Tema 3. Image processing

- Responsabili:
  - Alexandru-Ionuț Mîndru
  - Alexandru-Gabriel Anica
  - Andreea Dutulescu
- Data publicării: **20.12.2021**
- Deadline: **26.01.2022 23:59**

Atentie! deadline-ul soft coincide cu cel hard! Prin urmare, nu se vor mai accepta submisii dupa data de 26.01.2020!
Actualizări:

- **08.01.2022** removed some cpplint warnings, updated ref Task3/5;
- **03.01.2022** adaugat checker;
- **20.12.2021** adaugat enunt;

Scopul temei:

- utilizarea structurilor de date;
- lucrul cu fisiere binare;
- alocarea dinamica a memoriei;
- abordarea cu pasi mici a unei probleme mai complexe;

Introducere

Procesarea de imagini se refera la aplicarea unor algoritmi specifici pe continutul unei imagini pentru a obtine anumite efecte (blur, sharpening, etc.) sau rezultate (face detection/recognition, etc.). In aceasta tema vom lucra cu unul dintre cele mai simple formate de imagini si anume formatul BMP.

O imagine BMP are următoarea structură:

- un **File Header** care are următoarele câmpuri:
  1. **signature** – 2 octeți - literele 'B' și 'M' în ASCII;
  2. **file size** – 4 octeți – dimensiunea întregului fișier;
  3. **reserved** – 4 octeți – nefolosit;
  4. **offset** – 4 octeți – offsetul de la începutul fișierului până la începutului bitmap-ului, adica al matricii de pixeli.

- un **Info Header** care poate avea structuri diferite, însă noi vom lucra cu cel care se numește **BITMAPINFOHEADER**. Are următoarele câmpuri:
    1. **size** – 4 octeți – dimensiunea Info Header-ului, care are o valoare fixă, 40;
    2. **width** – 4 octeți – lățimea matricii de pixeli (numărul de coloane);
    3. **height** – 4 octeți – înălțimea matricii de pixeli (numărul de rânduri);
    4. **planes** – 2 octeți – setat la valoarea fixă 1;
    5. **bit count** – 2 octeți – numărul de biți per pixel. În cazul nostru va avea mereu valoarea 24, adică reprezentăm fiecare pixel pe 3 octeți, adică cele 3 canale, RGB;
    6. **compression** – 4 octeți – tipul de compresie. Acest câmp va fi 0;
    7. **image size** – 4 octeți – se referă la dimensiunea matricii de pixeli, inclusiv padding-ul adăugat (vedeți mai jos);
    8. **x pixels per meter** – 4 octeți – se referă la rezoluția de printare. Pentru a simplifica puțin tema, veți seta acest câmp pe 0. Nu o să printăm imaginile.
    9. **y pixels per meter** – la fel ca mai sus;
    10. **colors used** – numărul de culori din paleta de culori. Aceasta este o secțiune care va lipsi din imaginile noastre BMP, deoarece ea se află în general imediat după **Info Header** însă doar pentru imaginile care au câmpul **bit count** mai mic sau egal cu 8. Prin urmare, câmpul va fi setat pe 0;
    11. **colors important** – numărul de culori importante. Va fi, de asemenea, setat pe 0, ceea ce înseamnă că toate culorile sunt importante.
- **BitMap**-ul, care este matricea de pixeli și care ocupă cea mai mare parte din fișier. Trei lucruri trebuiesc menționate despre aceasta:
    1. pixelii propriu-ziși se află într-o matrice de dimensiune **height x width**, însă ea poate avea o dimensiune mai mare de atât din cauza **paddingului**. Acest padding este adăugat la sfârșitul fiecărei linii astfel încat fiecare linie să înceapă de la o adresă (offset față de începutul fișierului) multiplu de 4. Mare atenție la citire, pentru că acest

padding trebuie **ignorat** (fseek). De asemenea, la scriere va trebui să puneți **explicit** valoarea 0 pe toți octeții de padding.

2. este **răsturnată**, ceea ce înseamnă că prima linie în matrice conține de fapt pixelii din extremitatea de jos a imaginii. Vedeți exemplul de mai jos;
3. canelele pentru fiecare pixel sunt în ordinea BGR (**B**lue **G**reen **R**ed).

Header-ele pe care le puteți folosi în implementare se află în scheletul de cod asociat temei.

Urmatiti cu foarte mare atentie exemplul de aici si incercati sa intelegeti cum e reprezentata o imagine BMP **inainte** de a incepe implementarea. Daca e ceva neclar, puteti intreba oricand pe forum.

Puteti folosi utilitare precum **xxd** sau **hexdump** pentru a putea vizualiza continutul unui fisier binar (in cazul nostru, a unei imagini bmp). Vedeti si raspunsurile de aici .

Luați în considerare cum veți stoca în memorie matricea imaginii!

Cerinte

Veti avea de rezolvat 5 task-uri obligatorii si un task bonus.

Exemple de imagini in format bmp si de input-uri pentru task-uri gasiti in directorul `input` din checker.

Cand scrieti imagini in format BMP, intre ultimul byte din header si byte-ul la care incepe matricea de pixeli (adica pana la byte-ul de offset) trebuie sa scrieti peste tot byte-ul 0.

Task 0 - Interactive Console

Se va implementa un parser al liniei de comandă pentru a putea lucra interactiv cu transformările descrise în cerințele de mai jos și pentru a putea vedea la fiecare pas output-ul rezultat. Programul scris va citi în continuu de la standard input și va putea primi următoarele comenzi.

- save <path>
- edit <path>
- insert <path> y x
- set draw_color R G B
- set line_width x

- draw line y1 x1 y2 x2
- draw rectangle y1 x1 width height
- draw triangle y1 x1 y2 x2 y3 x3
- fill y x
- quit

Se garantează faptul că atât operațiile cât și parametrii trimiși la input sunt corecte.

Task 1 - Basic Commands

Pentru acest task vom implementa primele doua operatii de baza necesare unui editor de imagini:

- edit <path> - Incarcă imaginea în memoria aplicației pentru a fi editată
- save <path> - Salveaza/suprascrie imaginea din edit mode in calea path.
- quit - dezaloca memoria si inchide programul

Daca operatia e de salvare atunci fisierul in care se va salva imaginea va avea numele si extensia: <name>.bmp

Exemplu:

Input: edit poza1.bmp

Input: save ./output/poza_editata.bmp

Va crea fisierul poza_editata.bmp in folderul output(existent).

output

├── poza_editata.bmp

Pentru lucrul cu structurile de date cititi mai multe detalii aici.

Pentru a lucra cu fisiere veti folosi urmatoarele metode:

fopen, **fwrite**, **fread**, **fclose**

Veti deschide fisierele bmp cu ajutorul functiei **fopen** utilizand modul de access "rb". Pentru fisierul de output veti utiliza modul de access "wb".

fwrite si **fread** au antetul functiei similare.

size_t fread(void *restrict ptr, size_t size, size_t nmemb, FILE *restrict stream);

size_t fwrite(const void *restrict ptr, size_t size, size_t nmemb, FILE *restrict stream);

Ambele functii intorc numarul de elemente citite, respectiv scrise.

- Primul parametru reprezinta buffer-ul unde se vor pune elementele citite/se vor lua elementele pentru a fi scrise in fisier.
- Al doilea parametru reprezinta dimensiunea elementului citit din fisier. Daca vrem sa citim din fisier un unsigned int atunci functia de fread va fi apelata in felul urmator fread(&x, sizeof(unsigned int), 1, fr).
- Al treilea parametru reprezinta numarul de elemente citite/scrise.
- Al patrulea parametru reprezinta fisierul.

Inainte sa inchideti programul asigurati-va ca ati inchis toate fisierele deschise cu **fopen**! Pentru a inchide un fisier deschis folositi **fclose** unde pasati pointer-ul structurii FILE deschise.

Task 2 - Overlap Images

Pentru acest task vom implementa următoarele doua operații:

- insert <path> y x - Inserează imaginea (toata sau cropped) de la path peste cea din modul edit.

y si x sunt doua unsgined int care pot avea valori intre 0 si UINT_MAX

Ele reprezinta coordonatele punctului stanga-jos, in matricea imaginii din edit mode, de la care se va introduce noua imagine.

Atentie la notarea coordonatelor

dim 1024 × 768

- 1024 de pixeli pe fiecare linie
- 768 de linii

In cazul y si x - coordonata y reprezinta pozitia de pe linia si x linia din matrice.

Task 3 - Draw Lines

Pentru acest task vom implementa următoarele cinci operații:

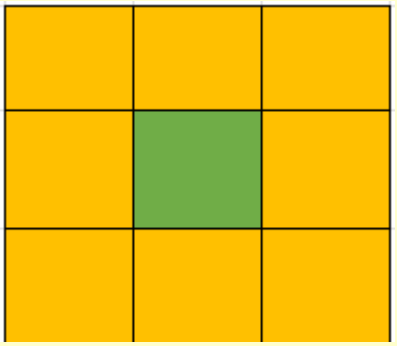
- set draw_color R G B - Va seta o culoare ce va fi folosită ori pentru a desena ori pentru a umple o zonă.
- set line_width x - Va seta dimensiuni creionului cu care vom desena liniile.
- draw line y1 x1 y2 x2
- draw rectangle y1 x1 width height
- draw triangle y1 x1 y2 x2 y3 x3

R G B au valori intre 0 și 255, valoarea default este 0 0 0

x are valori impare intre 1 si 255, valoarea default este 1

La desenarea cu un line_width diferit de 1, pixelul principal va fi incadrat intr-un patrat de latura x

Exemplu:



Pentru line_width = 3, pixelul principal este cel de culoare verde. La desenare, atat pixelul principal, cat si pixelii ce formeaza patratul(si interiorul lui) vor primi culoarea setata.

Pentru trasarea liniilor ce unesc doua puncte ce nu sunt paralele cu Ox sau Oy, se va folosi o functie de gradul I cu formula

(y - yA) / (yB - yA) = (x - xA) / (xB - xA).

Se stabileste pe ce axa se afla intervalul maxim (x max - x min) sau (y max - y min).

Se va parcurge de la minim la maxim intervalul de lungime maxima [x min, x max] sau [y min, y max] si pe baza formulei calculate PE INTREGI se afla a doua coordonata a punctului desenat.

Exemplu:

Pentru punctele (1, 1) si (2, 5), intervalul mai mare este cel de pe axa Oy.

Deci formula prin care vom afla x-ul corespunzator fiecarui y din intervalul [1, 5] este x = (y + 3) / 4.

y = 1 → x = 1

y = 2 → x = 1

y = 3 → x = 1

y = 4 → x = 1

y = 5 → x = 2

Pentru a evita erorile de rotunjire, punctele initiale primite ca parametru se vor desena indiferent de rezultatul functiei.

Daca intervalele sunt egale, atunci dreapta rezultata este de forma f(x) = x + C sau f(x) = -x + C, unde nu conteaza modul in care va fi desenata linia.

Task 4 - Fill color

Pentru acest task vom implementa operația finală:

- fill y x

În urma operației, pixelul de la pozitia **(y, x)** va fi suprascris cu pixeli setati prin comanda **set draw_color R G B**. Operația va fi repetată recursiv pentru toți vecinii acestuia (stânga, dreapta, sus, jos) care aveau aceeași culoare cu pixelul original din **(y, x)**.

Dacă pixelul original avea tot culoarea (r, g, b), operația nu produce nici un efect.

Exemplu

Imaginea Originală:

```
(0, 0, 255) (10, 10, 10) (255, 0, 0)    (10, 10, 10)
(10, 10, 10) (0, 255, 0) (10, 10, 10)  (10, 10, 10)
      (10, 10, 10) (10, 10, 10) (10, 10, 10) (10, 15, 10)
```

Instrucțiunea:

```
set draw_color 42 42 42
      fill_color 1 2
```

În urma aplicării operației, toți vecinii accesibili din poziția (1, 2), având culoarea (10, 10, 10), au fost suprascriși cu un pixel de valoarea (42, 42, 42). Pixelul de pe poziția (0, 1) a rămas neschimbat.

Imaginea Obținută:

```
(0, 0, 255)  (10, 10, 10)  (255, 0, 0)  (42, 42, 42)
(42, 42, 42) (0, 255, 0)  (42, 42, 42) (42, 42, 42)
      (42, 42, 42) (42, 42, 42) (42, 42, 42) (10, 15, 10)
```

Task 5 - Clean Valgrind BONUS (20p)

Intrucat unul din scopurile principale ale acestei teme este alocarea dinamica a memoriei, pentru a rezolva acest task trebuie sa nu aveti nicio eroare sau leak de memorie la rularea utilitarului valgrind pe aceasta.

Utilitarul se va rula folosind urmatoarea comanda:

```
valgrind --tool=memcheck --leak-check=full --error-exitcode=1 ./bmp < input_file
```

Atentie! Numele executabilului rezultat in urma comenzii

```
make build
```

trebuie sa fie neaparat **bmp**!

Atentie! Pentru a putea primi punctaj pe acest task trebuie sa aveti punctaj maxim pe restul cerintelor!

Resurse si checker-ul local

Resursele pentru tema se pot descarca:

- **bmp_header.h**: headerul care contine declaratiile struct-urilor pe care le veti folosi in citirea unui fisier BMP aici;
- **checker.zip**: arhiva zip ce contine checker-ul cu care puteti sa va testati implementarea local aici.

Trimitere tema

Tema va fi trimisa folosind vmchecker, cursul **Programarea Calculatoarelor**, tema **image_processing**.

Punctajul:

- 150p - teste (20p sunt bonus)
  - task 1 - 1 test = 10p
  - task 2 - 5 teste = 20p
  - task 3 - 5 teste = 30p
  - task 4 - 5 teste = 30p
  - mix de task-uri - 5 teste = 40p

        - task 6 - 20p pentru rularea fara erori a utilitarului valgrind (doar daca aveti deja 150p)
- 20p - coding style & README (checker-ul o sa va acorde aceste puncte doar pentru prezenta fisierului README, la corectare acestea vor fi validate)