

## Tema 2 - Stive, cozi, liste înlănțuite

1. **Implementare stivă sau coadă.** Să se implementeze o stivă (sau o coadă) utilizând liste înlănțuite. Vă trebuie:

- o structură NOD cu două câmpuri - un câmp pentru informație și un câmp de tip pointer la NOD pentru legătura la elementul următor.
- o structură STIVA (sau COADA) cu
  - un câmp de tip pointer la NOD = pointer la vârful stivei. Trebuie inițializat cu NULL. (Două câmpuri de tip pointer la nod, pentru primul și ultimul element în cazul cozii).
  - un câmp SIZE de tip int - numărul de elemente din stivă (coada)
  - funcția PUSH(elem) - pune *elem* în vârful stivei (la sfarsitul cozii)
  - funcția POP( ) - elimină elementul din vârful stivei (de la începutul cozii)
  - funcția TOP( ) - returnează valoarea aflată în vârful stivei. (FRONT() și BACK() pentru coadă)
  - funcția EMPTY( ) - verifică dacă stiva este vidă. (coada)
  - funcția CLEAR( ) - golește stiva (coada)

NU se utilizează o funcție de afișare a elementelor unei stive!!!

În funcția main a programului se va declara o stiva (coada). Apoi se vor introduce  $n$  elemente, cu  $n$  citit de la tastatură. Aceste elemente se extrag apoi pe rând și se afișază în ordinea de extragere. (1p)

2. **Parantezare corectă:** Se dă un șir de paranteze deschise și închise de tip (, ), [, ], {, }. Să se verifice dacă șirul este corect. **Exemplu:** șirul `[()()]` este corect, șirul `[()]` nu este corect, șirul `()]` (nu este corect. (1p)

3. **Evaluarea expresiilor aritmetice.** Etape:

- Se citește dintr-un fișier o expresie aritmetică alcătuită din numere întregi fără semn, operatorii aritmetici +, -, \*, /, ^ (reprezentând ridicare la putere) și paranteze rotunde. Atenție: algoritmul trebuie să funcționeze

corect și dacă există caracterele albe în expresie (spații, tab-uri) și dacă NU există! Spațiile albe se ignoră la analizarea expresiei.

- Să se construiască forma poloneză postfixată pentru expresia aritmetică dată și să se afișeze. (1p)
- Să se evalueze expresia și să se afișeze rezultatul. (1p)
- Să se semnaleze erori în expresie (de parantezare, de operatori, de caractere nepermise) (1p)
- Pentru funcționarea algoritmului și cu numere întregi de mai multe cifre - 1p
- Pentru funcționarea algoritmului și cu numere reale - 1p

Utilizați stive. Se poate folosi *stack* din STL. Atfel utilizați o structură STIVA!

### Exemple:

- $2 + + + 3$  nu este o expresie corectă. Programul semnalează că există prea mulți operatori.
- $2 * ((3 + 4)$  nu este o expresie corectă. Programul semnalează faptul că parantezarea nu este corectă.  
**Observație:** chiar dacă numărul de paranteze deschise este egal cu cel de paranteze închise, parantezarea poate să nu fie corectă. De exemplu  $(((($  sau  $)))(($ .
- $23\# + a$  nu este o expresie corectă. Programul semnalează faptul că apar caractere nepermise  $\#$ ,  $a$ .
- $3 * 4 - 3 * (24 - 12) - 7$ . Programul returnează valoarea  $-31$ . Forma poloneză postfixată este  $34 * 32412 - * - 7-$

Algoritmul este descris în documentația de pe e-learning.

4. **Eliminarea parantezelor redundante.** Se citește dintr-un fișier un șir de caractere reprezentând o expresie aritmetică formată din numere, operatori (+, -, /, \*), litere (reprezentând variabile), paranteze rotunde, eventual spații albe. Se cere eliminarea parantezelor redundante. (1p)

**Exemplu:** În expresia:  $((x + y) * 3 + ((z)))$  setul de paranteze care delimitează  $z$  din expresia  $((z))$  este redundant. Rezultatul algoritmului ar trebui să fie:  $((x + y) * 3 + z)$ .

5. Să se implementeze o coadă circulară. Pentru aceasta se cere crearea unei structuri COADA care să conțină:

- un câmp de tip vector de int numit DATA, care stochează elementele din coadă
- un câmp SIZE\_MAX de tip int, care reprezintă capacitatea maximă a cozii
- două câmpuri de tip int BEGIN = poziția primului element și END - poziția de după ultimul element din coadă.
- funcția PUSH(elem) - pune *elem* în coadă
- funcția POP() - elimină un element din coadă
- funcția FRONT( ) - returnează valoarea aflată la începutul cozii
- funcția EMPTY( ) - verifică dacă este vidă sau nu coada.
- funcția CLEAR( ) - golește coada

În funcția main se declară o variabilă de tip COADA, se inserează pe rând  $n$  elemente, cu  $n$  citit de la tastatură, apoi se extrag aceste elemente pe rând și se afișază în ordinea extragerii. (1p)

6. **Bucket-Sort.** Să se implementeze algoritmul de sortare *Bucket-sort* utilizând liste dublu înlanțuite. Utilizați o structură SORTED\_LIST care să aibă:

- membrii FIRST și LAST (de tip pointer la NOD) = pointeri către primul, respectiv ultimul element din listă
- funcția INSERT(key) - inserează o *key* în listă pe poziția potrivită, așa încât lista să rămână sortată .
- MERGE(L) - unește lista curentă cu lista L, așa încât elementele să rămână sortate și setează primul și ultimul element al liste L la NULL
- CLEAR( ) - golește lista.
- EMPTY( ) - verifică dacă lista e vidă
- PRINT( ) - afișază elementele stocate în listă.

Structura NOD utilizată în lista dublu înlanțuită trebuie să aibă câmpurile INFO, NEXT și PREV.

Algoritmul poate fi găsit în cartea "Introduction to Algorithms third edition" de T.C. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (2p)

7. **Implementarea unei liste simplu înlanțuite.** Să se implementeze o listă simplu înlanțuită cu funcționalitățile descrise în continuare. Se cere utilizarea unei structuri NOD care are două câmpuri: un câmp int pentru informație și un câmp de tip pointer la NOD pentru legătura către următorul element. Se cere utilizarea unei structuri LISTA care are ca membru un pointer la NOD reprezentând capul listei și metodele:

- membrii FIRST, LAST = pointeri la NOD reprezentând adresele primului, respectiv ultimului elem din listă
- un membru NR\_ELEM de tip int = numărul de elemente din listă
- funcția PUSH\_FRONT(key) - adaugă cheia *key* în capul listei (0.20 p)
- funcția PUSH\_BACK(key) - adaugă cheia *key* la finalul listei (0.20 p)
- funcția SEARCH(key) - caută o cheie *key* în listă - returnează pointer la nodul cu cheia *key* sau NULL (0.20 p)
- funcția ERASE(node) -șterge un element *node* din listă (NU implică căutare) (0.20 p)
- funcția ERASE(key) - șterge cheia *key* (implică căutare) (0.20 p)
- funcția EMPTY( ) - verifică dacă lista e vidă (0.20 p)
- funcția ERASE\_DOUBLES( ) - elimină dublurile din listă (0.5 p) (punctaj suplimentar pentru complexitate bună)
- funcția ERASE(listă L) - elimină din lista elementele comune cu cele din *L* (0.5 p) (punctaj suplimentar pentru complexitate bună)
- funcția CLEAR( ) - golește lista. (0.20 p)
- funcția PRINT( ) - afișează elementele listei (0.20 p)
- o funcționalitate proprie cu descrierea ei (între 0.25 și 1 punct în funcție de originalitate, complexitate și aplicabilitate)

În funcția *main* realizați un menu cu ajutorul unei instrucțiuni *switch*, prin care se oferă opțiuni, corespunzătoare fiecărei funcționalități, precum și o opțiune de EXIT. Într-o instrucțiune *while*, se citesc și se execută opțiuni până la alegerea opțiunii de EXIT.

**ATENȚIE:** Nici o funcție nu trebuie să dea eroare de execuție, dacă se apelează pe o listă vidă!!!

8. **Problema componentelor conexe:** Se consideră o imagine de dimensiuni  $Height \times Width$  reprezentată printr-o matrice de numere întregi. Imaginea conține obiecte albe (reprezentate prin poziții marcate cu 1) pe un fundal negru (elementele de fundal sunt marcate cu 0). Se consideră obiect, toate elementele albe (val 1) conectate între ele. Două poziții  $(x_1, y_1)$  și  $(x_2, y_2)$  (ambele marcate cu 1) sunt conectate dacă se poate ajunge de la  $(x_1, y_1)$  la  $(x_2, y_2)$  prin vecini marcați cu 1. Se consideră vecinii orizontali, verticali și diagonali. Un exemplu este dat în figura 1 (elementele marcate cu 0 sunt gri închis, cele marcate cu 1, adică obiectele, sunt marcate cu alb). În figură 1 punctele  $p_1$  și  $p_2$  sunt conectate, dar  $p_1$  și  $p_3$  nu sunt conectate. Imaginea conține în total 4 obiecte.

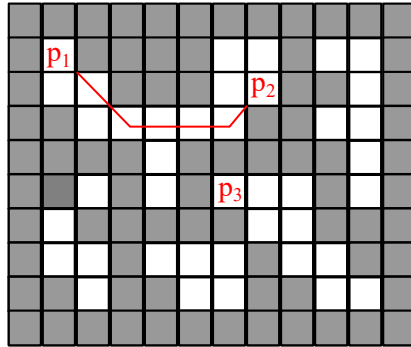


Figure 1: Exemplu de imagine conținând 4 obiecte. În matricea corespunzătoare pozițiile gri sunt 0, pozițiile albe sunt 1.

- Determinați numărul de obiecte din imagine. (1p)
- Determinați obiectul de suprafață maximă. Suprafața unui obiect este dată de numărul de puncte componente. (0.5 p)
- Separați obiectele în modul următor: pentru fiecare obiect se construiește o matrice nouă, care să conțină exact obiectul respectiv. În figura 2 este reprezentat rezultatul obținut asupra imaginii din fig 2. Aceste matrice se scriu în fișiere separate. (2.5p)

În figura 2 este ilustrată soluția pentru punctele a, b și c.  
Datele de intrare se citesc din fișier!

### Observații:

- NU utilizați funcții recursive! Puteți folosi *queue* din STL
- Pentru rezolvarea eficientă cu seturi disjuncte se oferă pentru punctele (a+b) 5 puncte.

- Problema labirintului:** Se consideră un labirint reprezentat printr-o matrice în care zidurile sunt marcate prin  $-1$  și drumurile prin  $0$ . În labirint se află un șoricel pe poziția  $(x_0, y_0)$  și o bucată de brânză pe poziția  $(x_1, y_1)$ . Să se afișeze un drum (de preferință de lungime minimă) de la șoricel la brânză. Se cere atât afișarea matricii în care se marchează drumul parcurs de șoricel, cât și un șir de perechi  $(x_i, y_i) =$  coordonatele căsuțelor ce reprezintă a drumul. Datele de intrare se citesc din fișier. **Nu utilizați recursivitate! Utilizați o coadă.** (2p)

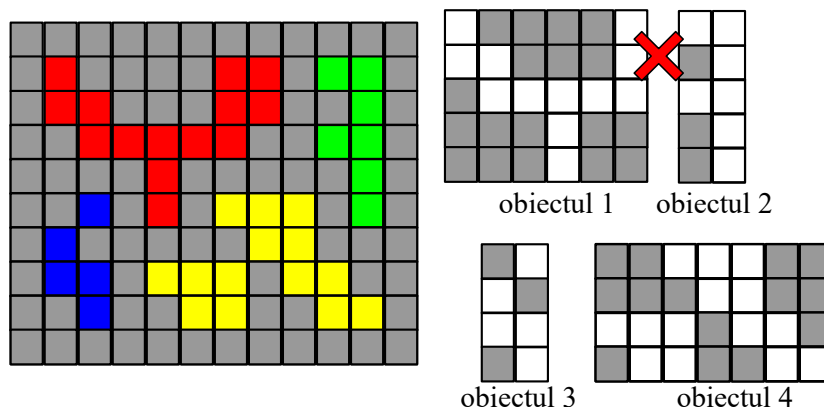


Figure 2: a. Obiectele colorate cu culori diferite. Sunt 4 obiecte. b. Obiectele cu suprafaa maximă sunt cel roșu și cel galben, ambele cu 14 căsuțe (pixeli). c. Dreptunghiurile reprezentând matricile corespunzătoare obiectelor.

### Evaluare:

- Nu este permisă utilizarea funcțiilor recursive!
- La problemele 2, 8 și 9 se permite utilizarea din STL a claselor *stack*, *queue*, *pair*, *tuple*
- Se permite folosirea containerului *vector* din STL, cu atașarea documentației aferente, dacă aceasta nu a fost deja atașată la tema 1.