remainder is in edx
cmpl $0, %edx
jz fivemultiple
incl %eax
fivemultiple:
#number of row determined it is in eax

#determine number of collumn
imull %eax, %ecx  #  5 x row
subl %esi, %ecx   #5 x row - position
```

```
subl $5, %ecx
imull $-1, %ecx
#number of collumn determined, in %ecx

#return values :
#		position %esi
#		row	%eax
#		collumn  %ecx


movl %ebp, %esp
popl %ebp
ret
```

3 – Encryption

```
.global _Encryption
_Encryption:

pushl %ebp
movl %esp, %ebp
subl $40, %esp  # four local variable

movl 8(%ebp), %ebx   #  ebx = first  pointer
movl 12(%ebp), %ecx  #  ecx = second pointer
movl 20(%ebp), %edx  #  edx = ommittedletter

movl (%ebx), %ebx    #  ebx = first
movl (%ecx), %ecx    #  ecx = second

movl $1, %edi       #  edi = 1 first flag
movl $2, %esi       #  esi = 2 second flag

cmpl %edx, %ebx     #  check if first is ommitted
jnz firstnotommitted
xorl %edi, %edi     #  make first flag 0
xorl $35, %ebx      #  encrypt ebx - ebx ^ '#'
firstnotommitted:

cmpl %edx, %ecx     #  check if second is ommitted
jnz secondnotommitted
xorl %esi, %esi     #  make second flag 0
xorl $35, %ecx      #  encrypt ecx - ecx ^ '#'
secondnotommitted:

orl %esi, %edi      #  check if both of them are encrypted
jz endofecryption    #  both are encrypted - end

cmpl $3, %edi       #  check if one of them is encrypted
jnz swapvalues       #  one of them is envrypted - swap values for final encryption

cmpl $97, %ebx      #  check if the first is not letter
js swapvalues
cmpl $122, %ebx
jg swapvalues

cmpl $97, %ecx      #  check if the second is not letter
```

```
js swapvalues
cmpl $122, %ecx
jg swapvalues

movl 16(%ebp), %edx  #  address to table

movl %edx, -40(%ebp) #  pass the table
movl %ebx, -36(%ebp) #  pass the first
movl %ecx, -4(%ebp)  #  save the second
movl %ebx, -20(%ebp) #  save the first

call _FindPosition

movl %esi, -24(%ebp) #  save position of the first
movl %eax, -28(%ebp) #  save row of the first
movl %ecx, -32(%ebp) #  save collumn of the first

movl -4(%ebp), %ecx
movl %ecx, -36(%ebp) #  pass the second
call _FindPosition

movl %esi, -8(%ebp)  #  save position of the second
movl %eax, -12(%ebp) #  save row of the second
movl %ecx, -16(%ebp) #  save collumn of the second

movl -28(%ebp), %ebx #  ebx = row of the first
movl -32(%ebp), %ecx #  ecx = collumn of the first

movl -12(%ebp), %edx #  edx = row of the second
movl -16(%ebp), %eax #  eax = collumn of the second

cmpl %ebx, %edx
jz samerow
cmpl %ecx, %eax
jz samecollumn
                #  update the first
decl %edx          #  --edx  dec row of the second
imull $5, %edx      #  5 x edx
addl %ecx, %edx
decl %edx          #  position of new value for the first

                #  update the second
decl %ebx
imull $5, %ebx
addl %eax, %ebx
decl %ebx          #   position of the new value for the second

movl 16(%ebp), %esi  #  pointer to the table
movl (%esi,%edx,4), %edx # new value for the first
movl (%esi,%ebx,4), %ebx # new value for the second

movl 8(%ebp), %eax   #  pointer to the first
movl %edx, (%eax)    #  update the first

movl 12(%ebp), %eax   #  pointer to the second
movl %ebx, (%eax)    #  update the second

jmp end
```

```
samerow:
cmpl %ecx, %eax
jz end # if they are the same letters
js firstIsRight
                # second is right
movl -24(%ebp), %ebx # ebx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
movl 16(%ebp), %esi  # esi = pointer to the table

movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first

cmpl $5, %eax
jnz notInfivecol
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
subl $5, %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second
jmp end
notInfivecol:
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second
jmp end

firstIsRight:       #  first is right
movl 16(%ebp), %esi  # esi = pointer to the table
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second

cmpl $5, %ecx
jnz notInfivecol2
movl -24(%ebp), %ebx  # ebx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
subl $5, %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first
jmp end

notInfivecol2:
movl -24(%ebp), %ebx # ebx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first
jmp end

samecollumn:
cmpl %ebx, %edx
js firstbelow
                #  second is below
movl -24(%ebp), %ecx # ecx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
movl 16(%ebp), %esi  # esi = pointer to the table
```

```
addl $4, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first

cmpl $5, %edx
jnz notinfifthrow
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
subl $21, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second
jmp end

notinfifthrow:
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
addl $4, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second
jmp end

firstbelow:
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
movl 16(%ebp), %esi  # esi = pointer to the table
addl $4, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second

cmpl $5, %ebx
jnz notinfifthrow2
movl -24(%ebp), %ecx  # ecx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
subl $21, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first
jmp end

notinfifthrow2:
movl -24(%ebp), %ecx  # ecx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
addl $4, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first
jmp end


swapvalues:
movl 8(%ebp), %edi  #   pointer to the first
movl 12(%ebp), %esi #   pointer to the second

movl %ebx, (%esi)   #   update the second value
movl %ecx, (%edi)   #   update the first value
jmp end

endofecryption:
movl 8(%ebp), %edi  #   pointer to the first
movl 12(%ebp), %esi #   pointer to the second
```

```
movl %ebx, (%edi)   #   update the first value
movl %ecx, (%esi)   #   update the second value
jmp end

end:
movl %ebp, %esp
pop %ebp
ret
```

4 - Decryption

```
.global _Dencryption
_Dencryption:
#ejfoej
pushl %ebp
movl %esp, %ebp
subl $40, %esp  # four local variable

movl 8(%ebp), %ebx   #  ebx = first  pointer
movl 12(%ebp), %ecx  #  ecx = second pointer
movl 20(%ebp), %edx  #  edx = ommittedletter

xorl $35, %edx

movl (%ebx), %ebx    #  ebx = first
movl (%ecx), %ecx    #  ecx = second

movl $1, %edi       #  edi = 1 first flag
movl $2, %esi       #  esi = 2 second flag

cmpl %edx, %ebx     #  check if first is ommitted
jnz firstnotommitted
xorl %edi, %edi     #  make first flag 0
xorl $35, %ebx      #  encrypt ebx - ebx ^ '#'
firstnotommitted:

cmpl %edx, %ecx     #  check if second is ommitted
jnz secondnotommitted
xorl %esi, %esi     #  make second flag 0
xorl $35, %ecx      #  encrypt ecx - ecx ^ '#'
secondnotommitted:

orl %esi, %edi      #  check if both of them are encrypted
jz endofecryption    #  both are encrypted - end

cmpl $3, %edi       #  check if one of them is encrypted
jnz swapvalues      #  one of them is envrypted - swap values for final encryption

cmpl $97, %ebx      #  check if the first is not letter
js swapvalues
cmpl $122, %ebx
jg swapvalues

cmpl $97, %ecx      #  check if the second is not letter
js swapvalues
cmpl $122, %ecx
jg swapvalues
```

```
movl 16(%ebp), %edx  #  address to table

movl %edx, -40(%ebp) #  pass the table
movl %ebx, -36(%ebp) #  pass the first
movl %ecx, -4(%ebp)  #  save the second
movl %ebx, -20(%ebp) #  save the first

call _FindPosition

movl %esi, -24(%ebp) #  save position of the first
movl %eax, -28(%ebp) #  save row of the first
movl %ecx, -32(%ebp) #  save collumn of the first

movl -4(%ebp), %ecx
movl %ecx, -36(%ebp) #  pass the second
call _FindPosition

movl %esi, -8(%ebp)  #  save position of the second
movl %eax, -12(%ebp) #  save row of the second
movl %ecx, -16(%ebp) #  save collumn of the second

movl -28(%ebp), %ebx #  ebx = row of the first
movl -32(%ebp), %ecx #  ecx = collumn of the first

movl -12(%ebp), %edx #  edx = row of the second
movl -16(%ebp), %eax #  eax = collumn of the second

cmpl %ebx, %edx
jz samerow
cmpl %ecx, %eax
jz samecollumn
                #  update the first
decl %edx          #  --edx  dec row of the second
imull $5, %edx      #  5 x edx
addl %ecx, %edx
decl %edx          #  position of new value for the first

                #  update the second
decl %ebx
imull $5, %ebx
addl %eax, %ebx
decl %ebx          #   position of the new value for the second

movl 16(%ebp), %esi  #  pointer to the table
movl (%esi,%edx,4), %edx # new value for the first
movl (%esi,%ebx,4), %ebx # new value for the second

movl 8(%ebp), %eax   #  pointer to the first
movl %edx, (%eax)    #  update the first

movl 12(%ebp), %eax   #  pointer to the second
movl %ebx, (%eax)    #  update the second

jmp end

samerow:
cmpl %ecx, %eax
jz end # if they are the same letters
```

```
jns firstIsleft
                # first is left
movl -24(%ebp), %ebx # ebx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
movl 16(%ebp), %esi  # esi = pointer to the table

decl %ebx
decl %ebx


movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first

cmpl $1, %eax
jnz notInfirstcol
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
addl $3, %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second
jmp end
notInfirstcol:
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
decl %ebx
decl %ebx

movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second
jmp end

firstIsleft:       #  first is left
movl 16(%ebp), %esi  # esi = pointer to the table
movl -8(%ebp), %ebx  # ebx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
decl %ebx
decl %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the second

cmpl $1, %ecx
jnz notInfirstcol2
movl -24(%ebp), %ebx  # ebx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
addl $3, %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first
jmp end

notInfirstcol2:
movl -24(%ebp), %ebx # ebx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
decl %ebx
decl %ebx
movl (%esi,%ebx,4), %ebx
movl %ebx, (%edi)    # update the first
jmp end
```

```
samecollumn:
cmpl %ebx, %edx
jns firstabove
                #  second is above
movl -24(%ebp), %ecx # ecx = position first
movl 8(%ebp), %edi   # edi = pointer to the first
movl 16(%ebp), %esi  # esi = pointer to the table
subl $6, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first

cmpl $1, %edx
jnz notinfirstrow
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
addl $19, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second
jmp end

notinfirstrow:
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
subl $6, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second
jmp end

firstabove:
movl -8(%ebp), %ecx  # ecx = position second
movl 12(%ebp), %edi  # edi = pointer to the second
movl 16(%ebp), %esi  # esi = pointer to the table
subl $6, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the second

cmpl $1, %ebx
jnz notinfirsthrow2
movl -24(%ebp), %ecx  # ecx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
addl $19, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first
jmp end

notinfirsthrow2:
movl -24(%ebp), %ecx  # ecx = position first
movl 8(%ebp), %edi  # edi = pointer to the first
subl $6, %ecx
movl (%esi,%ecx,4), %ecx
movl %ecx, (%edi)    # update the first
jmp end


swapvalues:
movl 8(%ebp), %edi  #   pointer to the first
movl 12(%ebp), %esi #   pointer to the second
```

```
movl %ebx, (%esi)   #   update the second value
movl %ecx, (%edi)   #   update the first value
jmp end

endofecryption:
movl 8(%ebp), %edi  #   pointer to the first
movl 12(%ebp), %esi #   pointer to the second

movl %ebx, (%edi)   #   update the first value
movl %ecx, (%esi)   #   update the second value
jmp end

end:
movl %ebp, %esp
pop %ebp
ret
```

4*. Specific tool set used

In order to run this program, Code::Blocks with GNU GCC compiler is used

5. User instructions:

When the user runs the program, the program will present him/her with three option:
1 – encryption
2 – decryption
3 – quit
Enter your choice:

If the user wants to encrypt the text, he/she has to choose 1. Then he/she has to enter keyword and text. The program will generate encrypted text.

If the user wants to decrypt the text, he/she has to choose 2. Then he/she has to enter keyword and text. The program will generate decrypted text.

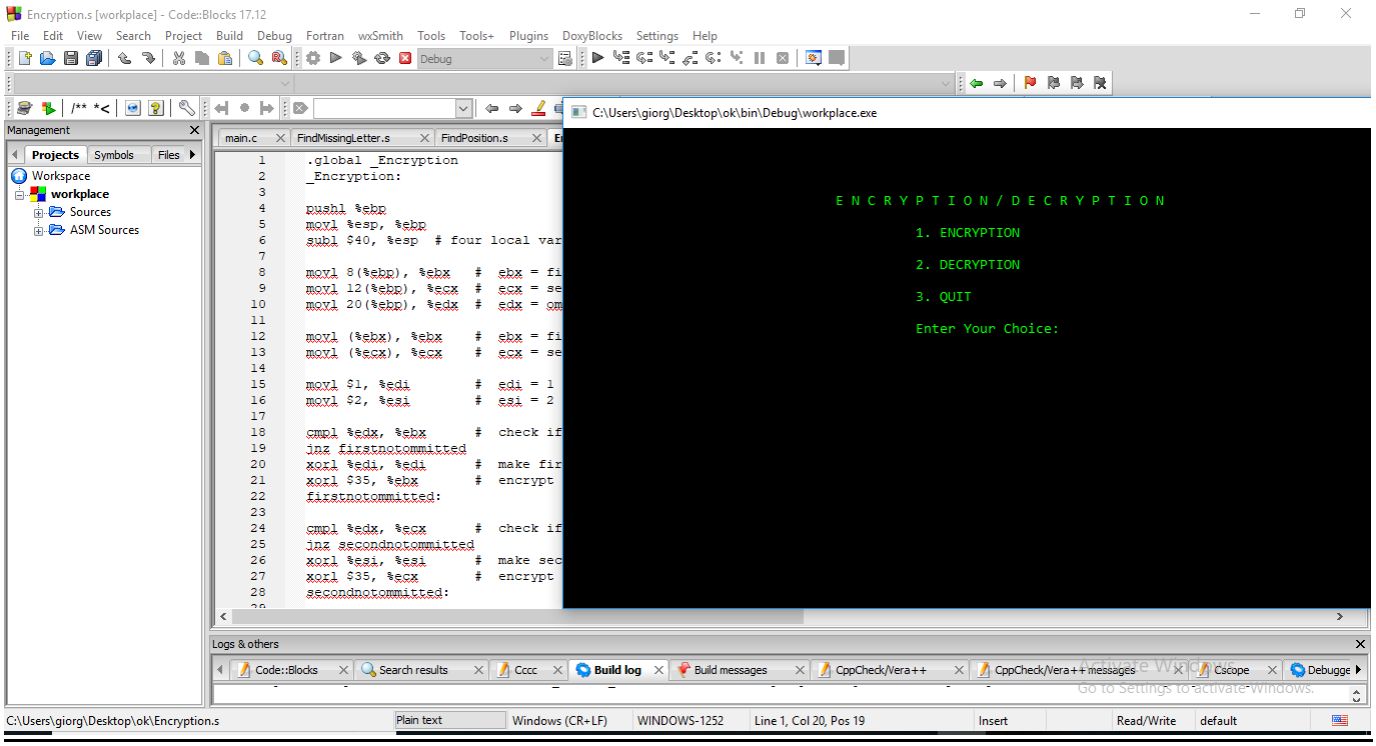
6*. Test results and any lab notes, bugs, project limitations

There are no bugs as far as the user might consider the proper execution of the program.
The program limitation is that, it cannot encrypt or decrypt text with more than 1000 characters.

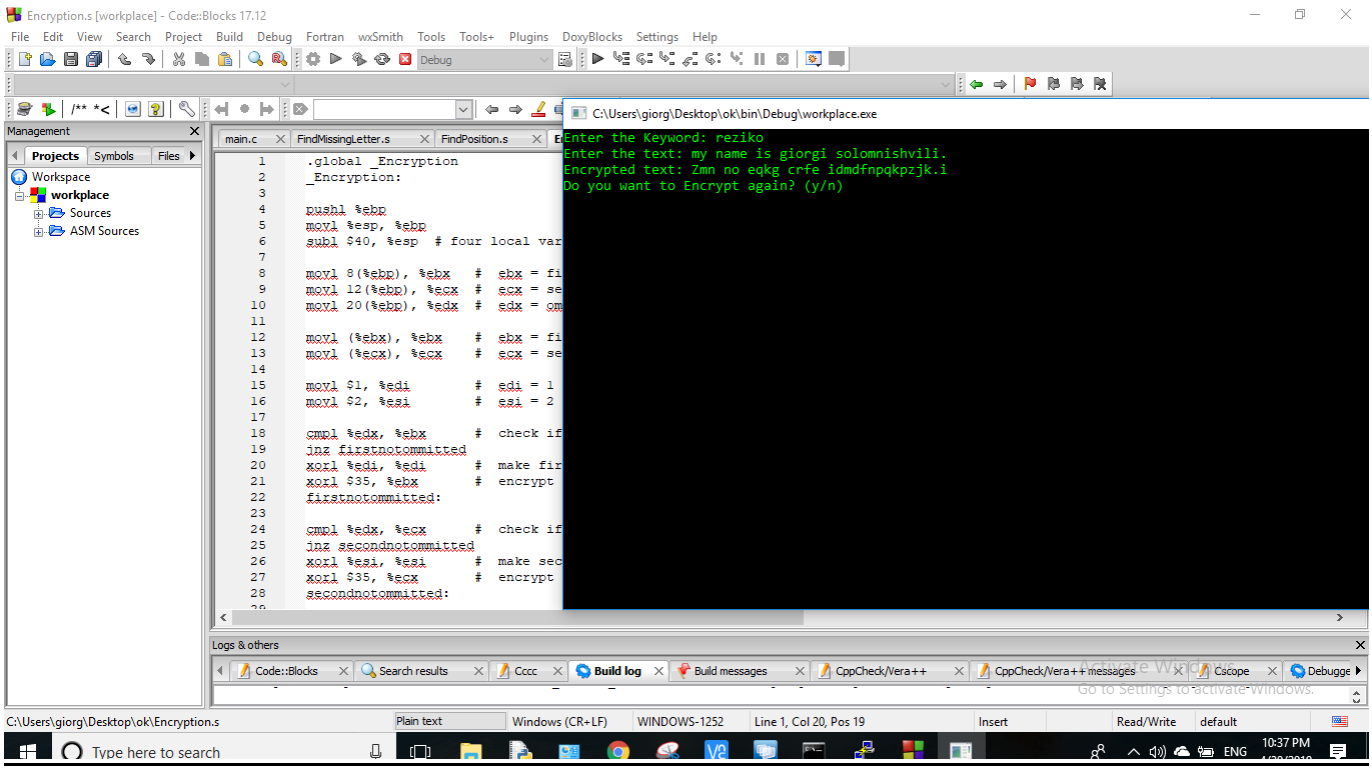
7*. Special hardware

There is no special hardware necessary to run the program. The user just needs a PC with windows and code::blocks.

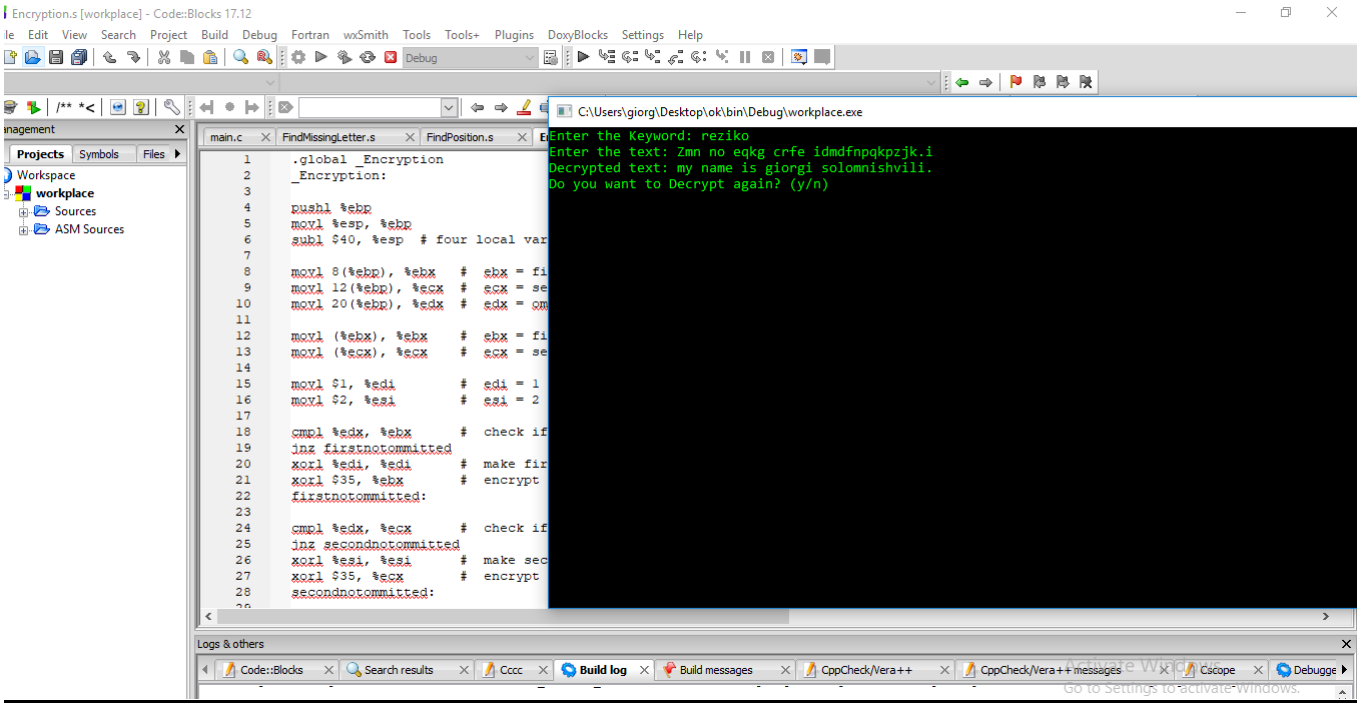
8*. Screen capture of demonstration


Encryption.s [workplace] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
workplace
Sources
ASM Sources
main.c
FindMissingLetter.s
FindPosition.s
.global _Encryption
_Encryption:
pushl %ebp
movl %esp, %ebp
subl $40, %esp # four local var
movl 8(%ebp), %ebx # ebx = fi
movl 12(%ebp), %ecx # ecx = se
movl 20(%ebp), %edx # edx = om
movl (%ebx), %ebx # ebx = fi
movl (%ecx), %ecx # ecx = se
movl $1, %edi # edi = 1
movl $2, %esi # esi = 2
cmpl %edx, %ebx # check if
jnz firstnotommitted
xorl %edi, %edi # make fir
xorl $35, %ebx # encrypt
firstnotommitted:
cmpl %edx, %ecx # check if
jnz secondnotommitted
xorl %esi, %esi # make sec
xorl $35, %ecx # encrypt
secondnotommitted:
C:\Users\giorg\Desktop\ok\bin\Debug\workplace.exe
ENCRYPTION/DECRYPTION
1. ENCRYPTION
2. DECRYPTION
3. QUIT
Enter Your Choice:
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugge
Activate Windows
Go to Settings to activate Windows.
C:\Users\giorg\Desktop\ok\Encryption.s
Plain text
Windows (CR+LF)
WINDOWS-1252
Line 1, Col 20, Pos 19
Insert
Read/Write
default

Picture 1



Picture 2



Picture 3

9*. What you would do differently if you had time.

If I had more time, I would add another option. I would have the program generate keyword on its own. The user would not have to enter the keyword if he/she did not want to.

10* Conclusion

This program is a mixed C and Assembly project. It utilizes 4 assembly subroutines, and 3 major C functions. Most of part (the core part) of the program is written in assembly.

11* Hours spent on the project

Approximately, 25 hours are spent on this project.

12 hours are spent on brainstorming – I had to decide what I was going to do and how I was going to do it.

8 hours are spent on writing Encryption and Decryption subroutines.

2 spent on writing findposition and findmissing letter subroutines

3 hours are spent on writing C functions