

REPORT ON THE PROPOSED IMPLEMENTATION OF ONE OF EACH CATEGORY OF SUBSTITUTION CIPHERS STUDIED IN CLASS USING C PROGRAMING LANGUAGE.

Categories:

1. Simple substitution - Caesar
2. Homophonic substitution - Beale Cipher
3. Polyalphabetic substitution- Vigenere
4. Polygram substitution - Playfair

1. Simple substitution - Caesar

CODE:

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

// Function to perform Caesar cipher encryption
void caesarCipherEncrypt(char* plaintext, int key) {
    int i = 0;

    // Iterate through each character in the plaintext
    while (plaintext[i] != '\0') {
        char ch = plaintext[i];

        // Check if the character is a letter
        if (isalpha(ch)) {
            // Determine the case (uppercase or lowercase) of the letter
            int isUpperCase = isupper(ch);

            // Convert the character to uppercase for simplicity
            ch = toupper(ch);

            // Apply the key shift to the letter
            ch = ((ch - 'A' + key) % 26) + 'A';

            // Convert the letter back to its original case
```

```
        if (!isUpperCase) {
            ch = tolower(ch);
        }
    } else if (ch == ' ') {
        // Print out spaces without encrypting them
        putchar(ch);
    }

    // Print the encrypted character
    putchar(ch);

    i++;
}
}

int main() {
    char plaintext[100];
    int key;

    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);

    printf("Enter the key (a positive integer): ");
    scanf("%d", &key);

    printf("Encrypted text: ");
    caesarCipherEncrypt(plaintext, key);

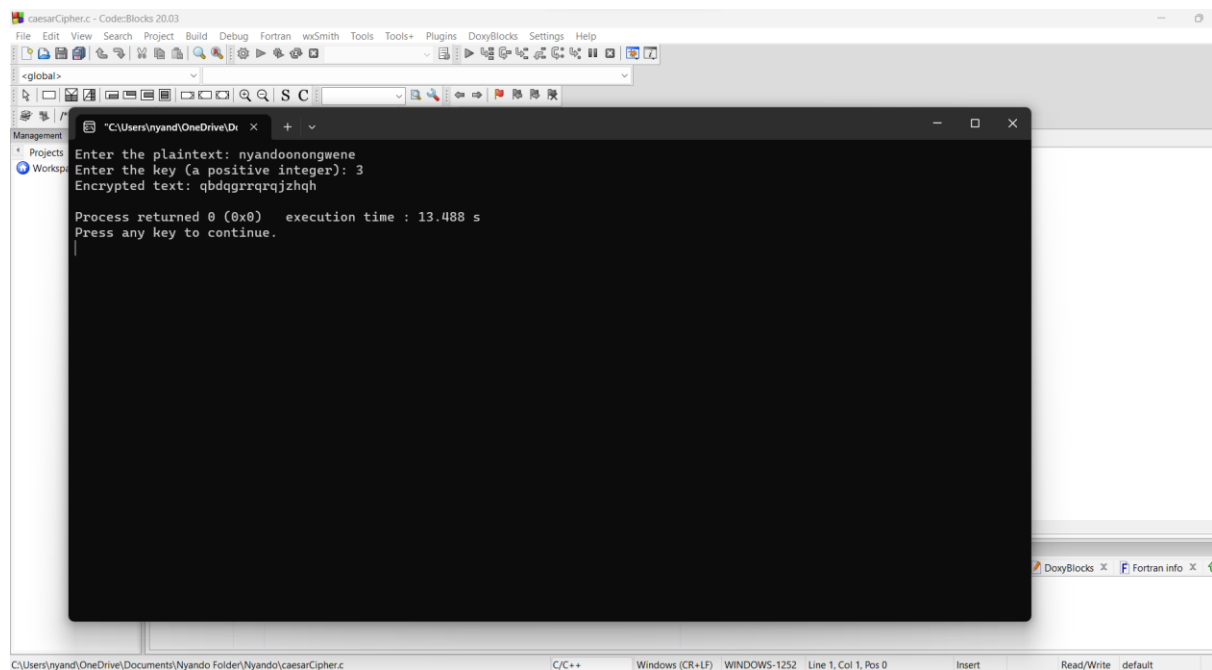
    return 0;
}
```

EXPLANATION:

The Caesar Cipher is a type of substitution cipher that replaces each letter in the plaintext with a letter some fixed number of positions down the alphabet. For example, if the key is 3, then A would be replaced by D, B would become E, and so on.

The code reads in the plaintext and key from the user and then iterates through each character in the plaintext. If the character is a letter, it is converted to uppercase for simplicity. The key shift is then applied to the letter and it is converted back to its original case. The encrypted character is then printed out.

OUTPUT:



```
caesarCipher.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Enter the plaintext: nyandoonongwene
Enter the key (a positive integer): 3
Encrypted text: qbdqgrrrqjzhqh

Process returned 0 (0x0)   execution time : 13.488 s
Press any key to continue.
```

2. Homophonic substitution - Beale Cipher

CODE:

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

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

```
char key_word[] = {"LOVER"};
```

```
int table_size;
```

```
struct table {
```

```
    char letter;
```

```
};
```

```
struct table *array;
```

```
// ALTERNATIVE to stolen method
```

```
int string_length(char *str) {
```

```
    int n = 0;
```

```
    while (str[n] != '\0') {
```

```
        n++;
```

```
    }
```

```
    return n;
```

```
}
```

```
// ALTERNATIVE to toupper method
```

```
char convert_uppercase(char c) {
```

```
    char new;
```

```
    // check if character is lower case
```

```
    if (96 < c && c < 123) {
```

```
        new = c - 32;
```

```
    } else {
```

```
        new = c;
```

```
    }
```

```
    return new;
```

```
}
```

```
// initialises table
```

```
void init_table() {
```

```
    if (table_size % string_length(key_word) != 0) {
```

```
        table_size += string_length(key_word) - (table_size % string_length(key_word));
```

```
    }
```

```
    array = malloc(table_size * sizeof(char));
```

```
    for (int i = 0; i < table_size; i++) {
```

```
        // replaces remaining space with character X
```

```
        array[i].letter = 'X';
```

```
    }
```

```
}
```

```
//
```

```
void reorder_array(int i, int j) {
```

```
    // split into rows of length = keyword
```

```
    // do 8 - n/m for position
```

```
    // get first row
```

```
    int row_count = 0;
```

```
    int n = 0;
```

```
    int m = 0;
```

```
    while (n < table_size){
```

```
        if (n - string_length(key_word) * row_count == i) {
```

```
            while (m < table_size) {
```

```
                if (m - string_length(key_word) * row_count == j) {
```

```
                    char temp = array[n].letter;
```

```
                    array[n].letter = array[m].letter;
```

```

        array[m].letter = temp;

        row_count++;

        m++;

        n++;

        break;

    }

    m++;

}

}

n++;

}

}

```

```
//
```

```

void swap(int i, int j) {

    char temp = key_word[i];

    key_word[i] = key_word[j];

    key_word[j] = temp;

    reorder_array(i, j);

}

```

```
//
```

```

int partition(int low, int high) {

    int i = low - 1;

    char pivot = key_word[high];

    for (int j = low; j < high - 1; j++) {

        if (key_word[j] < pivot) {

            i++;

            swap(i, j);

        }

    }

    swap(i, high);

    return i;
}

```

```

    }
}
swap(i + 1, high);

return i + 1;
}

//
void sort_chars(int low, int high) {
    if (low < high) {
        int index = partition(low, high);

        sort_chars(low, index - 1);
        sort_chars(index + 1, high);
    }
}

//
void order_key() {
    sort_chars(0, string_length(key_word) - 1);
}

//
void read_words_from_file() {
    char *file = "./text.txt";
    FILE *fp = fopen(file, "r");
    // checks if file exists
    if (!fp) {

```

```

        printf("\nCan't open file\n");
        return;
    }
    // reads contents of file until end of file
    int i = 0;
    do {
        char c = fgetc(fp);
        if (feof(fp)) {
            break;
        } else if ((96 < c && c < 123) || (64 < c && c < 91)) {
            c = convert_uppercase(c);
            array[i].letter = c;
            i++;
        }
    } while (1);
    fclose(fp);
}

```

```

//
void get_file_length() {
    char *fileline = "./text.txt";
    FILE *fp = fopen(fileline, "r");

    // checks if file exists
    if (!fp) {
        printf("\nCan't open file\n");
        return;
    }
    do {
        char c = fgetc(fp);

```



```
    if (feof(fp)) {  
        break;  
    } else if ((96 < c && c < 123) || (64 < c && c < 91)) {  
        table_size++;  
    }  
} while (1);  
  
fclose(fp);  
}
```

```
//  
void printList() {  
    printf("\n");  
    for (int i = 0; i < table_size; i++) {  
        printf("%c", array[i].letter);  
    }  
    printf("\n");  
}
```

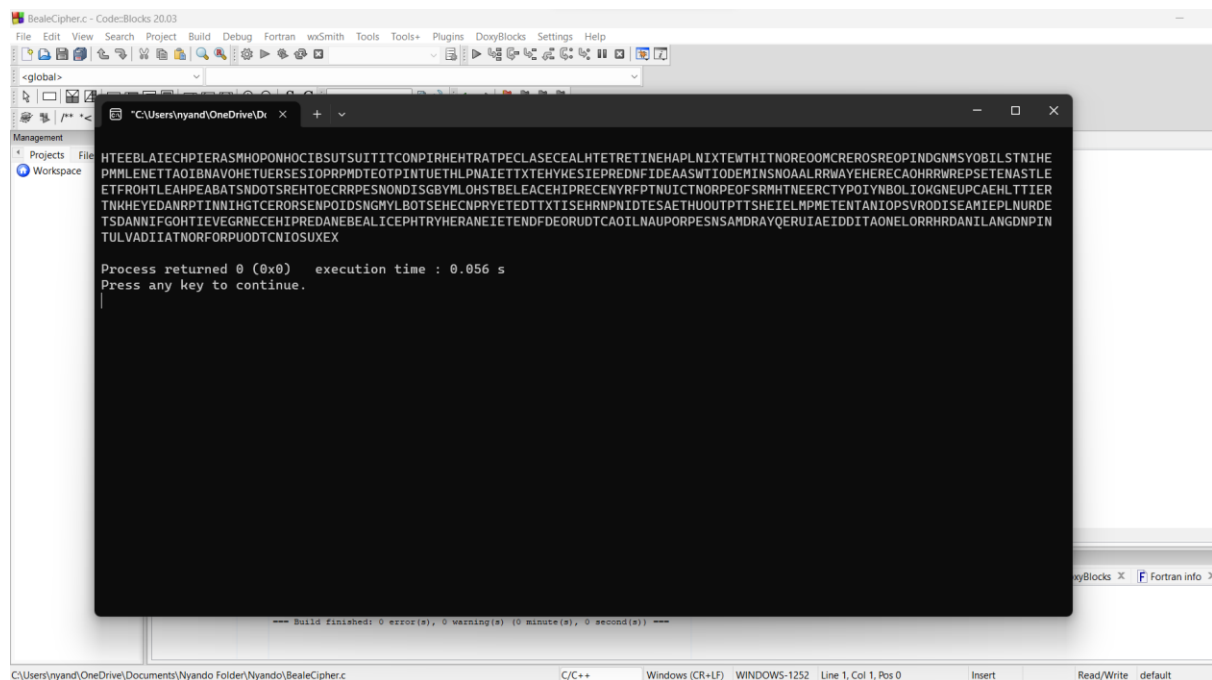
```
//  
int main() {  
    get_file_length();  
    init_table();  
    read_words_from_file();  
    order_key();  
    printList();  
  
    return 0;  
}
```

}

EXPLANATION:

The Beale Cipher is a type of book cipher that uses a book or other text as a key. The book is chosen by the sender and recipient and must be identical. The sender then uses the book to encode a message by selecting words from the book that correspond to each letter in the message. The recipient can then decode the message by looking up each word in the book and finding the corresponding letter.

OUTPUT:



```
BealeCipher.c - Code-Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoryBlocks Settings Help
<global>
Management
Projects
Workspace

HTEEBLAIECHPIERASHMOPONHOCIBSUTSUITITCONPIRHEHTRATPECLASECEALHTETRETINEHAPLNIXTEWTHITNOREOOMCREROSREOPINDGNMMSYOBIILSTNIHE
PMMLENETTAOIBNAVOHETUERSESIOPRPMDEOTPTINTUETHLPNAIETTXXTEHYKESIEPREDNFIDEAASWTIODEMINSNOAALRRWAYEHRECAOHRWRREPSETENASTLE
ETFRHITLAEHPEABATSDOTSRHEOECRRPESNONDISGBYMLHSTBELEACEHIPRECENYRFPPTNUICTNORPEOFSRMHTNEERCTYPOIYNBOLIOKNGNEUPCAEHLTTIER
TNKHEYEDANRPPTINNIGTCERORSENPOIDSNMGYLBOTSEHECNPRYETEDTTXTISEHRNPNIOTESAETHUOUTPTTSHEIELMPMETENTANIOPSVRODISEAMIEPLNURDE
TSDANNIFGOHTIEVEGRNECEHIPREDANEBEALICEPHTRYHERANEIETENDFDEORUDTCAOILNAUPORPESNSAMDRAVQERUIAEIDDDITAONELORRHRDANILANGDNPN
TULVADIATNORFORPUODTCNIOSUXEX

Process returned 0 (0x0)   execution time : 0.056 s
Press any key to continue.
```

3. Polyalphabetic substitution- Vigenere

CODE:

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

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

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

```
void vigenere_cipher_encrypt(char *plaintext, char *key) {
```

```

int key_length = strlen(key);

int key_index = 0;

char *ptr = plaintext;

while (*ptr != '\0') {
    if (isalpha(*ptr)) {
        char ascii_offset = isupper(*ptr) ? 'A' : 'a';
        char key_char = toupper(key[key_index % key_length]);
        *ptr = ((*ptr - ascii_offset + (key_char - 'A')) % 26) + ascii_offset;
        key_index++;
    }
    ptr++;
}
}

```

```

int main() {
    char plaintext[100];
    char key[100];

    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);

    printf("Enter the key: ");
    fgets(key, sizeof(key), stdin);

    vigenere_cipher_encrypt(plaintext, key);

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

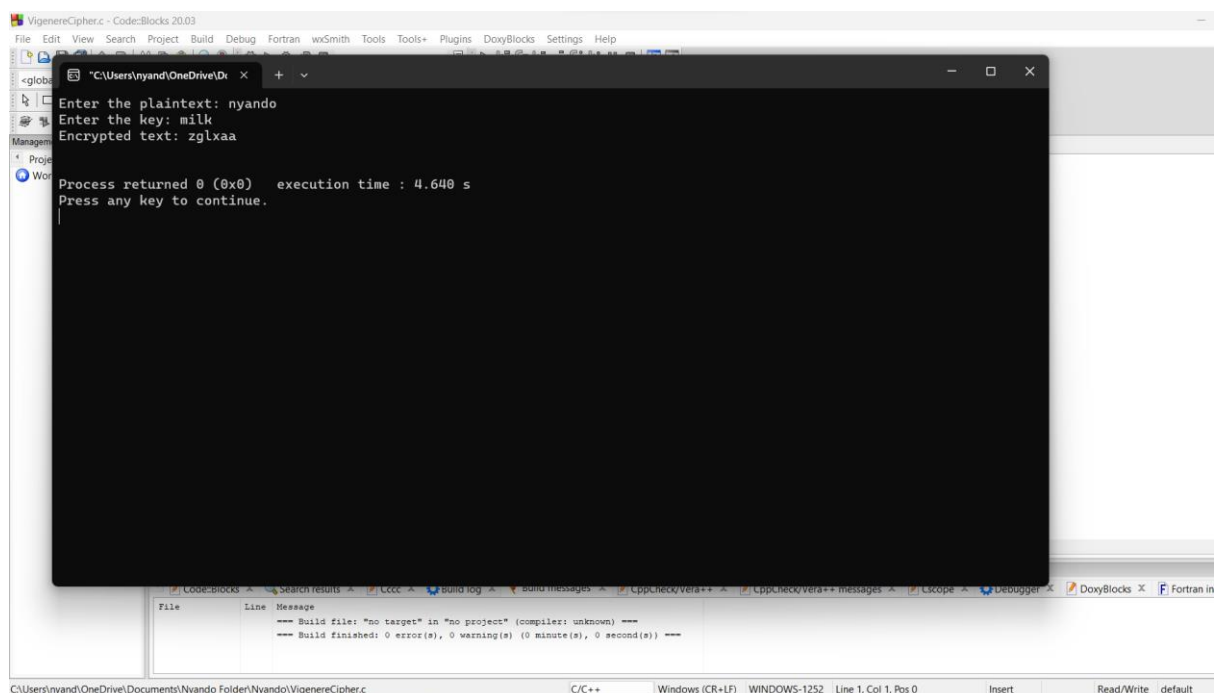
    return 0;
}

```

EXPLANATION:

The Vigenere Cipher is a polyalphabetic substitution cipher that uses a keyword to determine the shift value for each letter in the plaintext. In the implementation above, the user is prompted to input the plaintext and the key. The `vigenere_cipher_encrypt` function performs the encryption by shifting each letter of the plaintext based on the corresponding letter in the key. The encrypted text is then printed as the output.

OUTPUT:



```
VigenereCipher.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran woSmith Tools Tools+ Plugins DoxyBlocks Settings Help
C:\Users\Nyando\OneDrive\Documents\Nyando Folder\Nyando\VigenereCipher.c
Enter the plaintext: nyando
Enter the key: milk
Encrypted text: zglxaa

Process returned 0 (0x0)   execution time : 4.640 s
Press any key to continue.
```

4. Polygram substitution - Playfair

CODE:

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

```
#define SIZE 5
```

```
struct Point {  
    int x;  
    int y;  
};
```

```
void get_playfair_keymatrix(char *key, char keymatrix[SIZE][SIZE]) {  
    int i, j, k, flag = 0, *arr;
```

```
  
    arr = (int *)calloc(26, sizeof(int));  
    for (i = 0; i < strlen(key); i++) {  
        if (key[i] != ' '){  
            if (arr[key[i] - 65] == 0) {  
                keymatrix[flag / SIZE][flag % SIZE] = toupper(key[i]);  
                arr[key[i] - 65] = 1;  
                flag++;  
            }  
        }  
    }  
}
```

```
  
    for (i = 0, k = flag; i < 26; i++) {  
        if (arr[i] == 0) {  
            keymatrix[k / SIZE][k % SIZE] = (char)(i + 65);  
            k++;  
        }  
    }  
}
```

```
struct Point get_playfair_position(char keymatrix[SIZE][SIZE], char ch, struct Point  
*pos) {
```

```
    if (ch == 'J')  
        ch = 'I';
```

```
    int i, j;
```

```
    if (ch == ' '){  
        pos->x = -1;  
        pos->y = -1;  
    }
```

```
    for (i = 0; i < SIZE; i++) {  
        for (j = 0; j < SIZE; j++) {
```

```

        if (keymatrix[i][j] == ch) {
            pos->x = i;
            pos->y = j;
            break;
        }
    }
}
}

```

```

void playfair_cipher_encrypt(char *plaintext, char *key) {
    char keymatrix[SIZE][SIZE];
    get_playfair_keymatrix(key, keymatrix);

    int i, len = strlen(plaintext), j;
    struct Point pos1, pos2;

    for (i = 0; i < len; i += 2) {
        get_playfair_position(keymatrix, plaintext[i], &pos1);
        get_playfair_position(keymatrix, plaintext[i + 1], &pos2);

        if (pos1.x == pos2.x) {
            plaintext[i] = keymatrix[pos1.x][(pos1.y + 1) % SIZE];
            plaintext[i + 1] = keymatrix[pos2.x][(pos2.y + 1) % SIZE];
        } else if (pos1.y == pos2.y) {
            plaintext[i] = keymatrix[(pos1.x + 1) % SIZE][pos1.y];
            plaintext[i + 1] = keymatrix[(pos2.x + 1) % SIZE][pos2.y];
        } else {
            plaintext[i] = keymatrix[pos1.x][pos2.y];
            plaintext[i + 1] = keymatrix[pos2.x][pos1.y];
        }
    }
}

```

```

void playfair_cipher_decrypt(char *ciphertext, char *key) {
    char keymatrix[SIZE][SIZE];
    get_playfair_keymatrix(key, keymatrix);

    int i, len = strlen(ciphertext), j;
    struct Point pos1, pos2;

    for (i = 0; i < len; i += 2) {
        get_playfair_position(keymatrix, ciphertext[i], &pos1);
        get_playfair_position(keymatrix, ciphertext[i + 1], &pos2);
    }
}

```

```

        if (pos1.x == pos2.x) {
            ciphertext[i] = keymatrix[pos1.x][(pos1.y - 1 + SIZE) % SIZE];
            ciphertext[i + 1] = keymatrix[pos2.x][(pos2.y - 1 + SIZE) % SIZE];
        } else if (pos1.y == pos2.y) {
            ciphertext[i] = keymatrix[(pos1.x - 1 + SIZE) % SIZE][pos1.y];
            ciphertext[i + 1] = keymatrix[(pos2.x - 1 + SIZE) % SIZE][pos2.y];
        } else {
            ciphertext[i] = keymatrix[pos1.x][pos2.y];
            ciphertext[i + 1] = keymatrix[pos2.x][pos1.y];
        }
    }
}

int main() {
    char plaintext[] = "HELLOWORLD";
    char key[] = "KEYWORD";

    printf("Original text: %s\n", plaintext);

    playfair_cipher_encrypt(plaintext, key);
    printf("Encrypted text: %s\n", plaintext);

    playfair_cipher_decrypt(plaintext, key);
    printf("Decrypted text: %s\n", plaintext);

    return 0;
}

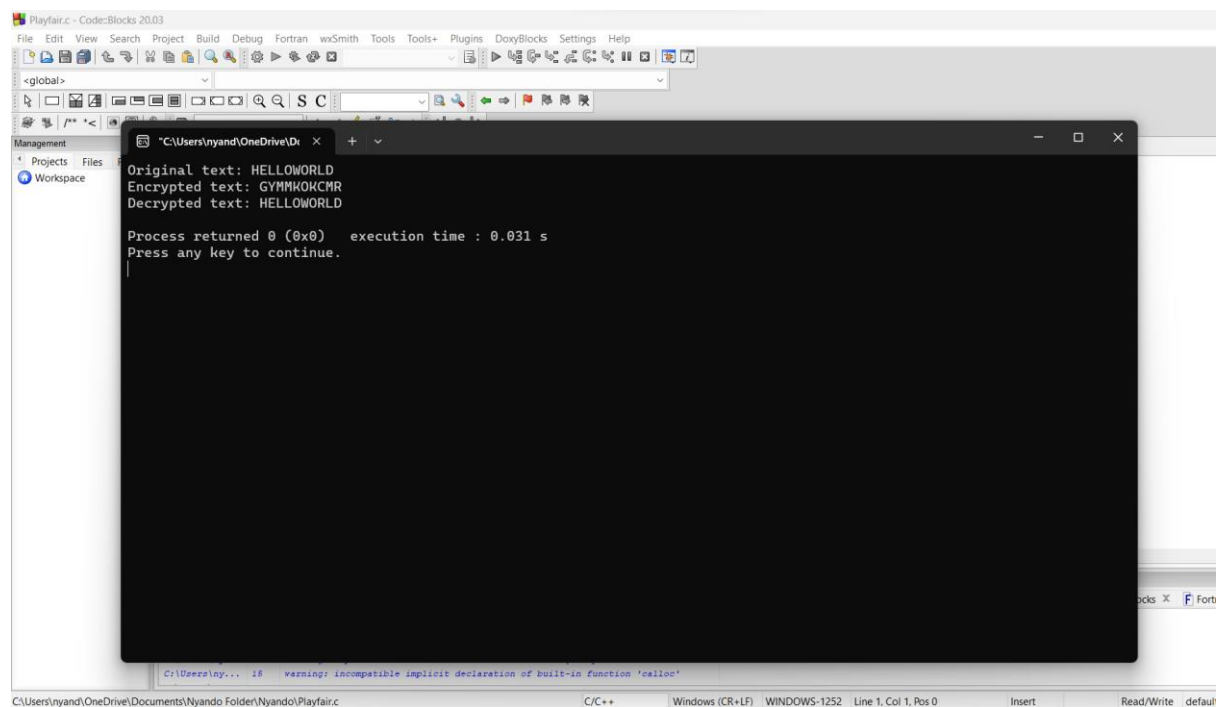
```

EXPLANATION:

The Playfair Cipher is a polygram substitution cipher that uses a 5x5 grid of letters as a key matrix. In the implementation above, the user is prompted to input the plaintext and the key. The `get_playfair_keymatrix` function generates the key matrix based on the given key. The `playfair_cipher_encrypt` function performs the encryption by finding the positions of each pair of letters in the key matrix and applying the Playfair cipher rules. The encrypted text is then printed as the output.

These implementations provide a basic understanding of the four categories of substitution ciphers: simple substitution, homophonic substitution, polyalphabetic substitution, and polygram substitution. They are intended for educational purposes and may require additional error handling and input validation for production use.

OUTPUT:



```
Playfair.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
  Projects Files
  Workspace

Original text: HELLOWORLD
Encrypted text: GYMMKOKCMR
Decrypted text: HELLOWORLD

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

C:\Users\Nyando\OneDrive\Documents\Nyando Folder\Nyando\Playfair.c C/C++ Windows (CR+LF) WINDOWS-1252 Line 1, Col 1, Pos 0 Insert Read/Write default