TABLOURI UNIDIMENSIONALE

1. CONTAINERUL VECTOR

Uneori, pentru a memora șiruri de valori este necesară utilizarea unor structuri speciale de date care să poată reține valorile acestora. Aceste structuri vor fi folosite doar atunci când sunt necesare imperios deoarece sunt consumatoare inutile de memorie. Cu ajutorul containerului *vector* pot fi memorate atât șiruri în care fiecare valoare poate să fie identificată printr-un singur indice (tablou unidimensional), doi indici (tablou bidimensional) sau chiar tablouri multidimensionale.

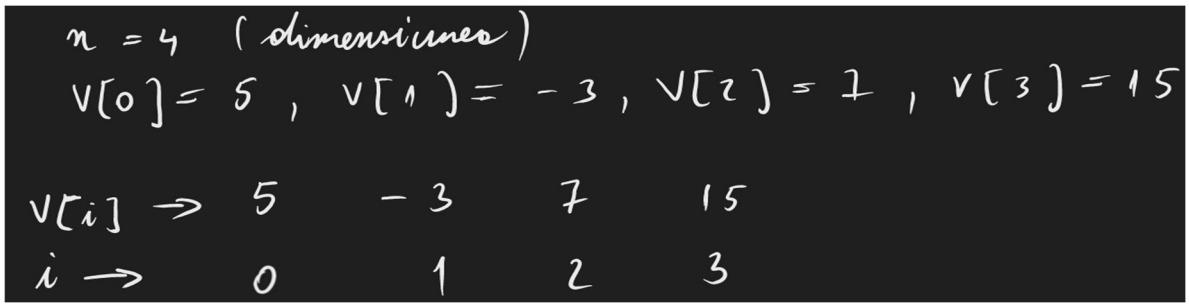
Pentru a le putea utiliza trebuie să fie utilizată biblioteca *vector*, iar unele operații pe vector pot fi efectuate folosind funcții din biblioteca *algorithm* sau *stdlib*. După declarare este necesar, în cele mai multe cazuri sa se aloce memoria necesară. Atunci când se cunoaște cu certitudine dimensiunea tabloului se poate aloca la declarare spațiul de memorie necesar, împreună cu inițializarea cu o valoare, dacă este nevoie.

Mai jos voi prezenta câteva exemple de utilizarea a acestui container.

1. Citesc si afișez un vector cu n elemente numere întregi.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n;
    vector<int> v;//declar un container gol
    cin >> n;//citesc cate elemente va avea tabloul
    v.resize(n);//aloc memorie pentru n elemente
    for(i = 0; i < n; i++)//citesc elementele vectorului
        cin >> v[i];
    for(i = 0; i < n; i++)//afisez elementele vectorului
        cout << v[i] << " ";
}
```

Cum funcționează un astfel de tablou?



Observăm că primul indice alocat pentru vector este 0.

Evident că exemplul de mai sus este efectiv unul de prezentare. Un astfel de container poate fi parcurs prin intermediul unor adrese de memorie la care se află memorate valorile, adrese numite *iteratori*. Pentru a parcurge vectorul cu un iterator, acesta trebuie declarat și va trebuie sa parcurgă vectorul între *v.begin()* și *v.end()*, respectiv adresa primului element și adresa care marchează sfârșitul vectorului. Pentru a afișa valoarea memorata de un iterator este folosit operatorul *.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n;
    vector<int> v;//declar un container gol
    vector<int> :: iterator it; ;//declar iteratorul
    cin >> n;//citesc cate elemente va avea tabloul
    v.resize(n);//aloc memorie pentru n elemente
    for(i = 0; i < n; i++)//citesc elemntele vectorului
        cin >> v[i];
    for(it = v.begin(); it != v.end(); it++)//afisez elementele vectorului
        cout << *it << " ";
}
```

Observați ca accesul la valoarea aflată pe poziția *i* se face folosind *v[i]*, sau **it* pentru valoarea aflată la adresa *it*.

2. Citesc un vector cu n elemente numere întregi. Calculez suma elementelor pare de pe poziții impare.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, s;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
```

```
        cin >> v[i];
    s = 0;
    for(i = 0; i < n; i++)
        if(i % 2 == 1)
            if(v[i] % 2 == 0)
                s += v[i];
    cout << s;
}
```

Tema.

Citesc un vector cu n elemente numere întregi.

- Calculez câte numere negative are vectorul
- Calculez produsul numerelor nenule din vector
- Calculez suma numerelor care au cifra zecilor 3 din vector

3. Citesc un vector cu n elemente numere întregi și un număr întreg x. Afișez DA, dacă x se află în vector sau NU în caz contrar.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, ok, x;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    cin >> x;
    ok = 0;
    for(i = 0; i < n and ok == 0; i++)
        if(x == v[i])
            ok = 1;
    if(ok == 1)
        cout << "DA";
    else
        cout << "NU";
}
```

Pentru rezolvarea problemei se poate folosi funcția *find* din biblioteca *algorithm*, funcție care returnează 0 daca *x* se află în vector sau *1* în caz contrar. Funcția returnează adresa la care găsesc valoarea *x*.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int i, n, ok, x;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    cin >> x;
    if(find(v.begin(), v.end(), x) != v.end())
        cout << "DA";
    else
        cout << "NU";
}
```

4. Citesc un vector cu n elemente numere întregi. Afișez cea mai mare valoare din vector.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, pmax;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    pmax = 0;
    for(i = 0; i < n; i++)
        if(v[pmax] < v[i])
            pmax = i;
    cout << v[pmax];
}
```

Sau

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    int i, n, max;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    max = v[0];
    for(i = 0; i < n; i++)
        if(max < v[i])
            max = v[i];
    cout << max;
}
```

Prima soluție are avantajul că reține atât poziția valorii maxime, cât și valoarea maximă.

5. Citesc un vector cu n elemente numere întregi. Verific dacă vectorul este palindrom. Daca n = 4 si v = {1, 2, 2, 1} atunci v este palindrom.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, ok;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    ok = 1;
    for(i = 0; i < n / 2 && ok == 1; i++)
        if(v[i] != v[n - i - 1])
        ok = 0;
    if(ok == 1)
        cout << "Palindrom";
    else
        cout << "Nu este palindrom";
}
```

sau

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, left, right, ok;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    left = 0;
    right = n - 1;
    ok = 1;
    while(left < right && ok == 1)
        if(v[left] != v[right])
            ok = 0;
        else
            {
                left++;
                right--;
            }
    if(ok == 1)
        cout << "Palindrom";
    else
        cout << "Nu este palindrom";
}
```

Cea de-a doua soluție are avantajul de a folosi tehnica *"two pointers"* prin care vectorul este parcurs simultan din ambele capete folosind doua variabile *left* și *right*. Această tehnică poate fi aplicată și în rezolvarea altor probleme. Următoarea problemă este un astfel de exemplu.

6. Citesc un vector cu n elemente numere întregi. Modificați vectorul astfel încât toate valorile de 0 să apară la început, iar celelalte la sfârșit. Dacă v = {0, 1, 0, 1} se va transforma în v = {0, 0, 1, 1}.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int i, n, left, right;
```

```
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    left = 0;
    right = n - 1;
    while(left < right)
        if(v[left] == 0)
            left++;
        else
            if(v[right] == 0)
                swap(v[left], v[right]);
            else
                right--;
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
}
```

Tema.

Citesc un vector cu n elemente numere întregi.

- Citesc un număr x. Afișez prima poziție pe care apare x în vector
- Citesc un număr x. Afișez ultima poziție pe care apare x în vector
- Afișez poziția pe care apare minimul în vector(vectorul are valori unice)
- Mutați valorile pare din vector la sfârșit și pe cele impare la început

7. Citesc un vector cu n elemente numere întregi. Permutați circular elementele vectorului la stânga cu o poziție.

v = {2, 1, 5, 3} se va transforma în v = {1, 5, 3, 2}

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, t;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    t = v[0];
```

```
    for(i = 0; i < n - 1; i++)
        v[i] = v[i + 1];
    v[n - 1] = t;
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
}
```

8. Citesc un vector cu n elemente numere întregi. Permutați circular elementele vectorului la dreapta cu o poziție.

v = {2, 1, 5, 3} se va transforma în v = {3, 2, 1, 5}

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, t;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    t = v[n - 1];
    for(i = n - 1; i > 0; i--)
        v[i] = v[i - 1];
    v[0] = t;
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
}
```

9. Citesc un vector cu n elemente numere întregi. Afișați toate permutările circulare la dreapta ale vectorului.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, j, n, t;
    vector<int> v;
```

```
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    for(j = 1; j <= n; j++)
    {
        t = v[n - 1];
        for(i = n - 1; i > 0; i--)
            v[i] = v[i - 1];
        v[0] = t;
        for(i = 0; i < n; i++)
            cout << v[i] << " ";
        cout << "\n";
    }
}
```

10. Citesc un vector cu n elemente numere întregi. Afișați toate permutările vectorului.

v = { 4, 5, 1} afisez

4 1 5

5 4 1

5 1 4

1 4 5

1 5 4

4 5 1

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int i, j, n, t;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
        cin >> v[i];
    do
    {
```

```
        for(i = 0; i < n; i++)
            cout << v[i] << " ";
        cout << endl;
    } while (next_permutation(v.begin(), v.end()));
}
```

Funcția *next_permutation* generează următoarea permutare a elementelor vectorului și verifică dacă permutarea a mai fost generată.

Vectorii pot fi construiți din diverse mulțimi de valori și pot avea dimensiuni cunoscute după construire sau necunoscute. Următoarele 2 programe construiesc vectori de dimensiune cunoscută.

Tema.

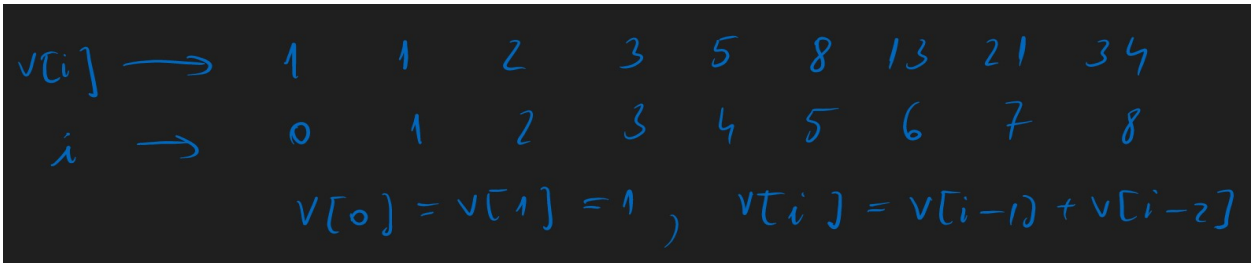
Citesc un vector cu n elemente numere întregi.

- Afișați toate permutările circulare la stânga
- Citesc un număr k. Afișez permutarea circulară la dreapta cu k poziții.

11. Citesc n. Construiesc un vector cu primele n numere pare.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, t;
    vector<int> v;
    cin >> n;
    v.resize(n);
    for(i = 0; i < n; i++)
       v[i] = 2 * i;
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
}
```

12. Citesc n. Construiesc un vector cu primii n termeni ai șirului lui Fibonacci.



```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, t;
    vector<int> v;
    cin >> n;
    v.resize(n);
    v[0] = v[1] = 1;
    for(i = 2; i < n; i++)
       v[i] = v[i - 1] + v[i - 2];
    for(i = 0; i < n; i++)
        cout << v[i] << " ";
}
```

13. Citesc n și un vector x cu n elemente. Construiesc un vector y cu valorile pare din x.

De această dată voi nu voi mai ști câte elemente are y. De aceea nu îi voi mai aloca memorie inițial și voi folosi *push_back* care adaugă elementele unele după altele la sfârșit și alocă automat memorie pentru ele. Numărul de elemente pe care îl are *y* va fi *y.size()*.


```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n;
    vector<int> x, y;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    for(i = 0; i < n; i++)
        if(x[i] % 2 == 0)
```

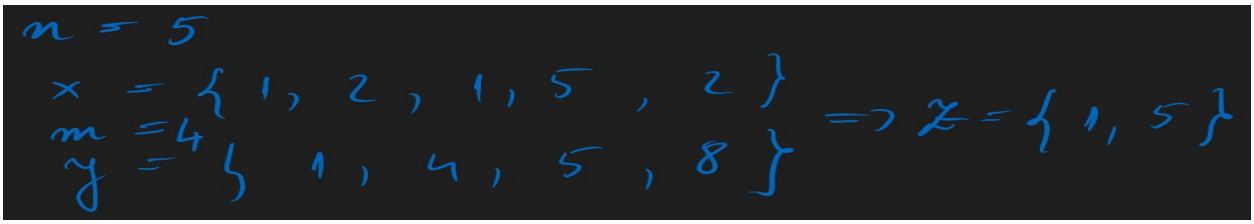
```
            y.push_back(x[i]);
    for(i = 0; i < y.size(); i++)
        cout << y[i] << " ";
}
```

13. Citesc n. Eliminați cifrele impare din n.

Construiesc un vector cu cifrele pare ale lui n și îl afișez invers, pentru că cifrele numărului sunt adăugate de la sfârșit la început sau îl inversez și îl afișez normal.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int i, n;
    vector<int> x;
    cin >> n;
    while(n != 0)
    {
        if(n % 2 == 0)
            x.push_back(n % 10);
        n /= 10;
    }
    //afisare invers
    for(i = x.size() - 1; i >= 0; i--)
        cout << x[i] << " ";
    cout << '\n';
    //inversare vector si afisare normal
    reverse(x.begin(), x.end());//reverse e in algorithm
    for(i = 0; i < x.size(); i++)
        cout << x[i] << " ";
}
```

14. Citesc n și un vector x cu n elemente, apoi m și un vector y cu m elemente. Construiți vectorul z cu elementele comune ale lui x și y.



Observați că în z toate elementele trebuie să fie unice. De aceea după ce am verificat dacă x[i] se află în y, am verificat imediat dacă el nu se află în z, asigurând în acest fel unicitatea elementelor.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
    int i, n, m;
    vector<int> x, y, z;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    cin >> m;
    y.resize(m);
    for(i = 0; i < m; i++)
        cin >> y[i];
    for(i = 0; i < n; i++)
        if(find(y.begin(), y.end(), x[i]) != y.end())//daca x[i] e in y
            if(find(z.begin(), z.end(), x[i]) == z.end())//daca x[i] nu e in z
                z.push_back(x[i]);
    for(i = 0; i < z.size(); i++)
        cout << z[i] << " ";
}
```

O altă modalitate de a asigura unicitatea este folosirea containerului *set*, container care nu permite duplicarea elementelor adăugate în el. Adăugarea elementelor se face cu *insert*, afișarea se face prin utilizarea unui obiect de tipul obiectelor din care este alcătuită mulțimea.

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

int main()
{
    int i, n, m;
    vector<int> x, y;
    set<int> z;
    cin >> n;
```

```
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    cin >> m;
    y.resize(m);
    for(i = 0; i < m; i++)
        cin >> y[i];
    for(i = 0; i < n; i++)
        if(find(y.begin(), y.end(), x[i]) != y.end())//daca x[i] e in y
            z.insert(x[i]);
    for(int element : z)
        cout << element << " ";
}
```

Tema.

Citesc un vector cu x cu n elemente numere întregi.

- Construiți un vector y cu valorile de 2 cifre din x
- Citesc un vector y cu m elemente numere întregi. Construiți un vector(sau o mulțime) z cu elementele unice care se află în x și nu se află în y.
- Citesc un număr n. Construiți un vector x cu reprezentarea în baza 2 a lui n.

Mutați valorile pare din vector la sfârșit și pe cele impare la început

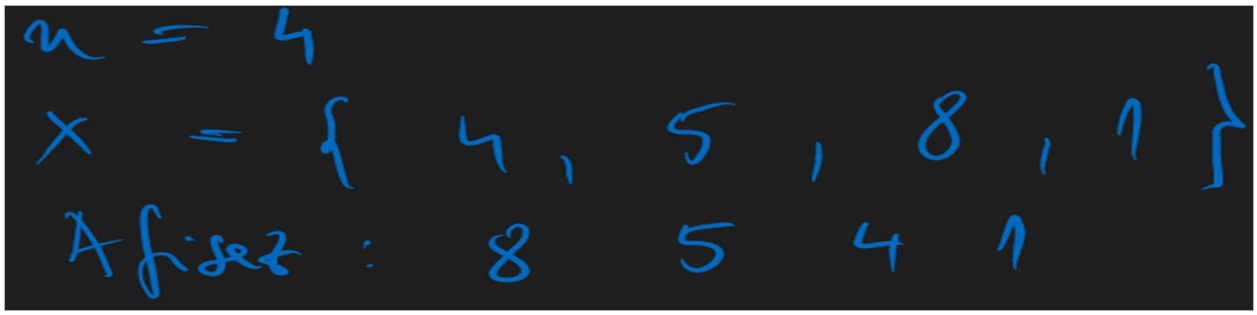
15. Citesc n și un vector x cu n elemente. Ordonați crescător elementele lui x.

Biblioteca *algorithm* conține o funcție care poate ordona în funcție de diverse criterii elementele unui vector. Trebuie precizate adresele între care se realizează sortarea și criteriul după care se realizează sortarea. Implicit, sortarea este crescătoare și foarte interesantă este modalitatea în care se precizează criteriul după care se face sortarea.


```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
```

```
int main()
{
    int i, n, m;
    vector<int> x;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
   sort(x.begin(), x.end());
   for(i = 0; i < n; i++)
      cout << x[i] << " ";
}
```

16. Citesc n și un vector x cu n elemente. Ordonați descrescător elementele lui x.



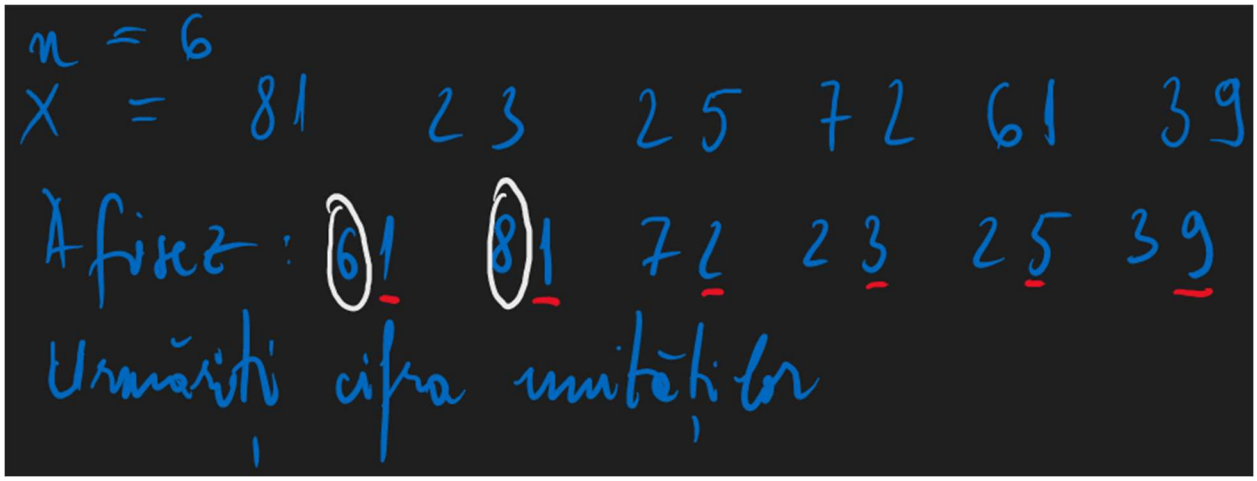
De această dată voi fi obligat să precizez criteriul de sortarea, deoarece implicit, vectorul va fi ordonat crescător. Pentru a realiza acest lucru voi folosi o funcție de tip *bool*, care va returna *true* sau *false* în care voi preciza situația în care două elemente de tipul elementelor vectorului sunt poziționate corect (adică descrescător). Numele funcției va fi adăugat la *sort*.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

bool Criteriu(int a, int b)
{
    return a > b;//asta e ordinea corecta in vector
}

int main()
{
    int i, n, m;
    vector<int> x;
    cin >> n;
    x.resize(n);
```

```
    for(i = 0; i < n; i++)
        cin >> x[i];
   sort(x.begin(), x.end(), Criteriu);//am adaugat criteriul dupa care sortez
   for(i = 0; i < n; i++)
      cout << x[i] << " ";
}
```

Criteriile de sortare pot fi mult mai fine sau mai complicat precizate așa cum este în următoarea problemă.

17. Funny sort : Citesc n și un vector x cu n elemente de 2 cifre. Ordonați crescător elementele lui x în funcție de cifra unităților – cele cu cifra unităților mai mică vor fi la început. Dacă două numere au aceeași cifră a unităților ele vor fi ordonate crescător în funcție de cifra zecilor – cele cu cifra zecilor mai mică vor fi la început.




```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

bool Criteriu(int a, int b)
{
    return a % 10 < b % 10 || a % 10 == b % 10 && a / 10 < b / 10;//asta e
ordinea corecta in vector
}

int main()
{
    int i, n, m;
    vector<int> x;
    cin >> n;
```

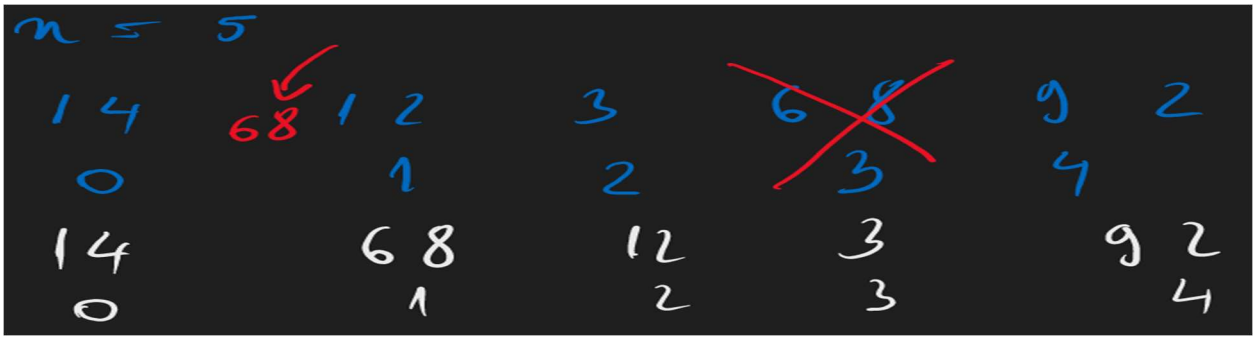
```
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
   sort(x.begin(), x.end(), Criteriu);//am adaugat criteriul dupa care sortez
   for(i = 0; i < n; i++)
      cout << x[i] << " ";
}
```

Tema.

Citesc un vector cu x cu n elemente numere întregi . Ordonați descrescător vectorul astfel în funcție de numărul alcătuit din ultimele două cifre ale fiecărui număr din x.

Un alt tip de problemă este acela în care în vector se șterge sau se inserează o valoare. Pentru aceasta există două metode dedicate: *v.insert* și *v.erase*. **Atenție:** Deoarece prin ștergere se modifică dimensiunea vectorului parcurgerea sau afișarea trebuie făcuta cu *v.size* care se actualizează automat.

18. Citesc un vector cu n elemente, n >= 5. Ștergeți elementul de pe poziția 3 si inserați -1 pe poziția 1.




```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, copie;
    vector<int> x;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    copie = x[3];
    x.erase(x.begin() + 3);
    for(i = 0; i < x.size(); i++)
```

```
        cout << x[i] << " ";
    cout << '\n';
    x.insert(x.begin() + 1, copie);
    for(i = 0; i < x.size(); i++)
        cout << x[i] << " ";
}
```

18. Citesc un vector cu n elemente. Ștergeți toate valorile egale cu 0 din vector.

La ștergerea repetată trebuie avut grijă la următorul aspect: Atunci când se elimină o valoare de pe poziția *i*, valoarea de pe poziția *i + 1* alunecă în locul valorii de pe poziția *i*, dar instrucțiunea *for* cu care parcurg vectorul mărește automat pe *i* cu 1 ceea ce face ca valoarea aflată inițial pe poziția *i + 1* să nu mai poată fi verificată. De aceea, la fiecare ștergere trebuie să anulez creșterea lui *i* adăugând un *i--*.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, copie;
    vector<int> x;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    for(i = 0; i < x.size(); i++)//nu mai folosesc n
        if(x[i] == 0)
            {
                x.erase(x.begin() + i);
                i--;//anulez cresterea lui i
            }
    for(i = 0; i < x.size(); i++)
        cout << x[i] << " ";
}
```

18. Citesc un vector cu n elemente. Inserați după fiecare număr par dublul lui.

La inserarea repetată trebuie avut grijă la următorul aspect: Atunci când adaug o valoare trebuie să trec peste ea ca să evit o inserare în funcție de valoarea nou adăugată și atunci trebuie să îl cresc suplimentar pe *i* cu *1*.

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main()
{
    int i, n, copie;
    vector<int> x;
    cin >> n;
    x.resize(n);
    for(i = 0; i < n; i++)
        cin >> x[i];
    for(i = 0; i < x.size(); i++)//nu mai folosesc n
        if(x[i] %2 == 0)
            {
                x.insert(x.begin() + i + 1, 2 * x[i]);
                i++;//sar peste elementul inserat
            }
    for(i = 0; i < x.size(); i++)
        cout << x[i] << " ";
}
```

Tema.

Citesc un vector cu x cu n elemente numere întregi .

- Inserați între fiecare 2 numere diferența lor în modul
- Ștergeți toate numerele pare din vector

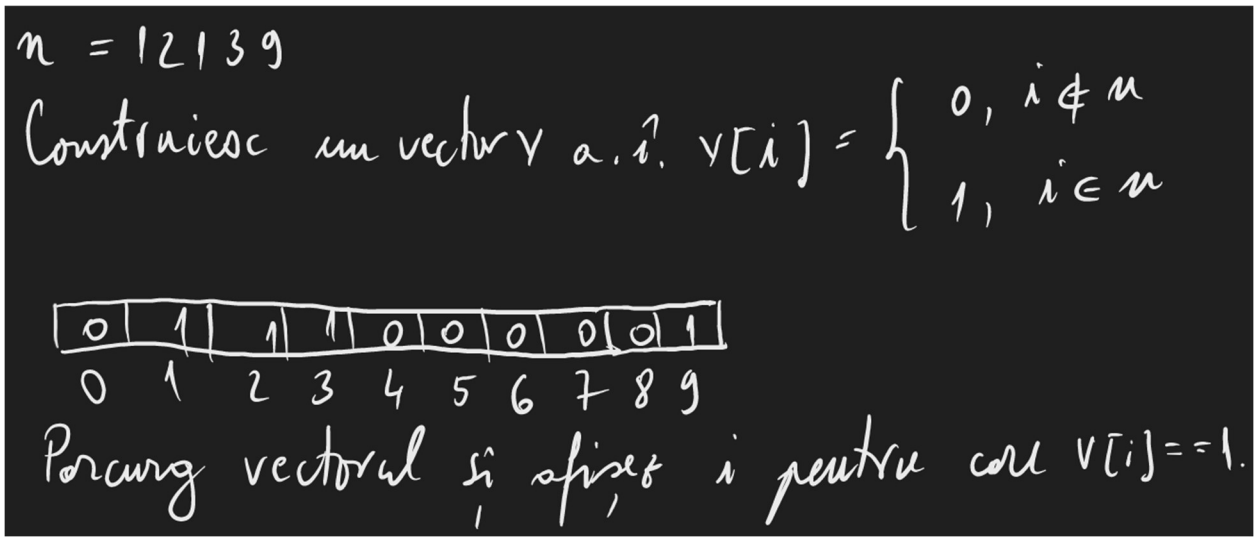
2. VECTORI CARACTERISTICI

Vectorii caracteristici sunt folosiți pentru rezolvarea unor probleme în timp optim. În general indicele *i* al vectorului reprezintă o valoare, iar *v[i]* reprezintă o caracteristică a acelei valori. Aș putea încadra vectorii caracteristici în două categorii: vectori de apariție în care *v[i]* poate lua două valori *0* sau *1* și vectorii de frecvență în care *v[i]* reprezintă numărul de apariții al lui *i*.

Cel mai ușor vectorii caracteristici se pot exemplifica pe probleme cu cifrele numerelor – acești vectori vor avea doar 10 elemente, pentru că există doar 10 cifre.

O alternativă modernă a vectorilor caracteristici este containerul *map*.

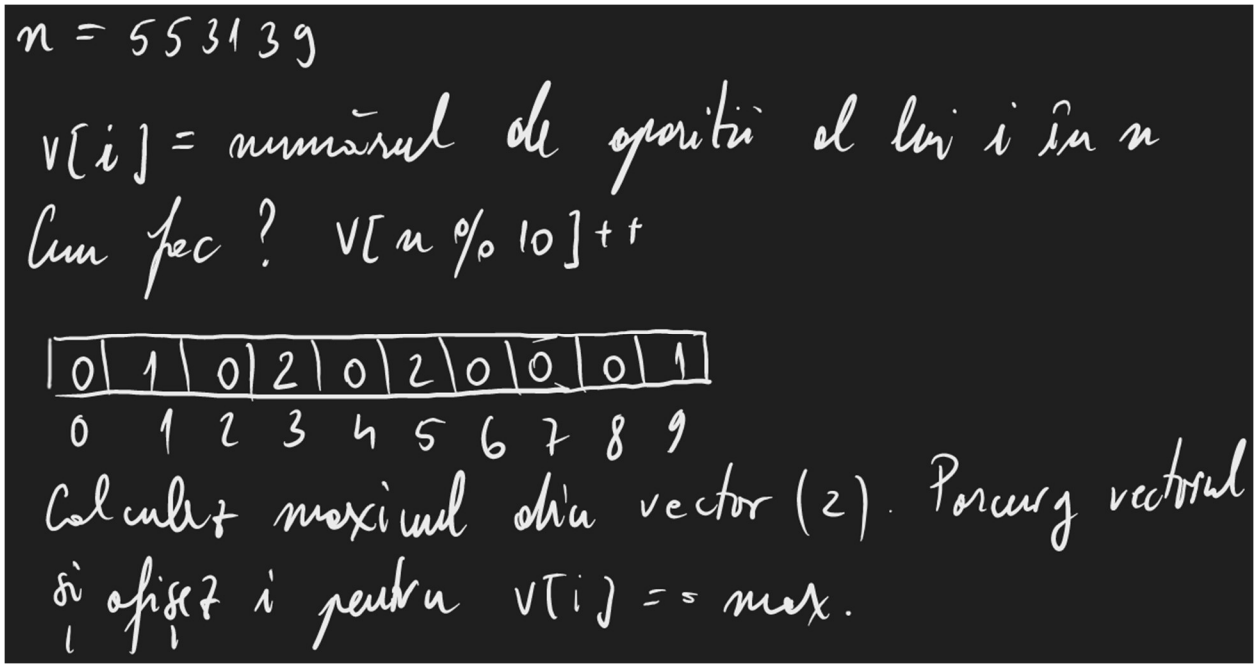
19. Citesc un număr n. Afișați cifrele distincte ale lui n. De exemplu dacă n = 121392 se va afișa 1 2 3 9.




```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, copie;
    vector<int> x(10, 0);//declar un vector cu 10 elemente egale cu 0
    cin >> n;
    while(n != 0)
    {
        x[n % 10] = 1;
        n /= 10;
    }
```

```
    for(i = 0; i < 10; i++)
        if(x[i] == 1)
            cout << i << " ";
}
```

20. Citesc un număr n. Afișați cifrele care apar de cele mai multe ori. De exemplu dacă n = 553139 voi afișa 5 și 3.




```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, m;
    vector<int> x(10, 0);//declar un vector cu 10 elemente egale cu 0
    cin >> n;
    while(n != 0)
    {
        x[n % 10]++;
        n /= 10;
    }
    m = 0;
    for(i = 0; i < 10; i++)
        if(x[i] > m)
            m = x[i];
```

```
    for(i = 0; i < 10; i++)
        if(m == x[i])
            cout << i << " ";
}
```

21. Citesc un număr n. Afișați cel mai mare număr care poate fi obținut din cifrele lui n. De exemplu dacă n = 553139 voi afișa 955331.

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int i, n, c;
    vector<int> x(10, 0);//declar un vector cu 10 elemente egale cu 0
    cin >> n;
    while(n != 0)
    {
        x[n % 10]++;
        n /= 10;
    }
    for(c = 9; c >= 0; c--)
        for(i = 1; i <= x[c]; i++)
            cout << c;
}
```

Tema.

- **Citesc n. Afișați câte cifre distincte are n.**
- **Citesc n si apoi n numere de 2 cifre. Afișați cel mai mare număr de 2 cifre care nu se află în șirul citit.**
- **Citesc n și apoi n numere de 2 cifre. Afișați numerele cu cifre distincte care apar de cele mai puține ori în șirul citit.**

3. CONTAINERUL MAP

În locul vectorilor caracteristici se poate folosi containerul *map*, container care are fiecare element compus din două valori: o cheie(indicele vectorului în problemele anterioare) și o valoare asociată – valoarea asociată indicelui în vectorul caracteristic. Aceste valori sunt ordonate crescător după chei, care sunt unice. Cheile se accesează cu *M.first*, iar valorile asociate cu *M.second*. Reprezintă o soluție bună pentru vectorii de frecvență

22. Citesc un număr n. Afișați cifrele care apar de cele mai multe ori. De exemplu dacă n = 553139 voi afișa 5 și 3.

```
#include <iostream>
#include <map>
using namespace std;

int main()
{
    int n, m;
    map<int, int> M;
    cin >> n;
    while(n != 0)
    {
        M[n % 10]++;
        n /= 10;
    }
    m = 0;
    for(auto c : M)
        if(c.second > m)
            m = c.second;
    for(auto c : M)
        if(c.second == m)
            cout << c.first << " ";
}
```