**Proiect la Programare Procedurală**

**Sava Daniel**

În acest proiect, se lucrează cu imagini color în formatul BMP (bitmap), care vor fi manipulate în limbajul C ca fișiere binare. Acest format nu comprimă imaginile, ci stochează 3 octeți per pixel pentru imaginile color.

Datorită modului în care este stocată imaginea, se va crea o structură ***Pixel***, care va conține cei 3 octeți a unui pixel, reprezentând cele 3 canale de culoare (R – roșu, G – verde, B – albastru). Intensitatea fiecărui canal de culoare este dată de o valoare naturală cuprinsă între 0 și 255 (deci datele vor fi de tip ***unsigned char***). Deoarece codarea unei imagini BMP într-un fișier binar respectă standardul little-endian, octeții corespunzători celor 3 canale de culoare R, G, B sunt memorați de la dreapta la stânga, adică în ordinea B, G, R.

Pentru o ușoară manipulare a imaginii, se va crea o structură ***Image***, care va memora date despre imagine: header-ul (vector de ***unsigned char***), conținutul (vector de ***Pixel***), lățimea, înălțimea și paddingul (toate de tip ***unsigned int***).

// Aici mai trebuie scris și despre structura imagine, dar asta intră la template-matching

**Modulul de criptare/ decriptare**

Funcția ***xorshift32*** generează numere întregi fără semn pe 32 de biți, cu un caracter pseudo aleator (numere având proprietăți statistice asemănătoare celor ale unei secvențe de numere perfect aleatoare, adică o secvență de numere pentru care probabilitatea de apariție a unei anumite valori este independentă de toate valorile generate anterior), plecând de la o valoare inițială, numită seed. Se va folosi generatorul propus de George Marsaglia în 2003 (https://en.wikipedia.org/wiki/Xorshift).

* Primul parametru este numărul de numere ce se doresc a fi generate
* Al doilea parametru este seed-ul
* Funcția returnează un vector alocat dinamic cu (n + 1)(seed-ul + n numere) numere pseudo aleatoare

Funcția ***loadImageIntoMemory*** încarcă o imagine BMP în memoria internă în formă liniarizată.

* Funcția are ca parametru calea imaginii BMP
* Funcția returnează o structură ***Image***, care conține date despre imagine

1. Imaginea BMP se va deschide ca fișier binar pentru citire (modul ***rb***)
2. Se obține header-ul, care ocupă primii 54 de octeți ai fișierului, conține informații despre formatul BMP, precum și informații despre dimensiunea imaginii, numărul de octeți utilizați pentru reprezentarea unui pixel etc.
3. Dimensiunea imaginii în pixeli este exprimată sub forma width×height, unde width reprezintă numărul de pixeli pe lățime (memorată pe patru octeți fără semn începând cu octetul al 18-lea din header), iar height reprezintă numărul de pixeli pe înălțime (memorată pe următorii 4 octeți fără semn, respectiv începând cu octetul al 22-lea din header)
4. Se calculează padding-ul, deoarece pentru rapiditatea procesării imaginilor la citire și scriere, imaginile în format BMP au proprietatea că fiecare linie are un număr de octeți care să fie multiplu de 4, lucru realizat prin adăugarea unor octeți de padding
5. Se citește conținutul imaginii începând cu colțul din stânga jos, datorită modului în care este stocată o imagine BMP, având grijă ca la finalul fiecărei linii să se sară peste octeții de padding

Funcția ***saveImageIntoFile*** salvează în memoria externă o imagine BMP stocată în formă liniarizată în memoria internă.

* Primul parametru este calea fișierului în care va fi salvată imaginea
* Al doilea parametru este structura care conține imaginea în formă liniarizată

1. Se deschide fișierul binar în care va fi salvată imaginea (modul ***wb***)
2. Se scrie header-ul
3. Se scrie conținutul imaginii începând de la final, datorită modului în care este stocată o imagine BMP, și având grijă ca la finalul fiecărei linii să se scrie și eventualii octeții de padding

Funcția ***durstenfeld*** generează o permutare aleatoare de dimensiune dată cu ajutorul algoritmului lui Durstenfeld (https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle) și a numerelor pseudo-aleatoare generate anterior cu ajutorul funției ***xorshift32***.

* Primul parametru este numărul de elemente al permutării
* Al doilea parametru este vectorul ce conține numerele pseudo-aleatoare
* Funcția returnează un vector alocat dinamic care conține permutarea obținută

Funcția ***shufflePixels*** permută pixelii unei imagini conform unei permutări date (în cazul de față, cea obținută în urma algoritmului lui Durstenfeld).

* Primul parametru este imaginea ai cărei pixeli se permută
* Al doilea parametru este vectorul ce conține permutarea

Funcția ***encryptImage*** aplică algoritmul de criptare.

* Primul parametru este calea imaginii ințiale
* Al doilea parametru este calea imaginii criptate
* Al treilea parametru este calea unui fișier text care conține cheia secretă

1. Se citesc valorile R0 și SV din fișierul text
2. Se încarcă imaginea BMP în memoria internă
3. Se generează o secvență 𝑅 = 𝑅1 ,𝑅2, … 𝑅2\*width\*height-1 de numere întregi aleatoare fără semn pe 32 de biți, folosind generatorul ***xorshift32*** inițializat cu valoarea nenulă 𝑅0
4. Se generează o permutare aleatoare 𝜎 de dimensiune width×height, folosind funcția ***dursenfeld***, care implementează algoritmul lui Durstenfeld și numerele pseudo-aleatoare 𝑅1,… 𝑅width\*height-1
5. Se permută prin intermediul funcției ***shufflePixels*** pixelii imaginii inițiale (P) conform permutării 𝜎, obținându-se o imagine intermediară (P’), folosing relația P’𝜎(k) = Pk
6. Imaginea criptată (𝐶) se obține aplicând asupra fiecărui pixel al imaginii intermediare (P’) următoarea relație de substituție:

Ck = SV ⨁ P0’ ⨁Rwidth\*height , k = 0

Ck-1 ⨁ Pk’ ⨁ Rwidth\*height+k , k = 1, 2, ... width\*height – 1

* Cu ⨁ se notează operația sau-exclusiv (XOR) între 2 octeți fără semn
* Pentru 2 pixeli P1 = (P1R, P1G, P1B) și P2 = (P2R, P2G, P2B), se notează cu P1 ⨁P2 pixelul (P1R ⨁ P2R, P1G ⨁ P2G, P1B ⨁ P2B)
* Pentru un pixel P = (PR, PG, PB) și un întreg fără semn X pe 32 de biți format din octeții (X3, X2,X1,X0), se va nota cu P ⨁ X pixelul (PR ⨁ X2, PG ⨁ X1, PB ⨁ X0). Pentru a accesa octeul i al lui X, se va folosi formula X ˃> (8 \* i)) & 0xFF

1. Se salvează imaginea în memoria externă

Funcţia ***inversePermutation*** calculează inversa unei permutări date.

* Primul parametru este numărul de elemente al permutării
* Al doilea parametru este permutarea
* Funcția returnează unn vector alocat dinamic ce connține inversa permutării date

Funcția ***decryptImage*** aplică algoritmul de decriptare.

1. Se citesc valorile R0 și SV din fișierul text
2. Se folosește funcția ***encryptImage*** pentru a cripta imaginea
3. Se încarcă imaginea BMP în memoria internă
4. Se generează o secvență 𝑅 = 𝑅1 ,𝑅2, … 𝑅2\*width\*height-1 de numere întregi aleatoare fără semn pe 32 de biți, folosind generatorul ***xorshift32*** inițializat cu valoarea nenulă 𝑅0
5. Se generează o permutare aleatoare 𝜎 de dimensiune width×height, folosind funcția ***dursenfeld***, care implementează algoritmul lui Durstenfeld și numerele pseudo-aleatoare 𝑅1,… 𝑅width\*height-1
6. Se calculează inversa permutării aleatoare, 𝜎-1
7. Se aplică asupra fiecărui pixel din imaginea criptată (C) inversa relației de substituție folosită în procesul de criptare:

Ck’ = SV ⨁ C0 ⨁Rwidth\*height , k = 0

Ck-1 ⨁ Ck ⨁ Rwidth\*height+k , k = 1, 2, ... width\*height – 1

1. Se permută prin intermediul funcției ***shufflePixels*** pixelii imaginii criptate (C’) conform permutării 𝜎-1, folosing relația D𝜎-1(k) = Ck’
2. Se salvează imaginea în memoria externă

Funcția ***printChiSquareTest*** afișează valorile testului χ2 pentru o imagine BMP pe fiecare canal de culoare (R, G, B), folosing formula.

* Funcția are ca parametru calea imaginii.