Documontație Proiect PP

**Modulul de criptare/decriptare**

# void incarca\_imagine(char \*caleImagine, unsigned int \*H, unsigned int \*W, unsigned char \*\*\*v, unsigned char \*\*header );

* funcția ia pixelii din imaginea primită în fișierul caleImagine și îi pune într-un vector
* caleImagine – calea imaginii inițiale
* H – va furniza înalțimea imaginii
* W- va furniza lățimea imaginii
* V - va furniza imaginea liniarizată
* Header - va furniza header-ul imaginii

# void descarca\_imagine(char \*caleImgExtern, unsigned int H, unsigned int W, unsigned char \*\*v, unsigned char \*header );

* funcția creaza o imagine nouă în fișierul caleImgExtern folosind imaginea liniarizată, header-ul, înălțimea, respective lățimea acesteia
* caleImgExtern – calea imaginii rezultate
* H – înalțimea imaginii
* W- lățimea imaginii
* v - imaginea liniarizată
* header -header-ul imaginii

# unsigned int xorshift(unsigned int \*seed);

* funcția returneaza un numar aleator pornind de la o valoare inițială seed trimisă ca parametru și care se modifica înăuntrul funcției
* funcția a fost scrisă pe baza codului din prezentare de la pagina 16

# void xorshift32(unsigned int seed, unsigned int \*\*v, unsigned int n);

* generează numere aleatoare pornind de la o valoare inițială pe care le pune in vectorul v
* seed – valoare inițială
* v – furnizeaza numerea aleatoare
* n – numarul de numere aleatoare din v

# void permutare(unsigned int n, unsigned int \*\*perm, unsigned int R0, unsigned int \*\*rand);

* furnizează o permutare aleatoare pe baza numerelor pseudo-aleatoare generate de xorshift32
* n - numarul de elemente
* perm – permutarea
* R0 - reprezintă valoare inițială de la care se generează numerele
* rand – conține numerele pseudo-aleatoare
* funcția a fost scrisă pe baza codului din prezentare de la pagina 17 și cu ajutorul cursurilor 10-12

# void permutare\_imagine(unsigned int n, unsigned int \*perm, unsigned char \*\*\*v);

* funcția permută pixelii din imagine conform permutării găsite
* n – numarul de pixeli
* perm -permutarea
* v – matricea linearizată, va furniza pixelii permutați

# unsigned char\* xor(unsigned char \*p1, unsigned char \*p2);

* returnează rezultatul operației xor dintre doi pixeli
* p1 – primul pixel
* p2 – al doilea pixel

# unsigned char\* xor\_int(unsigned char \*p, unsigned int x);

* returnează rezultatul operației xor dintre un pixel si un intreg
* p – pixelul
* x – intregul

# void xorare(unsigned char \*\*\*v,unsigned int n, unsigned int sv, unsigned int \*rand);

* funcția xorează pixelii din imagine conform reguli date pentru criptare
* v – matricea linearizată, va furniza pixelii xorați
* n – numarul de pixeli
* sv - *starting value,* face parte din cheia secreta
* rand – numerele aleatoare gasite
* funcția a fost scrisă pe baza regulii din prezentare de la pagina 18

# **void** criptare(**char** \*caleImagine, **char** \*caleImgExtern, **char** \*keyText);

* funcția criptează imaginea aflată în caleImagine și o furnizează in caleImgExtern
* caleImagine – calea imaginii inițiale
* caleImgExtern – calea imaginii criptate
* keyText – fișier text în care se gasește cheia secretă

# void permutare\_inversa(unsigned int n, unsigned int \*\*perm);

* funcția furnizează inversa unei permutări date ca parametru
* n – numarul de elemente
* perm - permutarea

# void xorare1(unsigned char \*\*\*v,unsigned int n, unsigned int sv, unsigned int \*rand);

* funcția xorează pixelii din imagine conform reguli date pentru decriptare
* v – matricea linearizată, va furniza pixelii xorați
* n – numarul de pixeli
* sv - *starting value,* face parte din cheia secreta
* rand – numerele aleatoare gasite

# **void** decriptare(**char** \*cale\_img\_initiala, **char** \*cale\_img\_criptata, **char** \*keyText);

* funcția decriptează imaginea aflată în cale\_img\_criptata și o furnizează in cale\_img\_initiala
* cale\_img\_initiala – calea imaginii inițiale
* cale\_img\_criptata – calea imaginii criptate
* keyText – fișier text în care se gasește cheia secretă

# double calcul(unsigned int \*f, double q);

* returneaza valoare testului chi patrat pe un canal de culoare
* f – frecvențele pentru fiecare valoare
* q - frecvenţa estimată teoretic a oricărei valori i

# **void** chi\_\_patrat(**char** \*cale\_imagine);

* funcția afișează pentru o imagine data testul chi patrat pentru fiecare canal de culoare
* cale\_imagine – calea imaginii

**Modulul de recunoașterea de pattern-uri într-o imagine**

# **void** grayscale(**char** \*cale\_sursa, **char** \*cale\_destinatie);

* funcția transforma imaginea inițială într-o imagine cu tonuri de gri
* cale\_sursa – imaginea inițială
* cale\_destinație – imaginea modificată

# void color(struct pixel \*\*a, unsigned int h, unsigned int w, unsigned int fi, unsigned int fj, struct pixel C);

* funcția colorează în matricea a fereastra f
* a – matricea de pixeli
* h – înălțimea șablonului
* w – lățimea șablonului
* fi – linia de pe care începe fereastra f
* fj – coloana de pe care începe fereastra f
* C – culoarea cu care colorăm fereastra

# void colorare(char \*imagine, unsigned int h, unsigned int w, unsigned int nv, struct detectii \*v);

* funcția colorează fiecare detecție găsită
* imagine – calea imaginii inițiale
* h – înălțimea șablonului
* w – lățimea șablonului
* nv – nr-ul de detecții găsite
* v – vectorul de detecții

# void adaugare(unsigned int \*n, struct detectii \*\*v, unsigned int i, unsigned int j, double corr, struct pixel C);

* adaugă o nouă detecție la vectorul v
* n – nr-ul de detecții
* v – vectorul de detecții
* i – linia de pe care începe fereastra f
* j – coloana de pe care începe fereastra f
* corr -corelația ferestrei
* C – culoarea cu care colorăm fereastra

# **void** template\_matching(**char** \*cale\_imagine, **char** \*cale\_sablon, **double** ps, **unsigned** \*nv, **struct** detectii \*\*v, **struct** pixel C);

* funcția gasește toate potrivirile dintre un șablon si o imagine data care trec de un prag dat
* cale\_imagine – calea imaginii inițiale
* cale\_sablon – calea sablonului
* ps - pragul
* n – nr-ul de detecții
* v – vectorul de detecții
* C – culoarea cu care colorăm fereastra

# **void** pattern\_recognition(**char** \*sursa, **unsigned int** \*n, **struct** detectii \*\*v);

* funcția detectează toate potrivirile cu sabloanele date
* sursa – imaginea inițială
* n – nr-ul de detecții
* v – vectorul de detecții

# int comp(const void \*a, const void \*b);

* funcție pentru sortarea cu qsort

# **double** suprapunere(**struct** detectii di, **struct** detectii dj)

* funcția returneaza valoarea suprapunerii dintre doua ferestre
* di – prima fereastră
* dj – a doua fereastră

# **void** non\_maxime(**unsigned int** \*n, **struct** detectii \*\*D)

* funcția elimină detecțiile suprapuse din imagine
* n – numarul de detecții
* D – vectorul de detecții