NAME: QUINUEL TABOT NDIP-AGBOR

MATRICULE NUMBER: FE21A300

COURSE: SECURITY AND CRYPTOSYSTEMS (CEF 350)

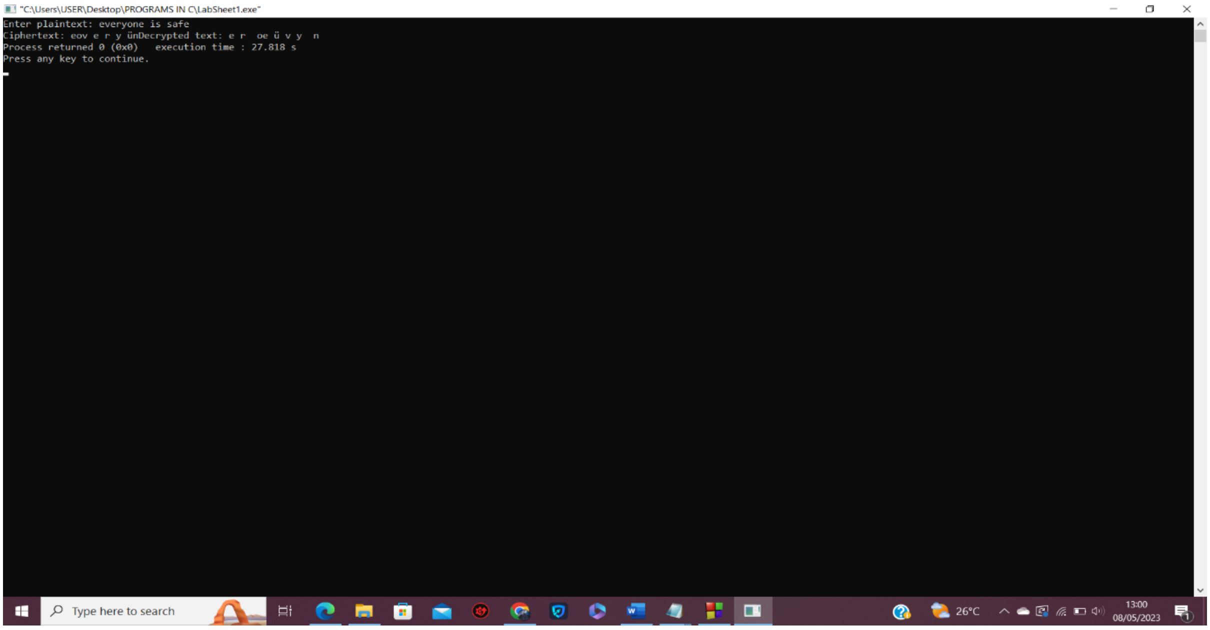
Lab Sheet 1.

Exercise 1 – Implementation of the Columnar Transposition cipher.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LEN 1000 // Maximum length of input string

// Function to perform columnar transposition encryption
void columnar_encrypt(char* plaintext, char* ciphertext, int key) {
 int len = strlen(plaintext);
 int rows = (len + key - 1) / key; // Number of rows in the transposition table

 // Allocate memory for the transposition table
 char** table = (char**) malloc(rows * sizeof(char*));
 for (int i = 0; i < rows; i++) {
  table[i] = (char*) malloc(key * sizeof(char));
 }

 // Fill the transposition table with the plaintext
 int index = 0;
 for (int i = 0; i < rows; i++) {
  for (int j = 0; j < key; j++) {
   if (index < len) {
    table[i][j] = plaintext[index++];
   } else {
    table[i][j] = ' ';
   }
```

```
    }
  }

  // Read the ciphertext from the transposition table column by column
  index = 0;
  for (int j = 0; j < key; j++) {
    for (int i = 0; i < rows; i++) {
      ciphertext[index++] = table[i][j];
    }
  }

  // Free the memory allocated for the transposition table
  for (int i = 0; i < rows; i++) {
    free(table[i]);
  }
  free(table);
}

// Function to perform columnar transposition decryption
void columnar_decrypt(char* ciphertext, char* plaintext, int key) {
  int len = strlen(ciphertext);
  int rows = (len + key - 1) / key; // Number of rows in the transposition table

  // Allocate memory for the transposition table
  char** table = (char**) malloc(rows * sizeof(char*));
  for (int i = 0; i < rows; i++) {
    table[i] = (char*) malloc(key * sizeof(char));
  }

  // Fill the transposition table with the ciphertext
  int index = 0;
  for (int j = 0; j < key; j++) {
    for (int i = 0; i < rows; i++) {
      if (index < len) {
        table[i][j] = ciphertext[index++];
      } else {
        table[i][j] = ' ';
      }
    }
  }

  // Read the plaintext from the transposition table row by row
  index = 0;
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < key; j++) {
      plaintext[index++] = table[i][j];
    }
  }

  // Free the memory allocated for the transposition table
  for (int i = 0; i < rows; i++) {
    free(table[i]);
  }
  free(table);
}

int main() {
  char plaintext[MAX_LEN];
  char ciphertext[MAX_LEN];
  int key = 5; // Change this to the desired key length

  // Read the plaintext from the user
  printf("Enter plaintext: ");
  fgets(plaintext, MAX_LEN, stdin);
  plaintext[strcspn(plaintext, "n")] = 0; // Remove trailing newline character

  // Encrypt the plaintext using columnar transposition
  columnar_encrypt(plaintext, ciphertext, key);
  printf("Ciphertext: %sn", ciphertext);

  // Decrypt the ciphertext using columnar transposition
  char decrypted_text[MAX_LEN];
  columnar_decrypt(ciphertext, decrypted_text, key);
  printf("Decrypted text: %sn", decrypted_text);

  return 0;
}
```

Code Implementation Test For Encrytion :



"C:\Users\USER\Desktop\PROGRAMS IN C\LabSheet1.exe"

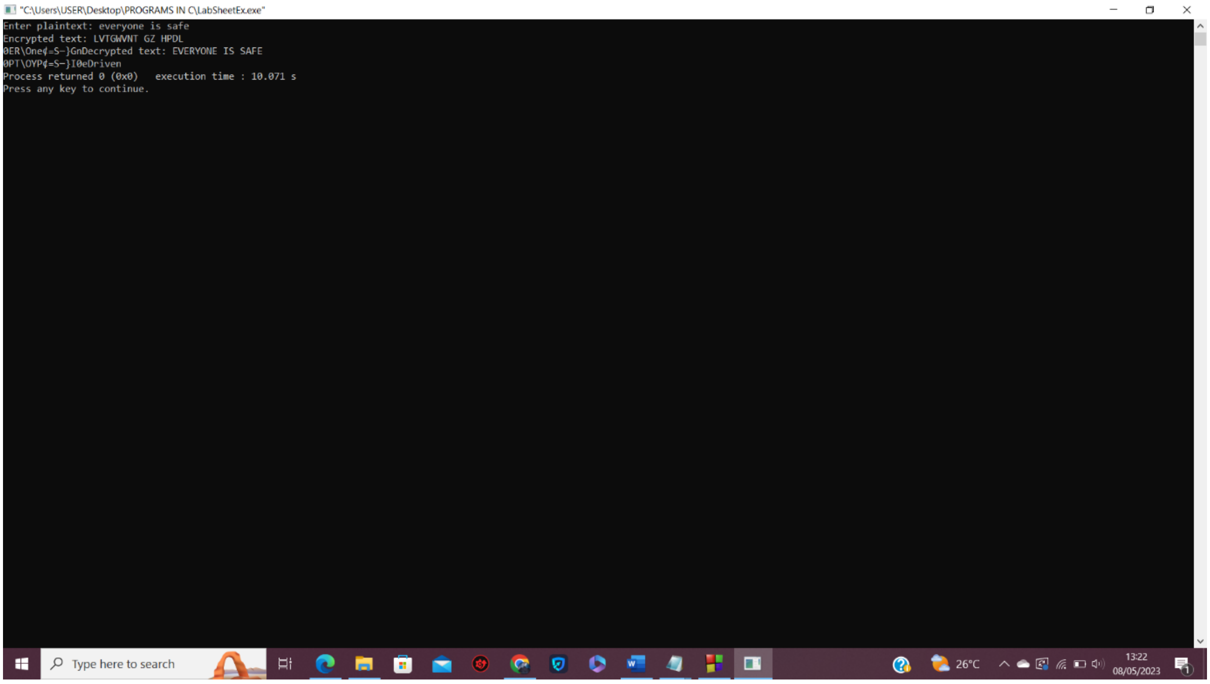
```
Enter plaintext: everyone is safe
Ciphertext: eov e r y ünDecrypted text: e r  oe ü v y  n
Process returned 0 (0x0)   execution time : 27.818 s
Press any key to continue.
```

Exercise 2 - Implementation of the Vigenere cipher with key K

```
#include <stdio.h>
```

```
#include <string.h>
#include <ctype.h>

char* vigenere_encrypt(char *plaintext, char *key) {
  int i, j, k = 0;
  char *ciphertext = malloc(strlen(plaintext) + 1);
  for (i = 0, j = 0; i < strlen(plaintext); i++, j = (j + 1) % 5) {
    if (isalpha(plaintext[i])) {
      ciphertext[i] = ((toupper(plaintext[i]) - 'A') + (toupper(key[j]) - 'A')) % 26 + 'A';
    } else {
      ciphertext[i] = plaintext[i];
      k++;
    }
  }
  ciphertext[i] = '0';
  return ciphertext;
}

char* vigenere_decrypt(char *ciphertext, char *key) {
  int i, j, k = 0;
  char *plaintext = malloc(strlen(ciphertext) + 1);
  for (i = 0, j = 0; i < strlen(ciphertext); i++, j = (j + 1) % 5) {
    if (isalpha(ciphertext[i])) {
      plaintext[i] = ((toupper(ciphertext[i]) - 'A') - (toupper(key[j]) - 'A') + 26) % 26 + 'A';
    } else {
      plaintext[i] = ciphertext[i];
      k++;
    }
  }
  plaintext[i] = '0';
  return plaintext;
}
```

Code Implementation For Encryption With key “happy”:



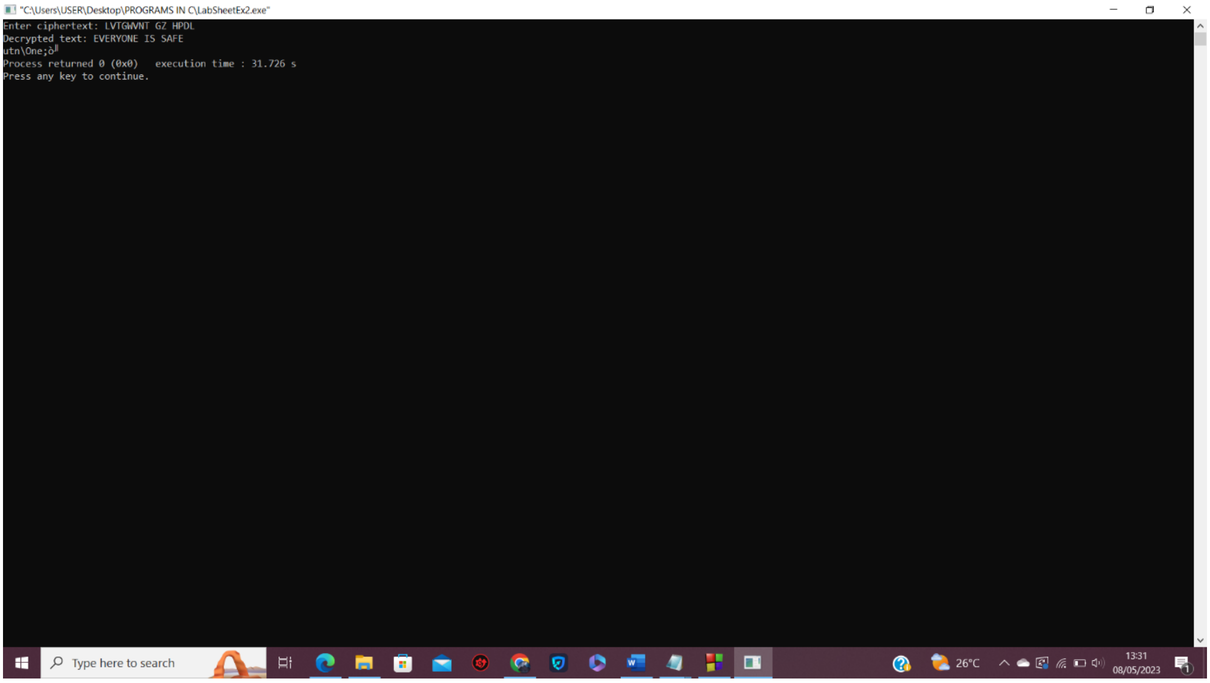
```
Enter plaintext: everyone is safe
Encrypted text: LVTGWVNT GZ HPDL
0ER\One¢=S-}GnDecrypted text: EVERYONE IS SAFE
0PT\OYP¢=S-}I0eDriven
Process returned 0 (0x0)   execution time : 10.071 s
Press any key to continue.
```

Code For Decryption:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>


char* vigenere_decrypt(char *ciphertext, char *key);


int main() {

   char ciphertext[100], plaintext[100];

   char key[] = "happy";


   printf("Enter ciphertext: ");

   fgets(ciphertext, sizeof(ciphertext), stdin);


   // Decrypt ciphertext
```

```
    strcpy(plaintext, vigenere_decrypt(ciphertext, key));


    printf("Decrypted text: %sn", plaintext);


    return 0;
}


char* vigenere_decrypt(char *ciphertext, char *key) {
    int i, j, k = 0;
    char *plaintext = malloc(strlen(ciphertext) + 1);
    for (i = 0, j = 0; i < strlen(ciphertext); i++, j = (j + 1) % 5) {
        if (isalpha(ciphertext[i])) {
            plaintext[i] = ((toupper(ciphertext[i]) - 'A') - (toupper(key[j]) - 'A') + 26) % 26 + 'A';
        } else {
            plaintext[i] = ciphertext[i];
            k++;
        }
    }
    plaintext[i] = '0';
    return plaintext;
}
```

Code Implementation For Decryption With key “happy”:



```
Enter ciphertext: LVTGWVNT GZ HPDL
Decrypted text: EVERYONE IS SAFE
utn\One;ò╜
Process returned 0 (0x0)   execution time : 31.726 s
Press any key to continue.
```