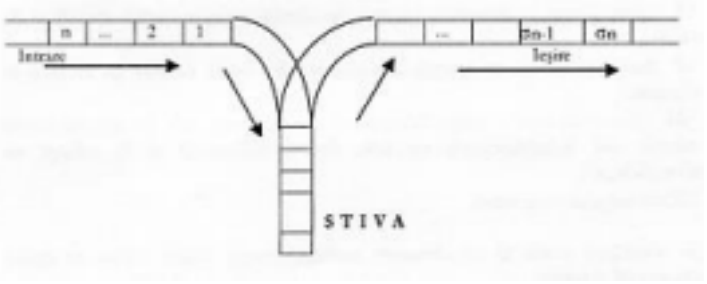
Structuri de date
Tema 3

1. Fie S o stivă de numere întregi şi x o variabilă întreagă. Folosind operaţiile standard pe stivă (Push, Pop, isEmpty, isFull) scrieţi cîte o metodă care să efectueze următoarele operaţii: a) citeşte un element din vârful stivei, fără a-l şterge; dacă S este vidă, x=MAXINT; b) citeşte în x al treilea element din vârful stivei, dacă există cel puţin trei elemente; dacă nu, x=MAXINT;
   c) citeşte în x elementul de la baza stivei (x=MAXINT dacă stiva este goală) şi lasă stiva neschimbată;
   d) şterge toate elementele egale cu x din stivă, lăsând celelalte elemente nealterate.

2. Uneori, un program necesită două stive conţinând acelaşi tip de elemente. Dacă cele două stive sunt memorate în vectori diferiţi, atunci o stivă poate să depăşească maximul capacităţii în timp ce cealaltă stivă are mult spaţiu nefolosit. O cale de a evita această situaţie este să punem ambele stive "împreună" într-un vector şi să permitem unui vector să crească de la un capăt, iar celuilalt să crească de la celălalt capăt, ambele stive crescând din direcţii diferite. În acst fel, dacă o stivă se umple prea tare şi cealaltă are spaţiu liber, atunci ele pot să coexiste fără să apară vreo eroare. Declaraţi şi implementaţi o nouă clasă numită **DoubleStack** care să conţină doi indici *topA* şi *topB* şi metodele *PushA ,PushB, PopA, PopB* care gestionează cele două stive.

3. Scrieţi un program care utilizează o stivă pentru a citi un întreg şi tipăreşte toţi divizorii săi primi.

4. O stivă poate fi utilizată ca o linie de cale ferată ce schimbă sensul ca în figura de mai jos.



Vagoanele numerotate 1,2,...,n sunt puse pe linia de intrare în această ordine. Să se demonstreze, că folosind o stivă de capacitate n se poate obţine la ieşire orice permutare $\sigma \in S_n$ a vagoanelor. Pentru o permutare dată σ, să se scrie o funcţie care afişează succesiunea operaţiilor care au loc.

5. Să se scrie o funcţie care citeşte de la tastatură o linie de caractare care se presupune că este compusă din două părţi, separate între ele prin "<". Ca rezultat funcţia trebuie să returneze: - 1 dacă nu există "<" în şirul de intrare;
   - 2 dacă partea dinaintea lui "<" este mai lungă decât cea de după "<" ;
   - 3 dacă partea dinaintea lui "<" este mai scurtă decât cea de după "<" ;
   - 4 dacă părţile stânga şi dreapta a lui "<" au aceiaşi lungime dar sunt diferite; - 0 dacă partea stângă este identică partea dreaptă.

Se va folosi ocoadă circulară pentru rezolvarea problemei.

6. Implementaţi clasa **Scroll,** care este o structură de date intermediară între stivă şi coadă. Într-un *Scroll*, toate intrările se fac la sfârşitul listei însă ştergerile se pot face la ambele capete.

7. Scrieţi forma poloneză inversă a următoarelor expresii:
   a) A*B*C+D*E
   b) A+B-C+D
   c) A*B+C*D+E*F*G
   d) A-B+C-D*E

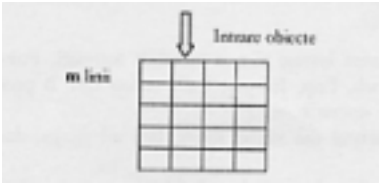
8. Scrieţi un algoritm care transformă o expresie din formă prefixată în formă postfixată. Analizaţi complexitatea lui.

9. Considerăm un aeroport mic, cu o singură pistă. La fiecare moment poate ateriza sau decola un singur avion, prioritate având avioanele care vor să aterizeze. Se pune problema simulării traficului în aeroport. Pentru aceasta vom construi două cozi „de aşteptare", una pentru avioanele ce aşteaptă să aterizeze, alta pentru cele care vor să decoleze. Scrieţi un program în limbajul C care să conţină două funcţii concurente care gestionează cele două cozi asociate.

10. Se dau n+1 tije dintre care n sunt ocupate şi una este liberă. Fiecare tijă din cele ocupate are capacitate de exact n bile. Fiecare tijă conţine exact n bile identice, de aceiaşi culoare (fiecare tijă conţine câte o culoare diferită). Ştiind că pe o tijă nu pot exista mai mult de n bile să se scrie o funcţie care arată şirul mutărilor astfel încât, folosind tija liberă, să se ajungă în final la situaţia în care pe fiecare tijă există exact n bile, toate de culori diferite.

11. (Jocul Tetris) Se consideră următoarele figuri geometrice plane:
    a) pătratul de latură 1;
    b) pătratul de latură 2;
    c) figura L;
    d) figura Z;

Se consideră o coadă în care sunt introduse obiecte din cele patru tipuri enunţate anterior şi din care se extrag obiecte pe rând, astfel încât la fiecare pas să se ştie obiectul curent de introdus în stivă şi obiectul care îl urmează. Dându-se un caroiaj dreptunghiular de dimensiune **m x n** compus din pătrate elementare ca cel de mai jos şi o coadă de obiecte, se cere să se dispună obiectele scoase din coadă şi introduse pe sus (ca o stivă) în caroiaj (vezi figura de mai jos) în aşa fel încât să nu existe obiect apărut mai devreme şi să fie peste unul apărut mai târziu, iar în final numărul de linii completate cu obiecte să fie maxim.



12. Se consideră un careu dreptunghiular M de **m x n** celule care au valoarea 0 sau 1. Definim acţiunea *Reverse(i,j,k)* care inversează lista valorilor celulelor de pe linia i de la coloana j la

coloana k astfel încât M(i,j) ⇔ M(i,k), M(i,j+1) ⇔ M(i,k+1), etc. Se cere: i. Să se scrie o funcţie care implementează procedura *Reverse*.

ii. Pornind de la o configuraţie dată a careului, să se scrie o secvenţă de execuţii ale procedurii *Reverse* care are ca rezultat faptul că toate celulele cu valoarea 1 se vor grupa în colţul din stânga sus.

13. Să se scrie o funcţie care interclasează valorile presupuse sortate din două stive S1 şi S2 şi depune rezultatul într-o coadă Q.

14. Să se scrie un program care afişează un prompter „>" şi citeşte de la acest prompter linii de comenzi, după care afişează rezultatul lor după cum urmează:
>**S** *identificator* ⇒ creează o stivă care va fi referită cu numele dat de identificator; >**Q** *identificator* ⇒ creează o coadă care va fi referită cu numele dat de identificator; >**D** *identificator* ⇒ şterge conţinutul şi variabilele referite de identificator; >**A** *identificator listă_valori* ⇒ adaugă în identificatorul specificat valorile din listă; >**X** *identificator număr* ⇒ extrage din identificator un număr specificat de valori; >**P** *identificator* ⇒ extrage din identificator toate valorile pe rând şi le afişează; >**M** *identificator1 identificator2 identificator3* ⇒ concatenează valorile din identificator1 cu cele din identificator2 şi le adaugă în identificator3;
>**E** ⇒ termină programul.

15. Se consideră coada **Q** cu elemente numere întregi asupra căreia se aplică algoritmul următor:

```
While(!Q.SqueueEmpty())
{
        Q.SqueueDelete(x);
        if(!Q.SqueueEmpty())
                {
                        Q.SqueueDelete(y);
                        Q.SqueueAdd(x+y);
                }
        else
                return x;
}
```

a) Ce returnează algoritmul anterior pentru coada Q? Justificaţi răspunsul.
b) Pentru o coadă dată de n elemente, în câţi paşi se termină algoritmul?

16. Arătaţi cum se poate implementa o coadă prin două stive. Analizaţi timpul de execuţie pentru operaţiile cozii.

17. Arătaţi cum se poate implementa o stivă prin două cozi. Analizaţi timpul de execuţie pentru operaţiile stivei.