Criptare imagine

In cadrul acestui proeiect pentru facucltate, am realizat un program care primeste ca input o imagine in format .bmp si o cripteaza / decripteaza, aplicand transformari asupra pixelilor.

Memorare

```
typedef struct
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
}pixel;
```

Am definit o structura “pixel” pentru a memora convenabil continutul unei imagini. Vor fi stocati cei 3 octeti ai unui pixel, pentru canalele rosu, verde şi albastru.

XorShift32

Funcţia primeste prin intermediul parametrilor:

- **v*- un tablou unidimensional alocat dinamic in care vor fi stocate numerele pseudo-aleatoare
- *latime* – latimea imaginii criptate
- *inaltime* – inaltimea imaginii criptate
- *seed* – valoarea initiala de la care se pleaca

si va genera o secventa de numere pseudo-aleatoare prin intermediul algoritmului XorShift32 propus de George Marsaglia.

```
void XorShift32(unsigned int **v,int latime,int inaltime,unsigned int seed)
{
    /* functie care va genera in vectorul v numere pseudo-aleatoare cu ajutorul
    algoritmului XORSHIFT32 */

    int n=2*inaltime*latime,i;
    unsigned int r;

    *v=(unsigned int *)malloc(sizeof(unsigned int)*n);
    if(!v)
    {
        printf("Eroare la alocarea de memorie pentru vectorul de numere aleatoare ");
        return;
    }

    (*v)[0]=seed;

    for(i=1;i<n;i++)
    {   r=(*v)[i-1];
        r=r^r<<13;
        r=r^r>>17;
        r=r^r<<5;
       (*v)[i]=r;
    }
}
```

dimensiuni_imagine_header

Functia va primi ca parametri:

- *header* – un tablou unidimensional alocat dinamic in care va fi stocat headerul imaginii transmis prin parametrul nume_fisier
- *nume_fisier* – calea unei imagini
- *inaltime* – inaltimea imaginii transmisa prin parametrul "*nume_fisier*"
- *latime* – latimea imaginii transmisa prin parametrul "*nume_fisier*"
- *padding* – paddingul imaginii transmisa prin parametrul "*nume_fisier*"

si retine principalele caracteristici ale unei imagini

```
void dimensiuni_imagine_header(unsigned char **header, char *nume_fisier,int *inaltime,int *latime,int *padding)
{
    /* functie care retine headerul, latimea,inaltimea si paddingul unei imagini */
    FILE *f=fopen(nume_fisier,"rb");
    if(!f)
    {
        printf("Nu s-a gasit fisierul cu imaginea initiala: %s",nume_fisier);
        return;
    }

    fseek(f,18,SEEK_SET);
    fread(latime,sizeof(unsigned int),1,f);
    fread(inaltime,sizeof(unsigned int),1,f);

    if((*latime) % 4 != 0)
        *padding = 4 - (3 * (*latime)) % 4;
    else
        *padding = 0;

    fseek(f,0,SEEK_SET);


    int i;
    *header=(unsigned char*)malloc(sizeof(char)* 54);

    if(!header)
    {
        printf("Nu s-a putut aloca memorie pentru header");
        return;
    }

    for(i=0;i<54;i++)
        {
            unsigned char c;
            fread(&c,sizeof(unsigned char),1,f);
            (*header)[i]=c;
        }

    fclose(f);

}
```

Functia deschide imaginea furnizata in modul “rb” si se pozitioneaza pe pozitia celui de al 18-lea octet pentru a citi latimea imaginii, iar in continuare inaltimea acestuia si calculeaza paddingul. Dupa care se pozitioneaza la inceput, si citeste primii 54 octeti, reprezentand header-ul imaginii.

IncarcaImagine

Functia va avea ca parametri:

- **v* – tablou unidimensional structura de tip pixel
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini
- **fisier* – sir de caractere ce reprezinta calea imaginii

si liniarizeaza imaginea transmisa prin parametrul *fisier*, incepand din coltul din stanga sus pana la cel din dreapta jos.

```
void IncarcaImagine(pixel **v,int latime,int inaltime,int padding,char *fisier)
{
    /* functie care liniarizeaza o imagine */
    *v=(pixel*)malloc(sizeof(pixel)*latime*inaltime);

    if(!(*v))
    {
         printf("Nu s-a putut aloca memorie pentru vector");
        return;
    }

    FILE*f=fopen(fisier,"rb");

    if(!f)
    {
        printf("Nu s-a gasit fisierul cu imaginea initiala: %s pentru liniarizarea ei",fisier);
        return;
    }


       fseek(f,sizeof(unsigned char)*54,SEEK_SET);

       int j,i,nr=latime*(inaltime-1);
       pixel pix;

       for(i=0;i<inaltime;i++)
       {
           for(j=0;j<latime;j++)
           {
               fread(&pix,sizeof(pixel),1,f);
               (*v)[nr++]=pix;
           }

           nr=nr-2*latime;
           fseek(f,sizeof(char)*padding,SEEK_CUR);
       }
       fclose(f);
   }
```

Functia sare peste primii 54 de octeti, ce reprezinta header-ul, dupa care citeste cate un pixel (3 octeti fara semn), punandu-l pe pozitia nr, actualizata dupa fiecare atribuire, in tabloul unidimensional *v .

DescarcaImagine

Functia va avea ca parametri:

- *v – tablou unidimensional structura de tip pixel ce memoreaza o imagine in forma liniarizata
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini
- **fisier* – sir de caractere ce reprezinta denumirea unei imagini
- *padding* – padding-ul imaginii
- *header* – headerul imaginii

si salveaza in memoria externa sub numele trimis ca parametru *fisier* imaginea memorata prin tabloul unidimensional *v

```
void DescarcaImagine(pixel *v,int latime, int inaltime,int padding,char *fisier,unsigned char *header)
{
    /* functie care deliniarizeaza o imagine */
    int i,j,poz=latime*inaltime-latime;
    unsigned char c='0';

    FILE*f=fopen(fisier,"wb");

    fwrite(header,sizeof(unsigned char),54,f);

    for(i=0;i<inaltime;i++)
    {
        for(j=0;j<latime;j++)
         fwrite(&v[poz++],sizeof(pixel),1,f);

        fwrite(&c,sizeof(unsigned char),padding,f);
        poz=poz-2*latime;
    }

    fclose(f);
}
```

Functia este complementara celei numite *IncarcaImagine*.

Durstenfeld

Functia va avea ca parametri:

- **perm* – tablou unidimensional in care va fi stocata o permutare aleatoare
- *latime – latimea unei imagini*
- *inaltime – inaltimea unei imagini*

- **R – tablou unidimensional cu numere aleatoare*

si construieste prin intermediul algoritmului lui Durstenfeld o permutare aleatoare, bazandu-se pe valorile tabloului *R*.

```
void Durstenfeld(unsigned int **perm,int latime,int inaltime, unsigned int *R)
{
    /* functie care genereaza o permutarea aleatoare, prin intermediul algoritmului lui Durstenfeld */
    unsigned int r,aux,n;
    int i;
    n=latime*inaltime;

    *perm=(unsigned int*)malloc(sizeof(unsigned int)*n);

    if(!*perm)
    {
        printf("Eroare la alocarea de memorie a vectorului");
        return;
    }

    for(i=n-1;i>=0;i--)
        (*perm)[i]=i;

    for(i=n-1;i>=1;i--)
    {
        r=R[n-i]%(i+1);

        aux=(*perm)[r];
        (*perm)[r]=(*perm)[i];
        (*perm)[i]=aux;
    }

}
```

permutarePixeli

Functia va avea ca parametri:

- **imag_liniar_permutat* - tablou unidimensional structura de tip pixel care memoreaza pixelii unei imagini in forma liniara, dupa permutarea acestora
- **Imag_liniar* – tablou unidimensional structura de tip pixel care retine o imagine in forma liniara
- **perm* – tablou unidimensional ce retine o permutare aleatoare
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini

si permuta pixelii unei imagini (retinuta in forma liniarizata – *imag_liniar*), pe baza unei permutari *perm* si vor fi stocati in tabloul *imag_liniar_permutat*.

```
void permutarePixeli(pixel **imag_liniar_permutat,pixel*imag_liniar,unsigned int *perm,int latime,int inaltime)
{
    /*functie care permuta pixelii imaginii liniarizate, pe baza unei permutari date */
    int i,n;
    n=latime*inaltime;
    *imag_liniar_permutat=(pixel*)malloc(sizeof(pixel)*n);

    if(!*imag_liniar_permutat)
    {
        printf("Eroare la alocarea de memorie a vectorului");
        return;
    }

    for(i=0;i<n;i++)
        (*imag_liniar_permutat)[perm[i]]=imag_liniar[i];
}
```

octet_nr

Functia va avea ca parametri:

- *x* – un numar natural
- *nr* – un numar

si returneaza al *nr*-lea octet al numarului *x*.

```
unsigned char octet_nr(unsigned int x,int nr)
{
    /* functie care returneaza octetul nr al unui numar x */
    unsigned char *p;
    p=(unsigned char*)&x;
    int i;
    for(i=0;i<nr;i++)
        p++;
    unsigned char y;
    y=*p;

    return y;
}
```

xorFinal

Functia va avea ca parametri :

- **imag_liniar* - tablou unidimensional structura de tip pixel care memoreaza pixelii unei imagini in forma liniarizata
- **R* – tablou unidimensional in care este memorata o permutare aleatoare
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini
- *SV* – numar natural (starting value)

- **imag_liniar_criptat* - tablou unidimensional structura de tip pixel care memoreaza pixelii unei imagini in forma liniara, dupa aplicarea operatiei de xorare pe baza formulei:

$$C_k = \begin{cases} SV \oplus P'_0 \oplus R_{W*H}, & k = 0 \\ C_{k-1} \oplus P'_k \oplus R_{W*H+k}, & k \in \{1,2,\dots,W*H-1\} \end{cases}$$

unde *SV* (*starting value*) este un număr întreg nenul fără semn pe 32 de biți.

Unde C = *imag_liniar_criptat
P' = *imag_liniar
R = *R

Iar $\oplus$ reprezinta operatia *xor* intre primii 3 octeti ai celor 3 componente

```
void xorFinal(pixel **imag_liniar_criptat,pixel *imag_liniar,unsigned int *R,int latime,int inaltime, unsigned int SV)
{
    /* functie care aplica operatia xor pe pixelii liniarizati ai unei matrice */
    int n=latime*inaltime,i;
    *imag_liniar_criptat=(pixel*)malloc(sizeof(pixel)*n);

    if(!*imag_liniar_criptat)
    {
        printf("Eroare la alocarea de memorie a vectorului");
        return;
    }

        (*imag_liniar_criptat)[0].b=(octet_nr(SV,0))^(imag_liniar[0].b)^(octet_nr(R[n],0));
        (*imag_liniar_criptat)[0].g=(octet_nr(SV,1))^(imag_liniar[0].g)^(octet_nr(R[n],1));
        (*imag_liniar_criptat)[0].r=(octet_nr(SV,2))^(imag_liniar[0].r)^(octet_nr(R[n],2));

    for(i=1;i<n;i++)
    {
        (*imag_liniar_criptat)[i].b = ( (*imag_liniar_criptat)[i-1].b ) ^ (imag_liniar[i].b) ^ (octet_nr(R[n+i],0));
        (*imag_liniar_criptat)[i].g = ( (*imag_liniar_criptat)[i-1].g ) ^ (imag_liniar[i].g) ^ (octet_nr(R[n+i],1));
        (*imag_liniar_criptat)[i].r = ( (*imag_liniar_criptat)[i-1].r ) ^ (imag_liniar[i].r) ^ (octet_nr(R[n+i],2));

    }

}
```

criptareImag

Functia va avea ca parametri:

- **imag_init* – sir de caractere care furnizeaza calea imaginii initiale
- **imag_final* – sir de caractere care furnizeaza calea imaginii finale
- **fisier_cheie* – sir de caractere care furnizeaza numele fisierului care contine cheia secreta si seed-ul pentru functia de generare de numere aleatoare
- *ok* – valoare care furnizeaza 1 sau 0 daca s-au gasit fisierele *imag_initial* si *fisier_cheie*

```
void criptareImag(char*imag_init,char*imag_final,char*fisier_cheie,int *ok)
{
    /* functia principala care cripteaza o imagine */

    FILE*Init=fopen(imag_init,"r");
    FILE*Final=fopen(imag_final,"wb");
    FILE*Cheie=fopen(fisier_cheie,"r");

    if(!Init)
    {
        printf("Eroare la deschiderea imaginii %s \n",imag_init);
        (*ok)=0;
        return;
    }

    if(!Cheie)
    {
        printf("Eroare la deschiderea fisierului %s\n",fisier_cheie);
         (*ok)=0;
        return;
    }

    unsigned int *R,*Perm,R0,SV;

    pixel *imag_liniar_criptat;
    pixel *imag_liniar,*imag_liniar_permutat;

    int latime,inaltime,padding;
    unsigned char * header;


    fscanf(Cheie,"%u %u",&R0,&SV); // citeste din fisier cheia secreta si seed-ul pentru algoritmul lui Durstenfeld

    dimensiuni_imagine_header(&header,imag_init,&inaltime,&latime,&padding);

    IncarcaImagine(&imag_liniar,latime,inaltime,padding,imag_init);  // transforma imaginea in mod liniar

    XorShift32(&R,latime,inaltime,R0);     // construieste prin XORSHIFT32 un set de numere pseudo-aleatoare

    Durstenfeld(&Perm,latime,inaltime,R);          // Prin alg lui Durstenfeld genereaza o permutare

    permutarePixeli(&imag_liniar_permutat,imag_liniar,Perm,latime,inaltime);   // Se permuta pixelii

    xorFinal(&imag_liniar_criptat,imag_liniar_permutat,R,latime,inaltime,SV); // se xor-eaza pixelii

    DescarcaImagine(imag_liniar_criptat,latime,inaltime,padding,imag_final,header); // se deliniarizeaza imaginea si se salveaza in m

    fclose(Init);
    fclose(Final);
    fclose(Cheie);

    free(R);
    free(Perm);
    free(imag_liniar);
    free(imag_liniar_criptat);
    free(imag_liniar_permutat);

}
```

Functia deschide imaginea, respectiv fisierul cheie, iar daca nu reuseste, ok=0 si se opreste. Citeste din fisierul *fisier_cheie* cheia secreta, repspectiv seed-ul, obtine caracteristicile imaginii *imag_init* prin apelul *dimensiuni_imagine_header(&header,imag_init,&inaltime,&latime,&padding),* liniarizeaza imaginea prin *IncarcaImagine (&imag_liniar, latime, inaltime, padding, imag_init)* , genereaza un set de numere aleatoare prin algoritmul xorShift32 (*XorShift32(&R, latime ,inaltime, R0)*),

genereaza o permutare aleatoare prin algoritmul lui Durstenfeld si pe baza numerelor aleatoare generate (*Durstenfeld(&Perm ,latime, inaltime, R)*) permuta pixelii *permutarePixeli (&imag_liniar_permutat, imag_liniar, Perm, latime, inaltime),* are loc xor-area *xorFinal(&imag_liniar_criptat,imag_liniar_permutat,R,latime,inaltime,SV)*, iar in final imaginea este deliniarizata *DescarcaImagine(imag_liniar_criptat,latime,inaltime,padding,imag_final,header)* . Dupa salvarea in memoria externa a noii imagini obtinute, *imag_final*, este eliberata memoria.

invers_perm

Functia va avea ca parametri:

- **perm_invers* - tablou unidimensional care memoreaza permutarea inversa a permutarii *perm*
- **perm* – tablou unidimensional care memoreaza o permutare
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini

Programul primeste o permutare prim intermediul parametrului *perm* si memoreaza in *perm_invers* permutarea inversa.

```
void invers_perm(unsigned int **perm_invers,unsigned int *perm,int latime,int inaltime)
{
    /* functie care genereaza inversa unei permutari */
    int n=latime*inaltime,i;

    *perm_invers=(unsigned int*)malloc(sizeof(unsigned int)*n);

    if(!perm_invers)
        {
        printf("eroare");
        return;
        }

   for(i=0;i<n;i++)
    (*perm_invers)[perm[i]]=i;

}
```

inversXOR

Functia primeste ca parametri:

- **imag_liniar_decript* - tablou unidimensional structura de tip pixel care memoreaza pixelii unei imagini in forma liniarizata, dupa aplicare operatiei de xor-are
- **imag_liniar* - tablou unidimensional structura de tip pixel care memoreaza pixelii unei imagini in forma liniarizata

- *latime* - latimea unei imagini
- *inaltime* – inaltimea unei imagini
- **R* – tablou unidimensional care
- *SV* – numar natural (starting value)

Si aplica operatia de xor-are pe pixelii unei imagini memorate in forma liniarizata

```
void inversXOR(pixel **imag_liniar_decript, pixel*imag_liniar,int latime,int inaltime,unsigned int *R,unsigned int SV)
{
    /* functie care aplica operatia xor pe pixelii liniarizati ai matricei, obtinand astfel varianta nemodificata
    a acestora */

    int i,n=latime*inaltime;
    *imag_liniar_decript=(pixel*)malloc(sizeof(pixel)*n);

    if(!*imag_liniar_decript)
    {
        printf("Eroare");
        return;
    }

        (*imag_liniar_decript)[0].b=(octet_nr(SV,0))^(imag_liniar[0].b)^(octet_nr(R[n],0));
        (*imag_liniar_decript)[0].g=(octet_nr(SV,1))^(imag_liniar[0].g)^(octet_nr(R[n],1));
        (*imag_liniar_decript)[0].r=(octet_nr(SV,2))^(imag_liniar[0].r)^(octet_nr(R[n],2));

    for(i=1;i<n;i++)
    {
        (*imag_liniar_decript)[i].b = ( imag_liniar[i-1].b ) ^ (imag_liniar)[i].b ^ (octet_nr(R[n+i],0));
        (*imag_liniar_decript)[i].g = ( imag_liniar[i-1].g ) ^ (imag_liniar)[i].g ^ (octet_nr(R[n+i],1));
        (*imag_liniar_decript)[i].r = ( imag_liniar[i-1].r ) ^ (imag_liniar)[i].r ^ (octet_nr(R[n+i],2));
    }
}
```

decriptareImag

Functia va avea ca parametri:

- **imag_init* – sir de caractere care furnizeaza calea imaginii initiale
- **imag_final* – sir de caractere care furnizeaza calea imaginii finale
- **fisier_cheie* – sir de caractere care furnizeaza numele fisierului care contine cheia secreta si seed-ul pentru functia de generare de numere aleatoare
- *ok* – valoare care furnizeaza 1 sau 0 daca s-au gasit fisierele *imag_initial* si *fisier_cheie*

```
void decriptareImag(char*imag_init,char*imag_final,char*fisier_cheie,int *ok)
{
    FILE*Init=fopen(imag_init,"rb");
    FILE*Cheie=fopen(fisier_cheie,"r");
    FILE*Final=fopen(imag_final,"wb");
    if(!Init)
    {
        printf("Eroare la deschiderea imaginii");
        (*ok)=0;
        return;
    }

    if(!Cheie)
    {
        printf("Eroare la deschiderea fisierului ce contine cheia secreta");
        (*ok)=0;
        return;
    }

    int latime,inaltime,padding;
    unsigned char * header;
    unsigned int *R,*Perm,*Perm_invers,R0,SV;

    pixel *imag_liniar_xor;
    pixel *imag_liniar;
    pixel *imag_decript;

    fscanf(Cheie,"%u %u",&R0,&SV);

    dimensiuni_imagine_header(&header,imag_init,&inaltime,&latime,&padding);

    IncarcaImagine(&imag_liniar,latime,inaltime,padding,imag_init);   // transforma imaginea in mod liniar

    XorShift32(&R,latime,inaltime,R0);          // construieste prin XORSHIFT32 un set de numere pseudo-aleatoare

    Durstenfeld(&Perm,latime,inaltime,R);                // Prin alg lui Durstenfeld genereaza o permutare

    invers_perm(&Perm_invers,Perm,latime,inaltime);  // se genereaza permutarea inversa

    inversXOR(&imag_liniar_xor,imag_liniar,latime,inaltime,R,SV); // se xor-eaza pixelii

    permutarePixeli(&imag_decript,imag_liniar_xor,Perm_invers,latime,inaltime); // se permuta pixelii pe baza permutarii inv

    DescarcaImagine(imag_decript,latime,inaltime,padding,imag_final,header); // se deliniarizeaza imaginea

    fclose(Final);
    fclose(Init);
    fclose(Cheie);

    free(R);
    free(Perm);
    free(Perm_invers);
    free(imag_decript);
    free(imag_liniar);
    free(imag_liniar_xor);
    free(header);

}
```

Functia deschide imaginea, respectiv fisierul cheie, iar daca nu reuseste, ok=0 si se opreste. Citeste din fisierul *fisier_cheie* cheia secreta, repspectiv seed-ul, obtine caracteristicile imaginii *imag_init* prin apelul *dimensiuni_imagine_header(&header,imag_init,&inaltime,&latime,&padding),* liniarizeaza

imaginea prin *IncarcaImagine (&imag_liniar, latime, inaltime, padding, imag_init)* , genereaza un set de numere aleatoare prin algoritmul xorShift32 (*XorShift32(&R, latime ,inaltime, R0)*), genereaza o permutare aleatoare prin algoritmul lui Durstenfeld precum si inversa sa (*invers_perm(&Perm_invers,Perm,latime,inaltime)*) si pe baza numerelor aleatoare generate (*Durstenfeld(&Perm ,latime, inaltime, R)* are loc xor-area inversa *inversXOR(&imag_liniar_xor, imag_liniar, latime, inaltime, R, SV),* se permuta pixelii pe baza permutarii inverse obtinute *permutarePixeli(&imag_decript, imag_ liniar_xor, Perm_invers, latime, inaltime)* iar in final imaginea este deliniarizata *DescarcaImagine(imag_liniar_criptat,latime,inaltime,padding,imag_final,header)* . Dupa salvarea in memoria externa a noii imagini obtinute, *imag_final*, este eliberata memoria.

chiTest

Functia va avea ca parametru:

- **imag* – sir de caractere care furnizeaza calea unei imagini

si afiseaza valorile testului chi pentru cele 3 canale RGB ale imaginii *imag.* Daca valorile testelor chi nu depasesc 293.15, inseamna ca procesul de criptare a avut loc cu success

```
void chiTest(char*imag)
{
    FILE*f=fopen(imag,"rb");

    if(!f)
    {
        printf("Nu s-a gasit imaginea");
        return;
    }
    unsigned int *frecvR,*frecvG,*frecvB;
    double chi_r=0,chi_g=0,chi_b=0,fmed;
    int i,j,latime,inaltime,padding;
    unsigned char*header;
    pixel pix;

    dimensiuni_imagine_header(&header,imag,&inaltime,&latime,&padding);

    frecvR=(unsigned int*)calloc(256,sizeof(unsigned int));
    frecvG=(unsigned int*)calloc(256,sizeof(unsigned int));
    frecvB=(unsigned int*)calloc(256,sizeof(unsigned int));

    if(!frecvR)
    {
        printf("Nu s-a putut aloca memorie");
        return;
    }
```

```
 if(!frecvG)
 {
     printf("Nu s-a putut aloca memorie");
     return;
 }

 if(!frecvB)
 {
     printf("Nu s-a putut aloca memorie");
     return;
 }

 fmed=latime*inaltime/256;

 fseek(f,sizeof(unsigned char)*54,SEEK_SET);

for(i=0;i<inaltime;i++)
 {
 for(j=0;j<latime;j++)
 {
    fread(&pix,sizeof(pixel),1,f);
    frecvB[pix.b]++;
    frecvG[pix.g]++;
    frecvR[pix.r]++;
 }
 fseek(f,sizeof(char)*padding,SEEK_CUR);
 }


   for(i=0;i<256;i++)
   {
       chi_b=chi_b+(frecvB[i]-fmed)*(frecvB[i]-fmed);
       chi_g=chi_g+(frecvG[i]-fmed)*(frecvG[i]-fmed);
       chi_r=chi_r+(frecvR[i]-fmed)*(frecvR[i]-fmed);
   }

   chi_b/=fmed;
   chi_g/=fmed;
   chi_r/=fmed;

   printf("Pentru imaginea %s, testul are pe canalul rosu %.2f\n",imag,chi_r);
   printf("Pentru imaginea %s, testul are pe canalul verde %.2f\n",imag,chi_g);
   printf("Pentru imaginea %s, testul are pe canalul albastru %.2f\n\n",imag,chi_b);

   free(frecvB);
   free(frecvG);
   free(frecvR);
   free(header);

}
```

In primare partea a main-ului, am apelat functiile de criptare si decriptare pentru o imagine citita dintr-un fisier:

```
int main()
{

    char *input,*encript,*output,*secret_key;
    int ok1=1,ok2=1;

    input=(char*)malloc(sizeof(char)*30);
    if(!input)
    {
        printf("Nu s-a putut aloca memorie");
        return 0;
    }

    encript=(char*)malloc(sizeof(char)*30);
    if(!encript)
    {
        printf("Nu s-a putut aloca memorie");
        return 0;
    }

    output=(char*)malloc(sizeof(char)*30);
    if(!output)
    {
        printf("Nu s-a putut aloca memorie");
        return 0;
    }

   secret_key=(char*)malloc(sizeof(char)*30);
   if(! secret_key)
   {
       printf("Nu s-a putut aloca memorie");
       return 0;
   }

   FILE*fin1=fopen("imagini_criptare.txt","r");

   if(!fin1)
   {
       printf("Nu s-a putut gasi fisierul imagini_criptare.txt ");
       return 0;
   }

   FILE*fin2=fopen("imagini_decriptare.txt","r");

   if(!fin2)
   {
       printf("Nu s-a putut gasi fisierul imagini_decriptare.txt ");
       return 0;
   }

   fscanf(fin1,"%s",input);
   fscanf(fin1,"%s",encript);
   fscanf(fin1,"%s",secret_key);

   criptareImag(input,encript,secret_key,&ok1);
```

```
fscanf(fin2,"%s",encript);
fscanf(fin2,"%s",output);
fscanf(fin2,"%s",secret_key);

if(ok1==1)
    decriptareImag(encript,output,secret_key,&ok2);

if(ok2==1)
{
    chiTest(input);
    chiTest(encript);
}
else
{
    printf("Nu s-au efectuat testele chi pentru ca nu s-a putut face criptarea / decriptarea\n");
    return 0;
}

fclose(fin1);
fclose(fin2);
```

Se aloca spatiu pentru sirurile de caractere ce vor memora numele fisierelor, se vor citi din fisierul imagini_criptare.txt numele imaginii care va fi criptata, numele imaginii in forma criptata si fisierul ce contine cheia secreta si starting value. Se deschide fisierul imagini_decriptare.txt ce contine numele imaginii in forma criptata, numele imaginii dupa ce a fost decriptata, precum si fisierul ce contine cheia secreta si starting value, iar in final se va afisa rezultatul testului chi pentru fiecare canal RGB.