Tema 3. Image processing

Responsabili:

- Sebastian Pîrtoacă
- Relu Drăgan

Termen de predare: **15.01.2017 23:59**

Pentru fiecare zi (24 de ore) de întârziere, se vor scădea 10 puncte din nota acordată. Deadline-ul hard este **18.01.2017 23:59**.

Actualizări:

- **[16/12/2016]** adaugat enunt;
- **[31/12/2016]** modificare gresela exemplu task 3 (pixel de valoare 10 inlocuit cu 30).

Scopul temei:

- utilizarea structurilor de date;
- lucrul cu fisiere binare;
- abordarea cu pasi mici a unei probleme mai complexe;
- observarea unor tehnici de baza care se folosesc in image processing.

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);

Search

Resurse generale

- Regulament: seria CA
- Regulament: seria CB/CD
- Punctaj seria CA
- Calendar
- Catalog laborator
- Debugging
- Coding style
- Checker laborator CB/CD

Cursuri

Continutul Tematic

Laboratoare

01. Unelte de programare
02. Tipuri de date. Operatori.
03. Instrucțiunile limbajului C
04. Funcții
05. Tablouri. Particularizare - vectori
06. Matrice. Operații cu matrice
07. Optimizarea programelor folosind operaţii pe biţi
08. Pointeri. Abordarea lucrului cu tablouri folosind pointeri
09. Alocarea dinamică a memoriei. Aplicaţii folosind tablouri şi matrice
10. Prelucrarea şirurilor de caractere. Funcţii. Aplicaţii
11. Structuri. Uniuni.

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.

Cerinte

Veti avea de rezolvat 4 task-uri. La primele 3 task-uri o sa procesati o singura imagine. Numele acesteia o sa fie citit de pe prima linie din fisierul de intrare (vezi formatul datelor de intrare). La task-ul 4 o sa

Aplicație: Matrice rare
12. Operaţii cu fişiere. Aplicaţii folosind fişiere.
13. Parametrii liniei de comandă. Preprocesorul. Funcţii cu număr variabil de parametri
14. Recapitulare

Teme de casa (general)

- Indicații generale

Teme de casă: seria CA

- Tema 1
- Tema 2
- Tema 3

Proiect: seria CA

- Proiect

Teme CB/CD

- Tema 1
- Tema 2
- Tema 3

Table of Contents

lucrati cu un format binar special (mai multe detalii in sectiunea corespunzatoare task-ului 4).

Task 1 (10p):

Acest task presupune transformarea unei imagini color intr-o imagine alb-negru. Fisierul .bmp la care se gaseste imaginea se va citi de pe prima linie a fisierului de intrare, numit **input.txt**. Daca numele imaginii este **<nume>.bmp** atunci imaginea alb-negru se va scrie in fisierul **<nume>_black_white.bmp** (de exemplu, daca prima linie din fisierul input.txt este image.bmp atunci imaginea alb negru se va scrie in fisierul image_black_white.bmp). Procedeul prin care o imagine color se transforma intr-o imagine alb-negru este urmatorul: **fiecare** pixel din imaginea color, fie acesta (R, G, B), va deveni (X, X, X) unde X = [(R + G + B) / 3] unde prin [] se intelege **parte intreaga** (inferioara). De exemplu pixelul (3, 2, 2) va deveni (2, 2, 2), iar pixelul (10, 20, 30) va deveni (20, 20, 20).

Exemplu: Daca imaginea color arata astfel:

Dupa transformare ar trebui sa arate astfel:

Nu descarcati imaginile pentru testare! Acestea nu respecta formatul BMP! Pentru testare folositi **numai** imaginile puse la dispozitie in arhiva de testare. Imaginile de mai sus au doar caracter informativ.

Task 2 (20p):

In cadrul acestui task va trebui sa aplicati anumite filtre pe o imagine. Un filtru este o matrice (in cazul acestei teme matricea va avea mereu 3 linii si 3 coloane) care este aplicata fiecarui pixel din imagine pentru a obtine noul pixel. In continuare o sa vedem ce inseamna sa aplici o matrice (filtru) unui pixel. Fie (R_{22}, G_{22}, B_{22}) un pixel din imagine care are urmatorii vecini:

$$\begin{bmatrix} (R_{11}, G_{11}, B_{11}) & (R_{12}, G_{12}, B_{12}) & (R_{13}, G_{13}, B_{13}) \\ (R_{21}, G_{21}, B_{21}) & (R_{22}, G_{22}, B_{22}) & (R_{23}, G_{23}, B_{23}) \\ (R_{31}, G_{31}, B_{31}) & (R_{32}, G_{32}, B_{32}) & (R_{33}, G_{33}, B_{33}) \end{bmatrix}$$

si fie filtrul:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Atunci, in imaginea rezultata in urma aplicarii filtrului **A**, pixel-ul (R_{22}, G_{22}, B_{22}) va fi inlocuit cu $(R'_{22}, G'_{22}, B'_{22})$ unde:

$$R'_{22} = \sum_{i=1}^{3} \sum_{j=1}^{3} R_{ij} a_{ij}$$

$$G'_{22} = \sum_{i=1}^{3} \sum_{j=1}^{3} G_{ij} a_{ij}$$

$$B'_{22} = \sum_{i=1}^{3} \sum_{j=1}^{3} B_{ij} a_{ij}$$

Daca in urma calculului uneia dintre cele 3 valori de mai sus, suma rezultata este < 0 atunci valoarea se va seta la 0. Asemanator, daca suma este > 255 atunci componenta corespunzatoare din pixel se va seta la 255.

Daca un pixel are vecini in afara imaginii, la calculul sumei acesti vecini vor avea valoarea (0, 0, 0).

Sa presupunem ca avem urmatoarea imagine:

```
(2 2 2)    (3 3 3)   (0 0 0)   (0 0 0)
(10 10 10) (1 1 1)   (0 0 0)   (0 0 0)
(0 0 0)    (0 0 0)  (50 50 5)  (0 0 0)
(0 0 0)    (0 0 0) (240 0 240) (0 0 0)
```

Primul pixel (cel de pe prima linie si de pe prima coloana) trebuie sa fie pixelul din coltul stanga sus al imaginii (atunci cand este afisata pe ecran)! Atentie la faptul ca imaginea se citeste rasturnata din fisierul BMP. La aplicarea filtrelor trebuie sa considerati dispunerea de mai sus (nu cea din fisier). Cu alte cuvinte, liniile sunt dispuse crescator de sus in jos, prima linie fiind si prima linie afisata pe ecran, iar coloanele sunt dispuse crescator de stanga la dreapta.

Pentru imaginea de mai sus, fisierul BMP ar arata astfel (ignorand cele 2 headere):

```
(0 0 0)    (0 0 0) (240 0 240) (0 0 0)      No pad
(0 0 0)    (0 0 0)  (50 50 5)  (0 0 0)      No pad
(10 10 10) (1 1 1)   (0 0 0)   (0 0 0)      No pad
(2 2 2)    (3 3 3)   (0 0 0)   (0 0 0)      No pad
```

Pe imaginea de mai sus dorim sa aplicam urmatorul filtru:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Imaginea rezultata va fi:

```
(15 15 15) (6 6 6)     (3 3 3)      (0 0 0)
(13 13 13) (14 14 14)  (51 51 6)    (0 0 0)
(10 10 10) (51 51 6)   (255 50 245) (50 50 5)
(0 0 0)    (240 0 240) (255 50 245) (240 0 240)
```

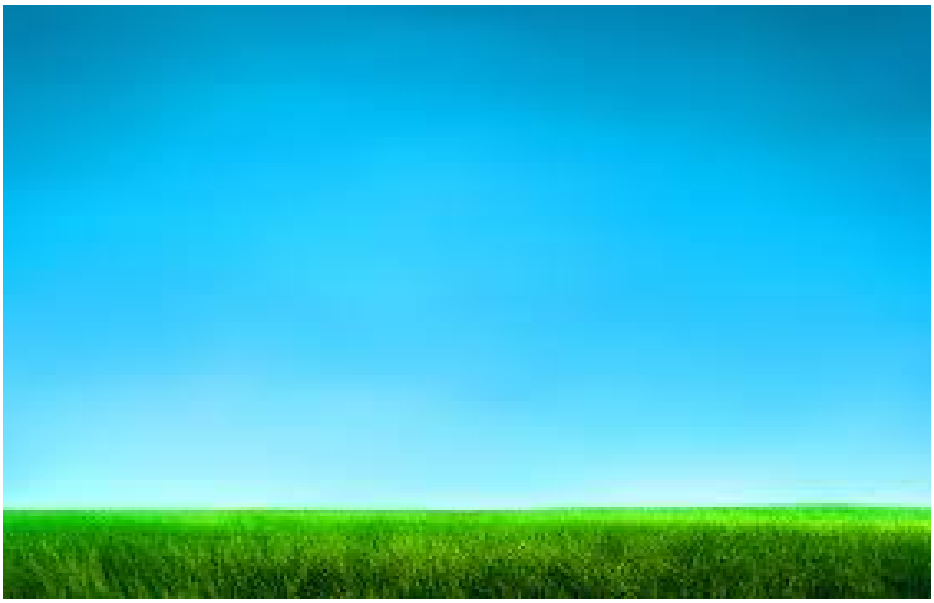
Pentru rezolvarea acestui task va trebui sa aplicati 3 filtre de edge-detection. Acestea sunt:

$$F_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} F_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} F_3 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

Aceste filtre se vor aplica pe rand imaginii alb-negru obtinute la Task-ul 1. Imaginea obtinuta prin aplicarea lui F_1 imaginii alb-negru se va salva in fisierul **<nume>_f1.bmp**, imaginea obtinuta prin aplicarea lui F_2 imaginii alb-negru se va salva in fisierul **<nume>_f2.bmp**, iar imaginea obtinuta prin aplicarea lui F_3 imaginii alb-negru se va salva in fisierul **<nume>_f3.bmp**.

Task 3 (30p)

Ne propunem sa realizam o compresie simpla pentru imaginile BMP. La un nivel inalt, algoritmii de compresie se impart in doua categorii: lossless si lossy. Compresia imaginilor este o compresie lossy. Nu se poate aplica o compresie lossless, pe o imagine, care sa aiba un grad ridicat de compresie deoarece algoritmii de compresie lossless se bazeaza pe eliminarea reduntantei statistice, care in cazul unei imagini este mica (deoarece intr-o imagine, putem intalnii, orice valoare cu aceeasi probabilitate). De exemplu, formatul JPEG presupune o compresie de tip loosy. Algoritmul de compresie pe care il vom aplica in aceasta tema este unul minimalist si se bazeaza pe urmatoarea idee: pixeli invecinati care sunt putin diferiti (culoarea este asemanatoare) pot fi grupati in pixeli cu aceeasi culoare fara sa se observe o diferenta majora la vizualizarea imaginii. Ganditi-va ca avem urmatoarea imagine



.

Se observa ca exista apoximativ 3 nuante de culoare: albastru inchis, albastru deschis si verde. In loc sa salvam fiecare pixel separat in fisier putem sa specificam doar cele trei zone si sa spunem ca fiecare are o anumita culoare. Astfel, se va pierde din calitate (compresie loosy) dar se va castiga spatiu. Cat se pierde din calitate este direct proportional cu cat de mare este pragul pentru care vom considera ca doi pixeli fac parte din aceeasi zona.

O zona este o multime de pixeli adiacenti (care se invecineaza pe orizontala sau veriticala, dar nu pe diagonala). O zona nu trebuie sa aiba o forma neaparat regulata.

Algoritmul de compresie pe care o sa il implementati (si descrie formal ceea ce am spus mai sus) este urmatorul:

1) Se alege un pixel din imagine care nu a fost inclus in nicio zona. Fie acest pixel (R, G, B). Daca toti pixelii au fost inclusi intr-o zona atunci algoritmul s-a terminat;

2) Pornind de la acest pixel se detemina zona de pixeli care satisface simultan urmatoarele conditii:

- Pixelul ales la punctul **1)** face parte din aceasta zona;
- Oricare ar fi un pixel din zona (R', G', B') trebuie ca $|R - R'| + |G - G'| + |B - B'| <= threshold$ unde threshold este o valoare data ca input algoritmului. De asemenea pixelul (R', G', B') nu trebuie sa fi fost deja inclus intr-o alta zona. Daca (R', G', B') era deja inclus intr-o alta zona, chiar daca s-ar indeplinii conditia ca suma modulelor sa fie mai mica decat un threshold atunci pixelul (R', G', B') nu va fi inclus in zona curenta;
- Zona gasita este maximala. Cu alte cuvinte, nu mai exista nici un alt pixel care sa fie vecin al zonei si sa indeplineasca conditiile de la punctul anterior.

3) Toti pixelii din zona determinata vor avea culoarea pixelului de la pasul **1)** adica (R, G, B).

4) Dupa deteminarea noii zone se reia algoritmul incepand cu pasul **1)**.

Modul in care alegem pixelul la pasul **1)** poate inflenta calitatea algoritmului. Pentru aceasta tema vom aplica urmatoarea metoda de selectie: daca exista mai multi pixeli care nu au fost deja inclusi intr-o zona, se va alege acel pixel care se afla pe linia mai mica, iar in caz de egalitate (pixeli pe aceeasi linie) se va alege cel care se afla pe coloana mai mica. **Atentie!** La fel ca si la **Task-ul 1** prima linie este linia cea mai de sus din imagine (adica cea mai de jos din fisier) iar prima coloana este coloana cea mai la stanga in imagine.

Sa consideram urmatoarea imagine (si threshold = 10):

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30 29
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (12 10
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (12 11
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (12 10
```

Initial nici un pixel nu este inclus in vreo zona. Se alege pixelul din coltul stanga-sus de valoare (10 10 10) deoarece acesta se afla pe cea mai mica linie (si pe cea mai mica coloana). Se determina zona maximala din care acesta face parte:

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30 2
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (12 1
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (12 1
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (12 1
```

Alegem din nou un pixel care nu a fost inclus intr-o zona conform algoritmului de selectie. Acest pixel este cel de pe linia 1 si coloana 4 (indexarea incepe la 1) de valoare (30 30 30). Se determina zona maximala din care acesta face parte (colorata cu verde):

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30 2
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (12 1
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (12 1
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (12 1
```

Alegem din nou un pixel care nu a fost inclus intr-o zona conform algoritmului de selectie. Acest pixel este cel de pe linia 2 si coloana 5 (indexarea incepe la 1) de valoare (12 10 13). Se determina zona maximala din care acesta face parte (colorata cu albastru):

```
(10 10 10) (12 10 13) (10 10 10) (30 30 30) (30 2
(10 10 10) (12 15 10) (10 10 10) (30 28 30) (12 1
(11 11 11) (11 12 11) (10 10 10) (30 33 30) (12 1
(12 12 12) (12 10 12) (10 10 10) (30 30 31) (12 1
```

Am determinat astfel 3 zone in imagine. Dupa ce se inlocuieste in fiecare zona culoarea pixelilor cu cea a pixelului initial (determinat la pasul **1**) obtinem imaginea:

```
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (30 3
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (12 1
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (12 1
(10 10 10) (10 10 10) (10 10 10) (30 30 30) (12 1
```

Aceasta imagine nu o vom scrie intr-un fiser .bmp, ci intr-un fisier binar dupa urmatorul format:

1) Cele doua header-e din formatul BMP se scriu nemodificate;

2) Incepand de la valoarea byte-ul de offset din header (adica unde ar fi trebuit sa inceapa matricea de pixeli din fomatul BMP) se for scrie tupluri (nr_linie, nr_coloana, R, G, B) cu urmatoarea semnificatie:

- nr_linie este o valoare be 2 bytes si reprezinta linia la care se gaseste pixelul in imagine (indexarea incepe de la 1);
- nr_coloana este o valoare pe 2 bytes si reprezinta coloana la care se gasete pixelul in imagine (indexarea incepe de la 1);
- R, G, B sunt 3 valori fiecare pe 1 byte si reprezinta cele trei canale Red, Green si Blue ale pixelului de la linia nr_linie si coloana nr_coloana.

In fisier se vor scrie doar pixelii care sunt la granita unei zone cu alta zona. Adica se vor scrie doar pixelii de pe marginea fiecarei zone. Ordinea in care acestia se vor scrie in fisier este crescator pe linii, iar in caz de egalitate crescator pe coloane.

Pixelii de pe linia cea mai mica/cea mai mare si pixelii de pe coloana cea mai mica/cea mai mare (adica marginile imaginii) se vor scrie intotdeauna in fisier.

Pentru exemplul de mai sus ceea ce se va scrie in fiserul **binar** dupa ce se scriu cele doua header-e BMP este:

```
(1, 1, 10, 10, 10)
(1, 2, 10, 10, 10)
(1, 3, 10, 10, 10)
(1, 4, 30, 30, 30)
(1, 5, 30, 30, 30)
(2, 1, 10, 10, 10)
(2, 3, 10, 10, 10)
(2, 4, 30, 30, 30)
(2, 5, 12, 10, 13)
(3, 1, 10, 10, 10)
(3, 3, 10, 10, 10)
(3, 4, 30, 30, 30)
(3, 5, 12, 10, 13)
(4, 1, 10, 10, 10)
(4, 2, 10, 10, 10)
(4, 3, 10, 10, 10)
(4, 4, 30, 30, 30)
(4, 5, 12, 10, 13)
```

Urmariti cu atentie exemplul si observati ce pixeli nu au fost/au fost scrisi in fisierul binar.

Aplicarea algoritmului de compresie se realizeaza pe imaginea al carei nume se gaseste pe primul rand in fisierul **input.txt** (cea pe care am lucrat si la primul task). Valoarea de prag (threshold) care se va utiliza in cadrul algoritmului se va citi de pe linia 2 din fisierul **input.txt**. Datele in format binar, rezultate in urma compresiei, se vor scrie in fisierul **compressed.bin**.

Task 4 (30p)

Se da un fisier binar in formatul celui de la Task-ul 3 (imagine comprimata). Sa se aplice algoritmul de decompresie. Numele fisierului din care cititi datele comprimate se gaseste in fisierul **input.txt** pe linia a treia. Imagina decomprimata se va scrie in format BMP in fisierul **decompressed.bmp**. Practic, trebuie sa delimitati zonele si sa atribuiti fiecarui pixel dintr-o zona culoarea marginii zonei (toata marginea va avea aceeasi culoare, deoarece asa am definit compresia in Task-ul 3). Daca o imagine este comprimata si apoi decomprimata, imaginea rezultata nu va fi egala cu imaginea initiala (loosy compression) ci va pierde din calitate. Pe de alta parte, un BMP de 6MBytes dupa compresie poate ocupa si 300KBytes (vezi fisierele din arhiva de testare).

Formatul datelor de intrare si iesire

Fisierul de intrare se numeste **input.txt**. Acesta are urmatoarea structura:

- Pe prima linie sa gaseste numele fisierului de unde cititi imaginea pe care o veti prelucra la primele 3 task-uri. Numele fisierului se termina intotdeauna cu ".bmp" si nu va contine spatii sau tab-uri;
- Pe a doua linie se gaseste valoarea de prag pentru task-ul 3;
- Pe a treia linie se gaseste numele fisierului binar de unde cititi o imagine comprimata (in formatul de la task-ul 3) pe care va trebui sa o decomprimati.

Exemplu:

```
image.bmp
25
arhive.bin
```

Ca output pentru exemplul de mai sus se vor scrie fisierele: image_black_white.bmp, image_f1.bmp, image_f2.bmp, image_f3.bmp, compressed.bin si decompressed.bmp.

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. Aceeasi observatie se aplica si in cazul formatului comprimat.

Restrictii si precizari

- Dimensiunile imaginilor nu vor depasii 2500×2500;
- Atunci cand scrieti o imagine (atat in format .bmp cat si in formatul special de la task-ul 3) **NU** modificati header-ele citite. Cu alte cuvinte ce headere ati citit aceleasi headere veti scrie (deoarece nici unul dintre task-uri nu necesita modificarea header-ului imaginii). In caz contrar, testul se va puncta cu 0;
- Exista o limita de timp de **60 de secunde** pentru fiecare test necesara pentru a testa tema pe vmchecker. Scopul cursului de programare nu este de a scrie algoritmi eficienti dar limita de 60 de secunde este de bun simt. In mod normal, o implementare obisnuita ar trebui sa ruleze in mai putin de 5s pe test.
- Tema se va trimite pe vmchecker si se va testa local cu ajutorul checker-ului care va fi disponibil **in curand**;
- Pe vmchecker veti uploada o arhiva in format .zip care sa contina:
  1. **Makefile**, cu cel puțin 3 targeturi, **build**, **run** și **clean**; Regula **run** trebuie sa ruleze executabilul care a fost obtinut la regula **build**;
  2. **sursele** voastre, adică fișierele .c și .h. **Inclusiv headerul bmp_header.h sau sub orice altă denumire îl folosiți**;
  3. **README**, în care trebuie să dați detalii despre implementare, de ce ați ales să rezolvați într-un anumit fel, etc.
- Daca rezolvati doar o parte din task-uri asigurati-va ca pe celelalte nu primiti erori la rulare (precum SEGFAULT) sau time limit exceeded, altfel tot testul va fi punctat cu 0. De exemplu, daca task-urile 1 si 2 sunt OK dar task-ul 3 primeste SEGFAULT sau dureaza prea mult atunci tot testul se va nota cu 0.

Fiti **foarte** atenti la erori de tipul stack overflow. Pe vmchecker dimensiunea stivei este de 1MB. Daca aveti functii recursive aproximati cata memorie se ocupa pe stiva la un apel al functiei respective.

Resurese si checker-ul local

Resursele pentru tema se pot descarca de aici. Sunt prezente:

- **bmp_header.h**: headerul care contine declaratiile struct-urilor pe care le veti folosi in citirea unui fisier BMP;
- **checker-ul** pe care il veti putea folosi pentru a va testa tema local.