Facultatea de Matematică și Informatică

Universitatea din București

PROIECT

PROGRAMARE PROCEDURALĂ

Cozma Laura-Elena

2019

Cuprins

Modulul de criptare și decriptare

Una din structurile folosite atât în partea de criptare, cât și în cea de template matching este cea denumită **Pixel**.

```
typedef struct
{
    unsigned char r,g,b;
}Pixel;
```

Câmpurile r, g și b corespund celor 3 canale de coloare R(red), G(green) și B(blue).

Funcția XORSHIFT32

```
void XORSHIFT32(unsigned int **R,unsigned int n,unsigned int R0)
{
    //functie care genereaza numere pseudo-aleatoare XORSHIFT32
    (*R)=(unsigned int *)malloc((n+1)*sizeof(unsigned int));
    unsigned int i;
    (*R)[0]=R0;
    for(i=1;i<=n;i++)
    {
        R0=R0^R0<<13;
        R0=R0^R0>>17;
        R0=R0^R0<<5;
        (*R)[i]=R0;
    }
}
```

Parametrii funcției:

- *Vectorul R*, al cărui spațiu este alocat în interiorul funcției. Acesta va reține numerele pseudo-aleatoare generate cu ajutorul algoritmului xorshift32. Acesta va avea dimensiunea n+1, pe poziția 0 aflându-se valoarea inițială R0 (seed-ul), iar pe următoarele n poziții numerele produse în interiorul funcției.
- *Numărul întreg fără semn n* dă dimensiunea vectorului.
- *Numărul întreg fără semn R0* reprezintă valoarea inițială de la care se pleacă în generare.

Funcția dimensiune_imagine

```
void dimensiune_imagine(char *caleImagine,unsigned int *latime,unsigned int *lungime)
{
    //functie care determina lungimea si latimea unei imagini
    FILE *f;
    f=fopen(caleImagine,"rb");

    if(f==NULL)
    {
        printf("Eroare deschidere fisier pentru aflarea dimensiunii imaginii");
        return ;
    }

    fseek(f,18,0);
    //latimea se afla incepand cu octetul 18 in headerul imaginii si este un numar nat
    fread(&(*latime),sizeof(unsigned int),1,f);
    //lungimea se afla incepand cu octetul 22 in headerul imaginii si este un numar na
    fread(&(*lungime),sizeof(unsigned int),1,f);

    fclose(f);
}
```

Funcția va fi folosită atât în prima parte a proiectului, cât și în cea de-a doua pentru a determina dimensiunile imaginilor.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii ale cărei dimensiuni urmează a fi calculate.
- *Numărul întreg fără semn latime* (parametru de ieșire) furnizează lățimea imaginii transmise ca parametru.
- *Numărul întreg fără semn lungime* (parametru de ieșire) furnizează lungimea imaginii transmise ca parametru.

Lungimea și lățimea imaginii exprimate în pixeli sunt specificate în header printr-o valoare întreagă fără semn. Lățimea este memorată începând cu octelul al 18-lea, iar lungimea de la octetul cu numătul de ordine 22. De aceea, cu ajutorul apelului funcției fseek(f,18,0), ne poziționăm pe al 18-lea octet din header. Apoi se citesc succesiv două blocuri a câte 4 octeți, ce vor reprezenta lățimea, respectiv lungimea imaginii.

Funcția header

```
void header(char * caleImagine,unsigned char **h)
{
    //functie care copiaza headerul unei imagini intr-un vector
    FILE *f;
    f = fopen ( caleImagine, "rb");
    if(f==NULL)
    {
        printf("Eroare deschidere fisier");
        return ;
    }

    (*h)=(unsigned char *)malloc(55*sizeof(unsigned char));

    if((*h)==NULL)
    {
        printf("Eroare alocare spatiu pentru header");
        return ;
    }
    unsigned char x;
    int i;
    for(i=0;i<54;i++)
    {
        fread(&x,sizeof(unsigned char),1,f);
        (*h)[i]=x;
    }

    fclose(f);
}
```

Această funcție furnizează headerul unei imagini.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii al cărui header va fi copiat
- *Vectorul de tip unsigned char *h* va reține headerul imaginii.

După deschiderea imaginii în modul „rb”, se va citi pe rând din aceasta cate un octet de 54 de ori, iar fiecare va fi copiat în vectorul h.

Funcția liniarizare

```
void liniarizare(char *caleImagine,Pixel **v)
{
    //functia care copiaza pixelii unei imagini intr-un vector
    FILE *f;
    f=fopen(caleImagine,"rb");

    if(f==NULL)
    {
        printf("\nEroare deschidere fisier la liniarizare\n");
        return ;
    }

    unsigned int lungime_img,latime_img,padding;
    dimensiune_imagine(caleImagine,&latime_img,&lungime_img);

    printf("Dimensiunea imaginii in pixeli inainte de liniarizare este (latime x inaltime): %u x %u\n",latime_img, lungime_img);

    if(latime_img %4 == 0)                      //calculam paddingul
        padding = 0;                            //daca latimea e multiplu de 4, atunci si 3*latimea e multiplu de 4
    else padding = 4-(3*latime_img)%4;

    (*v)=(Pixel *)malloc(lungime_img*latime_img*sizeof(Pixel));   //alocam spatiu pentru vectorul corespunzator imaginii liniarizat

    if((*v)==NULL)
    {
        printf("\nEroare alocare spatiu pentru vectorul liniarizat\n");
        return ;
    }

    fseek(f,54,0);                              //dezactiv headerul
    int i,j;
    Pixel p;
    for(i=0 ; i<lungime_img ; i++)
    {
        for(j=0 ; j<latime_img ; j++)           //parcurgem imaginea tip matrice
        {
            fread(&p.b,sizeof(unsigned char),1,f);
            fread(&p.g,sizeof(unsigned char),1,f);
            fread(&p.r,sizeof(unsigned char),1,f);

            (*v)[(lungime_img-1-i)*latime_img+j]=p;
        }

        fseek(f,padding,1);                     //sarim peste padding

    }

    fclose(f);

}
```

Funcția furnizează un vector ce va conține pixelii imaginii parcurse de la stânga la dreapta începând de la colțul din stânga sus și până la cel din dreapta jos.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii ai cărei pixeli vor fi copiați
- *Vectorul de tip Pixel *v* va reține pixelii imaginii, cu excepția celor din header.

Se deschide imaginea în modul „rb”. Poziționăm pointerul f de tip FILE * dincolo de header cu ajutorul apelului funcției fseek(f,54,0). Determinăm dimensiunile imaginii(lungime_img și latime_mg) prin intermediul apelului dimensiune_imagine (caleImagine, &latime_img, &lungime_img) și aflăm paddingul imaginii ce va fi reținut în variabila padding.

```
for(i=0 ; i<lungime_img ; i++)
   {
      for(j=0 ; j<latime_img ; j++)
      {
         fread(&p.b,sizeof(unsigned char),1,f);
         fread(&p.g,sizeof(unsigned char),1,f);
         fread(&p.r,sizeof(unsigned char),1,f);

         (*v)[(lungime_img-1-i)*latime_img+j]=p;
      }
      fseek(f,padding,1);                  //sarim peste padding

   }
```

Cu ajutorul celor două foruri parcugem imaginea linie cu linie de la stânga spre dreapta. Cum citirea imaginii începe din stanga jos, prima linie citită trebuie să fie copiată pe ultimele latime_img poziții în vector (la final), a doua pe penultimele poziții, etc. Formula găsită pentru un pixel citit de la poziția (i,j) este (lungime_img-1-i)*latime_img+j.

Funcția deliniarizare

```
void deliniarizare(unsigned char *header_imagine,char *caleImagine, Pixel *v)
{
    FILE *f;
    f=fopen(caleImagine,"wb+");
    if(f==NULL)
    {
        printf("\nEroare deschidere fisier la deliniarizare\n");
        return ;
    }

    int i,j;
    //scriem in fisierul tip bmp headerul imaginii trimis ca parametru
    for(i=0;i<54;i++)
    {
        fwrite(&header_imagine[i],sizeof(unsigned char),1,f);
    }
    fflush(f);                      //golim bufferul inainte unei citiri intrucat s-a efectuat o scriere

    unsigned int lungime_img,latime_img,padding;
    dimensiune_imagine(caleImagine,&latime_img,&lungime_img);

    printf("Dimensiunea imaginii in pixeli inainte de deliniarizare este (latime x inaltime): %u x %u\n",latime_img, lungime_img);

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


    Pixel p;
    for(i=(lungime_img-1)*latime_img ; i>=0 ; i=i-latime_img)
    {
        for(j=0;j<latime_img;j++)
        {
            p=v[i+j];                           //scriem pixelii in mod corespunzator in imagine
            fwrite(&p.b,sizeof(unsigned char),1,f);
            fwrite(&p.g,sizeof(unsigned char),1,f);
            fwrite(&p.r,sizeof(unsigned char),1,f);

        }
      unsigned char x=0;
      for(j=0;j<padding;j++)
            fwrite(&x,sizeof(unsigned char),1,f);       //scriem paddingul imaginii
    }

    fclose(f);
}
```

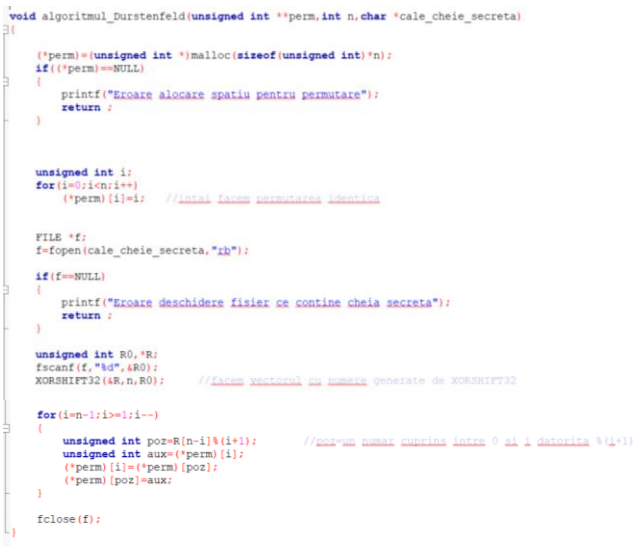
Reprezintă simetricul funcției ce liniarizează un vector.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii ce urmează să fie construită cu ajutorul vectorului liniarizat
- *Vectorul de tip Pixel *v* reține pixelii imaginii, cu excepția celor din header.
- *Vectorul header_imagine* reține octeții din header.

Mai întâi se copiază headerul imaginii ce se construiește în fișierul te tip bmp. Asemănătoare funcției liniarizare, parcurgerea vectorului v se va face pe „bucăți” de lungimea unei linii din imagine, de la coadă către început. Fiecare porțiune e parcursă și copiată pixel cu pixel în imaginea finală. Nu trebuie uitat paddingul pe care îl punem sub forma unor octeți cu valoarea 0.

Funcția algoritmul_Durstenfeld



```
void algoritmul_Durstenfeld(unsigned int **perm,int n,char *cale_cheie_secreta)
{

    (*perm)=(unsigned int *)malloc(sizeof(unsigned int)*n);
    if((*perm)==NULL)
    {
        printf("Eroare alocare spatiu pentru permutare");
        return ;
    }


    unsigned int i;
    for(i=0;i<n;i++)
        (*perm)[i]=i;   //intai facem permutarea identica


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

    if(f==NULL)
    {
        printf("Eroare deschidere fisier ce contine cheia secreta");
        return ;
    }

    unsigned int R0,*R;
    fscanf(f,"%d",&R0);
    XORSHIFT32(&R,n,R0);     //facem vectorul cu numere generate de XORSHIFT32

    for(i=n-1;i>=1;i--)
    {
        unsigned int poz=R[n-i]%(i+1);       //pozeaza un numar cuprins intre 0 si i datorita %(i+1)
        unsigned int aux=(*perm)[i];
        (*perm)[i]=(*perm)[poz];
        (*perm)[poz]=aux;
    }

    fclose(f);
}
```

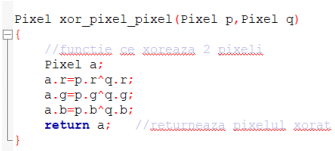
Funcția generează o permutare aleatoare perm.

Parametrii funcției:

- *Şirul de caractere cale_cheie_secreta*, ce reprezintă calea fişierului care conţine cheia secretă pentru generarea numerelor pseudo-aleatoare.
- *Vectorul *perm* va conține permutarea generată.
- *Numărul n* e numărul de elemente ale permutării

Se alocă spațiu pentru permutare. Se construiește permutarea identică (*perm)[i]=i. Parcurg vectorul perm de la poziția n-1 până la poziția 1 și pentru fiecare indice generez un număr poz plecând de la valorile vectorului R. Pentru poziția n-1 folosesc numărul pseudo-aleator R[1] %(n) ce va avea valori între 0 și n-1, poziției n-2 îi corespunde numărul R[2], etc. Valoarea aflată la indicele poz determinat este interschimbată cu valoarea de pe poziția i, conform algoritmului lui Durstenfeld, obținându-se astfel o permutare aleatoare.

Funcția xor_pixel_pixel



```
Pixel xor_pixel_pixel(Pixel p,Pixel q)
{
    //functia ce xoreaza 2 pixeli
    Pixel a;
    a.r=p.r^q.r;
    a.g=p.g^q.g;
    a.b=p.b^q.b;
    return a;   //returneaza pixelul xorat
}
```

Funcția realizează operația xor între doi pixeli. Aceasta se face canal cu canal, fiecare culoare fiind xorată cu cea corespunzătoare din celălalt pixel.

Parametrii funcției:

- *Două variabile de tip pixel* ce vor fi xorate.

Funcția xor_pixel_numar

```
Pixel xor_pixel_numar(Pixel p,unsigned int nr)
{
    //functie ce xoreaza un pixel cu un numar
    Pixel a;
    unsigned char *octet;
    octet=(unsigned char *)&nr;
    a.b=(p.b)^(*octet); //(*octet)contine valoarea aflata la adresa octet
    octet++;            //trecem la adresa urmatorului octet
    a.g=p.g^(*octet);
    octet++;
    a.r=p.r^(*octet);
    return a;
}
```

Funcția realizează operația xor între un număr și un pixel. Aceasta se face începând cu cel mai puțin semnificativ octet al numărului ce va fi grupat împreună cu p.b, canalul corespunzător culorii blue. Octeții numărului sunt parcurși cu ajutorul unui pointer octet de tip unsigned char *.

- octet=(unsigned char *)&nr – octet reține adresa celui mai puțin semnificativ octet din nr
- (*octet) - reține valoarea de la adresa celui mai puțin semnificativ octet din nr
- octet ++ - trece la următorul octet/următoarea adresă
- Funcția returnează pixelul rezultat în urma operației xor.

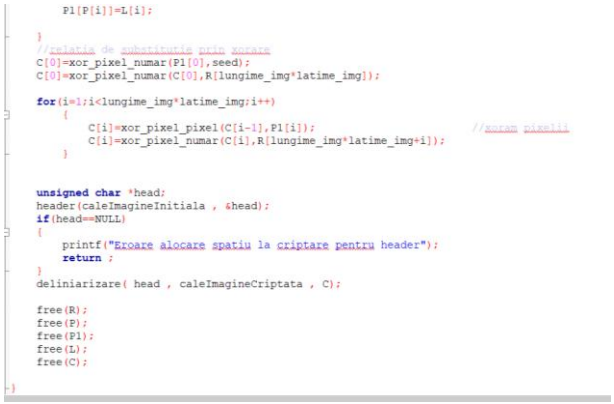
Parametrii funcției:

- *O variabilă de tip Pixel p*
- *Un număr natural nr*

Funcția criptare

```
void criptare(char *caleImagineInitiala,char *caleImagineCriptata,char *cheieSecreta)
{
    FILE *fkey;

    fkey=fopen(cheieSecreta,"rb");

    if(fkey==NULL)
    {
        printf("Eroare deschidere fisier in functia criptare pentru cheia secreta");
        return ;
    }

    unsigned int R0,seed;
    fscanf(fkey,"%u %u",&R0,&seed);
    fclose(fkey);

    unsigned int lungime_img,latime_img;
    dimensiune_imagine(caleImagineInitiala,&latime_img,&lungime_img);

    printf("Dimensiunea imaginii in pixeli inainte de criptare este (latime x inaltime): %u x %u\n",latime_img, lungime_img);

    unsigned int *R,*P,i;
    Pixel *L,*C,*P1;
    //generarea numerelor pseudo-aleatoare
    XORSHIFT32(&R,2*lungime_img*latime_img,R0);
    if(R==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru R");
        return ;
    }
```

```
    P1=(Pixel *)malloc((lungime_img*latime_img+1)*sizeof(Pixel));
    if(P1==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru P1");
        return ;
    }

    C=(Pixel *)malloc((lungime_img*latime_img+1)*sizeof(Pixel));
    if(C==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru C");
        return ;
    }
    //liniarizarea imaginii de criptat
    liniarizare(caleImagineInitiala,&L);
    if(L==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru vectorul liniarizat");
        return ;
    }
    //generarea permutarii aleatoare
    algoritmul_Durstenfeld(&P,lungime_img*latime_img,cheieSecreta);
    if(P==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru vectorul generat de algoritmul Durstenfeld");
        return ;
    }
    //permutarea pixelilor imaginii liniarizate conform permutarii aleatoare
    for(i=0;i<lungime_img*latime_img;i++)
    {
```



```
        P1[P[i]]=L[i];

    }
    //relatia de substitutie prin xorare
    C[0]=xor_pixel_numar(P1[0],seed);
    C[0]=xor_pixel_numar(C[0],R[lungime_img*latime_img]);

    for(i=1;i<lungime_img*latime_img;i++)
        {
            C[i]=xor_pixel_pixel(C[i-1],P1[i]);                    //xoram pixelii
            C[i]=xor_pixel_numar(C[i],R[lungime_img*latime_img+i]);
        }


    unsigned char *head;
    header(caleImagineInitiala , &head);
    if(head==NULL)
    {
        printf("Eroare alocare spatiu la criptare pentru header");
        return ;
    }
    deliniarizare( head , caleImagineCriptata , C);

    free(R);
    free(P);
    free(P1);
    free(L);
    free(C);

}
```

Funcția criptează o imagine și o transmite ca parametru.

Parametrii funcției:

- *Șirul de caractere caleImagineInitiala*, ce reprezintă calea/denumirea imaginii ce urmează să fie criptată.
- *Șirul de caractere caleImagineCriptata*, ce reprezintă calea/denumirea imaginii ce este criptată.
- *Șirul de caractere cale_cheie_secreta*, ce reprezintă calea fișierului care conține cheia secretă pentru generarea numerelor pseudo-aleatoare și seed-ul pentru xorarea pixelilor.

Cu ajutorul pointerului la fișier *fkey se deschide fișierul ce conține valoarea secretă inițială pentru algoritmul xorshift32 și seed-ul cu ajutorul căruia se va începe modificarea valorilor pixelilor. Se generează 2 *lungime_img*latime_img numere în vectorul R. Se generează permutarea aleatoare cu primele lungime_img*latime_img numere din R. Se liniarizează imaginea transmisă ca parametru în vectorul L, după care se formează un nou vector P1 cu elementele lui L permutate conform lui P după formula P1[P[i]]=L[i]. În vectorul C se pun valorile lui P1 modificate cu ajutorul formulei de recurență descrise în funcție. Vectorul C se deliniarizează, formându-se astfel imaginea criptată.

Funcția permutare_inversa

Funcția calculează permutarea inversă q permutării p cu n elemente.

```
void permutare_inversa(unsigned int *p,unsigned int **q,int n)
{
    //functie care calculeaza inversa unei permutari p date ca parametru cu n elemente
    (*q)=(unsigned int *)malloc(n*sizeof(unsigned int));
    if((*q)==NULL)
    {
        printf("Eroare alocare spatiu la permutarea inversa");
        return ;
    }

    int i;
    for(i=0;i<n;i++)
        (*q)[p[i]]=i;
}
```

Parametrii funcției:

- *Vectorul *p* ce va fi permutat

- *Vectorul *q* permutat
- *n*, numărul de elemente

Funcția decriptare

```c
void decriptare(char *caleImagineCriptata,char *caleImagineInitiala,char *cheieSecreta)
{
    //functie care realizeaza decriptarea imaginii data prin calea caleImagineCriptata
    FILE *fkey;

    fkey=fopen(cheieSecreta,"rb");
    if(fkey==NULL)
    {
        printf("Eroare deschidere fisier ce contine cheia secreta");
        return ;
    }
    unsigned int R0,seed;
    fscanf(fkey,"%u %u",&R0,&seed);
    fclose(fkey);


    unsigned int lungime_img,latime_img;
    dimensiune_imagine(caleImagineCriptata,&latime_img,&lungime_img);

    printf("Dimensiunea imaginii in pixeli inainte de decriptare este (latime x inaltime): %u x %u\n",latime_img, lungime_img);

    unsigned int *R,*P,i,*Inversa;
    Pixel *Init,*C,*C1;
    //generarea numerelor pseudo-aleatoare
    XORSHIFT32(&R,2*lungime_img*latime_img,R0);

    algoritmul_Durstenfeld(&P,lungime_img*latime_img,cheieSecreta);
    permutare_inversa(P,&Inversa,lungime_img*latime_img);

    C1=(Pixel *)malloc((lungime_img*latime_img+1)*sizeof(Pixel));
    C=(Pixel *)malloc((lungime_img*latime_img+1)*sizeof(Pixel));
    //liniarizarea imaginii criptate
    liniarizare(caleImagineCriptata,&C);
     //relatia de substitutie aplicata pe imaginea criptata
    C1[0]=xor_pixel_numar(C[0],seed);
    C1[0]=xor_pixel_numar(C1[0],R[lungime_img*latime_img]);
    for(i=1;i<lungime_img*latime_img;i++)
        {
            C1[i]=xor_pixel_pixel(C[i-1],C[i]);
            C1[i]=xor_pixel_numar(C1[i],R[lungime_img*latime_img+i]);
        }

    Init=(Pixel *)malloc(lungime_img*latime_img*sizeof(Pixel));

    for(i=0;i<lungime_img*latime_img;i++)
    {
        Init[Inversa[i]]=C1[i];

    }


    unsigned char *head;
    header(caleImagineCriptata , &head);                //aflam headerul

    deliniarizare( head , caleImagineInitiala , Init);   //transformam vectorul in imagine

    free(R);
    free(P);
    free(Inversa);
    free(Init);
    free(C);
    free(C1);
}
```

Funcția decriptează o imagine și o transmite ca parametru.

Parametrii funcției:

- *Șirul de caractere caleImagineCriptata*, ce reprezintă calea/denumirea imaginii ce este criptată.
- *Șirul de caractere caleImagineInitiala*, ce reprezintă calea/denumirea imaginii ce urmează să fie decriptată.
- *Șirul de caractere cale_cheie_secreta*, ce reprezintă calea fișierului care conține cheia secretă pentru generarea numerelor pseudo-aleatoare și seed-ul pentru xorarea pixelilor.

Cu ajutorul pointerului la fișier *fkey se deschide fișierul ce conține valoarea secretă inițială pentru algoritmul xorshift32 și seed-ul cu ajutorul căruia se va începe modificarea valorilor pixelilor. Se generează 2 *lungime_img*latime_img numere în vectorul R. Se generează permutarea aleatoare P cu primele lungime_img*latime_img numere din R. Se realizează permutarea inveră permutării P, numită Inversa. Se liniarizează imaginea transmisă ca parametru

în vectorul C, după care se formează un nou vector C1 cu elementele lui C modificate cu ajutorul formulei de recurență descrise în funcție. Se formează vectorul Init cu elementele permutate conform lui Inversa după formula Init[Inversa[i]]=C1[i]. Vectorul Init se deliniarizează, formându-se astfel imaginea decriptată.

Funcția chi_patrat

```
void chi_patrat(char *caleImagine)
{
    //testul chi patrat
    Pixel *v;
    liniarizare(caleImagine,&v);
    if(v==NULL)
    {
        printf("Nu s-a reusit alocarea spatiului pentru liniarizare in testul chi patrat");
        return ;
    }

    unsigned int *frecv;
    frecv=(unsigned int *)calloc(256,sizeof(unsigned int));

    unsigned int lungime_img,latime_img;
    dimensiune_imagine(caleImagine,&latime_img,&lungime_img);

    double f_bar,s_r=0,s_b=0,s_g=0;
    f_bar=(lungime_img*latime_img)/256.0;


    unsigned int i;
    //chi patrat pentru canalul r
    for(i=0;i<lungime_img*latime_img;i++)
        frecv[v[i].r]++;    //calculam frecventa fiecarei culori

    for(i=0;i<256;i++)
    {
        s_r+=(float)((frecv[i]-f_bar)*(frecv[i]-f_bar))/f_bar;
        frecv[i]=0;
    }

    for(i=0;i<lungime_img*latime_img;i++)
        frecv[v[i].g]++;

    for(i=0;i<256;i++)
    {
        s_g+=(float)((frecv[i]-f_bar)*(frecv[i]-f_bar))/f_bar;
        frecv[i]=0;    //initializam si cu 0 ca sa nu facem un nou for
    }

    for(i=0;i<lungime_img*latime_img;i++)
        frecv[v[i].b]++;

    for(i=0;i<256;i++)
    {
        s_b+=(float)((frecv[i]-f_bar)*(frecv[i]-f_bar))/f_bar;
        frecv[i]=0;
    }

    printf("\nValorile testului chi-patrat sunt: %.3f  %.3f  %.3f\n",s_r,s_g,s_b);

    free(v);
    free(frecv);
}
```

Aceasta furnizează valorile testului chi-patrat aplicat asupra unei imagini.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea unei imagini

Imaginea se liniarizează în vectorul *v. Vectorul *frecv are 256 de elemente și e inițializat cu 0 cu ajutorul funcției calloc. Se calculează succesiv frecvența fiecărei culori a fiecărui canal, calculându-se de asemenea și valorile medii s_r+=(float)((frecv[i]-f_bar)*(frecv[i]-f_bar)) /f_bar, s_g+ = (float)((frecv[i]-f_bar)*(frecv[i]-f_bar))/ f_bar, s_b+ = (float) ((frecv[i] - f_bar)*(frecv[i]-f_bar))/ f_bar. Funcția afișează în consolă aceste 3 valori.

Modulul de recunoaștere de pattern-uri într-o imagine

Funcția imagine2matrice

```
void imagine2matrice(char *caleImagine,Pixel ***a,unsigned int *nr_linii,unsigned int *nr_coloane)
{
    //transforma o imagine intr-o matrice, sarind peste header
    FILE *f;
    f=fopen(caleImagine,"rb");

    if(f==NULL)
    {
        printf("Eroare deschidere fisier");
        return ;
    }

    unsigned int padding;
    int i,j;
    Pixel x;
    dimensiune_imagine(caleImagine,nr_coloane,nr_linii);

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

    (*a)=(Pixel **)malloc((*nr_linii)*sizeof(Pixel *));

    if((*a)==NULL)
    {
        printf("Nu s-a reusit alocarea pentru matrice");
        return ;
    }

    fseek(f,54,0);

    for(i=(*nr_linii)-1; i>=0;i--)
    {
        //cum citirea imaginii incepe cu coltul din stanga jos, asa ca parcurgem matricea invers
        (*a)[i]=(Pixel *)malloc((*nr_coloane)*sizeof(Pixel));
        if((*a)[i]==NULL)
        {
              printf("Nu s-a reusit alocarea pentru liniile matricei");
              return ;
        }

        for(j=0;j<(*nr_coloane);j++)
        {
            fread(&x,sizeof(Pixel),1,f);
            (*a)[i][j].b=x.r;
            (*a)[i][j].g=x.g;
            (*a)[i][j].r=x.b;
        }
        fseek(f,padding,1);      //sarim peste padding
    }

    fclose(f);
}
```

Transformă o matrice într-o imagine.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea unei imagini.
- *Matricea de tip Pixel **a*, ce va salva imaginea sub forma unui tablou bidimensional.
- *Numerele nr_linii, nr_coloane*, reprezentând dimensiunile tabloului bidimensional.

Se calculează dimensiunile imaginii, padding-ul acesteia. Fseek(f,54,0) sare peste header. Se parcurge matricea începând cu ultima linie, fiecare linie fiind parcursă de la stânga la dreapta. Se citesc rând pe rând pixelii imaginii, fiecare fiind pus pe pozițiile (i,j) din matrice, canalul b în locul canalului r, iar canalul r în locul lui b, datorită citirii inverse a culorilor unui pixel. După fiecare linie se sare peste padding.

Funcția matrice2imagine

```
void matrice2imagine(char *caleImagineInitiala,char *caleImagineFinala,Pixel **a,unsigned int nr_linii,unsigned int nr_coloane)
{
    //transforma o matrice intr-o imagine
    FILE *fin,*fout;
    fin=fopen(caleImagineInitiala,"rb");

    if(fin==NULL)
    {
        printf("Eroare deschidere fisier la transformarea unei matrice intr-o imagine");
        return ;
    }

    fout=fopen(caleImagineFinala,"wb");

    if(fout==NULL)
    {
        printf("Eroare deschidere fisier la transformarea unei matrice intr-o imagine");
        return ;
    }

    unsigned int padding;
    int i,j;
    unsigned char elem,pad=0;

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

    for(i=0;i<54;i++)
    {
        //copiem headerul
        fread(&elem,1,1,fin);
        fwrite(&elem,1,1,fout);
    }
     for(i=nr_linii-1; i>=0;i--)
    {
        for(j=0;j<nr_coloane;j++)
        {
            fwrite(&a[i][j].b,1,1,fout);
            fwrite(&a[i][j].g,1,1,fout);
            fwrite(&a[i][j].r,1,1,fout);

        }
        for(j=0;j<padding;j++)
           fwrite(&pad,1,1,fout);   //scriem si paddingul
    }

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

Este simetricul funcției imagine2matrice.

Parametrii funcției:

- *Șirul de caractere caleImagineInitiala*, ce reprezintă calea/denumirea unei imagini ce a fost transformată anterior într-o matrice. Aceasta este folosită pentru a copia headerul imaginii în cea finală.
- *Șirul de caractere caleImagineFinala*, ce reprezintă calea/denumirea imaginii construite din matrice.
- *Matricea de tip Pixel **a*, care are salvată imaginea sub forma unui tablou bidimensional.
- *Numerele nr_linii, nr_coloane*, reprezentând dimensiunile tabloului bidimensional.

Se copiază headerul imaginii inițiale în imaginea finală. Se parcurg liniile de la final către început și se scrie pixel cu pixel în imagine(deoarece atunci când se citește o imagine se începe cu ultima linie, colțul din stânga jos). Se scrie și paddingul la finalul fiecărei linii cu valori de 0.

Funcția grayscale_image

Transformă o imagine color nume_fisier_sursa în nuanțe de gri, furnizând imaginea finală prin parametrul nume_fisier_destinatie.

```
void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie)
{
    FILE *fin, *fout;
    unsigned int 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");
         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 a imaginii ce urmeaza sa fie transformata in grayscale e (latime x inaltime): %u x %u\

    //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);
}
```

Funcția medie_intensitati_s

```
float medie_intensitati_s(Pixel **s,int nr_linii,int nr_coloane)
{
    //calculeaza media intensitatilor culorilor unui sablon
    float s_bar=0;
    int i,j;
    for(i=0;i<nr_linii;i++)
        for(j=0;j<nr_coloane;j++)
           s_bar+=s[i][j].r;

    s_bar=s_bar/(float)(nr_linii*nr_coloane);
    return s_bar;
}
```

Aceasta calculează media intensităților canalului r din pixelii unui șablon.

Parametrii funcției:

- *Matricea de tip Pixel **s*, care are salvat un șablon.
- *Numerele nr_linii*, *nr_coloane*, reprezentând dimensiunile tabloului bidimensional.

Tabloul este parcurs element cu element și se adaugă lui s_bar valoarea canalului s[i][j].r. s_bar se împarte la nr_linii*nr_coloane pentru a determina media intensităților.

Funcția deviatia_standard_s

```
float deviatia_standard_s(Pixel **s,int nr_linii,int nr_coloane,float s_bar)
{
    //s_bar reprezinta media intensitatilor sablonului s
    float deviatie=0;
    int i,j;
    for(i=0 ; i<nr_linii ; i++)
    {
        for(j=0 ; j<nr_coloane ; j++)
            deviatie+=((s[i][j].r-s_bar)*(s[i][j].r-s_bar));
    }

    deviatie=deviatie/(float)(nr_linii*nr_coloane-1);

    deviatie=sqrt(deviatie);

    return deviatie;
}
```

Aceasta returnează deviația standard a unui șablon.

Parametrii funcției:

- *Matricea de tip Pixel* ***s*, care are salvat un șablon.
- *Numerele nr_linii, nr_coloane*, reprezentând dimensiunile tabloului bidimensional.
- *Numărul real s_bar*, reprezentând media intensităților pixelilor șablonului.

Tabloul este parcurs element cu element și se calculează valoarea deviației conform formulei.

Funcția medie_intensitati_i

```
float medie_intensitati_i(Pixel **img,unsigned int nr_linii,unsigned int nr_coloane,int x,int y)
{
    //calculam media intensitatii imaginii i pe o portiune de 11*15 pornind de la pozitia x y

    float fi_bar=0;
    int i,j;
    for(i=x;i<x+nr_linii;i++)
    {
        for(j=y;j<y+nr_coloane;j++)
            fi_bar+=img[i][j].r;
    }
    fi_bar=fi_bar/(float)(nr_linii*nr_coloane);

    return fi_bar;
}
```

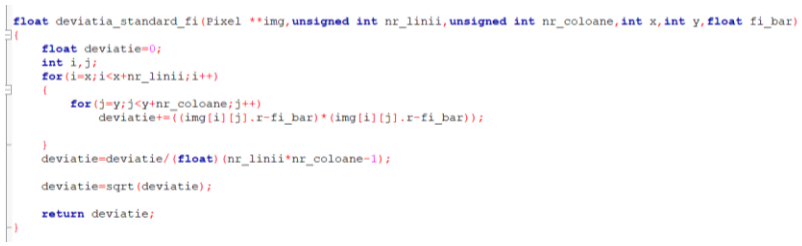
Aceasta calculează media intensităților canalului r din pixelii unei ferestre din imagine.

Parametrii funcției:

- *Matricea de tip Pixel* ***img*, care are salvată o imagine
- *Numerele nr_linii, nr_coloane*, reprezentând dimensiunile ferestrei asupra căreia calculăm media intensităților.
- *Numerele x și y,* reprezentând coordonatele colțului din stânga sus al ferestrei.

Valorile ce trebuie adăugate mediei fi_bar sunt parcurse începând cu linia x până la linia x+nr_linii-1, respectiv coloana y, până la coloana y+nr_coloane-1.

Funcția deviatia_standard_fi



```
float deviatia_standard_fi(Pixel **img,unsigned int nr_linii,unsigned int nr_coloane,int x,int y,float fi_bar)
{
    float deviatie=0;
    int i,j;
    for(i=x;i<x+nr_linii;i++)
    {
        for(j=y;j<y+nr_coloane;j++)
            deviatie+=((img[i][j].r-fi_bar)*(img[i][j].r-fi_bar));

    }
    deviatie=deviatie/(float)(nr_linii*nr_coloane-1);

    deviatie=sqrt(deviatie);

    return deviatie;
}
```

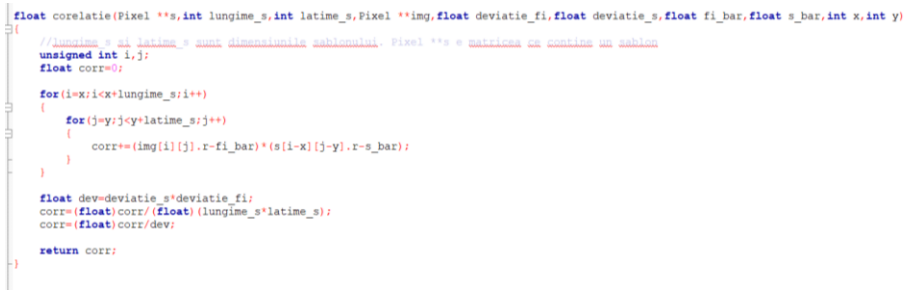
Aceasta returnează deviația standard a unei ferestre determinate de coordonatele colțului din stânga sus și dimensiunile laturilor.

Parametrii funcției:

- *Matricea de tip Pixel **img*, care are salvată o imagine.
- *Numerele nr_linii*, *nr_coloane*, reprezentând dimensiunile ferestrei asupra căreia calculăm media intensităților.
- *Numerele x și y,* reprezentând coordonatele colțului din stânga sus al ferestrei.
- *Numărul real fi_bar*, desemnând media intensităților pixelilor ferestrei.

Fereastra este parcursă element cu element și se calculează valoarea deviației conform formulei.

Funcția corelatie



```
float corelatie(Pixel **s,int lungime_s,int latime_s,Pixel **img,float deviatie_fi,float deviatie_s,float fi_bar,float s_bar,int x,int y)
{
    //lungime_s si latime_s sunt dimensiunile sablonului. Pixel **s e matricea ce contine un sablon
    unsigned int i,j;
    float corr=0;

    for(i=x;i<x+lungime_s;i++)
    {
        for(j=y;j<y+latime_s;j++)
        {
            corr+=(img[i][j].r-fi_bar)*(s[i-x][j-y].r-s_bar);
        }
    }

    float dev=deviatie_s*deviatie_fi;
    corr=(float)corr/(float)(lungime_s*latime_s);
    corr=(float)corr/dev;

    return corr;
}
```

Funcția returnează un număr real, reprezentând valoarea corelației dintre un șablon și o fereastră dintr-o imagine, calculată conform formulei.

Parametrii funcției:

- *Matricea de tip Pixel **s*, care are salvat un șablon.
- *Numerele latime_s, lungime_s*, reprezentând dimensiunile șablonului.
- *Matricea de tip Pixel **img*, care are salvată o imagine.
- *Numărul real deviatie_fi,* conținând deviația standard a ferestrei date ca parametru.

- *Numărul real deviatie_s,* conținând deviația standard a sablonului.
- *Numărul real fi_bar*, desemnând media intensităților pixelilor ferestrei.
- *Numărul real s_bar*, reprezentând media intensităților pixelilor șablonului.
- *Numerele x și y,* reprezentând coordonatele colțului din stânga sus al ferestrei.

Structura Identificare

```
typedef struct
{
    int lin_st_sus , col_st_sus , lin_dr_jos , col_dr_jos , cifra;
    float corelatie;
}Identificare;
```

Structura va reține caracteristicile unor ferestre dintr-o imagine:

- Coordonatele colțului din stânga sus: lin_st_sus, col_st_sus
- Coordonatele colțului din dreapta jos: lin_dr_jos, col_dr_jos
- Cifra identificată în fereastră, care ne ajută la determinarea culorii pe care o va avea chenarul ce va încadra cifra
- Corelația dintre fereastră și șablonul ce reprezintă cifra.

Funcția contur

```
void contur(Pixel **img,unsigned int lungime_s,unsigned int latime_s,int x,int y,Pixel c)
{
  //functie care realizeaza conturul unei ferestre
  int i;
  for(i=y;i<y+latime_s;i++)
      img[x][i]=c;   //linia paralela cu axa ox de sus

  for(i=x+1;i<x+lungime_s;i++)
    {
        img[i][y]=c;
        img[i][y+latime_s-1]=c;   //laturile laterale paralele cu o
    }

   for(i=y;i<y+latime_s;i++)
     img[x+lungime_s-1][i]=c;   //linia paralela cu axa ox de jos
}
```

Funcția desenează conturul unei ferestre cu o anumită culoare.

Parametrii funcției:

- *Matricea de tip Pixel **img*, care are salvată o imagine.
- *Numerele latime_s, lungime_s*, reprezentând dimensiunile șablonului (în același timp sunt și dimensiunile ferestrei).
- *Numerele x și y,* reprezentând coordonatele colțului din stânga sus al ferestrei.

```
for(i=y;i<y+latime_s;i++)
    img[x][i]=c;  - colorează în culoarea c latura de sus paralelă cu axa OX
for(i=y;i<y+latime_s;i++)
   img[x+lungime_s-1][i]=c;  - colorează în culoarea c latura de jos paralelă cu axa OX
for(i=x+1;i<x+lungime_s;i++)
  {
    img[i][y]=c;
    img[i][y+latime_s-1]=c;   - colorează în culoarea c laturile paralele cu OY
  }
```

Colorarea propriu-zisă se realizează în momentul în care matricea este transformată în imagine, iar fereastra va avea pe margine pixeli de o anumită culoare.

Funcția template_matching_pentru_un_sablon

```
void template_matching_pentru_un_sablon(char *caleImagine,char *caleSablon,float prag,Identificare **d,int *nr)
{


    float deviatie_fi,deviatie_s,fi_bar, s_bar;

    Pixel **img,**s;
    unsigned int lungime_img,latime_img,lungime_s,latime_s;

    imagine2matrice(caleImagine,&img,&lungime_img,&latime_img);
    if(img==NULL)
    {
        printf("Eroare la alocarea spatiului pentru matricea imaginii test.bmp in template_matching_pentru_un_sablon");
        return ;
    }
    imagine2matrice(caleSablon,&s,&lungime_s,&latime_s);
    if(s==NULL)
    {
        printf("Eroare la alocarea spatiului pentru matricea sablonului in template_matching_pentru_un_sablon");
        return ;
    }

    s_bar=medie_intensitati_s(s,lungime_s,latime_s);
    deviatie_s=deviatia_standard_s(s,lungime_s,latime_s,s_bar);//calculam media intensitatilor si deviatia standard a sa


    int i,j;
    for(i=0;i<=lungime_img-lungime_s;i++)
    {
        for(j=0;j<=latime_img-latime_s;j++)
            {
            fi_bar=medie_intensitati_i(img,lungime_s,latime_s,i,j);  //calculam media intensitatilor si deviatia standard
            deviatie_fi=deviatia_standard_fi(img,lungime_s,latime_s,i,j,fi_bar);
            float corr=corelatie(s,lungime_s,latime_s,img,deviatie_fi,deviatie_s,fi_bar,s_bar,i,j);
            //   printf("lin = %d  col = %d  f_med = %f  s_med = %f  dev_sablon = %f  dev_matrice = %f corr = %f \n",i,j,
            if(corr>=prag)
             {
                //daca corelatia ferestrei depaseste pragul, adaugam datele despre ea in vectorul d
                Identificare *aux;
                (*nr)++;
                aux=(Identificare *)realloc((*d),(*nr)*sizeof(Identificare));

                if(aux==NULL)
                {
                    printf("Eroare realocare");
                    return ;
                }

                (*d)=aux;
                (*d)[(*nr)-1].cifra=caleSablon[1]-'0';
                (*d)[(*nr)-1].corelatie=corr;
                (*d)[(*nr)-1].lin_st_sus=i;
                (*d)[(*nr)-1].col_st_sus=j;
                (*d)[(*nr)-1].lin_dr_jos=i+lungime_s-1;
                (*d)[(*nr)-1].col_dr_jos=j+latime_s-1;

             }

            }
    }
    //dezalocam memoria
    for(i=0;i<lungime_s;i++)
        free(s[i]);
    free(s);
    for(i=0;i<lungime_img;i++)
        free(img[i]);
    free(img);
}
```

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii asupra căreia se aplică operația de template matching.
- *Șirul de caractere caleSablon*, ce reprezintă calea/denumirea unui șablon ce se suprapune cu imaginea în fiecare punct al acesteia, pentru a se calcula corelația.
- *Numărul real prag,* fiind pragul impus pentru corelație.
- *Vectorul de tip Identificare *d,* ce va reține ferestrele ale căror corelații depășesc pragul.
- *Numărul nr,*dimensiunea vectorului d.

Imaginea se transformă în matricea **img, de dimensiune lungime_img și latime_img. Șablonul se transformă în matricea **s, de dimensiune lungime_s și latime_s. Se calculează media intensităților pixelilor șablonului și deviația standard a acestuia.

Se parcurge imaginea de la linia 0 și coloana 0, însă ultima linie este lungime_img-lungime_s, iar ultima coloană este latime_img-latime_s, întrucât atunci când suprapunem șablonul

peste imagine, nu trebuie să ieșim în exteriorul acesteia. Pe fiecare punct parcurs al imaginii îl considerăm colțul din stânga sus al unei ferestre pentru care calculăm media intensităților, deviația și corelația. Dacă corelația depășește pragul, atunci adăugăm datele despre fereastră în vectorul *d, pe care il redimensionăm și mărim nr (numărul de elemente ale vectorului).

Funcția cmpDescrescator_dupa_corelatie

```
int cmpDescrescator_dupa_corelatie(const void *a,const void *b)
{
    //functie de comparare
    Identificare va=*(Identificare *)a;
    Identificare vb=*(Identificare *)b;

    if(va.corelatie>vb.corelatie)return -1;
    else if(va.corelatie<vb.corelatie)return 1;
    return 0;
}
```

Este o funcție ce compară două elemente de tip identificare și stabilește ordinea lor dacă vectorul ar fi sortat descrescător după corelație. Return -1 înseamnă că elementul a se află la stânga lui b, return 0, au corelațiile egale și 1 altfel. Ea va fi folosită la apelul funcției qsort.

Funcția sortare_detecții

```
void sortare_detectii(Identificare *d,int nr_detectii)
{
    //functie ce sorteaza descrescator dupa corelatie vectorul d cu nr detectii elemente
    qsort(d,nr_detectii,sizeof(Identificare),cmpDescrescator_dupa_corelatie);
}
```

Sortează descrescător elementele vectorului d cu nr_detectii elemente, după corelație.

Funcțiile minim și maxim

```
int minim(int a,int b)
{
    if(a<b)return a;
    return b;
}


int maxim(int a,int b)
{
    if(a>b)return a;
    return b;
}
```

Returnează minimul, respectiv maximul dintre două numere. Acestea vor fi utilizate în funcția suprapunere.

Funcția suprapunere

```
float suprapunere(Identificare a,Identificare b)
{
    int x1=maxim(a.lin_st_sus,b.lin_st_sus);//coltul stanga sus al intersectiei
    int y1=maxim(a.col_st_sus,b.col_st_sus);

    int x2=minim(a.lin_dr_jos,b.lin_dr_jos); //coltul din dreapta jos al intersectiei
    int y2=minim(a.col_dr_jos,b.col_dr_jos);

    float arie_intersectie,suprapus;
    if(x1>x2 || y1>y2)arie_intersectie=0;      //nu se intersecteaza
    else arie_intersectie=(x2-x1+1)*(y2-y1+1);

    if(arie_intersectie!=0)
    {
        // aria_sablon=165;
        suprapus=(float)arie_intersectie/(165+165-arie_intersectie);
        return suprapus;
    }
    return -1;   //nu se suprapun

}
```

Parametrii funcției:

- *Elementele a și b de tip Identificare, între care se calculează suprapunerea spațială.*

X1= linia colțului din stânga sus al posibilei intersecții
Y1= coloana colțului din stânga sus al posibilei intersecții
X2= linia colțului din dreapta al posibilei intersecții
Y2= coloana colțului din dreapta jos al posibilei intersecții

Dacă linia colțului din stânga sus e mai mare decât cea a colțului din dreapta jos sau coloana colțului din stânga sus e mai mare decât cea a colțului din dreapta jos, atunci intersecția celor două ferestre e vidă, iar funcția returnează -1. Altfel, putem calcula aria intersecției și suprapunerea spațială a ferestrelor.

Funcția eliminare

```
void eliminare(Identificare *v,int n,int poz)
{
    //functie care elimina un element
    int i;
    for(i=poz+1;i<n;i++)
        v[i-1]=v[i];
}
```

Elimină elementul de pe poziția poz din vectorul de tip Identificare *v cu n elemente.

Funcția suprimarea_non_maximelor

```
void suprimarea_non_maximelor(Identificare *d,int *nr_detectii)
{
    //elimina ferestrele a caror corelatie se suprapun
    int i,j;
    for(i=0;i<(*nr_detectii)-1;i++)
    {
        for(j=i+1;j<(*nr_detectii);j++)
            {
                float suprapus=suprapunere(d[i],d[j]);
                if(suprapus>0.2)
                {
                    eliminare(d,(*nr_detectii),j);
                    j--;
                    (*nr_detectii)--;
                }
            }
    }
}
```

Parametrii funcției:

- *Vectorul de tip Identificare *d,* ce va reține ferestrele ale căror corelații depășesc pragul.
- *Numărul nr_detectii,*dimensiunea vectorului d.

Cu ajutorul a 2 for-uri determinăm toate perechile ordonate de ferestre, le calculam suprapunerea spațială, iar dacă aceasta depașește valoarea 0.2, eliminăm fereastra ce are corelația mai mică.

Funcția suprimarea_non_maximelor

```
void colorare_imagine(Pixel **initial,int lungime_s,int latime_s,Identificare *detectie,int nr_detectii)
{
    int i;

    sortare_detectii(detectie,nr_detectii);
    suprimarea_non_maximelor(detectie,&nr_detectii);


   for(i=0;i<nr_detectii;i++)
    {
        Pixel c;
        int cif=detectie[i].cifra;   //alegem culoarea in functie de cifra
        switch(cif)
        {
        case 0:c.r=255;   c.g=0;  c.b=0;
            break;
        case 1:c.r=255;   c.g=255;  c.b=0;
            break;
        case 2:c.r=0;   c.g=255;  c.b=0;
            break;
        case 3:c.r=0;   c.g=255;  c.b=255;
            break;
        case 4:c.r=255;   c.g=0;  c.b=255;
            break;
        case 5:c.r=0;   c.g=0;  c.b=255;
            break;
        case 6:c.r=192;   c.g=192;  c.b=192;
            break;
        case 7:c.r=255;   c.g=140;  c.b=0;
            break;
        case 8:c.r=128;   c.g=0;  c.b=128;
            break;
        case 9:c.r=128;   c.g=0;  c.b=0;
            break;
        }

        contur(initial,lungime_s,latime_s,detectie[i].lin_st_sus,detectie[i].col_st_sus,c);
    }
}
```

Funcția colorează conturul tuturor ferestrelor ce se află în vectorul *detective după suprimarea non-maximelor.

Parametrii funcției:

- *Matricea de tip Pixel **Initial*, care are salvată o imagine.
- *Numerele latime_s*, *lungime_s*, reprezentând dimensiunile șablonului (în același timp sunt și dimensiunile ferestrei).
- *Vectorul de tip Identificare *detectie*, ce va reține ferestrele ale căror corelații depășesc pragul.
- *Numărul nr_detectii*,dimensiunea vectorului d.

Se sortează vectorul *detecție descrescător după corelație după care se elimină suprapunerile. Contururile ferestrelor rămase în vectorul *detectie sunt colorate în culoarea corespunzătoare cifrei cu ajutorul switch-ului (fiecărei cifre îi corespunde o culoare).

Funcția construire_denumire

```
void construire_denumire(int cif,char **denumireGrayscale)
{
    //contruim denumirea sablonanelor in grayscale in functie de cifra pe care o reprezinta.
    //are formatul "gcifra.bmp"
    (*denumireGrayscale)=NULL;
    (*denumireGrayscale)=(char *)malloc(8*sizeof(char));
    if((*denumireGrayscale)==NULL)
    {
        printf("Eroare alocare spatiu");
        return ;
    }
    char cifra[2];
    cifra[0]=cif+'0';
    cifra[1]='\0';
    strcpy((*denumireGrayscale),"g");
    strcat((*denumireGrayscale),cifra);
    strcat((*denumireGrayscale),".bmp");
    (*denumireGrayscale)[6]='\0';


}
```

Funcția furnizează denumirea șablonului cifrei cif după ce este transformat în grayscale.

Parametrii funcției:

- *Un număr cif*, reprezentând o cifră.
- *Șirul de caractere *denumireGrayscale,* denumirea finală a șablonului

Fiecare șablon după ce e transformat în grayscale va avea denumirea „gcifra.bmp”, cifra fiind cea desenată în șablon. Alocăm spațiu pentru șirul de caractere, pe prima poziție punem litera ‘g’, apoi cifra, iar pe următoarele 5 poziții „.bmp\0”.

Funcția template_matching

```
void template_matching(char *caleImagine,float prag)
{
    Identificare *detectie;
    detectie=NULL;
    int nr_detectii=0;


    grayscale_image(caleImagine,"test_grayscale.bmp");

    FILE *fin;
    fin=fopen("caleSabloane.fin","r");

    if(fin==NULL)
    {
        printf("Eroare deschidere imagine la template matching");
        return ;
    }
    int i;

    char *denum_grayscale,*nume_sablon;
    nume_sablon=(char *)malloc(12*sizeof(char));
    if(nume_sablon==NULL)
    {
        printf("Eroare alocare spatiu pentru numele sabloanelor");
        return ;
    }
    for(i=0;i<=9;i++)
    {

        fscanf(fin,"%s",nume_sablon);
        construire_denumire(i,&denum_grayscale);
        grayscale_image(nume_sablon,denum_grayscale);
        template_matching_pentru_un_sablon("test_grayscale.bmp",denum_grayscale,prag,&detectie,&nr_detectii);
    }

    fclose(fin);

    Pixel **matrice_test;
    unsigned int nr_linii_test,nr_coloane_test;
    imagine2matrice(caleImagine,&matrice_test,&nr_linii_test,&nr_coloane_test);
    //coloram chenarele in culorile corespunzatoare
    colorare_imagine(matrice_test,15,11,detectie,nr_detectii); //15 si 11 sunt dimensiunile unui sablon
    matrice2imagine(caleImagine,"rezultat_template_matching.bmp",matrice_test,nr_linii_test,nr_coloane_test);
    free(matrice_test);
    free(detectie);
}
```

Funcția reunește toate cele 10 apeluri ale funcției template_matching_pentru_un_sablon și colorează contururile ferestrelor.

Parametrii funcției:

- *Șirul de caractere caleImagine*, ce reprezintă calea/denumirea imaginii asupra căreia se aplică operația de template matching.
- *Numărul real prag,* fiind pragul impus pentru corelație.

Se tranformă imaginea inițială în grayscale și i se atribuie denumirea „test_grayscale.bmp”. Parcurgem cu un for toate cifrele, iar la fiecare pas citim din fișierul „caleSabloane.fin” denumirea șablonului cifrei egale cu i(indicele). Construim denumirea șablonului în grayscale și transformăm

șablonul în grayscale. Realizăm operația de template matching între imaginea inițială grayscale și șablonul grayscale.

La ieșirea din for colorăm conturul identificărilor, iar matricea rezultată e transformată în imagine.

Funcția main

```
int main()
{
    char *nume_imagine,*nume_imagine_criptata,*nume_imagine_decriptata,*nume_fisier_cheie_secreta;
    nume_imagine=(char *)malloc(40*sizeof(char));
    if(nume_imagine==NULL)
    {
        printf("Eroare alocare");
        return 0;
    }
    nume_imagine_criptata=(char *)malloc(40*sizeof(char));
    if(nume_imagine_criptata==NULL)
    {
        printf("Eroare alocare");
        return 0;
    }
    nume_imagine_decriptata=(char *)malloc(40*sizeof(char));
    if(nume_imagine_decriptata==NULL)
    {
        printf("Eroare alocare");
        return 0;
    }
    nume_fisier_cheie_secreta=(char *)malloc(40*sizeof(char));
    if(nume_fisier_cheie_secreta==NULL)
    {
        printf("Eroare alocare");
        return 0;
    }

    FILE *f;
    f=fopen("denumire_imagini.fin","r");
    if(f==NULL)
    {
        printf("Eroare deschidere fisier");
        return 0;
    }
    fscanf(f,"%s%s%s%s",nume_imagine,nume_imagine_criptata,nume_imagine_decriptata,nume_fisier_cheie_secreta);
    criptare(nume_imagine,nume_imagine_criptata,nume_fisier_cheie_secreta);
    decriptare(nume_imagine_criptata,nume_imagine_decriptata,nume_fisier_cheie_secreta);
    chi_patrat(nume_imagine);
    chi_patrat(nume_imagine_criptata);
    template_matching(nume_imagine_decriptata,0.5);

    fclose(f);

    free(nume_imagine);
    free(nume_imagine_criptata);
    free(nume_imagine_decriptata);
    free(nume_fisier_cheie_secreta);
    return 0;
}
```

Se citesc din fișierul „denumire_imagini.fin” denumirea imaginii inițiale, denumirea imaginii criptate, denumirea imaginii decriptate, denumirea fișierului ce conține cheia secretă. Se criptează imaginea inițială, se decriptează, se afișează valorile testului chi_patrat atât pentru imaginea inițială, cât și pentru cea criptată, după care se execută funcția template_matching.