8. Tablouri

Tablourile (*arrays*) reprezintă un tip important de structură de date şi sunt colecții de obiecte de acelaşi tip reunite sub un singur nume.

Uneori este necesar să referim anumite variabile ca un singur grup pentru că este dificil să definim şi să folosim individual fiecare variabilă. De exemplu, dacă dorim să tipărim un set de date în ordine inversă celei în care au fost introduse, trebuie să le citim şi să le salvăm pe toate înainte de a putea face prima tipărire. Dacă avem de a face cu 1000 de valori, trebuie să declarăm 1000 de variabile ca să le stocăm, să scriem 1000 de instrucțiuni de citire şi 1000 de instrucțiuni de tipărire.

Tabloul este un tip de dată care ne permite să programăm mai uşor operații asupra grupurilor de valori de acelaşi tip.

8.1 Tipuri de dată simple şi tipuri de dată structurate

O valoare căreia îi este asociat un tip simplu de dată este un element singular care nu poate fi împărțit mai departe în părți componente. De exemplu, o valoare de tip `int` este un număr întreg şi nu mai poate fi descompus. În contrast cu acesta, o valoare având un tip de dată structurat este o colecție de elemente. Întreaga colecție este referită printr-un singur nume şi fiecare componentă poate fi accesată individual.

Un tip de dată structurat este `ifstream` pe care l-am folosit pentru citirea valorilor dintr-un fişier. Atunci când declarăm `inFile` de tip `ifstream`, `inFile` nu reprezintă o singură valoare. El reprezintă întreaga colecție de date din fişier. Fiecare coomponentă, însă, poate fi accesată individual printr-o operație de intrare.

Tipurile simple de date sunt elemente componente pentru tipurile structurate. Un tip de dată structurat pune împreună mulțimea de valori componente şi impune un aranjament specific asupra valorilor.

Aşa cum am văzut deja în clasificarea pe care am făcut-o deja, C++ oferă mai multe tipuri de dată structurate:

- Tipuri structurate
  - tablou (*array*)
  - `struct`
  - `union`
  - `class`

În acest capitol vom discuta despre tablouri.

8.2 Tablouri unidimensionale

Să reluăm exemplul de citire şi afişare în ordine inversă a 1000 de valori. O variantă de implementare a sa este:

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

int main()
{
 int val0;
 int val1;
 ...
 int val999;
 cin >> val0;
```

```
 cin >> val1;
 ...
 cin >> val999;
 cout << val999 << endl;
 ...
 cout << val1 << endl;
 cout << val0 << endl;
 return 0;
}
```

Programul are peste 3000 de linii şi foloseşte 1000 de variabile cu nume asemănătoare. Ar fi mult mai comod dacă numărul din componența numelor variabilelor ar fi, de fapt, un contor pe care să îl putem folosi pentru a citi şi scrie bucle `while` în care contorul să ia valori între 0 şi 999.

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

int main()
{
 int val[1000];  //declararea tabloului
 int contor = 0;
 while(contor < 1000)
 {
      cin >> val[contor];
      contor++;
 }
 contor = 999;
 while(contor >= 0)
 {
      cout << val[contor] << endl;
      contor--;
 }
 return 0;
}
```

În acest program am declarat `val` ca fiind un tablou unidimensional, adică o colecție de variabile, toate de acelaşi tip, în care prima parte a numelui variabilelor este acelaşi, iar ultima parte este un indice cuprins între `[]`. În acest exemplu, valoarea stocată în `contor` se numeşte *indice*.

Declararea tabloului este similară cu declarația unei variabile simple, cu o singură excepție: trebuie declarată şi dimensiunea tabloului. Numărul de componente se declară între `[]`. Declarația

```
int val[1000];
```

creează un tablou cu 1000 de componente, toate de tip int. Prima are indicele 0, a doua are indicele 1, iar ultima are indicele 999.

Declararea tablourilor

Un tablou unidimensional este o colecție structurată de componente (elemente) care pot fi fi accesate individual specificând poziția componentei printr-un indice o constantă întreagă. Şablonul sintactic al unei declarații de tablou unidimensional este următorul:

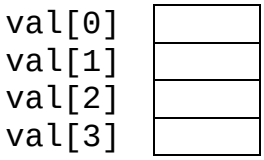
TipDată NumeTablou`[`*ExpresieConstInt*`];`

Componentele unui tablou pot avea aproape orice tip de dată, dar ne vom limita în acest capitol doar la tipurile atomice. Expresia dintre [] este o constantă întreagă. Poate fi un literal sau o constantă simbolică. Ea trebuie să fie strict mai mare decât 0 şi defineşte numărul de componente ale tabloului. Dacă valoarea este n, domeniul indicilor va fi între 0 şi n-1, nu între 1 şi n.

Exemplu

```
int val[4];
```

Prin această declarație, compilatorul rezervă într-o zonă compactă de memorie 4 locații de tip int:



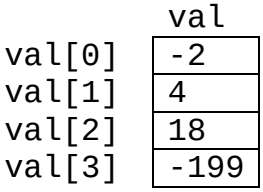
Accesarea componentelor individuale

Pentru a folosi componentele individuale scriem numele tabloului urmat de o expresie indice între []. Expresia specifică numărul componentei accesate şi poate fi atât constantă cât şi variabilă ori o expresie mai complicată. Oricare ar fi, însă, forma indicelui, acesta trebuie să fie o valoare întreagă.

Cea mai simplă formă a expresiei indice este o constantă.

Exemplu

```
int val[4];
val[0] = -2;
val[1] = 4;
val[2] = 18;
val[3] = -199;
```



Spre deosebire de declarații, expresiile indice pot fi şi variabile sau expresii întregi.

Exemplu

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

int main()
{
 int i, n[10]; //declararea tabloului
 i = 0;
 while(i < 10)
 {
      n[i] = 0; //initializarea elementelor taboului
      i++;
 }

 cout << "Element" << setw(13) << "Valoare" << endl;
 i = 0;
 while(i < 10)
 {
      cout << setw(7) << i << setw(13)
           << n[i] << endl;
```

```
        i++;
    }

    return 0;
}
```

Dacă `i` este `0`, atunci se foloseşte `n[0]`, când `i` este `1`, atunci se foloseşte `n[1]` etc.

Fiecare componentă a unui tablou poate fi tratată exact ca o variabilă simplă.

Exemplu

```
int val[4];
val[0] = -2;
cin >> val[2];
cout << val[1];
double x = sqrt(val[2]);
double y = 6.8 * val[2] + 7.5;
```

Indicii din afara limitelor

Să considerăm declarația

```
double alfa[100];
```

pentru care domeniul valid al indicilor este 0 – 99. Ce se întâmplă dacă executăm instrucțiunea

```
alfa[i] = 62.4;
```

când `i` < 0 sau `i` > 99 ? Rezultatul este că se accesează locații de memorie din afara tabloului. C++ nu verifică încadrarea indicilor între limite şi aceasta este răspunderea programatorului.

Algoritmii de procesare a tablourilor folosesc adeseori bucle pentru a parcurge elementele.

Exemplu

```
int i = 0;
while(i < 10)
{
      alfa[i] = 0.0;
      i++;
}
```

Aceeaşi buclă se poate scrie şi într-o a doua variantă în care variabila de control se compară cu limita superioară a domeniului indicilor.

Exemplu

```
int i = 0;
while(i <= 9)
{
      alfa[i] = 0.0;
      i++;
}
```

Prima variantă este preferată pentru că valoarea cu care se compară variabila de control este aceeaşi cu dimensiunea tabloului, fiind mai sugestivă.

Inițializarea tablourilor

În limbajul C++ este permisă inițializarea unei variabile odată cu declararea acesteia.

Exemplu

```
int i = 0;
```

Elementele unui tablou pot fi, de asemenea, inițializate în instrucțiunea de declarare prin adăugarea unei liste de valori separate prin virgulă, plasate între acolade.

Exemplu

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

int main()
{
 int n[10] = {32, 2, 64, 18, 95, 14, 90, 70, -60, 37};

 cout << "Element" << setw(13) << "Valoare" << endl;
 int i = 0;
 while(i < 10)
 {
      cout << setw(7) << i << setw(13)
           << n[i] << endl;
      i++;
 }

 return 0;
}
```

În acest fel, `n[0]` este inițializat cu valoarea `32`, `n[1]` cu `2` etc. Între acolade trebuie să se găsească cel puțin o valoare. Dacă se înscriu prea multe valori, compilatorul semnalează o eroare. Dacă sunt mai puține valori în listă, restul sunt inițializate cu valoarea `0`.

O facilitate a limbajului C++ este aceea prin care se permite omiterea dimensiunii tabloului atunci când declarația şi inițializarea se fac în aceeaşi instrucțiune.

Exemplu

```
int n[] = {32, 2, 64, 18, 95, 14, 90, 70, -60, 37};
```

Compilatorul stabileşte dimensiunea tabloului la numărul de elemente dintre acolade.

Pentru declararea dimensiunii unui tablou se pot folosi şi constante simbolice. Programul din exemplul următor va citi numerele dintr-un tablou de valori întregi şi va afişa sub formă grafică segmente ale căror lungimi sunt proporționale cu aceste valori. Acest grafic se numeşte histogramă.

Exemplu

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

int main()
{
 const int dim = 5;
 int val[dim] = {19, 3, 15, 7, 11};

 cout << "Element" << setw(13) << "Valoare"
      << setw(17) << "Histograma" << endl;
```

```
    int i = 0;
    while(i < dim)
    {
         cout << setw(7) << i << setw(13) << val[i]
              << setw(9);
         int j = 0;
         while(j < val[i])
         {
              cout << '*';
              j++;
         }
         cout << endl;
         i++;
    }

    return 0;
}
```

Tabourile cu valori tip char

Tablourile pot avea orice tip de dată. Vom discuta acum despre stocarea şirurilor de caractere (string-uri) în tablouri de tip `char`. Am văzut că pentru a afişa un string pe ecran putem folosi o instrucțiune de tipărire prin care textul este inserat în stream-ul de ieşire.

Exemplu

```
cout << "caractere";
```

Un string precum `"caractere"` este, de fapt, un tablou de valori de tip `char`.

Aceste tablouri au câteva caracteristici speciale care le diferențiază de celelalte tipuri de tablouri.

Un tablou de caractere poate fi inițializat printr-un string literal.

Exemplu

```
char clasament[] = "primul";
```

Prin această declarație, elementele tabloului `clasament` sunt inițializate cu valorile caracterelor individuale din stringul literal `"primul"`. Dimensiunea tabloului `clasament` este determinata de compilator prin lungimea sirului la care se adaugă automat un caracter special numit *caracterul null* care se numeşte *terminator de şir*. Astfel, tabloul `clasament` va avea şase elemente. Caracterul *null* are codul ASCII 0 iar reprezentarea sa ca şi constantă literală este `'\0'`. Toate string-urile se termină cu acest caracter. Un tablou de caractere care reprezintă un string trebuie declarat întotdeauna suficient de mare ca să cuprindă caracterele şirului şi terminatorul de şir.

Tablourile de caractere pot fi inițializate şi prin constante individuale printr-o listă de inițializare.

Exemplu

```
char clasament[] = {'p', 'r', 'i', 'm', 'u', 'l',
'\0'};
```

Putem accesa componentele individuale ale unui tablou de caractere prin folosirea indicilor. În felul acesta, `clasament[0]` este caracterul `'p'`, `clasament[1]` este caracterul `'r'` etc.

Putem introduce valori de la tastatură direct într-un tablou de tip `char` folosind `cin` şi >>.

Exemplu

```
char sir[20];
cin >> sir;
```

Prin prima instrucțiune, tabloul `sir` este declarat ca un şir de caractere care poate să stocheze 19 caractere şi terminatorul de şir. A doua instrucțiune citeşte un string de la tastatură adăugându-i automat caracterul null. Este răspunderea programatorului să stabilească dimensiunea tabloului în care se vor păstra aceste caractere. Manipulatorul `setw` poate fi folosit în acest caz pentru a ne asigura că numărul de caractere citite din stream-ul cin nu va depăşi dimensiunea tabloului în care sunt transferate acestea.

Exemplu

Vor fi citite 19 caractere de la tastatură după care se adaugă automat caracterul *null*.

```
cin >> setw(20) >> sir;
```

8.3 Transmiterea tablourilor ca parametri de funcții

Dacă dorim să transmitem o variabilă ca parametru unei funcții şi dorim ca funcția să nu poată să îi modifice valoarea atunci variabila trebuie transmisă prin valoare şi nu prin referință. De la această regulă fac excepție stream-urile. Tot o excepție o reprezintă şi tablourile pentru că în C++ acestea sunt transmise *întotdeauna* prin referință.

Pentru ca o variabilă simplă sau un obiect să fie transmis prin referință trebuie ataşat semnul `&` tipului de dată din lista de parametri formali. Fiind transmise întotdeauna prin referință, pentru declararea tablourilor nu se foloseşte `&`. Când un tablou este transmis ca parametru, i se transmite de fapt *adresa de bază* care este adresa de memorie a primului său element. În acest fel funcția îl va putea localiza în memoria calculatorului şi îi va putea accesa elementele.

Exemplu

```
void ModificaTablou(int b[], int dimensiune)
{
  int j = 0;
  while(j < dimensiune)
  {
    b[j] *= 2;
    j++;
 }
}
```

În acest exemplu am folosit operatorul `*=` care este un operator abreviat. În C++ se pot folosi următorii operatori aritmetici de asignare:

Operator	**Exemplu**	**Explicație**
`+=`	`a += 7`	`a = a + 7`
`-=`	`b -= 6`	`b = b – 6`
`*=`	`c *= 5`	`c = c * 5`
`/=`	`d /= 4`	`d = d / 4`
`%=`	`e %= 3`	`e = e % 3`

Operatorul %= se poate aplica doar variabilelor întregi.

În lista parametrilor formali, declararea unui tablou nu include şi dimensiunea sa între `[]`. Dacă se include dimensiunea, compilatorul o ignoră. Compilatorului îi este necesară doar informația referitoare la natura parametrului, adică faptul că este vorba despre un tablou, şi la tipul componentelor sale. Acesta este motivul pentru care trebuie adăugat un al doilea parametru al funcției prin care se precizează numărul de componente.

În prototipul unei funcții care are parametri de tip tablou nu este necesară prezența numelor parametrilor formali. De fapt această regulă este valabilă pentru orice funcție, indiferent de tipul parametrilor săi.

Exemplu

```
void ModificaTablou(int [], int);
```

Programul următor ilustrează diferența între trimiterea unui tablou şi a unui element al unui tablou ca parametri de funcții.

Exemplu

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

void ModificaTablou(int [], int);
void ModificaElement(int);

int main()
{
 const int dimTablou = 5;
 int i, a[dimTablou] = {0, 1, 2, 3, 4};

 cout << "Efectele transmiterii unui tablou "
      << "prin referinta:"
      << "\n\nValorile tabloului original:\n";

 i = 0;
 while(i < dimTablou)
 {
      cout << setw(3) << a[i];
      i++;
 }

 cout << endl;

 //Tablou transmis prin referinta
 ModificaTablou(a, dimTablou);

 cout << "Valorile modificate sunt:\n";

 i = 0;
 while(i < dimTablou)
 {
      cout << setw(3) << a[i];
      i++;
 }
```

```
 cout << "\n\n\n"
      << "Efectele transmiterii unui element "
      << "al tabloului prin valoare:"
      << "\n\nValoarea lui a[3] este "
      << a[3] << '\n';

 ModificaElement(a[3]);
 cout << "Valoarea lui a[3] este " << a[3] << endl;
 return 0;
}

//In functia ModificaTablou, "b" este un alt nume
//pentru tabloul original "a"
void ModificaTablou(int b[], int dimensiune)
{
  int j = 0;
  while(j < dimensiune)
  {
    b[j] *= 2;
   j++;
 }
}

//In functia ModificaElement, "e" este o copie a
//elementului a[3] transmis prin valoare
void ModificaElement(int e)
{
 cout << "Valoarea in functia ModificaElement este "
      << (e *= 2) << endl;
}
```

Programul tipăreşte pe ecran următoarele rezultate:

```
Efectele transmiterii unui tablou prin referinta:

Valorile tabloului original:
  0  1  2  3  4
Valorile modificate sunt:
  0  2  4  6  8


Efectele transmiterii unui element al tabloului prin
valoare:

Valoarea lui a[3] este 6
Valoarea in functia ModificaElement este 12
Valoarea lui a[3] este 6
```

Atunci când se apelează funcția `ModificaTablou`, i se transmite funcției o copie a adresei de memorie a tabloului `a`, iar modificările asupra tabloului `b` vor fi de fapt modificări ale tabloului `a`. Funcția `ModificaElement` are un parametru de tip valoare, o modificare asupra lui e neavând niciun efect asupra parametrului actual `a[3]`.

Pot apărea situații în programele noastre când o funcție nu trebuie să poată modifica elemente ale unui tablou care îi este transmis ca parametru. Deoarece tablourile sunt transmise întotdeauna prin referință, modificarea valorilor dintr-un tablou este dificil de controlat. Limbajul C++ oferă mecanismul implementat prin cuvântul cheie `const` care se poate folosi pentru a împiedica modificarea valorilor elementelor unui tablou printr-o funcție. Dacă parametrul tablou al unei funcții este de tip `const`, elementele sale devin constante în interiorul funcției şi orice intenție de modificare a valorilor lor este interpretată de compilator ca o eroare de sintaxă.

Exemplu

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

void IncearcaSaModificeTablou(const int []);

int main()
{
     int a[] = {10, 20, 30};
     IncearcaSaModificeTablou(a);
     cout << a[0] << ' ' << a[1] << ' ' << a[2] << endl;
     return 0;
}

void IncearcaSaModificeTablou(const int b[])
{
     b[0] /= 2; //eroare
     b[1] /= 2; //eroare
     b[2] /= 2; //eroare
}
```

Compilatorul semnalează eroare pentru că `b[0]`, `b[1]` şi `b[2]` sunt nemodificabile.

8.4 Tablouri multidimensionale

Tabourile în C++ pot avea mai multe dimensiuni. O modalitate comună de a le folosi este prin *matrici* cu linii şi coloane, fiind vorba în acest caz despre *tablouri cu două dimensiuni*. Pentru a identifica un element al unei astfel de matrici trebuie să specificăm doi indici: primul reprezintă linia iar al doilea reprezintă coloana. Tablourile multidimensionale pot avea şi mai mult de două dimensiuni.

Tablourile multidimensionale pot fi inițializate odată cu declararea lor în mod asemănător cu inițializarea tablourilor unidimensionale. De exemplu, un tablou bidimensional `b[2][2]` poate fi declarat şi inițializat prin instrucțiunea

```
int b[2][2] = { {1,2}, {3,4} };
```

Valorile sunt grupate pe linii, între acolade. Astfel, `1` şi `2` sunt valorile inițiale pentru `b[0][0]` şi `b[0][1]`, iar `3` şi `4` sunt valorile inițiale pentru `b[1][0]` şi `b[1][1]`.

Dacă nu sunt suficiente valori pentru o linie, elementele care rămân sunt inițializate cu valoarea `0`. În felul acesta, declarația

```
int b[2][2] = { {1}, {3,4} };
```

inițializează `b[0][0]` cu `1`, `b[0][1]` cu `0`, `b[1][0]` cu `3` şi `b[1][1]` cu `4`.

Programul următor demonstrează folosirea matricilor multidimensionale.

Exemplu

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

void TiparesteTablou(int [][3]);

int main()
{
     int array1[2][3] = {{1,2,3}, {4,5,6}};
     int array2[2][3] = {1,2,3,4,5};
     int array3[2][3] = {{1,2}, {4}};

     cout << "Valorile din array1 pe randuri sunt:" << endl;
     TiparesteTablou(array1);

     cout << "Valorile din array2 pe randuri sunt:" << endl;
     TiparesteTablou(array2);

     cout << "Valorile din array3 pe randuri sunt:" << endl;
     TiparesteTablou(array3);

     return 0;
}

void TiparesteTablou(int a[][3])
{
     int i = 0;
     while(i < 2)
     {
          int j = 0;
          while(j < 3)
          {
               cout << a[i][j] << ' ';
               j++;
          }
          cout << endl;
          i++;
     }
}
```

Programul apelează funcția `TiparesteTablou` pentru a tipări conținutul fiecăruia dintre cele trei tablouri multidimensionale. Atunci când se transmit tablouri unidimensionale ca parametri de funcții, indicele nu trebuie precizat. În cazul tablourilor multidimensionale, se ignoră doar valoarea primului indice, următorii trebuind să aibă valori. Compilatorul foloseşte aceste dimensiuni pentru a determina poziția în memorie a elementelor tabloului multidimensional. Toate elementele unui astfel de tablou sunt aşezate fizic în locații consecutive, mai întâi cele de pe primul

rând urmate imediat de elementele de pe al doilea rând etc. Această poziționare în memorie este ilustrată şi de modul în care sunt inițializate componentele tabloului `array2` din exemplul anterior. Valorile sunt asignate elementelor de pe primul rând, apoi celor de pe al doilea rând, iar `array2[1][2]` primeşte valoarea `0`.