Documentatie proiect programare procedurala

Proiectul reuneste 2 mari teme ale informaticii: criptarea continutului unui document, respectiv procesarea digitala si recunoasterea unor pattern-uri dintr-o imagine.

In prima parte, vor fi detaliati pasii necesari criptarii unei imagini în format bmp.

Criptare si decriptare

```
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 memoria externa

    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 inverse

    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*

```
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:

```c
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.

Template matching

In cea de-a doua parte vor fi descrisi pasii prin care se doreste recunoasterea unor pattern-uri intr-o imagine.

grayscale_image

Functia va avea ca parametri:

- **nume_fisier_sursa* - sir de caractere care furnizeaza calea imaginii initiale
- **nume_fisier_destinatie* - sir de caractere care furnizeaza calea imaginii finale

```
void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie)
{
   FILE *fin, *fout;
   unsigned int dim_img, latime_img, inaltime_img;
   unsigned char pRGB[3], aux;

//  printf("nume fisier sursa = %s \n",nume_fisier_sursa);

   fin = fopen(nume_fisier_sursa, "rb");
   if(fin == NULL)
    {
        printf("Nu am gasit imaginea sursa din care citesc \n");
        return;
    }

   fout = fopen(nume_fisier_destinatie, "wb+");

   fseek(fin, 2, SEEK_SET);
   fread(&dim_img, sizeof(unsigned int), 1, fin);
 // printf("Dimensiunea imaginii in octeti: %u\n", dim_img);

   fseek(fin, 18, SEEK_SET);
   fread(&latime_img, sizeof(unsigned int), 1, fin);
   fread(&inaltime_img, sizeof(unsigned int), 1, fin);
//  printf("Dimensiunea imaginii in pixeli (latime x inaltime): %u x %u\n",latime_img, inaltime_img);
```

```
//copiaza octet cu octet imaginea initiala in cea noua
fseek(fin,0,SEEK_SET);
unsigned char c;
while(fread(&c,1,1,fin)==1)
{
    fwrite(&c,1,1,fout);
    fflush(fout);
}
fclose(fin);

//calculam padding-ul pentru o linie
int padding;
if(latime_img % 4 != 0)
    padding = 4 - (3 * latime_img) % 4;
else
    padding = 0;

fseek(fout, 54, SEEK_SET);
int i,j;
for(i = 0; i < inaltime_img; i++)
{
    for(j = 0; j < latime_img; j++)
    {
        //citesc culorile pixelului
        fread(pRGB, 3, 1, fout);
        //fac conversia in pixel gri
        aux = 0.299*pRGB[2] + 0.587*pRGB[1] + 0.114*pRGB[0];
        pRGB[0] = pRGB[1] = pRGB[2] = aux;
        fseek(fout, -3, SEEK_CUR);
        fwrite(pRGB, 3, 1, fout);
        fflush(fout);
    }
    fseek(fout,padding,SEEK_CUR);
}
fclose(fout);
}
```

Functia transforama o imagine normala intr-una grayscale, folosindu-se de formula :

$$R' = G' = B' = [0.299 * R + 0.587 * G + 0.114 * B]$$

- R’ = noul octet pentru canlul rosu
- G’ = noul octet pentru canalul verde
- B’ = noul octet pentru canalul albastru

dimensiuni_imagine

Functia va avea ca parametri:

- **nume_fisier* – sir de caractere care furnizeaza calea unei imagini
- *inaltime* - inaltimea imaginii
- *latime* - latimea imaginii
- *padding* – padding-ul imaginii

```
void dimensiuni_imagine(char *nume_fisier,int *inaltime,int *latime,int *padding)
{
    FILE *f=fopen(nume_fisier,"rb");
    if(!f)
    {
        printf("Nu s-a gasit fisierul cu imaginea initiala: %s \n",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);

    fclose(f);

}
```

imagToMatrice

Functia va avea ca parametri:

- **imag* - sir de caractere ce furnizeaza calea unei imagini
- **v – tablou unidimensional structura de tip pixel in care va fi retinuta imaginea (colt stanga sus -> dreapta jos)
- *inaltime* – inaltimea imaginii
- *latime* – latimea imaginii
- *padding* – padding-ul imaginii

```
void imagToMatrice(char *imag,pixel ***v,int inaltime,int latime,int padding)
{
    // functie care primind o imagine, o transforma in matrice
    int i,j;
    pixel c;

    *v=(pixel**)malloc(sizeof(pixel*) * inaltime);

    if(!*v)
    {
        printf("Eroare la alocarea de memorie \n");
        return;
    }

    FILE*fin=fopen(imag,"rb");

    if(!fin)
    {
        printf("Nu s-a putut gasi imaginea %s \n",imag);
        return;
    }

    fseek(fin,54*sizeof(char),SEEK_SET);
```

```
    for(i=inaltime-1;i>=0;i--)
        {
            (*v)[i]=(pixel*)malloc(sizeof(pixel)*latime);

            if(!(*v)[i])
            {
                printf("Eroare la alocarea de memorie\n");

                for(j=inaltime-1;j>=i;j--)
                    free((*v)[j]);
                free(*v);

                return;
            }
            for(j=0;j<latime;j++)
            {
                fread(&c,sizeof(pixel),1,fin);
                (*v)[i][j].b=c.b;
                (*v)[i][j].g=c.g;
                (*v)[i][j].r=c.r;
            }
            fseek(fin,padding,SEEK_CUR);
        }

    fclose(fin);

}
```

Functia va transforma o imagine intr-un tablou bidimensional pentru a o prelucra mai usor.

matriceToimagine

Functia va avea ca parametri:

- *imag_init* – sir de caractere ce transmite calea imaginii initiale
- *imag_final* – sir de caractere ce transmite numele imaginii dupa ce va fi transformata din matrice in imagine
- ***v* – tablou bidimensional ce memoreaza o imagine
- *inaltime* – inaltimea imaginii
- *latime* - latimea imaginii
- *padding* – padding-ul imaginii

Functia este simetrica celei imagineToMatrice, salvand in memoria externa o imagine salvata sub forma unui tablou bidimensional de tip pixel. Este nevoie si de imaginea initiala, in forma normala, pentru header-ul acesteia.

```
void matriceToimagine(char *imag_init,char *imag_fin,pixel **v,int inaltime,int latime,int padding)
{

    FILE *fin,*fout;
    fin=fopen(imag_init,"rb");
    fout=fopen(imag_fin,"wb");

    if(fin==NULL)
    {
        printf("Nu s-a gasit imaginea %s \n",imag_init);
        return ;
    }


    int i,j;
    pixel pix={'0','0','0'};
    char c;

    for(i=0;i<54;i++)
    {
        fread(&c,sizeof(char),1,fin);
        fwrite(&c,sizeof(char),1,fout);
    }


     for(i=inaltime-1;i>=0;i--)
    {
        for(j=0; j<latime;j++)
        {
            fwrite(&v[i][j].b,sizeof(unsigned char),1,fout);
            fwrite(&v[i][j].g,sizeof(unsigned char),1,fout);
            fwrite(&v[i][j].r,sizeof(unsigned char),1,fout);

        }
           fwrite(&pix,sizeof(pixel),padding,fout);
    }

    fclose(fin);
    fclose(fout);
}
```

medie_pixeli

Functia va avea ca parametri:

- ***v* – tablou bidimensional ce memoreaza o imagine
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini
- *linie* – linia tabloului de unde se va calcula media pixelilor
- *coloana* – coloana tabloului de unde se va calcula media pixelilor

Functia va calcula media intensitatii pixelilor imaginii memorate in tabloul ***v* incepand cu coordonatele (linie,coloana) pentru o imagine de dimensiuni (latime,inaltime)

```
float medie_pixeli(pixel **v,int inaltime,int latime,int linie,int coloana)
{
    float S=0;
    int i,j;

    for(i=linie+0;i<linie+inaltime;i++)
        for(j=coloana+0;j<coloana+latime;j++)
            S=S+v[i][j].r;

    S=S/latime;
    S=S/inaltime;

    return S;
}
```

deviatie_standard

- ***v* - tablou bidimensional ce memoreaza o imagine
- *latime* – latimea unei imagini
- *inaltime* – inaltimea unei imagini
- *med* – media intensitatii pixelilor pentru portiunea respectiva
- *linie* – linia tabloului de unde se va calcula deviatia standard
- *coloana* – coloana tabloului de unde se va calcula deviatia standard

Functia calculeaza deviatia standard a unei portiuni dintr-o imagine, incepand cu coordonatele (linie,coloana) pentru un sablon de dimensiuni (inaltime,latime), cu formula:

$$\sigma_{f_I} = \sqrt{\frac{1}{n-1}\sum_{(i,j)\in f_I}\left(f_I(i,j) - \overline{f_I}\right)^2}$$

```
float deviatie_standard(pixel **v,int inaltime,int latime,float med, int linie,int coloana)
{
    int i,j;
    float S=0;
    for(i=linie+0;i<inaltime+linie;i++)
        for(j=coloana+0;j<coloana+latime;j++)
            S=S+(v[i][j].r-med)*(v[i][j].r-med);

    S=S/(latime*inaltime-1);
    S=sqrt(S);

    return S;
}
```

corelatie

Functia va avea ca parametri:

- **imag_matrice – tablou bidimensional ce memoreaza o imagine
- **sablon_matrice – tablou bidimensional ce memoreaza o imagine
- *inaltime* – inaltimea imaginii *sablon_matrice*
- *latime* – latimea imaginii *sablon_matrice*
- *linie* – linia tabloului *imag_matrice* de unde se va calcula corelatia dintre cele 2 imagini
- *coloana* – coloana tabloului *imag_matrice* de unde se va calcula corelatia dintre cele 2 imagini
- *f_med – intensitatea* medie a pixelilor a imaginii *imag_matrice*
- s_med – intensitatea medie a pixelilor a imaginii *sablon_matrice*
- *deviatie_mat* – deviatia standard a imaginii *imag_matrice*
- *deviatie_sablon* – deviatia standard a imaginii *sablon_matrice*

Functia calculeaza corelatia dintre portiunea imaginii *imag_matrice* incepand cu coordonatele (linie,coloana) cu imaginea *sablon_matrice*, dupa formula:

$$corr(S, f_I) = \frac{1}{n} \sum_{(i,j)\in S} \frac{1}{\sigma_{f_I}\sigma_S} \left(f_I(i,j) - \overline{f_I}\right) \left(S(i,j) - \bar{S}\right), unde$$

```
float corelatie(pixel**imag_matrice,pixel**sablon_matrice,int inaltime,int latime,int linie,
                int coloana,float f_med,float s_med,float deviatie_mat,float deviatie_sablon)
{
    int i,j;
    float S=0;

    for(i=0;i<inaltime;i++)
    for(j=0;j<latime;j++)
        S=S+ (imag_matrice[linie+i][coloana+j].r-f_med) * (sablon_matrice[i][j].r-s_med);


    S=S/deviatie_mat;
    S=S/deviatie_sablon;
    S=S/(latime*inaltime);

    return S;

}
```

```
typedef struct
{
    int l;
    int c;
    float corr;
    int imagine;
}fereastra;
```

Am definit o structura ce retine linia,coloana, corelatia si imaginea (cifra reprezentata) pentru o anumita fereastra

template_matching_sablon

Functia va acea ca parametri:

- **imag* – sir de caractere care transmite calea unei imagini
- ***imag_matrice* – tabloul bidimensional de tip pixel in care este memorata imaginea *imag*
- **sablon* – sir de caractere care transmite calea unei imagini
- *nr_sablon* – numar natural
- *prag* – numar real
- **D* – tablou unidimensional de tip *fereastra*
- *n* – numarul de elemente al tabloului **D*

Functia primeste ca parametru o imagine *imag*, si transformata in forma sa de matrice, precum si un sablon *sablon*, nr_sablon care reprezinta un numar natural reprezentat in imaginea *sablon*, un numar real *prag* precum si un tablou unidimensional de tip *fereastra* in care vor fi stocate toate ferestrele pentru care corelatia dintre *sablon* si o portiune din imaginea *imag* este mai mare decat pragul respectiv.

Functia deschide fiecare imagine si obtine principalele caracteristici ale fiecare imagini, prin apelurile dimensiuni_imagine(imag, &inaltime_imag, &latime_imag, &pad_imag) si dimensiuni_imagine (sablon, &inaltime_sablon, &latime_sablon, &pad_sablon). Transforma imaginea sablon in grayscale grayscale_image(sablon,sablon_gray) , transforma noua imagine grayscale intr-un tablou bidimensional imagToMatrice(sablon_gray,&sablon_matrice,inaltime_sablon,latime_sablon,pad_sablon) , calculeaza intensitatea medie a pixelilor si deviatia standard a sablonului S_med=medie_pixeli(sablon_matrice, inaltime_sablon, latime_sablon, 0, 0), S_dev=deviatie_standard(sablon_matrice, inaltime_sablon, latime_sablon, S_med, 0, 0), iar pentru fiecare pozitie a tabloului bidimensional ***imag_matrice* calculeaza corelatia cu sablonului grayscale corr=corelatie(imag_matrice, sablon_matrice, inaltime_sablon, latime_sablon, i, j, f_med, S_med, f_dev, S_dev) si daca este mai mare decat *prag*, se retine in tabloul *D.

```
void template_matching_sablon(char *imag,pixel **imag_matrice,char *sablon,int nr_sablon,float prag, fereastra **D,int *n)
{

    FILE*finI=fopen(imag,"rb");
    FILE*finS=fopen(sablon,"rb");

    if(!finI)
    {
        printf("Nu s-a putut gasi imaginea %s \n",imag);
        return;
    }

    if(!finS)
    {
        printf("Nu s-a putut gasi imaginea %s \n",sablon);
        return;
    }

    int latime_imag,inaltime_imag;
    int latime_sablon,inaltime_sablon;
    int pad_imag,pad_sablon,i,j;
    float S_med,f_med,S_dev,f_dev,corr;

    pixel **sablon_matrice;

    dimensiuni_imagine(imag,&inaltime_imag,&latime_imag,&pad_imag);
    dimensiuni_imagine(sablon,&inaltime_sablon,&latime_sablon,&pad_sablon);
```

```
char sablon_gray[35];

strcpy(sablon_gray,sablon);
strcpy(sablon_gray+strlen(sablon_gray)-4,"_grey.bmp");
sablon_gray[strlen(sablon_gray)]='\0';

grayscale_image(sablon,sablon_gray); // transforma sablonul grayscale

imagToMatrice(sablon_gray,&sablon_matrice,inaltime_sablon,latime_sablon,pad_sablon); // transforma sablonul in matrice

S_med=medie_pixeli(sablon_matrice,inaltime_sablon,latime_sablon,0,0);
S_dev=deviatie_standard(sablon_matrice,inaltime_sablon,latime_sablon,S_med,0,0);

for(i=0;i<=inaltime_imag-inaltime_sablon;i++)
    for(j=0;j<latime_imag-latime_sablon;j++)
    {
        f_med=medie_pixeli(imag_matrice,inaltime_sablon,latime_sablon,i,j);
        f_dev=deviatie_standard(imag_matrice,inaltime_sablon,latime_sablon,f_med,i,j);

        corr=corelatie(imag_matrice,sablon_matrice,inaltime_sablon,latime_sablon,i,j,f_med,S_med,f_dev,S_dev);

       if(corr>=prag) // daca corelatia este mai mare decat pragul, se adauga in tabloul unidimensional *D
         {
               fereastra *aux=NULL;
               aux=(fereastra*)realloc(*D,sizeof(fereastra)*(*n+1));

               if(!aux)
               {
                   printf("Eroare la alocarea de memorie\n ");
                   for(i=0;i<*n;i++)
                       free( (D)[i] );
                   return ;
               }
               *D=aux;
               (*D)[*n].c=j;
               (*D)[*n].l=i;
               (*D)[*n].corr=corr;
               (*D)[*n].imagine=nr_sablon;

               (*n)++;
        }
    }

}
```

deseneaza_contur

Functia va avea ca parametri:

- ***imag_matrice*- tablou bidimensional ce memoreaza o imagine
- *inaltime_sablon*- inaltimea unui sablon (imaginile cu cifrele 0,1 ...9)
- *latime_sablon* – latimea unui sablon (imaginile cu cifrele 0,1…9)
- *culoare* – structura de tip pixel ce semnifica o culoare specificata printr-un triplet RGB
- *x* – structura de tip fereastra ce reprezinta o portiune a unei imagini

si deseneza in tabloul unidimensional *imag_matrice* conturul unui sablon cu latimea si inaltimea date ca parametri si de culoarea *culoare*, plecand de la coordonatele initiale din fereastra X.

```
void deseneaza_contur(pixel **imag_matrice,int inaltime_sablon,int latime_sablon,pixel culoare,fereastra x)
{
    int i;
    int l=x.l;
    int c=x.c;

    for(i=l;i<l+inaltime_sablon;i++)
    {
        imag_matrice[i][c]=culoare;
        imag_matrice[i][c+latime_sablon-1]=culoare;
    }

    for(i=c;i<c+latime_sablon;i++)
    {
        imag_matrice[l][i]=culoare;
        imag_matrice[l+inaltime_sablon-1][i]=culoare;
    }
}
```

marcheaza_cifre

Functia va avea ca parametri:

- ***imag_matrice* – tablou bidimensional ce memoreaza o imagine
- *inaltime* – inaltimea unui sablon
- *latime* – latimea unui sablon
- **D* – tablou unidimensional structura de tip pixel in care sunt retinute ferestrele
- *n* – numarul de elemente al tabloului *D

si pentru fiecare fereastra, va decide cu ce culoare va fi desenat conturul.

```
void marcheaza_cifre(pixel **imag_matrice,int inaltime,int latime, fereastra*D,int n)
{
    int i;
    pixel culoare;
    for(i=0;i<n;i++)
    {

        switch(D[i].imagine)
        {
            case 0: culoare.r=255; culoare.g=0;   culoare.b=0; break;
            case 1: culoare.r=255; culoare.g=255; culoare.b=0; break;
            case 2: culoare.r=0;   culoare.g=255; culoare.b=0; break;
            case 3: culoare.r=0;   culoare.g=255; culoare.b=255; break;
            case 4: culoare.r=255; culoare.g=0;   culoare.b=255; break;
            case 5: culoare.r=0;   culoare.g=0;   culoare.b=255; break;
            case 6: culoare.r=192; culoare.g=192; culoare.b=192; break;
            case 7: culoare.r=255; culoare.g=140; culoare.b=0; break;
            case 8: culoare.r=128; culoare.g=0;   culoare.b=128; break;
            case 9: culoare.r=128; culoare.g=0;   culoare.b=0; break;
            break;
        }
    deseneaza_contur(imag_matrice,inaltime,latime,culoare,D[i]);
    }
}
```

min max

```
int min(int x,int y)
{
    if(x<y)
        return x;
    return y;
}

int max(int x,int y)
{
    if(x>y)
        return x;
    return y;
}
```

Functii care returneaza cel mai mic sau cel mai mare element dintre 2 numere date ca parametru

intersect_ferestre

Functia va avea ca parametri:

- *x* - variabila de tip fereastra
- *y* – variabila de tip fereastra
- *inaltime_sablon* – inaltimea unui sablon
- *latime_sablon* – latimea unui sablon

si va calcula aria de interesectie dintre doua ferestre. Functia calculeaza coordonatele celor 4 colturi ale fiecarei ferestre si decide daca cele 2 se intersecteaza ,iar in caz afirmativ, va retruna aria de intersectie.

```
int intersect_ferestre(fereastra x,fereastra y,int inaltime_sablon,int latime_sablon)
{
   punct StUp1,StDw1,DrUp1,DrDw1;
   punct StUp2,StDw2,DrUp2,DrDw2;

    StUp1.x=x.l;
    StUp1.y=x.c;

    DrDw1.x=StUp1.x+inaltime_sablon-1;
    DrDw1.y=StUp1.y+latime_sablon-1;

    DrUp1.x=StUp1.x;
    DrUp1.y=StUp1.y+latime_sablon-1;

    StDw1.x=StUp1.x+inaltime_sablon-1;
    StDw1.y=StUp1.y;

    StUp2.x=y.l;
    StUp2.y=y.c;

    DrDw2.x=StUp2.x+inaltime_sablon-1;
    DrDw2.y=StUp2.y+latime_sablon-1;

    DrUp2.x=StUp2.x;
    DrUp2.y=StUp2.y+latime_sablon-1;

    StDw2.x=StUp2.x+inaltime_sablon-1;
    StDw2.y=StUp2.y;

   int ok=0;
   int a,b;
```

```
    if( StUp2.x>=StUp1.x && StUp2.x <=DrDw1.x && StUp2.y>=StUp1.y && StUp2.y <=DrDw1.y )
        ok=1;

    if( DrUp2.x>=StUp1.x && DrUp2.x <=DrDw1.x && DrUp2.y>=StUp1.y && DrUp2.y <=DrDw1.y )
        ok=1;

    if( StDw2.x>=StUp1.x && StDw2.x <=DrDw1.x && StDw2.y>=StUp1.y && StDw2.y <=DrDw1.y )
        ok=1;

     if( DrDw2.x>=StUp1.x && DrDw2.x <=DrDw1.x && DrDw2.y>=StUp1.y && DrDw2.y <=DrDw1.y )
        ok=1;


    if( StUp1.x>=StUp2.x && StUp1.x <=DrDw2.x && StUp1.y>=StUp2.y && StUp1.y <=DrDw2.y )
        ok=1;

    if( DrUp1.x>=StUp2.x && DrUp1.x <=DrDw2.x && DrUp1.y>=StUp2.y && DrUp1.y <=DrDw2.y )
        ok=1;

    if( StDw1.x>=StUp2.x && StDw1.x <=DrDw2.x && StDw1.y>=StUp2.y && StDw1.y <=DrDw2.y )
        ok=1;

     if( DrDw1.x>=StUp2.x && DrDw1.x <=DrDw2.x && DrDw1.y>=StUp2.y && DrDw1.y <=DrDw2.y )
        ok=1;
    if(ok==0)
        return 0;

    a=min(DrDw1.x,DrDw2.x)-max(StUp1.x,StUp2.x)+1;
    b=min(DrDw1.y,DrDw2.y)-max(StUp1.y,StUp2.y)+1;

    return a*b;
}
```

suprapunere

- *x* – variabila de tip fereastra
- *y* – variabila de tip fereastra
- *inaltime* – inaltime unui sablon
- *latime* – latimea unui sablon

si va returna suprapunerea spatiala a 2 ferestre, data de formula:

$$suprapunere(d_i, d_j) = \frac{aria(d_i \cap d_j)}{aria(d_i \cup d_j)} = \frac{aria(d_i \cap d_j)}{aria(d_i) + aria(d_j) - aria(d_i \cap d_j)}$$

eliminare

Functia va avea ca parametri:

- **D* – tablou unidimensional structura de tip pixel in care sunt retinute ferestrele
- *n* – numarul de elemente al tabloului **D*
- poz – numar natural

si va elimina elementul de pe pozitia *poz*.

```
void eliminare(fereastra *D,int n,int poz)
{
    int i;
    for(i=poz;i<n-1;i++)
        D[i]=D[i+1];
}
```

elim_non_maxime

Functia va avea ca parametri:

- **D* – tablou unidimensional structura de tip pixel in care sunt retinute ferestrele
- *n* – numarul de elemente
- *inaltime* – inaltimea unui sablon
- *latime* – latimea unui sablon

si compara cate 2 ferestre, iar in cazul in care suprapunerea spatiala este mai mare de 0.2, se elimina cea cu corelatia mai mica.

```
void elim_non_maxime(fereastra *D,int *n,int inaltime,int latime)
{
    int i,j;
    float d;
    for(i=0;i<*n-1;i++)
        for(j=i+1;j<*n;j++)

        {
            d=suprapunere(D[i],D[j],inaltime,latime);
                if(d>0.2)
                    {
                        eliminare(D,*n,j);
                        (*n)--;
                        j--;
                    }
        }
}
```

sortare_corelatii

Functia avea ca parametri:

- **D* - tablou unidimensional structura de tip pixel in care sunt retinute ferestrele
- *n* – numarul de elemente

si sorteaza tabloul descrescator in functie de corelatii

```
int cmpdescresc(const void* a, const void *b)
{
    fereastra x=*(fereastra*)a;
    fereastra y=*(fereastra*)b;

    if(x.corr < y.corr)
        return 1;
    if(x.corr == y.corr)
        return 0;
    return -1;
}

void sortare_corelatii(fereastra *D,int n)
{
    qsort(D,n,sizeof(fereastra),cmpdescresc);
}
```

In cea de-a doua parte a main-ului, are loc operatia de template matching dintre o imagine si diferite sabloane. Intai se deschide fisierul *imagini.txt* care contine numele imaginii, numele imaginii dupa ce aceasta a fost criptata, precum si numele fisierului ce contine cheia secreta. Se creeaza numele noii imagini in grayscale (strcpy (imagineI_gray, imagineI), strcpy(imagineI_gray + strlen (imagineI) -4, "_gray.bmp "))si se transforma in grayscale (grayscale_image(imagineI, imagineI_gray)). Se obtin dimensiunile imaginii dimensiuni_imagine(imagineI_gray, &inaltime, &latime, &pad)) dupa care ambele imagini (initiala si grayscale) se transforma in matrice: imagToMatrice(imagineI_gray, &imagineI_matrice_gray, inaltime, latime, pad) si imagToMatrice(imagineI, &imagineI_matrice, inaltime, latime, pad). Pentru fiecare din cele 10 sabloane are loc operatia de template matching: se citeste si deschide fiecare imagine si apeleaza functia de template matching dintre imaginea grayscale si un sablon template_matching_sablon(imagineI_gray, imagineI_matrice_gray, cifra_sablon, i, prag, &D, &n) . Are loc operatia de sortare a corelatiilor (sortare_corelatii(D,n)) si eliminare a non-maximelor (elim_non_maxime(D, &n, inaltime_sablon, latime_sablon)), se marcheaza pe matricea imaginii initiale conturul fiecarei ferestre ramase (marcheaza_cifre(imagineI_matrice, inaltime_sablon, latime_sablon, D, n))si are loc salvarea in memoria externa a noii imagini (matriceToimagine(imagineI, "template_final.bmp", imagineI_matrice, inaltime, latime, pad)), cu denumirea template_final.bmp. In final, are loc eliberarea de memorie.