```
#include "funcoes.h"

void chave(char C[16][28], char D[16][28], char CD[15][56], char K[15][48],
    FILE * arq, char key_hex[16]) {

    int i, len;
    char key_bin[64], key_PC1[56];
    FILE * entrada;
    //************************************************************ Key in Binary
        form************************************

    printf("\n\n\n  __CHAVE__\n");
    fprintf(arq, "\n\n\n    __CHAVE__\n");

    //Orientações para o usuário
    printf("\nChave em Hexadecimal: ");
    fprintf(arq, "\nChave em Hexadecimal: ");
    for (i = 0; i < 16; i++) {
        printf("%c",key_hex[i]);
        fprintf(arq, "%c",key_hex[i]);
    }

    //Copia a chave para a forma binária
    hex_to_bin (key_hex,key_bin);

    printf("\n\nChave em Binário    : ");
    fprintf(arq, "\n\nChave em Binário  : ");
    for (i = 0; i < 64; i++) {
            printf("%c",key_bin[i]);
            fprintf(arq, "%c",key_bin[i]);
            if ( (i + 1) % 8 == 0 ) {
                printf(" ");
                fprintf(arq, " ");
            }
    }

    printf("\n");
    fprintf(arq, "\n");




    //************************************************************ First Round of
        Permutation ***********************************

    //Realiza a Permutação Inicial da Chave
    permutation (key_bin,key_PC1);

    //imprimindo na tela
    printf("\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DA_CHAVE__");
    fprintf(arq, "\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DA_CHAVE__");


    printf("\n\nPrimeira Permutação da Chave: ");
    fprintf(arq, "\n\nPrimeira Permutação da Chave: ");
    for (i = 0; i < 56; i++) {
            printf("%c",key_PC1[i]);
            fprintf(arq, "%c",key_PC1[i]);
            if ( (i + 1) % 7 == 0 ) {
```

```
                printf(" ");
                fprintf(arq, " ");
        }
}
```

```
printf("\n\n\n\n\n\t__ROTAÇÕES_DAS_CHAVES__");
fprintf(arq, "\n\n\n\n\n\t__ROTAÇÕES_DAS_CHAVES__");
//********************************************************** Shifting Begins
    ***************************************************************************
    ***********
//Divide a chave permutada em duas

make_half (key_PC1,C[0],D[0]);

printCD(C[0], D[0], '0', arq);


//Deslocamento rotacional simples da Direita para a Esquerda
single_shift (C[0],C[1]);
single_shift (D[0],D[1]);

printCD(C[1], D[1], '1', arq);

//Deslocamento rotacional simples da Direita para a Esquerda
single_shift (C[1],C[2]);
single_shift (D[1],D[2]);


printCD(C[2], D[2], '2', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[2],C[3]);
double_shift (D[2],D[3]);

printCD(C[3], D[3], '3', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[3],C[4]);
double_shift (D[3],D[4]);

printCD(C[4], D[4], '4', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[4],C[5]);
double_shift (D[4],D[5]);


printCD(C[5], D[5], '5', arq);
```

```
//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[5],C[6]);
double_shift (D[5],D[6]);


printCD(C[6], D[6], '6', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[6],C[7]);
double_shift (D[6],D[7]);


printCD(C[7], D[7], '7', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[7],C[8]);
double_shift (D[7],D[8]);


printCD(C[8], D[8], '8', arq);

//Deslocamento rotacional Simples da Direita para a Esquerda
single_shift (C[8],C[9]);
single_shift (D[8],D[9]);


printCD(C[9], D[9], '9', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[9],C[10]);
double_shift (D[9],D[10]);


printCD(C[10], D[10], 'A', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[10],C[11]);
double_shift (D[10],D[11]);


printCD(C[11], D[11], 'B', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[11],C[12]);
double_shift (D[11],D[12]);


printCD(C[12], D[12], 'C', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[12],C[13]);
double_shift (D[12],D[13]);
```

```
printCD(C[13], D[13], 'D', arq);


//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[13],C[14]);
double_shift (D[13],D[14]);


printCD(C[14], D[14], 'E', arq);

//Deslocamento rotacional Duplo da Direita para a Esquerda
double_shift (C[14],C[15]);
double_shift (D[14],D[15]);


printCD(C[15], D[15], 'F', arq);


//Deslocamento rotacional Simples da Direita para a Esquerda
single_shift (C[15],C[16]);
single_shift (D[15],D[16]);


printCD(C[16], D[16], 'G', arq);


//********************************** 16 Keys Generation Begins
    **********************************


        printf("\n\n\n\n\n\t__PERMUTAÇÕES_FINAIS_DAS_CHAVES__");
        fprintf(arq, "\n\n\n\n\n\t__PERMUTAÇÕES_FINAIS_DAS_CHAVES__");

make_key (C[1],D[1],CD[0]);
permutation_48 (CD[0],K[0]);


printCDK (CD[0], K[0], '1', arq);

make_key (C[2],D[2],CD[1]);
permutation_48 (CD[1],K[1]);


printCDK (CD[1], K[1], '2', arq);

make_key (C[3],D[3],CD[2]);
permutation_48 (CD[2],K[2]);

printCDK (CD[2], K[2], '3', arq);

make_key (C[4],D[4],CD[3]);
permutation_48 (CD[3],K[3]);

printCDK (CD[3], K[3], '4', arq);

make_key (C[5],D[5],CD[4]);
permutation_48 (CD[4],K[4]);

printCDK (CD[4], K[4], '5', arq);
```

```
    make_key (C[6],D[6],CD[5]);
    permutation_48 (CD[5],K[5]);

    printCDK (CD[5], K[5], '6', arq);

    make_key (C[7],D[7],CD[6]);
    permutation_48 (CD[6],K[6]);

    printCDK (CD[6], K[6], '7', arq);

    make_key (C[8],D[8],CD[7]);
    permutation_48 (CD[7],K[7]);

    printCDK (CD[7], K[7], '8', arq);

    make_key (C[9],D[9],CD[8]);
    permutation_48 (CD[8],K[8]);

    printCDK (CD[8], K[8], '9', arq);

    make_key (C[10],D[10],CD[9]);
    permutation_48 (CD[9],K[9]);

    printCDK (CD[9], K[9], 'A', arq);

    make_key (C[11],D[11],CD[10]);
    permutation_48 (CD[10],K[10]);

    printCDK (CD[10], K[10], 'B', arq);

    make_key (C[12],D[12],CD[11]);
    permutation_48 (CD[11],K[11]);

    printCDK (CD[11], K[11], 'C', arq);

    make_key (C[13],D[13],CD[12]);
    permutation_48 (CD[12],K[12]);

    printCDK (CD[12], K[12], 'D', arq);

    make_key (C[14],D[14],CD[13]);
    permutation_48 (CD[13],K[13]);

    printCDK (CD[13], K[13], 'E', arq);

    make_key (C[15],D[15],CD[14]);
    permutation_48 (CD[14],K[14]);

    printCDK (CD[14], K[14], 'F', arq);

    make_key (C[16],D[16],CD[15]);
    permutation_48 (CD[15],K[15]);

    printCDK (CD[15], K[15], 'G', arq);

}

void DES (char entrada[][16], int indice, char cryptoOut[][16], FILE * arq,
    char K[15][48]) {
```

```
int i,j,len,m = 0;

char input_hex[16],input_bin[64];
char key_PC1[56];
char ch,*decryption,*encryption;
char encrypted[64],decrypted[64],encry_permut[64],decry_permut[64];
int length,p = -1,q = -1;


char L0[32],R0[32],ER0[48];
char L1[32],R1[32],ER1[48],F1[48],
L2[32],R2[32],ER2[48],F2[48],
L3[32],R3[32],ER3[48],F3[48],
L4[32],R4[32],ER4[48],F4[48],
L5[32],R5[32],ER5[48],F5[48],
L6[32],R6[32],ER6[48],F6[48],
L7[32],R7[32],ER7[48],F7[48],
L8[32],R8[32],ER8[48],F8[48],
L9[32],R9[32],ER9[48],F9[48],
L10[32],R10[32],ER10[48],F10[48],
L11[32],R11[32],ER11[48],F11[48],
L12[32],R12[32],ER12[48],F12[48],
L13[32],R13[32],ER13[48],F13[48],
L14[32],R14[32],ER14[48],F14[48],
L15[32],R15[32],ER15[48],F15[48],
L16[32],R16[32],ER16[48],F16[48];




//************************************************************ Key in Binary
    form************************************
printf("\n\n\n
    \n-------------------------------------------------------------------------
    ----------\n\t__INÍCIO__\n\n");
fprintf(arq, "\n\n\n
    \n-------------------------------------------------------------------------
    ----------\n\t__INÍCIO__\n\n");


//************************************************************ Plain Text in
    Hex ************************************
//Será copiado o primeiro bloco para ser trabalhado.
        //A variável input_hex será responsável por receber todas as
            alterações no decorrer do programa
for (i = 0; i < 16; i++)
    input_hex[i] = entrada[indice][i];

//Orientações ao usuário
printf("Bloco de texto selecionado: %d - ", indice);
fprintf(arq, "Bloco de texto selecionado: %d - ", indice);
for (i = 0; i < 16; i++) {
    printf("%c",input_hex[i]);
    fprintf(arq, "%c",input_hex[i]);
}
```

```
    printf("\n");
    fprintf(arq, "\n");



    //************************************************************ Plain Text in
        Binary **********************************
    //Copia o texto para um formato binário
    hex_to_bin (input_hex,input_bin);

    //imprimindo na tela
    printf("\nBloco de texto em Binário : ");
    fprintf(arq, "\nBloco de texto em Binário : ");
    for (i = 0; i < 64; i++) {
            printf("%c",input_bin[i]);
            fprintf(arq, "%c",input_bin[i]);
            if ( (i + 1) % 8 == 0 ) {
                printf(" ");
                fprintf(arq, " ");
            }
    }


    //****************************** Initial Permutation of Plain Text
        ***********************************

    printf("\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DO_TEXTO_PURO__");
    fprintf(arq, "\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DO_TEXTO_PURO__");

    //Realiza a Permutação inicial para a as 16 iterações quer ocorreção do do
        texto
    permutation_64 (input_bin,L0,R0);


    printf("\n\nResultado da permutação inicial:");
    fprintf(arq, "\n\nResultado da Permutação inicial:");


    printLiRi(L0, R0, '0', arq);


    //****************************** 16 Rounds of Encryption
        ***********************************


    printf ("\n\n\n\n\n\t__16_ITERAÇÕES_SOBRE_O_TEXTO_PURO__\n");
    fprintf (arq, "\n\n\n\n\n\t__16_ITERAÇÕES_SOBRE_O_TEXTO_PURO__\n");
    //Uma iteração
    printLiRi(L0, R0, '0', arq);

        printf("\nOK");

    des_round (arq, L1,R1,L0,R0,ER0,K[0],F1, '0');
printf("\nOK");

    printLiRi(L1, R1, '1', arq);
```

```
des_round (arq, L2,R2,L1,R1,ER1,K[1],F2, '1');
printLiRi(L2, R2, '2', arq);

des_round (arq, L3,R3,L2,R2,ER2,K[2],F3, '2');
printLiRi(L3, R3, '3', arq);

des_round (arq, L4,R4,L3,R3,ER3,K[3],F4, '3');
printLiRi(L4, R4, '4', arq);

des_round (arq, L5,R5,L4,R4,ER4,K[4],F5, '4');
printLiRi(L5, R5, '5', arq);

des_round (arq, L6,R6,L5,R5,ER5,K[5],F6, '5');
printLiRi(L6, R6, '6', arq);

des_round (arq, L7,R7,L6,R6,ER6,K[6],F7, '6');
printLiRi(L7, R7, '7', arq);

des_round (arq, L8,R8,L7,R7,ER7,K[7],F8, '7');
printLiRi(L8, R8, '8', arq);

des_round (arq, L9,R9,L8,R8,ER8,K[8],F9, '8');
printLiRi(L9, R9, '9', arq);

des_round (arq, L10,R10,L9,R9,ER9,K[9],F10, '9');
printLiRi(L10, R10, 'A', arq);

des_round (arq, L11,R11,L10,R10,ER10,K[10],F11, 'A');
printLiRi(L11, R11, 'B', arq);

des_round (arq, L12,R12,L11,R11,ER11,K[11],F12, 'B');
printLiRi(L12, R12, 'C', arq);
```

```
des_round (arq, L13,R13,L12,R12,ER12,K[12],F13, 'C');

printLiRi(L13, R13, 'D', arq);


des_round (arq, L14,R14,L13,R13,ER13,K[13],F14, 'D');

printLiRi(L14, R14, 'E', arq);


des_round (arq, L15,R15,L14,R14,ER14,K[14],F15, 'E') ;

printLiRi(L15, R15, 'F', arq);


des_round (arq, L16,R16,L15,R15,ER15,K[15],F16, 'F') ;

printLiRi(L16, R16, 'G', arq);


//************************************************************ Final permutation
    ***********************************

//Orientações ao usuário
printf ("\n\n\n\n\n\t__PERMUTAÇÃO_FINAL_DO_TEXTO__");
fprintf (arq, "\n\n\n\n\n\t__PERMUTAÇÃO_FINAL_DO_TEXTO__");

//Copia o resultado das criptografias (L16 e R16) para uma string final no
    qual será realizada mais uma permutação.
for (i = 0; i < 32; i++) {
    encrypted[i] = R16[i];
    encrypted[i+32] = L16[i];
}




//Realiza a Ãºltima permutação do texto puro.
common_permutation (encrypted,encry_permut);

printf ("\n\nPré-Permutação: ");
fprintf (arq, "\n\nPré-Permutação: ");
for (i = 0; i < 64; i++) {
    printf ("%c", encrypted[i]);
    fprintf (arq,"%c", encrypted[i]);
}

printf ("\n\nPós-Permutação: ");
fprintf (arq, "\n\nPós-Permutação: ");
for (i = 0; i < 64; i++) {
    printf ("%c", encry_permut[i]);
    fprintf (arq,"%c", encry_permut[i]);
}
```

```
    printf ("\n\n\n\n\n\t__RESULTADO_DA_ITERAÇÃO__");
    fprintf (arq, "\n\n\n\n\n\t__RESULTADO_DA_ITERAÇÃO__");

    printf("\n\nBloco de texto selecionado: %d - ", indice);
    fprintf(arq, "\n\nBloco de texto selecionado: %d - ", indice);
    for (i = 0; i < 16; i++) {
        printf("%c",input_hex[i]);
        fprintf(arq, "%c",input_hex[i]);
    }


    //Converte o texto a forma Hexadecimal para uma visualização curta e mais
        fácil
    encryption = bin_to_hex (encry_permut);


    printf ("\n\nTexto Hexadecimal Encriptado desta Iteração: \n");
    fprintf (arq, "\n\nTexto Hexadecimal Encriptado desta Iteração: \n");
    for (i = 0; i < 16; i++) {
        cryptoOut[indice][i] = * (encryption+i);
        printf ("%c",cryptoOut[indice][i]);
        fprintf (arq, "%c", cryptoOut[indice][i]);
    }

    cryptoOut[indice][i] = '\0';
}


void UnDES (char entrada[][16], int indice, char cryptoOut[][16], FILE * arq,
    char K[15][48]){


    int i,j,len,m = 0;

    char input_hex[16],input_bin[64];
    char key_PC1[56];
    char ch,*decryption,*encryption,encryption_final[4000],
        decryption_final_hex[4000],decryption_final_plain[2000];
    char encrypted[64],decrypted[64],encry_permut[64],decry_permut[64];
    int length,p = -1,q = -1;

    char L0[32],R0[32],ER0[48];
    char L1[32],R1[32],ER1[48],F1[48],
    L2[32],R2[32],ER2[48],F2[48],
    L3[32],R3[32],ER3[48],F3[48],
    L4[32],R4[32],ER4[48],F4[48],
    L5[32],R5[32],ER5[48],F5[48],
    L6[32],R6[32],ER6[48],F6[48],
    L7[32],R7[32],ER7[48],F7[48],
    L8[32],R8[32],ER8[48],F8[48],
    L9[32],R9[32],ER9[48],F9[48],
    L10[32],R10[32],ER10[48],F10[48],
    L11[32],R11[32],ER11[48],F11[48],
    L12[32],R12[32],ER12[48],F12[48],
    L13[32],R13[32],ER13[48],F13[48],
    L14[32],R14[32],ER14[48],F14[48],
    L15[32],R15[32],ER15[48],F15[48],
    L16[32],R16[32],ER16[48],F16[48];
```

```
//********************************************************** Key in Binary
    form************************************
printf("\n\n\n
    \n-------------------------------------------------------------------------
    -----------\n\t__INÍCIO__\n\n");
fprintf(arq, "\n\n\n
    \n-------------------------------------------------------------------------
    -----------\n\t__INÍCIO__\n\n");


//********************************************************** Plain Text in
    Hex ************************************
//Será copiado o primeiro bloco para ser trabalhado.
        //A variável input_hex será responsável por receber todas as
            alterações no decorrer do programa
for (i = 0; i < 16; i++)
    input_hex[i] = entrada[indice][i];

//Orientações ao usuário
printf("Bloco de texto selecionado: %d - ", indice);
fprintf(arq, "Bloco de texto selecionado: %d - ", indice);
for (i = 0; i < 16; i++) {
    printf("%c",input_hex[i]);
    fprintf(arq, "%c",input_hex[i]);
}

printf("\n");
fprintf(arq, "\n");




//********************************************************** Plain Text in
    Binary ************************************
//Copia o texto para um formato binário
hex_to_bin (input_hex,input_bin);

//imprimindo na tela
printf("\nBloco de texto em Binário : ");
fprintf(arq, "\nBloco de texto em Binário : ");
for (i = 0; i < 64; i++) {
    printf("%c",input_bin[i]);
    fprintf(arq, "%c",input_bin[i]);
    if ( (i + 1) % 8 == 0 ) {
        printf(" ");
        fprintf(arq, " ");
    }
}


//****************************** Initial Permutation of Plain Text
    ************************************

printf("\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DO_TEXTO_CIFRADO__");
fprintf(arq, "\n\n\n\n\n\t__PERMUTAÇÃO_INICIAL_DO_TEXTO_CIFRADO__");

    //Realiza a Permutação inicial para a as 16 iterações quer ocorreção
        do do texto
permutation_64 (input_bin,L16,R16);
```

```
printf("\n\nResultado da permutação inicial:");
fprintf(arq, "\n\nResultado da Permutação inicial:");



printLiRi(L16, R16, 'G', arq);

printf ("\n\n\n\n\n\t__16_ITERAÇÕES_SOBRE_O_TEXTO_CIFRADO__\n");
fprintf (arq, "\n\n\n\n\n\t__16_ITERAÇÕES_SOBRE_O_TEXTO_CIFRADO__\n");

//Uma iteração
printLiRi(L16, R16, 'G', arq);


des_round_decry (arq, L16,R16,L15,R15,ER15,K[15],F16, 'F');

printLiRi(L15, R15, 'F', arq);



des_round_decry (arq, L15,R15,L14,R14,ER14,K[14],F15, 'E');

printLiRi(L14, R14, 'E', arq);



des_round_decry (arq, L14,R14,L13,R13,ER13,K[13],F14, 'D');

printLiRi(L13, R13, 'D', arq);



des_round_decry (arq, L13,R13,L12,R12,ER12,K[12],F13, 'C');

printLiRi(L12, R12, 'C', arq);



des_round_decry (arq, L12,R12,L11,R11,ER11,K[11],F12, 'B');

printLiRi(L11, R11, 'B', arq);



des_round_decry (arq, L11,R11,L10,R10,ER10,K[10],F11, 'A');

printLiRi(L10, R10, 'A', arq);



des_round_decry (arq, L10,R10,L9,R9,ER9,K[9],F10, '9');

printLiRi(L9, R9, '9', arq);



des_round_decry (arq, L9,R9,L8,R8,ER8,K[8],F9, '8');
```

```
printLiRi(L8, R8, '8', arq);


des_round_decry (arq, L8,R8,L7,R7,ER7,K[7],F8, '7');

printLiRi(L7, R7, '7', arq);


des_round_decry (arq, L7,R7,L6,R6,ER6,K[6],F7, '6');

printLiRi(L6, R6, '6', arq);


des_round_decry (arq, L6,R6,L5,R5,ER5,K[5],F6, '5');

printLiRi(L5, R5, '5', arq);


des_round_decry (arq, L5,R5,L4,R4,ER4,K[4],F5, '4');

printLiRi(L4, R4, '4', arq);


des_round_decry (arq, L4,R4,L3,R3,ER3,K[3],F4, '3');

printLiRi(L3, R3, '3', arq);


des_round_decry (arq, L3,R3,L2,R2,ER2,K[2],F3, '2');

printLiRi(L2, R2, '2', arq);


des_round_decry (arq, L2,R2,L1,R1,ER1,K[1],F2, '1');

printLiRi(L1, R1, '1', arq);


des_round_decry (arq, L1,R1,L0,R0,ER0,K[0],F1, '0');

printLiRi(L0, R0, '0', arq);


for (i = 0; i < 32; i++) {
    decrypted[i] = R0[i];
    decrypted[i+32] = L0[i];
}
common_permutation (decrypted,decry_permut);
//decry_permut[64] = '\0';

decryption = bin_to_hex (decry_permut);
```

```
    // printf ("%s\n",decryption);

    printf ("\n\nTexto Hexadecimal Descriptografado desta Iteração: \n");
    fprintf (arq, "\n\nTexto Hexadecimal Descriptografado desta Iteração: \n")
        ;
    for (i = 0; i < 16; i++) {
        decryption_final_hex[++q] = * (decryption+i);
        printf ("%c",decryption_final_hex[q]);
        fprintf (arq, "%c", decryption_final_hex[q]);
    }

    decryption_final_hex[q+1] = '\0';

    for (i = 0; i < 16; i++) {
        cryptoOut[indice][i] = decryption_final_hex[i];
    }

}

long calcularTamanhoArquivo(FILE *arquivo) {

    // guarda o estado ante de chamar a função fseek
    long posicaoAtual = ftell(arquivo);

    // guarda tamanho do arquivo
    long tamanho;

    // calcula o tamanho
    fseek(arquivo, 0, SEEK_END);
    tamanho = ftell(arquivo);

    // recupera o estado antigo do arquivo
    fseek(arquivo, posicaoAtual, SEEK_SET);

    return tamanho;
}

void leArquivo(FILE * arq, unsigned char * input, long size) {

    long i = 0;

    while(i < size) {
        input[i] = (unsigned char) getc(arq);
        i++;
    }
    //adiciona o fim da string
    input[i] = '\0';
}


void leChave(char * nomeArquivo, char keys[2][16], FILE * arq) {

    FILE * entrada;


    if ((entrada = fopen(nomeArquivo, "rb")) == NULL) {
        printf("\n\nErro ao abrir arquivo de entrada da senha!!!\n");
        fprintf(arq, "\n\nErro ao abrir arquivo de entrada da senha!!!\n");
        exit(1);
    }
```

```
    //Lê do arquivo o texto a ser trabalhado


    int i = 0;
    int j = 0;
    char c;
    while(j < 3) {
        i = 0;
        while (!feof(entrada) && i < 16) {

            c = getc(entrada);

            //if (c == '\n') i = 16;

            keys[j][i] = c;
            i++;
        }

        fseek(entrada, 2, SEEK_CUR);

        j++;
    }

    fclose(entrada);

    //Torna todos os caracteres em caixa alta.

    for (j = 0; j < 3; j++)
        upper_char (keys[j]);


}

int main (int argc, char *argv[]) {

    printf("\n                      ALGORITMO DES DE CRIPTOGRAFIA");
    printf("\n                    ----------- --- -- --------------");


    /*//nomeDoPrograma acao chave texto saida
    if (argc != 5) {
        printf("\n\nEntrada de dados inválida!");
        printf("%d", argc);
        printf("\n\tNomeDoPrograma acao chave texto saida\n\n\n" );

        exit(1);
    }

    char acao = argv[1][0];
    char * textoChave = argv[2];
    char * textoPuro = argv[3];
    char * textoSaida = argv[4];*/


    char acao = 'e';

    FILE * orienta;

        char * textoChave;
```

```
    char * textoPuro;
    char * textoSaida;

if (acao == 'e' || acao == 'E') {
    textoChave = "senha.txt";
    textoPuro = "entrada.txt";
    textoSaida = "out.txt";
        if ((orienta = fopen("RelatorioDeExecucaoEncripta.txt", "wb")) ==
            NULL) {
        printf("Erro ao criar arquivo de orientações!!!\n");
        exit(1);
    }

} else{

    textoChave = "senha.txt";
    textoPuro = "out.txt";
    textoSaida = "out2.txt";


    if ((orienta = fopen("RelatorioDeExecucaoDesencripta.txt", "wb")) ==
        NULL) {
        printf("Erro ao criar arquivo de orientações!!!\n");
        exit(1);
    }
}

fprintf(orienta, "\n                          ALGORITMO DES DE CRIPTOGRAFIA"
    );
fprintf(orienta, "\n                        ---------- --- --
    --------------");




FILE * entrada;

if((entrada = fopen(textoPuro, "r+b")) == NULL) {
    printf("\n\nErro ao abrir arquivo de entrada!!!\n");
    fprintf(orienta, "\n\nErro ao abrir arquivo de entrada!!!\n");
    exit(1);
}



char key_PC1[56], C[16][28], D[17][28], CD[15][56], K[15][48]; FILE * arq;
int len = 0, i = 0, j = 0, k = 0, temp = 0, r = 0, x = 0;
int d, e ,f ;


char key_hex[3][16];
leChave(textoChave, key_hex, orienta);




//************************************************************ Input Plain Text
    ************************************************************
//Recebe o texto a ser trabalhado
```

```
if (acao == 'e' || acao == 'E') {
    printf("\n\n\n  __TEXTO_PLANO__\n");
    fprintf(orienta, "\n\n\n    __TEXTO_PLANO__\n");

} else
    if (acao == 'd' || acao == 'D') {
        printf("\n\n\n  TEXTO CIFRADO\n");
        fprintf(orienta, "\n\n\n    TEXTO CIFRADO\n");
    } else
        {
            printf("\n\nAção [%c] inválida!!!\n\tEscolha (E)ncriptar ou
                (D)escifrar.\n\n", acao);
            fprintf(orienta, "\n\nAção [%c] inválida!!!\n\tEscolha
                (E)ncriptar ou (D)escifrar.\n\n", acao);
            exit(1);
        }

//Lê do arquivo o texto a ser trabalhado


long tamArquivo = calcularTamanhoArquivo(entrada);

unsigned char input[tamArquivo + 1], initial_hex[tamArquivo * 2 + 1],
    output[tamArquivo * 2 + 1];

for (i = 0; i < tamArquivo + 2; i++) {
    input[i] = '\0';
    initial_hex[i] = '\0';
}


leArquivo(entrada, input, tamArquivo);


fclose(entrada);

len = strlen (input);
printf("\n\n%d", len);
printf("%s\n", input);


if (acao == 'e' || acao == 'E') {
    printf("\nTexto Plano: ");
    fprintf(orienta, "\nTexto Plano: ");
    int flagCasaDecimal;
    for (i = 0; i < len; i++) {
        flagCasaDecimal = 0;
        //imprime o Texto recebido pelo usuário
        printf("%c", input[i]);
        fprintf(orienta, "%c", input[i]);

        //Transforma todo o texto lido em uma sequênca em Hexadecial que
            represente o mesmo
        if (input[i] / 16 == 0) {
            flagCasaDecimal = 1;
        }
        while (input[i]!= 0 && flagCasaDecimal == 1) {
            //Descobre o valor em hexadecimal do caractere
            r = input[i]%16;
            input[i] = 0;
```

```
                if (r>9) {
                    x = r-10;
                    r = 65+x;
                    initial_hex[k] = r;
                }
                else
                    initial_hex[k] = r+48;

                initial_hex[++k] = '0';

                //Conta quantos caracteres foram convertidos - Variável K
                k++;
            }

            while (input[i]!= 0 && flagCasaDecimal == 0) {
                //Descobre o valor em hexadecimal do caractere
                r = input[i]%16;
                input[i] = input[i]/16;

                if (r>9) {
                    x = r-10;
                    r = 65+x;
                    initial_hex[k] = r;
                }
                else

                initial_hex[k] = r+48;

                //Conta quantos caracteres foram convertidos - Variável K
                k++;
            }
        }
        printf("\n\n");
        fprintf(orienta, "\n\n");

            //inverte a cada dois caracteres
        for (i = 0; i < k; i = i+2) {
            temp = initial_hex[i];
            initial_hex[i] = initial_hex[i+1];
            initial_hex[i+1] = temp;
        }


        //Orienta o usuário informando o Texto Plano em Hexadecimal
        printf("Texto Hexad: ");
        fprintf(orienta, "Texto Hexad: ");

        for (i = 0; i  <  k; i++) {
                printf("%c",initial_hex[i]);
                fprintf(orienta, "%c",initial_hex[i]);
        }

    } else if (acao == 'd' || acao == 'D') {
        printf("\nTexto Cifrado: ");
        fprintf(orienta, "\nTexto Cifrado: ");

        for (i = 0; i  <  len; i++) {
            initial_hex[i] = input[i];
            printf("%c",initial_hex[i]);
```

```
        fprintf(orienta, "%c",initial_hex[i]);
    }

    k = i;
}

printf("\n");
fprintf(orienta, "\n");


//Quantidade de caracteres por 16
d = k/16;
//Resto da quantidade de caracteres por 16
e = k%16;
f = 0;


char hex_arr[d][16];
char manipulaTexto1[d][16], manipulaTexto2[d][16], manipulaTexto3[d][16];


//O texto será trabalhado em pedaços (blocos) com 16 caracteres.
    //Nesse pequeno procedimento, o Texto plano será dividino nesses
        blocos para que a criptografia seja feita em pedaços do texto.
for (i = 0; i <= d; i++) {
    //Verifica quantos blocos ainda faltam
    if (i < d) {
        //Coloca a sequência de caracteres numa matriz.
        for (j = 0; j <= 15; j++)
            hex_arr[i][j] = initial_hex[f++];

    } else
        //Verifica se os blocos não possuem resto
        if (k%16 == 0)
                break;

        else {
            //Se possuirem restos
                //Percore todos os 16 caracteres de cada bloco
            for (j = 0; j <= 15; j++) {
                // Enquanto houver caracteres no qual o usuário digitou,
                    será adicionado a matriz.
                if (j < e)
                        hex_arr[i][j] = initial_hex[f++];
                else {
                    //Todos os outros espaços restantes do bloco que não
                        foi completado será adicionado 0 a matriz
                        representando vazio/nulo
                    hex_arr[i][j] = '0';
                    hex_arr[i][++j] = '0';
                }
            }
        }

}

if (k%16!= 0)
    d++;

//imprimindo na tela
```

```
        printf("\nBlocos Parciais Hexadecimais do Texto:\n");
        fprintf(orienta, "\nBlocos Parciais Hexadecimais do Texto:\n");
        for (i = 0; i < d; i++) {
            printf("\t%d - ", i);
            fprintf(orienta, "\t%d - ", i);

            for (j = 0; j <= 15; j++) {
                printf("%c",hex_arr[i][j]);
                fprintf(orienta, "%c",hex_arr[i][j]);
            }

            printf("\n");
            fprintf(orienta, "\n");
        }




        if (acao == 'e' || acao == 'E') {

            chave(C, D, CD, K, orienta, key_hex[0]);

            for (i = 0; i < d; i++) {
                DES (hex_arr, i, manipulaTexto1, orienta, K);
                printf("\n\n\n\n\n");
                fprintf(orienta, "\n\n\n\n\n");
            }

            chave(C, D, CD, K, orienta, key_hex[1]);

            for (i = 0; i < d; i++) {
                DES (manipulaTexto1, i, manipulaTexto2, orienta, K);
                printf("\n\n\n\n\n");
                fprintf(orienta, "\n\n\n\n\n");
            }

            chave(C, D, CD, K, orienta, key_hex[2]);

            for (i = 0; i < d; i++) {
                DES (manipulaTexto2, i, manipulaTexto3, orienta, K);
                printf("\n\n\n\n\n");
                fprintf(orienta, "\n\n\n\n\n");
            }

        } else
            if (acao == 'd' || acao == 'D') {

                chave(C, D, CD, K, orienta, key_hex[2]);

                for (i = 0; i < d; i++) {
                    UnDES (hex_arr, i, manipulaTexto1, orienta, K);
                    printf("\n\n\n\n\n");
                    fprintf(orienta, "\n\n\n\n\n");
                }

                chave(C, D, CD, K, orienta, key_hex[1]);

                for (i = 0; i < d; i++) {
                    UnDES (manipulaTexto1, i, manipulaTexto2, orienta, K);
                    printf("\n\n\n\n\n");
```

```
            fprintf(orienta, "\n\n\n\n\n");
        }

        chave(C, D, CD, K, orienta, key_hex[0]);

        for (i = 0; i < d; i++) {
            UnDES (manipulaTexto2, i, manipulaTexto3, orienta, K);
            printf("\n\n\n\n\n");
            fprintf(orienta, "\n\n\n\n\n");
        }

    }



if (acao == 'e' || acao == 'E') {
    printf ("\n\n\t__RESULTADO_DA_ENCRIPTAÇÃO_DO_TEXTO__");
    fprintf (orienta, "\n\n\t__RESULTADO_DA_ENCRIPTAÇÃO_DO_TEXTO__");

    copyResultado(d, manipulaTexto3, output);


    printf ("\n\nTexto Encriptado : ");
    fprintf (orienta, "\n\nTexto Encriptado : ");
    printf ("\n%s\n\n",output);
    fprintf (orienta, "\n%s\n\n",output);

    arquivoResposta(output, textoSaida, orienta);


    printf ("\n\nFim da execução do algoritmo -- Texto Encriptado.\n");
    fprintf (orienta, "\n\nFim da execução do algoritmo -- Texto
        Encriptado." );
}
else
    if (acao == 'd' || acao == 'D') {
        printf ("\n\n\t__RESULTADO_DA_DESENCRIPTAÇÃO_DO_TEXTO__");
        fprintf (orienta, "\n\n\t__RESULTADO_DA_DESENCRIPTAÇÃO_DO_TEXTO__"
            );


        hex_to_plain (manipulaTexto3, output, d);


        for (i = 0; i < tamArquivo; i++) {
            printf("%c, %d,  %X\n", output[i], output[i], output[i]);
        }

        printf ("\n\nTexto Descriptografado : ");
        fprintf (orienta, "\n\nTexto Descriptografado : ");
        printf ("\n%s\n\n",output);
        fprintf (orienta, "\n%s\n\n",output);

        arquivoResposta(output, textoSaida, orienta);


        printf ("\n\nFim da execução do algoritmo -- Texto Descifrado.\n
            \n");
        fprintf (orienta, "\n\nFim da execução do algoritmo -- Texto
            Descifrado." );
```

```
        }

        fclose(orienta);
    return 0;
}
```