Fundamentele Programarii

Tablouri si vectori.

Tablouri si vectori.

- Tablouri
  - Ce sunt tablourile?
  - De ce folosim tablourile?
  - Declararea si initializarea tablourilor
  - Accesarea elementelor unui tablou
- Tablouri multidimensionale
- Vectori
  - Ce sunt vectorii?
  - Avantaje vs. tablouri
  - Declararea si initializarea vectorilor
  - Accesarea elementelor unui vector

Tablouri (arrays)

- Un **tablou** este un tip de date compus sau o structura de date

- Exemplu: colectie de elemente

- Toate elementele trebuie sa fie de acelasi tip!
- Fiecare element poate fi accesat direct!

De ce avem nevoie de tablouri?

```
int scor_1  {0};
int scor_2 {0};
int scor_3 {0};
…..

int scor_100 {0};
```

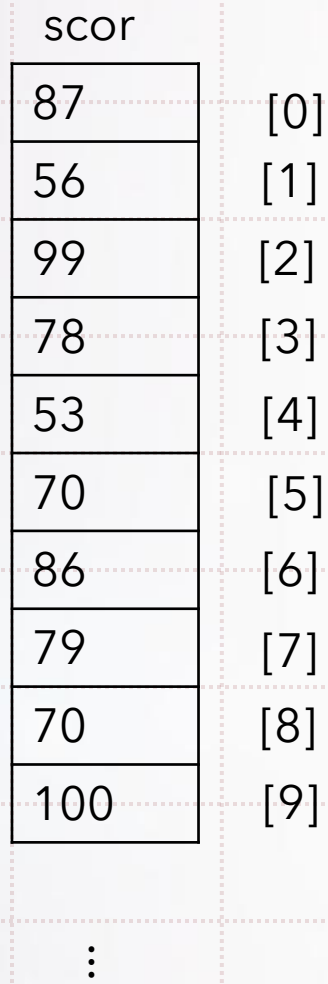
- Tablourile ne permit sa oferim compilatorului informatii despre anumite colectii cu cate elemente dorim, dand colectiei un singur nume

Caracteristici:

- Au marime fixa
- Elementele trebuie sa aiba acelasi tip
- Toate elementele sunt stocate impreuna in memorie
- Fiecare element poate fi accesat prin pozitia sau indexul sau

- Primul element are indexul **0**
- Ultimul element are indexul **marimea colectiei-1**

- Foarte eficiente
- Iteratia/ buclelele (looping) sunt foarte des intalnite



Declararea si initializarea tablourilor

Declararea

```
Tipul_elementului numele_tabloului [numar constant de elemente];
```

- Exemple:

```
int scorElevi [5];
int noteElevi [10];
const int nrZileAn {365};
double temperatura[nrZileAn];
```

- Aceste tablouri nu sunt initializate! In acest moment, ele vor contine niste valori aleatoare (garbage).

Declararea si initializarea tablourilor

Declararea

Ce se intampla daca scriem urmatorul cod?

```
int main(){
        int numar_studenti{ 20 };
        int note[numar_studenti];
        cout << "Notele studentilor sunt: " << endl;
        for (int i = 0; i < numar_studenti; i++) {
                cout << note[i] << " ";
        }
        return 0;
}
```

EROARE!

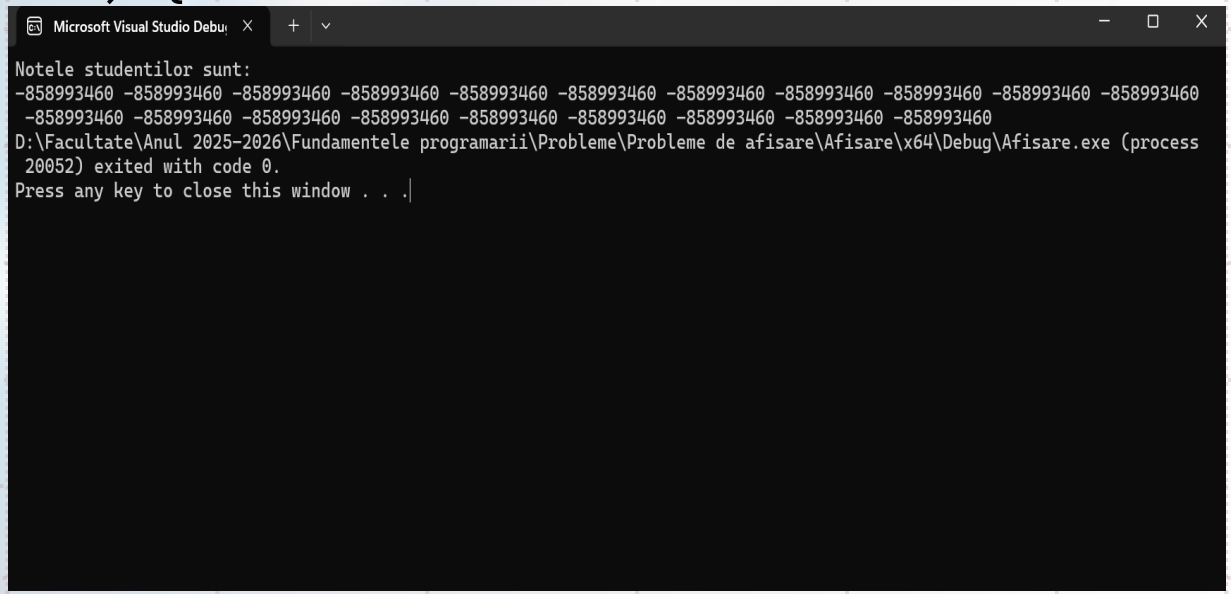
Declararea si initializarea tablourilor

Declararea

Avertizare!

Ce se intampla daca scriem urmatorul cod?

```
int main(){
        const int numar_studenti{ 20 };
        int note[numar_studenti];
        cout << "Notele studentilor sunt: " << endl;
        for (int i = 0; i < numar_studenti; i++) {
                cout << note[i] << " ";
        }
        return 0;
}
```



- **Initializeaza intotdeauna tablourile!**

Declararea si initializarea tablourilor

Initializarea:

`Tipul_elementului numele_tabloului [numar constant de elemente] {init lista};`

Sau

`Tipul_elementului numele_tabloului [numar constant de elemente] = {init lista};`

- Exemple:

```
int scorElevi [5] {100,95,99,67,87};
int noteElevi [10] {3,5};        //primele doua elemente sunt initializate cu 3 si 5 , iar restul cu 0

const int nrZileAn {365};
double temperatura[nrZileAn] {0}; // toate elementele sunt initializate cu 0

int altTablou [] {1,2,3,4,5} ; //marimea tabloului este automat calculata
```

Accesarea elementelor tablourilor

numeTablou [indexul_elementului]

- Exemplu: `scorElevi [1]`

```
int scorElevi[5]{ 100,95,99,67,87 };
cout << "Primul scor la indexul 0 este: " << scorElevi[0] << endl;
cout << "Al doilea scor la indexul 1 este: " << scorElevi[1] << endl;
cout << "Al treilea scor la indexul 2 este: " << scorElevi[2] << endl;
cout << "Al patrulea scor la indexul 3 este: " << scorElevi[3] << endl;
cout << "Al cincilea scor la indexul 4 este: " << scorElevi[4] << endl;
```

Schimbarea continutului elementelor tablourilor

numeTablou [indexul_elementului]

```
int scorElevi[5]{ 100,95,99,67,87 };
cout << "Primul scor la indexul 0 se modifica in:  " ;
cin >> scorElevi[0];
cout << "Al doilea scor la indexul 1 se modifica in: ";
cin >> scorElevi[1];
cout << "Al treilea scor la indexul 2 se modifica in: ";
cin >> scorElevi[2];
cout << "Al patrulea scor la indexul 3 se modifica in: ";
cin >> scorElevi[3];
cout << "Al cincilea scor la indexul 4 se modifica in: ";
cin >> scorElevi[4];
```

Sau, direct in program:

```
scorElevi[0]=90;
```

Cum functioneaza tablourile?

- **Numele tabloului** reprezinta locatia primului element din tablou (cel cu indexul 0)
- **[indexul]** reprezinta decalajul fata de inceputul tabloului. Astfel, C++ face un calcul simplu pentru a identifica elementul corect
- **Atentie la limitele tabloului!** Daca avem un tablou de 5 elemente si noi vrem sa afisam valoarea elementului cu indexul 5 atunci compilatorul nu va genera o eroare, se va genera doar o avertizare si se va afisa o valoare aleatorie!

Vezi programul “Tablouri”!

Cum functioneaza tablourile?

scorElevi

1000	100	0
1004	95	1
1008	99	2
1012	67	3
1016	87	4

1000+4=1004
(1 int --->4 bytes)

```
int scorElevi[5]={100,95,99,67,87};
cout << "\nPrimul scor la indexul 0 este: " << scorElevi[0] << endl;
cout << "Al doilea scor la indexul 1 este: " << scorElevi[1] << endl;
cout << "Al treilea scor la indexul 2 este: " << scorElevi[2] << endl;
cout << "Al patrulea scor la indexul 3 este: " << scorElevi[3] << endl;
cout << "Al cincilea scor la indexul 4 este: " << scorElevi[4] << endl;

cout << "\nValoarea numelui tabloului este " << scorElevi << endl;
```

1000

Tablouri multidimensionale

- Ne vom concentra atentia asupra tablourilor bidimensionale, dar se pot declara tablouri de orice dimensiune

```
TipulElementului numeleTabloului [dim1][dim2]
```

Exemplu: `int matrice[3][4];`

- Nu am initializat acest tablou, deci in acest moment tabloul va avea 12 valori aleatorii “garbage” (gunoi)!

Tablouri bidimensionale

```
const int nrLinii{ 3 };
const int nrColoane{ 4 };
int matrice[nrLinii][nrColoane];
```

	0	1	2	3
0	0	4	3	5
1	2	3	3	5
2	1	4	4	5

Tablouri bidimensionale

Accesarea elementelor unui tablou bidimensional

```
cin >> matrice[1][2];
cout<< matrice[1][2];
```

	0	1	2	3
0	0	4	3	5
1	2	3	**3**	5
2	1	4	4	5

Tablouri bidimensionale

Initializarea tablourilor bidimensionale

	0	1	2	3
0	0	4	3	5
1	2	3	3	5
2	1	4	4	5

```
int matrice[3][4] = { {0,4,3,5},{2,3,3,5},{1,4,4,5} };
```

sau

```
int matrice[3][4]  { {0,4,3,5},{2,3,3,5},{1,4,4,5} };
```

Tablouri bidimensionale

Ce se va afisa?

```
int a[2][2] = { 1,2,3,4 };
cout << a[0][3] << endl;
```

4

```
int a[2][2] = { {1,2},{3,4} };
cout << a[0][3] << endl;
```

Avertizare!!!

4

- Vezi programul “TablouriBidemensionale”!

Vectori

- Presupunem ca vrem sa monitorizam rezultatele la admitere la o scoala

- Nu avem de unde sa stim cati studenti se vor inregistra la admitere

- Optiuni:
  - Sa folosim un tablou cu o dimensiune suficient de mare astfel incat aceasta sa nu fie depasita
  - Sa folosim un tablou dinamic asa cum este un **vector**

Vectori

Ce este un vector?

- Un **vector** este un tablou unidimensional a carui dimensiune poate fi marita sau micsorata in timpul executiei programului (obiecte C++)
- Este un container din Libraria Standard C++.
- Sintaxa si semantica vectorilor sunt asemanatoare cu ale tablourilor
- **Putem face verificari asupra limitelor unui vector!**
- Putem folosi multe functii foarte utile precum **sortare, cautare, inversare si altele**

Vectori

Declararea vectorilor

```
#include<vector>

vector <char> vocale;
//se creaza o colectie (vector) de caractere (nu stim cate caractere),
//care acum este goala, dar in care vom putea introduce caractere mai
tarziu
vector <int> scorElevi;
```

SAU

```
vector <char> vocale(5); // se creaza un vector si ii spunem compilatorului ca acesta
va contine 5 caractere vide(nule, '\0')

vector <int> scorElevi(10); // se creaza un vector format din 10 nr intregi care
automat sunt initializate cu 0
```

Vectori

Initializarea vectorilor

```
vector <char> vocale{ 'a','e','i','o','u'};

vector <int> scorElevi {100, 98, 89, 85, 93};

vector <double> temperaturi(365,90.0); //365 reprezinta dimensiunea vectorului si toate elementele vectorului sunt initializate cu 90.0
```

Vectori

Caracteristici:

- Dimensiune **dinamica**
- Toate elementele lor sunt de acelasi tip
- Toate elementele sunt stocate impreuna in memorie
- Elementele individule pot fi accesate prin pozitia lor sau index

- Primul element are indexul **`0`**
- Ultimul element are indexul **`dimensiuneVector-1`**

- Elementele sunt initializate automat cu `0` sau cu `\0`
- Foarte eficienti
- Iteratia (looping) este foarte des folosita

Accesarea elementelor unui vector (sintaxa de la tablouri)

numeVector [indexulElementului]

Exemplu: scorElevi[1]

```
vector <int>  scorElevi{ 100,95,99,67,87 };
cout << "Primul scor la indexul 0 este: " << scorElevi[0] << endl;
cout << "Al doilea scor la indexul 1 este: " << scorElevi[1] << endl;
cout << "Al treilea scor la indexul 2 este: " << scorElevi[2] << endl;
cout << "Al patrulea scor la indexul 3 este: " << scorElevi[3] << endl;
cout << "Al cincilea scor la indexul 4 este: " << scorElevi[4] << endl;
```

Accesarea elementelor unui vector (sintaxa caracteristica vectorilor)

numeVector.at(indexulElementului)

Exemplu: scorElevi.at(1)

```
vector <int>  scorElevi{ 100,95,99,67,87 };
cout << "Primul scor la indexul 0 este: " << scorElevi.at(0) << endl;
cout << "Al doilea scor la indexul 1 este: " << scorElevi.at(1) <<
endl;
cout << "Al treilea scor la indexul 2 este: " << scorElevi.at(2) <<
endl;
cout << "Al patrulea scor la indexul 3 este: " << scorElevi.at(3) <<
endl;
cout << "Al cincilea scor la indexul 4 este: " << scorElevi.at(4) <<
endl;
```

Schimbarea elementelor unui vector (sintaxa caracteristica vectorilor)

numeVector.at(indexulElementului)

```
vector <int>  scorElevi{ 100,95,99,67,87 };
cin >> scorElevi.at(0);
cin >> scorElevi.at(1);
cin >> scorElevi.at(2);
cin >> scorElevi.at(3);
cin >> scorElevi.at(4);


scorElevi.at(0) = 90; //atribuirea valorii 90 elementului de pe
indexul 0
```

Cum putem mari dimensiunea vectorului atunci cand dorim acest lucru?

numeVector.push_back(element)

```
vector <int>  scorElevi{ 100,95,99};  //dimensiunea vectorului este 3

scorElevi.push_back(80); // 100,95,99,80
scorElevi.push_back(90); //100,95,99,80,90
```

Vectorul va aloca automat spatiul necesar!

Ce se intampla daca depasim dimensiunea?

```
vector <int>  scorElevi{ 100,95,99};  //dimensiunea vectorului este 3

cout<<scorElevi.at(5);
```

- **Compilatorul va genera eroare sau o exceptie (spre deosebire de cazul cand aveam tablouri si se genera doar o avertizare)!**

Metode/ functii folosite pentru vectori

```
vector <int>  scorElevi{ 100,95,99};  //dimensiunea vectorului este 3

std::vector<int>::iterator it = scorElevi.begin();
 // Afișăm primul element utilizând iteratorul

std::cout << "Primul element: " << *it << std::endl;


it = scorElevi.end()-1;
 // Afișăm ultimul element utilizând iteratorul

std::cout << “Ultimul element: " << *it << std::endl;
```

Un **iterator** în C++ este un obiect special care permite accesul secvențial la elementele unei colecții (cum ar fi un vector), fără a expune detaliile interne ale structurii de date. Practic, iteratorii îți permit să parcurgi și să manipulezi elementele dintr-o colecție într-un mod similar cu pointerii despre care vom discuta intr-un curs ulterior.

Metode/ functii folosite pentru vectori

- **sort**(scorElevi.begin(), scorElevi.end()); //sorteaza crescator vectorul; sort este o functie din biblioteca **algorithm** si ea lucreaza pe intervalul [begin,end);

begin() -> iterator la primul element din vector;
end()->iterator la “un element dupa ultimul”

- **reverse**(scorElevi.begin(), scorElevi.end()); // inverseaza elementele intr-un vector
- scorElevi.**pop_back(); //**elimina ultimul element din vector
- scorElevi.**erase**(scorElevi.begin() + i) ;// sterge elementul de pe pozitia i
- scorElevi.**clear** (); goleste vectorul
- n=scorElevi.size(); // numarul de elemente din vector
- **find**(scorElevi.begin(), scorElevi.end(), x); //returneaza iteratorul/adresa unde il gaseste prima data pe x in vector, iar daca nu il gaseste returneaza un iterator care indica “dupa ultimul element”; find lucreaza pe intervalul [begin, end)

Vezi programul “Vectori”!

Programul “Vectori”!

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

int main() {
vector <char> vocale{ 'a','e','i','o','u' };

cout << "vocale[0]="<< vocale[0] << endl;
cout << "vocale[4]= "<< vocale[4] << endl;
//cout << vocale[10] << endl; // vom avea o eroare

//char v[3] { 'a','b','c' };
//cout << v[3];   //nu vom avea o eroare, ci va fi afisat un "gunoi"

vector <int> scorElevi (3); //toate trei elementele sunt intializate cu 0
//vector <int> scorElevi(3, 100); // toate trei elementele sunt initializate cu 100
    //vector <int> scorElevi{ 100,99,87 };

cout << "\nScorul elevilor folosind sintaxa de la tablouri: " << endl;
cout << scorElevi[0] << endl;
cout << scorElevi[1] << endl;
cout << scorElevi[2] << endl;
```

Programul “Vectori”!-continuare1

```
cout << "\nScorul elevilor folosind sintaxa de la vector: " << endl;
cout << scorElevi.at(0) << endl;
cout << scorElevi.at(1) << endl;
cout << scorElevi.at(2) << endl;
cout << "\nExista " << scorElevi.size() << " scoruri in vector" << endl;
//size este o metoda din clasa vector

cout << "\nIntroduceti trei scoruri: "<<endl;
cin >> scorElevi.at(0);
cin >> scorElevi.at(1);
cin >> scorElevi.at(2);

cout << "\n Scorurile modificate sunt: "<<endl;
cout << scorElevi.at(0) << endl;
cout << scorElevi.at(1) << endl;
cout << scorElevi.at(2) << endl;
```

Programul “Vectori”!-continuare2

```
cout << "\n Adaugati un scor in vector ";

int scorNou;
cin >> scorNou;
scorElevi.push_back(scorNou);

cout << "\n Mai adaugati un alt scor in vector ";
cin >> scorNou;
scorElevi.push_back(scorNou);

cout << "\nScorurile sunt acum:" << endl;
cout << scorElevi.at(0) << endl;
cout << scorElevi.at(1) << endl;
cout << scorElevi.at(2) << endl;
cout << scorElevi.at(3) << endl;
cout << scorElevi.at(4) << endl;

cout << "\nExista " << scorElevi.size() << " scoruri in vector" << endl;
//determinam numarul de elemente din vector
```

Programul “Vectori”!-continuare3

```
cout << "Elementul de pe prima pozitie din scorElevi este: " << *scorElevi.begin() <<
endl;
cout << "Elementul de pe ultima pozitie din scorElevi este: " << *(scorElevi.end()-1)
<< endl;

//ordonam crescator elementele vectorului

sort(scorElevi.begin(), scorElevi.end());

cout << "\nScorurile ordonate crescator sunt:" << endl;
cout << scorElevi.at(0) << endl;
cout << scorElevi.at(1) << endl;
cout << scorElevi.at(2) << endl;
cout << scorElevi.at(3) << endl;
cout << scorElevi.at(4) << endl;
```

Programul “Vectori”!-continuare4

```
//le inversam ordinea

reverse(scorElevi.begin(), scorElevi.end());

cout << "\nScorurile ordonate descrescator sunt:" << endl;
cout << scorElevi.at(0) << endl;
cout << scorElevi.at(1) << endl;
cout << scorElevi.at(2) << endl;
cout << scorElevi.at(3) << endl;
cout << scorElevi.at(4) << endl;
```

Programul “Vectori”!-continuare5

```
//cautarea unui element dat intr-un vector

vector <int>::iterator it;
//interatorii sunt folositi pentru a face referire la adresa de memorie
// la care sunt stocate elementele vectorului
int x{ 20};

it=find(scorElevi.begin(), scorElevi.end(), x);
// adresa unde se gaseste x prima data in vectorul nostru
if (it != scorElevi.end()) {
cout << "Elementul " << x << " este pe pozitia " << it - scorElevi.begin() << endl;
}
else
cout << "Elementul " << x << " nu a fost gasit!" << endl;

//cout << "Acesta va genera o eroare! " << scorElevi.at(10);
```

```
//Exemplu de vector 2-dimensional

vector <vector <int>> matrice{ {1,2,3,4},{9,1,4,4},{3,6,4,5} };
cout << "\nElementele matricei de pe linia 1 (folosind sintaxa de la tablouri) sunt: " <<
endl;
cout << matrice[0][0] << " ";
cout << matrice[0][1] << " ";
cout << matrice[0][2] << " ";
cout << matrice[0][3] << " ";
cout << "\nElementele matricei de pe linia 1 (folosind sintaxa de la vector) sunt: " << endl;
cout << matrice.at(0).at(0) << " ";
cout << matrice.at(0).at(1) << " ";
cout << matrice.at(0).at(2) << " ";
cout << matrice.at(0).at(3) << " ";
cout << endl << "Matricea noastra este " << endl;
cout << matrice[0][0] << " "<< matrice[0][1] << " "<<matrice[0][2] << " " << matrice[0][3] <<
endl;
cout << matrice[1][0] << " " << matrice[1][1] <<" "<< matrice[1][2] << " " << matrice[1][3]
<< endl;
cout << matrice.at(2).at(0) << " " << matrice.at(2).at(1) <<" "<< matrice.at(2).at(1) << " "
<< matrice.at(2).at(3) << endl;

return 0;
}
```

https://test-master.space