Tema 2 - Stive, cozi, liste înlănţuite

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

   - o structură NOD cu două câmpuri - un câmp pentru informaţie şi 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 iniţializat cu NULL. (Două câmpuri de tip pointer la nod, pentru primul şi ultimul element în cazul cozii).
     - un câmp SIZE de tip int - numărul de elemente din stivă (coada)
     - funcţia PUSH(elem) - pune *elem* în vârful stivei (la sfarsitul cozii)
     - funcţia POP( ) - elimină elementul din vârful stivei (de la începutul cozii)
     - funcţia TOP( ) - returnează valoarea aflată în vârful stivei. (FRONT() şi BACK() pentru coadă)
     - funcţia EMPTY( ) - verifică dacă stiva este vidă. (coada)
     - funcţia CLEAR( ) - goleşte stiva (coada)

   NU se utilizează o funcţie de afişare a elementelor unei stive!!!

   În funcţia 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 şi se afişază în ordinea de extragere. (1p)

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

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

   - Se citeşte dintr-un fişier o expresie aritmetică alcătuită din numere întregi fără semn, operatorii aritmetici $+$, $-$, $*$, $/$, $\hat{}$ (reprezentând ridicare la putere) şi paranteze rotunde. Atenţie: algoritmul trebuie să funcţioneze

corect şi dacă există caracterele albe în expresie (spaţii, tab-uri) şi dacă NU există! Spaţiile albe se ignoră la analizarea expresiei.

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

Utilizaţi stive. Se poate folosi *stack* din STL. Atfel utilizaţi o structură STIVA!

Exemple:

- $2 + + + 3$ nu este o expresie corectă. Programul semnalează că există prea mulţi operatori.
- $2 * ((3 + 4)$ nu este o expresie corectă. Programul semnalează faptul că parantezarea nu este corectă.
  Observaţie: 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 documentaţia de pe e-learning.

4. **Eliminarea parantezelor redundante**. Se citeşte dintr-un fişier un şir de caractere reprezentând o expresie aritmetică formată din numere, operatori (+, - /, *), litere (reprezentând variabile), paranteze rotunde, eventual spaţii albe. Se cere elminarea 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 sa fie: $((x + y) * 3 + z)$.

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

- 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 = poziţia primului element şi END - poziţia de după ultimul element din coadă.
- funcţia PUSH(elem) - pune *elem* în coadă
- funcţia POP() - elimină un element din coadă
- funcţia FRONT( ) - returnează valoarea aflată la începutul cozii
- funcţia EMPTY( ) - verifică dacă este vidă sau nu coada.
- funcţia CLEAR( ) - goleşte coada

În funcţia 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 şi se afişază în ordinea extragerii. (1p)

6. **Bucket-Sort**. Să se implementeze algoritmul de sortare *Bucket-sort* utilizând liste dublu înlănţuite. Utilizaţi o structură SORTED_LIST care să aibă:

   - membrii FIRST şi LAST (de tip pointer la NOD) = pointeri către primul, respectiv ultimul element din listă
   - funcţia INSERT(key) - inserează o *key* în listă pe poziţia potrivită, aşa încât lista să rămână sortată .
   - MERGE(L) - uneşte lista curentă cu lista L, aşa încât elementele să rămână sortate şi setează primul şi ultimul element al liste L la NULL
   - CLEAR( ) - goleşte lista.
   - EMPTY( ) - verifică dacă lista e vidă
   - PRINT( ) - afişază elementele stocate în listă.

   Structura NOD utilizată în lista dublu înlănţuită trebuie să aibă câmpurile INFO, NEXT şi 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 înlănţuite**. Să se implementeze o listă simplu înlănţuită cu funcţionalităţile descrise în continuare. Se cere utilizarea unei structuri NOD care are două câmpuri: un câmp int pentru informaţie şi un câmp de tip pointer la NOD pentru legătura către următorul element. Se cere utilizarea unei structuri LISTA care are ca menbru un pointer la NOD reprezentând capul listei şi 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
- funcţia PUSH_FRONT(key) - adaugă cheia *key* în capul listei (0.20 p)
- funcţia PUSH_BACK(key) - adaugă cheia *key* la finalul listei (0.20 p)
- funcţia SEARCH(key) - caută o cheie *key* în listă - returnează pointer la nodul cu cheia *key* sau NULL (0.20 p)
- funcţia ERASE(node) -şterge un element *node* din listă (NU implică căutare) (0.20 p)
- funcţia ERASE(key) - şterge cheia *key* (implică căutare) (0.20 p)
- funcţia EMPTY( ) - verifică dacă lista e vidă (0.20 p)
- funcţia ERASE_DOUBLES( ) - elimină dublurile din lisă (0.5 p) (punctaj suplimentar pentru complexitate bună)
- funcţia ERASE(lista L) - elimină din lista elementele comune cu cele din L (0.5 p) (punctaj suplimentar pentru complexitate bună)
- funcţia CLEAR( ) - goleşte lista. (0.20 p)
- funcţia PRINT( ) - afişează elementele listei (0.20 p)
- o funcţionalitate proprie cu descrierea ei (intre 0.25 şi 1 punct în funcţie de originalitate, complexitate şi aplicabilitate)

În funcţia *main* realizaţi un menu cu ajutorul unei instrucţiuni *switch*, prin care se oferă opţiuni, corespunzătoare fiecărei funcţionalităţi, precum şi o opţiune de EXIT. Într-o instrucţiune *while*, se citesc şi se execută opţiuni până la alegerea opţiunii de EXIT.

ATENŢIE: Nici o funcţie nu trebuie să dea eroare de execuţie, 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 conţine obiecte albe (reprezentate prin poziţii 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 poziţii (x_1, y_1) şi (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 marcaţi cu 1. Se consideră vecinii orizontali, verticali şi 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 şi p_2 sunt conectate, dar p_1 şi p_3 nu sunt conectate. Imaginea conţine în total 4 obiecte.

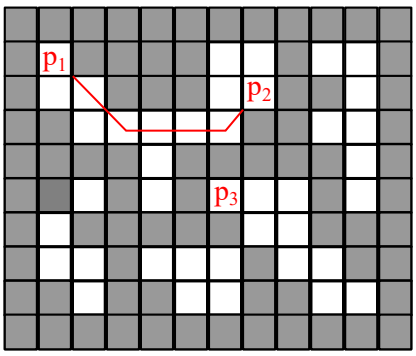

Figure 1: Exemplu de imagine conţinând 4 obiecte. În matricea corespunzătoare poziţiile gri sunt 0, poziţiile albe sunt 1.

a. Determinaţi numărul de obiecte din imagine. (1p)

b. Determinaţi obiectul de suprafaţă maximă. Suprafaţa unui obiect este dată de numărul de puncte componente. (0.5 p)

c. Separaţi obiectele în modul următor: pentru fiecare obiect se construieşte o matrice nouă, care să conţină exact obiectul respectiv. În figura 2 este reprezentat rezultatul obţinut asupra imaginii din fig 2. Aceste matrice se scriu în fişiere separate. (2.5p)

În figura 2 este ilustrată soluţia pentru punctele a, b şi c.
Datele de intrare se citesc din fişier!

Observaţii:

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

9. **Problema labirintului**: Se consideră un labirint reprezentat printr-o matrice în care zidurile sunt marcate prin -1 şi drumurile prin 0. În labirint se află un şoricel pe poziţia (x_0, y_0) şi o bucată de brânză pe poziţia (x_1, y_1). Să se afişeze un drum (de preferinţă de lungime minimă) de la şoricel la brânză. Se cere atât afişarea matricii în care se marchează drumul parcurs de şoricel, cât şi un şir de perechi (x_i, y_i) = coordonatele căsuţelor ce reprezint a drumul. Datele de intrare se citesc din fişier. **Nu utilizaţi recursivitate! Utilizaţi o coadă**. (2p)

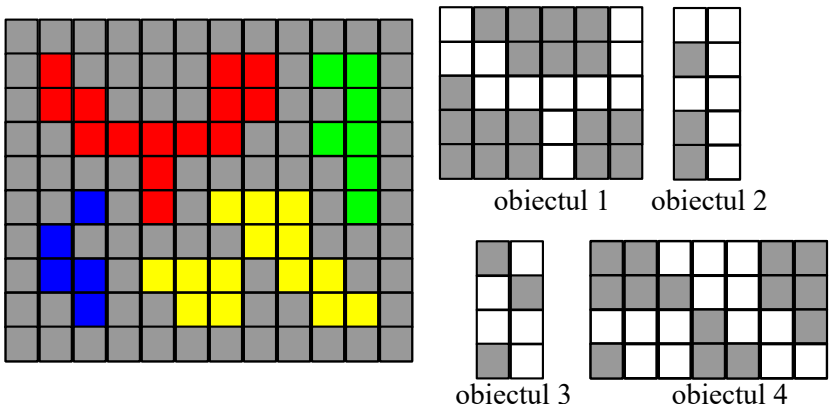

Figure 2: a. Obiectele colorate cu culori diferite. Sunt 4 obiecte. b. Obiectele cu suprafaa̧ maximă sunt cel roşu şi cel galben, ambele cu 14 căsuţe (pixeli). c. Dreptunghiurile reprezentând matricile corespunzătoare obiectelor.

Evaluare:

- Nu este permisă utilizarea funţiilor recursive!
- La problemele 2, 8 şi 9 se permite utilizarea din STL a claselor *stack*, *queue*, *pair*, *tuple*
- Se permite folosirea containerului *vector* din STL, cu ataşarea documentaţiei aferente, dacă aceasta nu a fost deja ataşată la tema 1.