Stiva(stack) , coada(queue)

Structuri implementate la nivel logic(conceptual) pentru care se definesc următoarele operații:

- inițializarea structurii
- verificarea dacă structura este vidă
- verificarea dacă structura este plină
- adăugare a unui element (**push**)
- vizualizarea elementului din capul structurii (**top** , **front**)
- eliminare a unui element (**pop**)

cu respectarea principiilor aferente stivei și cozii:

stiva **LIFO (Last In First Out)**

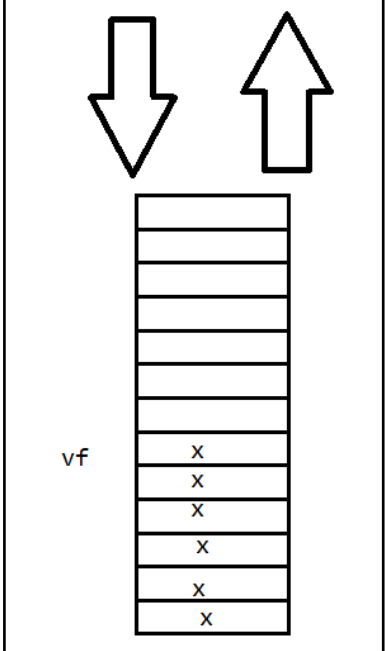
coada **FIFO (First In First Out)**

Implementarea:

- static (vectori)
- dinamic (liste alocate dinamic)
- containere STL

STIVA

* Apelul funcțiilor, recursivitatea, folosesc memoria de tip stivă.




```
const int capacitate = 100;
int st[capacitate], vf;

vf = 1; // Inițializarea stivei (stivă vidă)

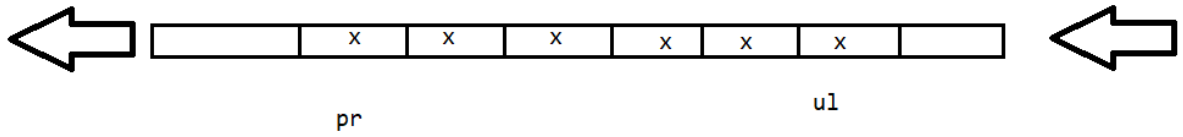
vf == 0 // stivă vidă
vf > 0 // stivă ne-vidă

st[++vf] = val ; // adăugare element - push(val)

vf --;// eliminarea unui element – POP

var = st[vf] ; // identificarea valoare din vârf – TOP
```

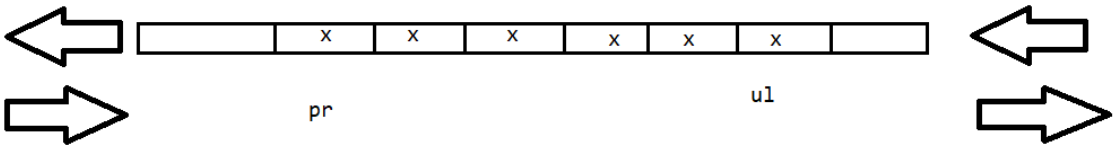
COADA




```
const int capacitate = 1000;
int c[capacitate], pr, ul;
```

```
pr = 1 , ul = 0;
```**// inițializarea cozii**

```
pr > ul //
```**coadă vidă**

```
pr <= ul //
```**coadă ne-vidă**

```
c[++ul] = val ;//
```**adăugare element – push(val)**

```
ul ++; //
```**Eliminarea unui element – pop**

```
c[pr] //
```**Identificarea valorii de la începutul cozii – front**

*Coada dublă - operațiile de push/pop/ top se realizează pe la ambele capete



#875

Să se scrie un program care gestionează o stivă de numere întregi. Inițial stiva este vidă. Programul va citi de la tastatură o listă de operații, care pot fi:

push X – adaugă valoarea întreagă X pe stivă;

pop – elimină elementul din vârful stivei;

top – afișează elementul din vârful stivei.

Programul va realiza asupra stivei operațiile citite, în ordine. Afișările se fac pe ecran, câte o valoare pe linie.

Date de intrare:Programul citește de la tastatură un număr n, apoi cele n operații,

Date de ieșire:Programul va afișa pe ecran numere rezultat ale operațiilor top, câte unul pe o linie.

Restricții și precizări : stiva nu va conține la un moment dat mai mult de 1000 elemente; elementele stivei vor fi cuprinse între -1.000.000 și 1.000.000; dacă la un moment dat se aplică operația pop și stiva nu conține niciun element, operația nu are efect;

dacă la un moment dat se aplică operația top și stiva nu conține niciun element, operația nu are efect (nu se afișează nimic)

Exemplu

Intrare

12

push 5

push 4

top

push 7

push -5

pop

top

pop

top

push 11

top

pop

Ieșire

4

7

4

11

```
#include<cstdio>
const int N=1001;
int st[N];
char s[6];
int main()
{
  int n,vf=0,x;
  scanf("%d ",&n);/// cin>>n;
  for(int i=1;i<=n;i++)
  {
    scanf("%s ",&s); /// cin>>s;
    if(s[1]=='u')
        { scanf("%d ",&x); /// cin>>x;
          st[++vf]=x;
          }
    else
    {
      if(vf!=0)
      {
        if(s[1]=='o' && s[0]=='p') vf--;
        else printf("%d\n",st[vf]);/// cout<<st[vf];
      }
    }
  }
  return 0;
}
```

#876

Să se scrie un program care gestionează o coadă de numere întregi. Inițial coada este vidă. Programul va citi de la tastatură o listă de operații, care pot fi:

push X – adaugă valoarea întreagă X în coadă;

pop – elimină elementul din coadă;

front – afișează elementul de la începutul cozii.

Programul va realiza asupra cozii operațiile citite, în ordine. Afișările se fac pe ecran, câte o valoare pe linie.

Date de intrare: Programul citește de la tastatură un număr n, apoi cele n operații,

Date de ieșire: Programul va afișa pe ecran numerele rezultat ale operațiilor front, câte unul pe o linie.

Restricții și precizări

coada nu va conține la un moment dat mai mult de 1000 elemente

elementele cozii vor fi cuprinse între -1.000.000 și 1.000.000

dacă la un moment dat se aplică operația pop și coada nu conține niciun element, operația nu are efect

dacă la un moment dat se aplică operația front și coada nu conține niciun element, operația nu are efect (nu se afișează nimic)

Exemplu

Intrare

12

push 5

push 4

front

push 7

push -5

pop

front

pop

front

push 11

front

pop

Ieșire

5

4

7

7

```
#include <cstring>
#include <iostream>
using namespace std;
char op[25];
int n , c[1005] , pr = 1 , ul = 0, X ;
int main()
{
    cin >> n;
    for(int i =1 ; i <= n ; i ++)
    {
        cin >> op;
        if(strcmp(op,"push") == 0)
        {
            cin >> X;
            if(ul <= 1000)
                c[++ul] = X;
            else
            {
                for(int i = pr ; i <= ul ; i ++)
                    c[i - pr + 1] = c[i];
                ul = ul - pr + 1;
                pr = 1;
                c[++ul] = X;
            }
        }
        if(strcmp(op,"pop") == 0)
            if(pr<= ul)
                pr ++;
        if(strcmp(op,"front") == 0)
            if(pr<= ul)
                cout << c[pr] << " ";

    }
    return 0;
```

```
}
```

#877

Gigel are un set de n cuburi. Fiecare cub este marcat cu un număr natural, de la 1 la n și i se cunoaște lungimea laturii – număr natural. Cu o parte dintre aceste cuburi Gigel va construi o stivă, astfel:

fiecare cub se analizează o singură dată, în ordinea numerelor marcate;

dacă stiva nu conține niciun cub, cubul curent devine baza stivei

dacă cubul curent are latura mai mică sau egală cu cubul din vârful stivei, se adaugă pe stivă;

dacă cubul curent are latura mai mare decât cubul din vârful stivei, se vor înlătura de pe stivă cuburi (eventual toate) până când cubul curent are latura mai mică sau egală cu cubul din vârful stivei.

Să se afișeze numerele de pe cuburile existente la final în stivă, de la bază spre vârf.

Programul citește de la tastatură numărul n, apoi n numere naturale, reprezentând, în ordine, lungimile laturilor cuburilor.

Programul va afișa pe ecran numărul de cuburi existente pe stivă, iar pe linia următoare, separate prin câte un spațiu, numerele marcate pe aceste cuburi.

Restricții și precizări: 1 ≤ n ≤ 1000, lungimile cuburilor vor fi mai mici decât 1000

Exemplu

Intrare

6

7 4 3 5 1 2

Ieșire

3

1 4 6

```
#include <cstdio>
using namespace std;
struct cub
{
  int l , poz;
}st[1005];
int main()
{
```

```
    int n , i , much , vf=0;
    scanf("%d" , &n);
    for(i = 1 ; i <= n ; i ++)
    {
      scanf("%d" , &much);
      while(vf > 0 && much > st[vf].l)
        vf --;
      vf++;
      st[vf].l = much;
      st[vf].poz = i;
    }
    printf("%d\n" , vf);
    for(i = 1 ; i <= vf ; i ++)
      printf("%d " , st[i].poz);
    return 0;
}
```

#864

Se dă o matrice cu n linii și m coloane și elemente 0 sau 1, reprezentând planul unui teren în care 0 reprezintă o zonă accesibilă, iar 1 reprezintă o zonă inaccesibilă. O zonă a terenului are ca și coordonate linia și coloana corespunzătoare din matrice. Într-o zonă cunoscută a matricei se află un robot, iar în altă zonă, de asemenea cunoscută, se află o roboțică. Determinați numărul minim de pași prin care robotul va ajunge la roboțică. Dacă nu este posibil ca robotul să ajungă la roboțică, rezultatul va fi -1.

Fișierul de intrare roboti.in conține pe prima linie numerele n m. Următoarele n linii conțin câte m valori, 0 sau 1. Următoarele două linii conțin câte două valori, reprezentând coordonatele robotului, respectiv ale roboțicii.

Fișierul de ieșire roboti.out va conține pe prima linie valoarea cerută.

Restricții și precizări

1 ≤ n , m ≤ 1000

zonele pe care se află inițial cei doi roboți sunt libere și sunt diferite

un pas reprezintă trecerea robotului din zona curentă într-o zonă vecină cu aceasta pe linie sau pe coloană, fără a părăsi matricea.

Exemplu

roboti.in

4 5

1 0 0 0 1

0 0 1 0 0

0 0 0 0 1

1 1 0 0 1

1 2

2 5

roboti.out

4

Explicație

Un traseu al robotului format din 4 pași este evidențiat mai jos.

10001

00100

00001

11001

Există și alte trasee posibile, dar lungimea lor este mai mare.

1

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream fin("roboti.in");
ofstream fout("roboti.out");
int a[1002][1002],x1 , y1 , x2 , y2,n,m;
short x[1000005], y[1000005];
const int dx[]={0,0,1,-1}, dy[]={1,-1,0,0};
int main(){
    fin >> n >> m;
    for(int i = 1 ; i <= n ; i ++)
        for(int j = 1 ; j <= m ; j ++)
            fin >> a[i][j];
    fin >> x1 >> y1 >> x2 >> y2;
    a[x1][y1] == 0; a[x2][y2] == 0;
    for(int i = 1 ; i <= n ; i ++)
        for(int j = 1 ; j <= m ; j ++)
            a[i][j] = - a[i][j];
    for(int i = 0 ; i <= n + 1 ; i ++)
        a[i][0] = a[i][m+1] = -1;
    for(int j = 0 ; j <= m + 1; j ++)
        a[0][j] = a[n+1][j] = -1;
    int st=1 , dr=1;
    x[dr] = x1, y[dr] = y1;
    a[x1][y1] = 1;
    while(st <= dr)
    {
        int i = x[st], j = y[st];
        for(int k = 0 ; k < 4 ; k ++)
        {
            int ii = i + dx[k], jj = j + dy[k];
            if(a[ii][jj] == 0)
            {
                a[ii][jj] = a[i][j] + 1;
                dr ++;
                x[dr] = ii, y[dr] = jj;
            }
        }
        st ++;
    }
```

```
    if(a[x2][y2] == 0)
        fout << -1;
    else
        fout << a[x2][y2] - 1;
    fout.close();
    return 0;
}
```