

# PROGRAMAREA CALCULATOARELOR ȘI LIMBAJE DE PROGRAMARE I

## Tema #2 Alocare dinamică de memorie

Deadline soft: **27.11.2024**. Deadline hard: **27.11.2024**.

**Coordonator:** Taisia Coconu

**Responsabili temă:** Taisia COCONU, Ion-Dorinel FILIP, Cătălin RÎPANU, Bogdan MOCANU, Corina-Elena ULARIU, Radu-Andrei GEORGESCU, Dominic STANCIU, Nicolae-Cristian MACOVEI, Valentin Gabriel CĂRĂULEANU

**Responsabil checker:** Bogdan Mocanu

### Changelog:

- 12.11.2024: Publicare tema 2.  
Vă rugăm să adresați toate întrebările legate de teme pe forumul asociat temei 2 de pe site-ul de curs. Orice clarificare / corectare / actualizare legată de enunț, checker sau lucruri administrative etc., vor fi anunțate exclusiv pe forumul dedicat temei 2.
- 18.11.2024: Testele au devenit publice și în enunț s-au detaliat suplimentar anumite aspecte privind calculul valorilor din grid (marcate cu **roșu**).

### Cuprins

<b>Definirea problemei și Obiectivele</b>	<b>3</b>
Obiectivele temei . . . . .	3
<b>Fișiere PPM</b>	<b>3</b>
Antetul imaginii . . . . .	3
Datele pixelilor . . . . .	4
Corpul imaginii . . . . .	4
<b>Prezentarea algoritmului Marching Squares</b>	<b>6</b>
<b>Structura codului și cerințele pentru implementare</b>	<b>11</b>
<b>Comenzi și exemple de rulare</b>	<b>11</b>
Comenzi . . . . .	11
Exemplu de cod - citire imagine . . . . .	11
Tabel mesaj de output . . . . .	13

<b>Regulament</b>	<b>14</b>
Arhivă	14
Checker	14
Punctaj	14
Reguli și precizări	14
Alte precizări	15

## Definirea problemei și Obiectivele

Algoritmul ***Marching Squares*** reprezintă o tehnică de grafică computerizată utilizată pentru generarea contururilor dintr-un câmp scalar bidimensional (2D). Este folosit pe scară largă pentru vizualizarea datelor în aplicații precum hărțile topografice, hărțile termice și hărțile meteorologice, prin desenarea izolinilor sau izobenzilor. Izolinile reprezintă linii de valori constante (de exemplu, altitudine sau temperatură), în timp ce izobenzile ilustrează regiuni umplute între aceste linii. Simplitatea și eficiența algoritmului ***Marching Squares*** îl fac ideal pentru a transforma datele 2D numerice în reprezentări grafice, permitând utilizatorilor să interpreteze cu ușurință modelele și gradienții din cadrul datelor.

### Obiectivele temei

Obiective generale:

- **Modelarea și proiectarea** unor soluții reale utilizate în aplicații curente, folosind concepte fundamentale ale programării calculatoarelor precum: *variabile, tipuri de date, tablouri* (unidimensionale și bidimensionale); pointeri și *alocarea dinamică a memoriei*;
- **Dezvoltarea** unei soluții originale la o problemă cunoscută, care respectă principiile de bază precum: *modularizare și utilizarea eficientă a memoriei*.

Obiective specifice:

- **Implementarea Algoritmului Marching Squares:** Realizarea unei implementări originale eficiente a algoritmului, capabilă să proceseze rapid rețele de date bidimensionale și să genereze contururi vizuale precise în funcție de valorile pragului predefinit;
- **Utilizarea formatului de imagine PPM:** lucrul cu imagini PPM-text pentru manipularea imaginilor și stocarea rezultatelor algoritmului;
- **Interpolarea liniară:** integrarea conceptelor de algebră liniară, precum interpolarea, pentru determinarea exactă a poziției contururilor la marginile celulelor, cu scopul de a obține o imagine finală clară și detaliată;
- **Alocare de memorie:** lucrul eficient cu tablouri unidimensionale și bidimensionale folosind alocarea dinamică a memoriei;
- **Redimensionarea imaginii:** procesarea eficientă a imaginilor prin mărirea rezoluției.

## Fișiere PPM

Formatul de imagine **PPM** (Portable Pix Map) este un format simplu și ușor de citit. Acesta utilizează o reprezentare text pentru a codifica informația unei imagini. În general este folosit pentru a reprezenta imagini color în care valorile pixelilor sunt exprimate prin componente (canalele) de culoare **RGB** (roșu, verde, albastru). În continuare, vom detalia structura și cum se interpretează un fișier PPM pentru a ajuta la înțelegerea și manipularea acestui format.

Dacă sunteți interesat de mai multe informații, specificația formală a imaginii poate fi găsită [aici](#).

**Structura unui fișier PPM** este următoarea:

1. **Antetul imaginii** – conține informații de bază despre imagine;
2. **Datele pixelilor** – valorile RGB ale fiecărui pixel din imagine.

### Antetul imaginii

Primele câteva linii sunt definite ca **antetul** imaginii. Ele oferă o prezentare generală a conținutului imaginii. Antetul PPM conține patru intrări, care vor fi definite folosind exemplul:

P3  
4 4  
255

- **Tipul de format:** P3 definește **formatul imaginii**; adică tipul de imagine PPM (culoare completă, codificare ASCII). Pentru această sarcină, formatul va fi *întotdeauna P3*;
- **Dimensiunile imaginii:** Următoarele sunt numărul de **coloane** și numărul de **linii** din imagine. Exemplul este o imagine de 4 pixeli pe 4 pixeli;
- **Valoarea maximă a culorii:** În cele din urmă, avem **valoarea maximă a culorii**. Aceasta definește scara de valori posibile pentru intensitățile culorii. Aceasta poate fi orice valoare, dar o valoare comună este 255, ceea ce înseamnă că valorile roșu, verde și albastru pentru un pixel pot varia de la 0 până la 255.

Modul în care este prezentat antetul este cel corect și trebuie respectat.

## Datele pixelilor

După antet, urmează secțiunea de date, unde sunt listate valorile RGB pentru fiecare pixel al imaginii. Aceste valori sunt organizate sevențial, rând după rând și pixel după pixel, iar fiecare pixel este reprezentat de trei valori (roșu, verde, albastru). Fiecare valoare trebuie să fie între 0 și valoarea maximă specificată în antet (în exemplul nostru, 255).

Exemplu de fișier ppm:

```
P3
4 4
255
0 0 0 100 0 0 0 0 0 255 0 255
0 0 0 0 255 175 0 0 0 0 0 0
0 0 0 0 0 0 0 15 175 0 0 0
255 0 255 0 0 0 0 0 0 255 255 255
```

### Explicație (conform celor ilustrate anterior):

- **Primul rând (P3)** – definește că este o imagine PPM color, cu date în format ASCII;
- **Al doilea rând (4 4)** – indică dimensiunile imaginii: 4 pixeli pe 4 pixeli;
- **Al treilea rând (255)** – specifică valoarea maximă a culorii, ceea ce înseamnă că fiecare componentă de culoare poate avea valori între 0 și 255;
- **Datele pixelilor** – fiecare grup de trei valori corespunde componentelor RGB pentru un pixel. De exemplu, primul pixel are valorile 0 0 0 (negru), al doilea pixel 100 0 0 (roșu deschis), iar ultimul pixel din ultima linie 255 255 255 (alb).

### Observații importante:

1. Ordinea componentelor RGB este întotdeauna respectată pentru fiecare pixel;
2. Respectarea structurii antetului este esențială pentru ca fișierul să fie interpretat corect de orice program care încarcă imagini PPM.

## Corpul imaginii

**Corpul imaginii** conține informațiile reale despre imagine sub forma unei serii de valori RGB. Fiecare pixel al imaginii este un pătrat mic colorat. În fișierul PPM, fiecare pixel este definit de un triplet de valori

care reprezintă cât de mult roșu, verde și albastru (RGB) sunt prezente. Așadar, primul pixel, care are valoarea **0 0 0**, este negru, iar ultimul pixel, **255 255 255**, este alb. Prin variația nivelurilor de valori RGB, puteți obține orice culoare între acestea.

**Structura valorilor RGB:** Fiecare pixel din imagine are trei valori întregi separate prin spații: (i) prima valoare reprezintă intensitatea roșie (R), (ii) a doua valoare reprezintă intensitatea verde (G), (iii) iar a treia valoare reprezintă intensitatea albastră (B).

Valorile RGB pentru fiecare componentă de culoare variază între 0 și valoarea maximă specificată în antet (în mod tipic 255). De exemplu, valoarea 0 pentru toate componente (0 0 0) produce negru (lipsă completă a luminii), iar 255 255 255 produce alb (intensitate maximă pentru toate componente). Pentru orice alte combinații între 0 și 255, pixelul va avea o nuanță corespunzătoare: valorile mari de R creează o tentă roșie, valorile mari de G creează verde, iar valorile mari de B creează albastru. De exemplu, un pixel cu valorile 255 0 0 va fi roșu pur, iar 0 255 0 va fi verde pur.

Raportându-ne la exemplul de fișier PPM prezentat anterior, în Figura ?? putem identifica următoarele:

- Primul pixel 0 0 0 este **negru**.
- Al doilea pixel 100 0 0 are o intensitate mai mare de roșu, ceea ce îl face **roșu închis**.
- Pixelul 255 0 255 este o combinație de roșu și albastru la intensitate maximă, ceea ce dă un **magenta strălucitor**.
- Ultimul pixel 255 255 255 este **alb**.

Observați că valorile culorilor trebuie separate de un spațiu, însă orice spațiu suplimentar este ignorat de vizualizatorul de imagini. În exemplul PPM de mai sus, am folosit spații suplimentare pentru a formați valorile pixelilor astfel încât să fie ușor de înțeles pentru oameni, dar computerului nu îi pasă dacă totul este pe o linie, dacă există o linie de valori RGB pe linia imaginii sau o combinație. **Nu trebuie să presupuneți că o linie din fișier corespunde unei linii din imagine.** Deși este obișnuit ca fiecare linie din fișier să corespundă unei linii de pixeli din imagine, formatul PPM este flexibil în ceea ce privește aranjarea valorilor: orice spațiu suplimentar sau linie nouă poate fi adăugat între valorile RGB pentru a îmbunătăți lizibilitatea, fără a afecta imaginea finală. Editorul de imagine va ignora spațiile și liniile suplimentare, deci datele pot fi organizate pe o singură linie lungă sau grupate în linii pentru o mai ușoară interpretare vizuală de către utilizatori.

Figura 1 ar arăta aproximativ astfel: **Tineți cont că fiecare pătrat este un pixel, aşa că imaginea reală este mult mai mică (imaginea redată a fost mărită cu 5000%).**

Ca o paranteză, deși fișierele PPM sunt ușor de vizualizat ca text (puteți folosi vim, emacs sau Notepad, de exemplu), ele sunt foarte ineficiente. Majoritatea formelor moderne de imagine folosesc o formă de *compresie* pentru a reduce cantitatea de informații stocate (și, implicit, dimensiunea acestora) menținând în același timp aspectul imaginii. O utilizare modernă pentru PPM este ca format intermediu atunci când se convertesc imagini dintr-un tip în altul.

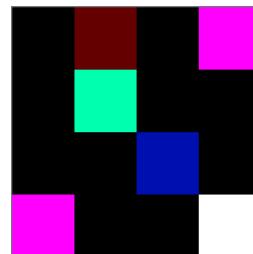


Figura 1: Exemplu de imagine PPM mărită cu 5000%.

## Prezentarea algoritmului Marching Squares

Algoritmul **Marching Squares**, introdus în anii 1980, este utilizat în procesarea imaginilor pentru a delimita contururile dintr-o imagine. Acesta poate fi aplicat pentru a desena linii de nivel pe hărți topografice, variații de temperatură pe hărți termice, puncte de presiune pe hărți de câmp de presiune și altele. Funcționarea algoritmului urmează pașii descriși mai jos.

În continuare presentăm pașii principali ai acestui algoritm, după care îi vom aplica pe un exemplu practic pentru a înțelege mai bine module de funcționare precum și importanța și aplicabilitatea acestui algoritm. Iată o prezentare generală a pașilor, inclusiv o reprezentare grafică pentru fiecare pas:

1. Se alege imagine în format PPM-P3 pentru care se dorește aplicarea algoritmului. Considerăm că imaginea este reprezentată de un domeniu  $D$  (width și height). Se poate observa mai jos imaginea pe baza căreia se va explica algoritmul:



Figura 2: Imaginea inițială.

Pentru început, **domeniul  $D$  al imaginii de intrare este împărțit în pătrate de dimensiune fixă** egală dimensiunile contururilor folosite. În cazul nostru, *valoare\_sampeling = 4*. Această valoare indică faptul că vom aveam conturi de dimensiune 4x4.

După definirea pătratelor, se construiește o rețea  $G$ , formată din colțurile acestor pătrate. Imaginea de intrare este împărțită în  $N^2$  pătrate, unde  $N = \text{width}/\text{valoare_sampeling}$ . Această valoare reprezintă numărul de pătrate pe latura unei dimensiuni a imaginii. Ea va fi utilizată ulterior pentru a determina modul în care conturul intersectează fiecare celulă. Prin împărțirea domeniului  $D$  în pătrate și construirea rețelei  $G$  algoritmul poate procesa eficient informațiile pentru a crea reprezentări grafice relevante.

Valorile din acest grid se calculează ca fiind media aritmetică a luminozității pixelilor (pătrătelelor din imaginea inițială). Luminozitatea unui pixel fiind calculată ca media aritmetică a valorilor celor 3 canale (Red, Green și Blue).

Gigel totuși vrea să implementeze o variantă alternativă, în care fiecare nod din grid este calculat ca media dintre valoarea presupusă în algoritmul original și valorile vecinilor de la Nord-Sud-Est-Vest (maxim 4 vecini pentru fiecare nod din grid). Majoritatea nodurilor din grid vor avea 4 vecini, cu excepția celor de la margine/colțuri pentru care vom considera mai puțini vecini. Gigel a ales această variantă pentru a avea o formulă de calcul și pentru ultima linie și ultima coloană din grid, care nu au corespondent în imaginea inițială, ci doar vecini. În Figura 3 puteți vedea un astfel de grid.

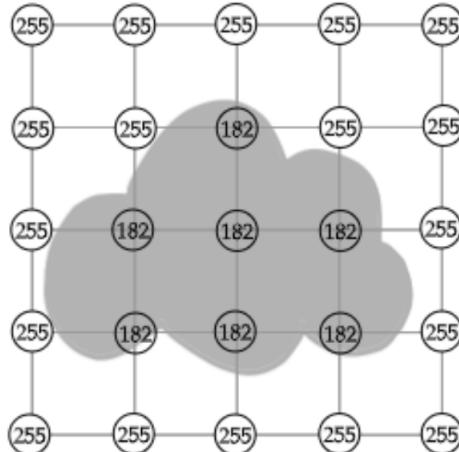


Figura 3: Grid-ul utilizat pentru înconjurarea imaginii.

Fiecare nod din grid corespunde unui pixel din imaginea inițială, grid-ul complet având aceeași dimensiune cu aceasta. Valorile relevante sunt totuși de STEP ori mai puține și doar acestea sunt reprezentate grafic și/sau luate în calcul de către program.

2. Urmează **etapa de thresholding**: Iterează prin rețeaua de conturare pentru a determina starea fiecărui punct în raport cu o valoare de izolare. Pe baza valorii de izolare și a rețelei, se creează o altă rețea binară  $F$  (de același dimensiune cu  $G$ ), prin compararea cu o valoare prag ( $\sigma$ ) asupra câmpului de date din  $G$ , transformând fiecare punct într-o stare binară (0 sau 1). În exemplul ilustrat mai jos, am considerat ( $\sigma = 200$ ) și, analizând imaginea anterioară se observă că valorile egale cu 255 sunt transformate în 1, iar valorile egale cu 182 sunt transformate în 0 (fiind mai mici decât 200). **ATENȚIE la margini!**

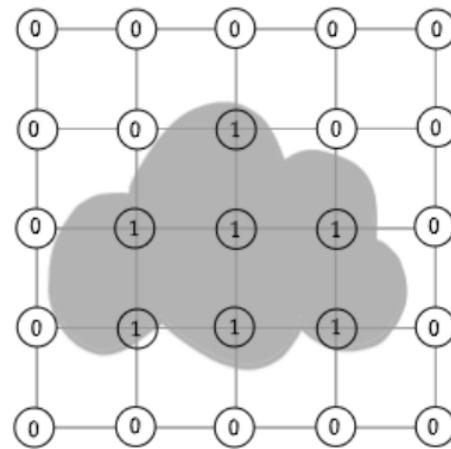


Figura 4: Grid binar.

3. Cea de a treia etapă este **formarea celulelor de conturare**: Fiecare pătrat înconjurat de grid, are acum atașate 4 valori binare reprezentând colțurile sale. Pentru alegerea pattern-ului care va înlocui respectiva regiune în imaginea rezultat, vom face conversia în binar a numărului obținut prin concatenarea bitilor de la colțurile (ca în Figura 5). Acest index reprezintă una dintre cele 16 configurații posibile pentru modul în care un contur intersectează celula. Pentru o înțelegere mai bună a acestei transformări a grilei, putem analiza imaginea următoare:

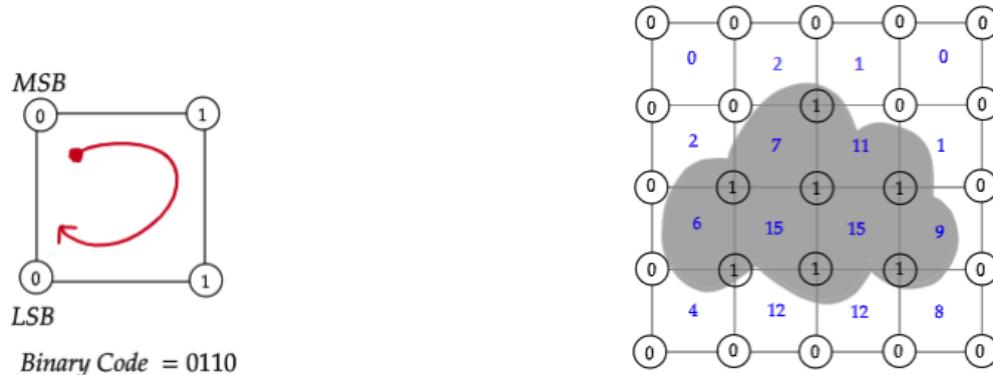


Figura 5: Grid binar și codificarea configurațiilor.

4. Pentru identificarea corectă a componentelor imaginilor, se va utiliza un lookup table (tabelă de căutare) care conține toate cele 16 combinații posibile de pătrate cu colțuri binare. Astfel, tabela de căutare pre-construită este utilizată pentru a determina liniile de contur pentru fiecare configurație posibilă a celulei. Acest lucru permite algoritmului să decidă rapid segmentele de contur care ar trebui desenate în fiecare celulă. Tabela pe care trebuie să o utilizați conține 16 reprezentări diferite, aşa cum se poate observa în imaginile de mai jos. Bitul 0 este reprezentat prin culoarea albă, iar bitul 1 este reprezentat prin culoarea neagră.

De exemplu, în Figura 6 avem un contur în care colțurile din stânga sunt 0 (albe) și conțurile din dreapta sunt 1 (negre). Conform convenției din Figura 5 îi este asociată poziția 6 (cu indexare de la 0) din tabela de căutare. Acest element este reprezentat în Figura 6, în care conturul este jumătate colorat și jumătate necolorat.

Restul tabelei de căutare se regăsește în Figura de mai jos:

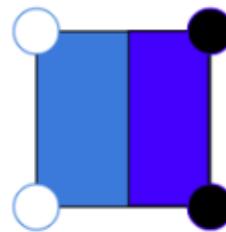


Figura 6: Exemplu element din tabela de căutare.

Reprezentarea acestor contururi se va realiza folosind următoarele culori greyscale (cele 3 canale vor avea valori egale):

- valoarea 255 pentru conturul zonelor ocupate. Conturul poate avea următoatele forme:
  - (a) triunghi (dacă există un singur colț negru pe o orice latură);
  - (b) dreptunghi (dacă există 2 colțuri negre și sunt situate pe aceeași latură);
  - (c) trapez (dacă există 3 colțuri negre);
  - (d) Un pătrat complet (dacă toate colțurile sunt negre).
- valoarea 180 pentru marcarea zonelor ocupate (din spatele conturului);
- valoarea 0 pentru marcarea zonelor libere.

Pentru reprezentarea conturului final, vom folosi cele 16 contururi de 4x4 pixeli, ca în Figura 7.

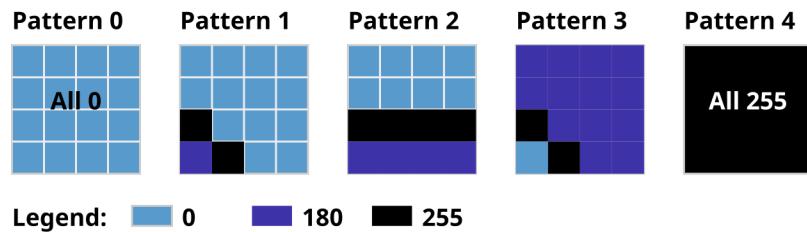


Figura 7: Pattern-uri de 4 pe 4 pixeli pentru contur.

Aceste 4 contururi pot fi folosite pentru a reprezenta ca pixeli discreți cele 16 pattern-uri din tabela de căutare aminitită anterior (vezi Figura 8).

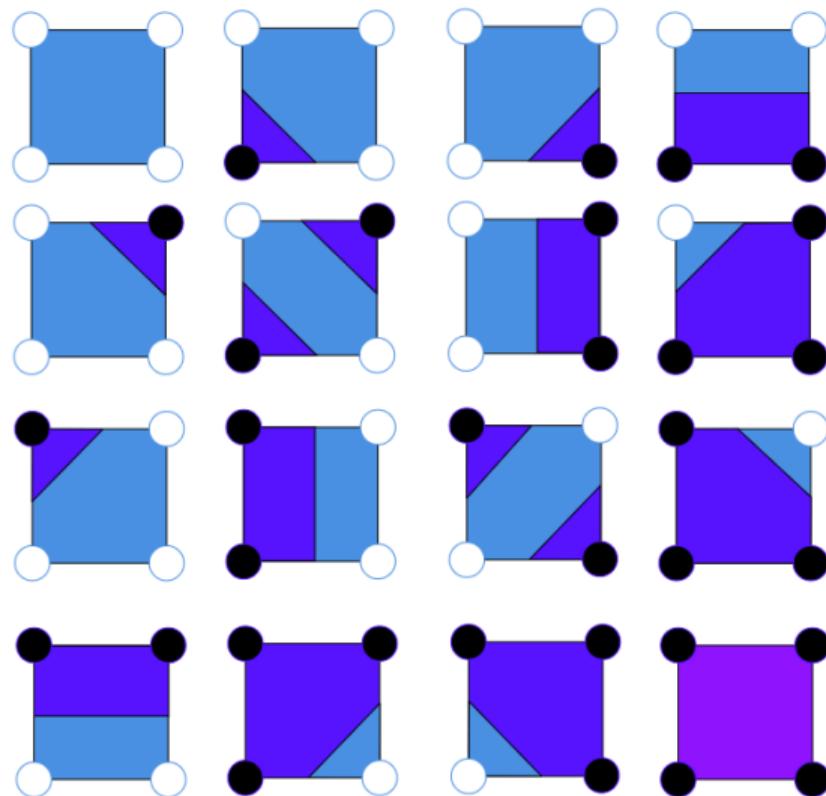


Figura 8: Cele 16 contururi din tabela de căutare.

Colturile negre reprezintă valori de 1, conform convenției din Figura 5.

În cadrul implementării, pentru a reține conturile în tabela de căutare, vă recomandăm să urmăriți imaginea de mai sus (de la stânga la dreapta și de sus în jos), notând faptul că un cerc alb este echivalent bitului 0, în timp ce un cerc negru este echivalent bitului 1.

5. Pasul final este interpolarea lineară: Pentru a determina locația exactă a linilor de contur de-a lungul marginilor celulelor. Se aplică interpolarea liniară între valorile de la colțurile celulei. Practic, în această etapă trebuie să construim o imagine nouă care pornește de la imaginea originală peste care se aplică patternurile identificate. Un exemplu de imagine, finală obținută în urma interpolării este:

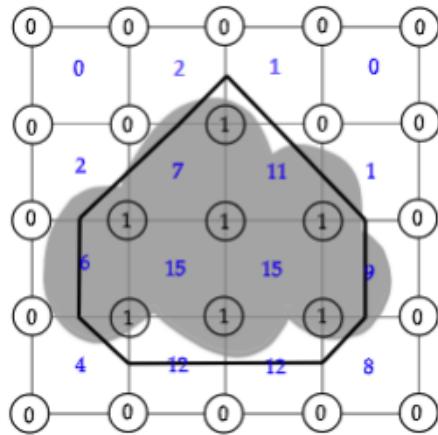
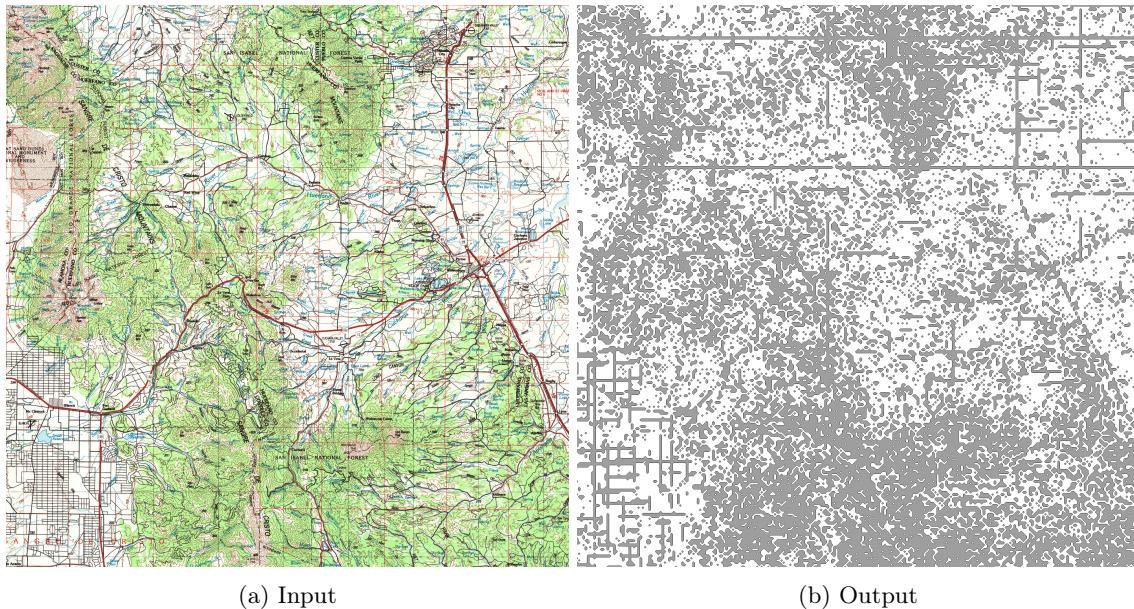


Figura 9: Imagine finală după interpolare.

Algoritmul Marching Squares are o deosebită aplicabilitate pe imagini topografice. În imaginea de mai jos, putem vedea un exemplu de input și output în format PNG (în implementarea temei se vor utiliza imagini în format PPM).

Figura 10: Comparație input/output pentru  $\sigma = 200$ ,  $pas = 8$ , dimensiune contur 8x8.

## Structura codului și cerințele pentru implementare

Pentru o rezolvare corectă, implementarea algoritmului trebuie să respecte următorii pași și concepte fundamentale:

1. **Modelarea structurii și manipularea imaginii:** Formatul PPM - P3 este un format text simplu în care fiecare pixel este reprezentat prin valorile de culoare pentru canalele R (Roșu), G (Verde) și B (Albastru), toate fiind în intervalul [0-255].
2. Imaginea trebuie să poată fi citită de la tastatură respectând formatul PPM - P3 descris anterior. rând cu rând și salvate aceste valori într-o structură de tip matrice.
3. În cazul unui format incorrect, programul trebuie să afișeze mesajele de eroare corespunzătoare.
4. **Gestiunea dinamică a memoriei:** De fiecare dată când citim o imagine, aceasta trebuie să fie stocată dinamic în memorie, pentru a evita limitările statice ale programului. Acest lucru se realizează prin alocarea dinamică de memorie folosind funcțiile specifice (malloc, realloc). Este important ca, atunci când nu mai este nevoie de o imagine, memoria să fie eliberată pentru a evita memory leaks.
5. **Redimensionarea imaginii:** Pentru a redimensiona imaginea, va fi necesară crearea unei imagini noi cu dimensiuni diferite (mai mare cu o valoare predefinită: *resize\_factor = 4*). Redimensionarea implică calcularea valorilor fiecărui pixel în imaginea redimensionată, folosind multiplicarea pixelului pe fiecare dimensiune;
6. **Inițializarea matricei pentru contururi:** Pentru identificarea contururilor, este necesar un mecanism de căutare a acestora în lista de contururi.
7. **Aplicarea algoritmului Marching Squares:** Algoritmul Marching Squares se aplică conform explicațiilor de mai sus.

## Comenzi și exemple de rulare

### Comenzi

READ, WRITE, INIT\_CONTUR, GRID, MARCH, RESIZE, EXIT

1. **READ:** Această comandă permite citirea de la tastatură a unei imagini în format PPM-P3;
2. **WRITE:** Comanda afișează pe ecran o imagine în format PPM-P3;
3. **INIT\_CONTUR:** Această comandă inițiază cele 16 patternuri pentru contururi;
4. **GRID:** Comanda GRID calculează grid-ul format din valori 1 și 0 conform descrierii algoritmului;
5. **MARCH:** Comanda MARCH rulează algoritmul Marching Squares pentru a analiza și completa contururile detectate;
6. **RESIZE:** Această comandă permite redimensionarea imaginii, prin mărire cu un parametru definit (*RESIZE\_FACTOR = 4*);
7. **EXIT:** Comanda EXIT finalizează programul și eliberează resursele utilizate. Aceasta încheie toate operațiile și permite utilizatorului să părăsească aplicația în siguranță.

### Exemplu de cod - citire imagine

Date de intrare (input):

READ

```
P3
2 2
255
1 1 1
0 0 0
1 1 1
0 0 0
WRITE
RESIZE
MARCH
EXIT
```

**Explicații:** Comanda READ este utilizată pentru a citi o imagine de tip PPM în format text. Aceasta presupune următorii pași:

1. **READ** - Indică faptul că programul va începe procesul de citire a imaginii.
2. **P3** - Specifică faptul că imaginea este în format PPM ASCII. Este important ca imaginea să fie în acest format; altfel, programul va genera o eroare.
3. **2 2** - Dimensiunile imaginii, în acest caz, 2 x 2 (lățime x înălțime).
4. **255** - Valoarea maximă a intensității culorii pentru fiecare canal (roșu, verde, albastru). Aceasta înseamnă că fiecare valoare a pixelului trebuie să fie între 0 și 255.
5. **Valorile pixelilor (1 1 1, 0 0 0, etc.)** - Fiecare grup de trei valori reprezintă un pixel, iar aceste valori corespund intensității culorilor roșu, verde și albastru pentru acel pixel. În acest exemplu:
  - (a) **1 1 1** reprezintă un pixel cu valori scăzute de intensitate pentru toate cele trei canale (aproape negru).
  - (b) **0 0 0** reprezintă un pixel complet negru.
  - (c) **WRITE** scrie la tastatură imaginea
  - (d) **RESIZE** redimensionează imaginea
  - (e) **WRITE** scrie la tastatură imaginea redimensionată
  - (f) **MARCH** aplică algoritmul March Squares
  - (g) **EXIT** închiderea în siguranță a programului.

Date de ieșire (output):

```
Imagine citita cu succes [2-2]
P3
2 2
255
1 1 1
0 0 0
1 1 1
0 0 0
Imagine redimensionata cu succes [8-8]
Marching Squares aplicat cu succes [8-8]
Gigel a terminat.
```

**Tabel mesaj de output**

MESAJ	COMANDA
Imagine citita cu succes [height-width]	READ
Eroare: trebuie sa fie P3	READ
Eroare: valoare pixel necorespunzatoare	READ
Eroare: eroare citire pixeli	READ
Imagine redimensionata cu succes [new_height-new_width]	RESIZE
P3 height width valoare maximă pixel (255) R G B R G B ... R G B	WRITE
P3 4 4 255 R G B R G B ... R G B	INIT_CONTUR
//obs. se afisează 16 contururi de acest fel.	
Grid calculat cu success [height-width] matrice grid	GRID
Marching Squares aplicat cu succes [height-width]	MARCH
Gigel a terminat.	EXIT

## Regulament

Regulamentul general al temelor se găsește pe ocw ([Temele de casă](#)). Vă rugăm să îl citiți integral înainte de a continua cu regulile specifice acestei teme.

### Arhivă

Soluția temei se va trimite ca o arhivă **zip**. Numele arhivei trebuie să fie de forma **Grupă\_NumePrenume\_TemaX.zip** - exemplu: **311CA\_NichitaRadu\_Tema2.zip**.

Arhiva trebuie să conțină în directorul **RĂDĂCINĂ** doar următoarele:

- Codul sursă al programului vostru (fișierele **.c** și **.h**).
- Un fișier **Makefile** care să conțină regulile **build** și **clean**.
- Un fișier **README** care să conțină prezentarea implementării alese de voi. **NU** copiați bucăți din enunț.

Arhiva temei **NU** va conține: fișiere binare, fișiere de intrare/ieșire folosite de checker, checkerul, orice alt fișier care nu este cerut mai sus.

Numele și extensiile fișierelor trimise **NU** trebuie să conțină spații sau majuscule, cu excepția fișierului **README** (care are numele scris cu majuscule și nu are extensie).

Nerespectarea oricărei reguli din secțiunea **Arhivă** aduce un punctaj **NUL** (0) pe temă.

### Checker

Pentru corectarea aceste teme vom folosi scriptul **check** din secțiunea de resurse asociată temei. Vă rugăm să citiți **README.md** pentru a știi cum să instalați și utilizați checkerul.

### Punctaj

Distribuirea punctajului:

- Punctaj pe teste: 80p;
- Claritatea și calitate cod: 10p
- Claritatea explicațiilor din **README**: 10p;
- Modularizare + implementări deosebite: 10p (bonus acordat manual)

**ATENȚIE!** Punctajul maxim pe temă este **100p**. Acesta reprezintă **1p** din nota finală la această materie. La această temă se pot obține până la **110p** (există un bonus de până la 10p acordat pe baza modularizării și a unor implementări deosebite, ce va fi acordat la corectarea manuală). Punctajul bonus de la teme se acordă doar la nota finală, dacă toate condițiile de promovare au fost deja satisfăcute.

### Reguli și precizări

- Punctajul pe teste este cel acordat de script-ul **check**, rulat pe **Moodle**. Echipa de corectare își rezervă dreptul de a depuncta pentru orice încercare de a trece testele fraudulos (de exemplu prin hardcodare, etc).
- Punctajul pe calitatea explicațiilor și a codului se acordă în mai multe etape:
  - **corectare automată**
    - \* Checkerul va încerca să detecteze în mod automat probleme legate de coding style și alte aspecte de organizare a codului.

- \* Aceasta va puncta cu maxim 20p dacă nu sunt probleme detectate în mod automat.
- \* Punctajul se va acorda proporțional cu numărul de puncte acumulate pe teste din cele 80p.
- \* Checkerul poate să aplique însă și penalizări (exemplu pentru warninguri la compilare) sau alte probleme descoperite la runtime.

#### – corectare manuală

- \* Tema va fi corectată manual și se vor verifica și alte aspecte pe care checkerul nu le poate prinde. Recomandăm să parcurgeți cu atenție tutorialul de **coding-style** de pe [ocw.cs.pub.ro](http://ocw.cs.pub.ro).
- \* Codul sursă trebuie să fie însoțit de un fișier README care trebuie să conțină informațiile utile pentru înțelegerea funcționalității, modului de implementare și utilizare a programului. Aceasta evaluează, de asemenea, abilitatea voastră de a documenta complet și concis programele pe care le produceți și va fi evaluat, în mod analog CS, de către echipa de asistenți. În funcție de calitatea documentației, se vor aplica depunctări sau bonusuri.
- \* La corectarea manuală se va acorda un bonus de maximum 10 puncte pentru modularizare. Deprinderea de a scrie cod sursă de calitate, este un obiectiv important al materiei. Sursele greu de înțeles, modularizate neadecvat sau care prezintă hardcodări care pot afecta semnificativ menținabilitatea programului cerut, pot fi depunctate adițional. Penalizarea pentru modularizare defectuoasă sau absentă complet, poate să fie oricât de mare.
- \* În această etapă se pot aplica depunctări oricât de mari (chiar și totale). Orice rezultat/punctaj acordat automat de checker, poate fi anulat sau suprascris de către echipa de PCLP la corectarea manuală.
- \* O temă care NU compilează cu -Wall -Wextra este depunctată la corectarea manuală cu 5p (punctajul echivalent pentru warnings).
- \* **Tema trebuie să reprezinte exclusiv munca studentului!** Echipa va aplica regulamentele în vigoare pentru situațiile de fraudă/plagiat, pentru orice nerespectare a acestei reguli, inclusivând, dar nelimitându-se la: copierea de la colegi (se aplică pentru ambele persoane implicate) sau din alte surse a unor părți din temă, prezentarea ca rezultat personal a oricărei bucăți de cod provenită din alte surse necitate sau neaprobată în prealabil (site-uri web, alte persoane, cod generat folosind AI/LLM-uri etc.).

## Alte precizări

- Implementarea se va face în limbajul C, iar tema va fi compilată și testată **DOAR** într-un mediu **LINUX**. Nerespectarea acestor reguli aduce un punctaj **NUL**.
- Tema trebuie trimisă sub forma unei arhive pe site-ul cursului [curs.upb.ro](http://curs.upb.ro).
- Tema poate fi submisă de oricâte ori fără depunctări până la deadline. Mai multe detalii se găsesc în regulamentul de pe [ocw](http://ocw.cs.pub.ro).
- O temă care NU compilează **NU** va fi punctată.
- O temă care compilează, dar care **NU** trece niciun test **NU** va fi punctată.
- Punctajul pe teste este cel acordat de **check** rulat pe **platforma remote a echipei de PCLP**. Echipa de corectare își rezervă dreptul de a depuncta pentru orice încercare de a trece testele frauduloase (de exemplu prin hardcodare).
- Ultima temă submisă pe Moodle poate fi rulată de către responsabili/echipa de PCLP de mai multe ori în vederea verificării faptului că nu aveți buguri în sursă. În cazul obținerii punctajelor diferite la rulări multiple, vom păstra minimul dintre punctaje. Vă recomandăm să verificați local tema de mai multe ori pentru a verifica că punctajul este mereu același, apoi să încărcați tema. De asemenea, verificați că punctajul de pe platformă este cel așteptat - singurul punctaj / feedback relevant este cel de pe platforma remote PCLP.