Programarea calculatoarelor și limbaje de programare 1

Laborator 4
Matrice. Aritmetica pointerilor

Dr. Ing. Liviu-Daniel ȘTEFAN
liviu_daniel.stefan@upb.ro

Dr. Ing. Mihai DOGARIU
mihai.dogariu@upb.ro

Prof. Dr. Ing. Bogdan IONESCU
bogdan.ionescu@upb.ro

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Universitatea POLITEHNICA din Bucureşti

27-29 Noiembrie, 2023

Cuprins

Matrice unidimensionale

Semantică

O matrice este o colecție de variabile de același tip, apelate cu același nume și poate avea una sau mai multe dimensiuni. Accesul la un anumit element al matricei se face cu ajutorul unui indice. Cel mai mic indice corespunde primului element, iar cel mai mare, ultimului element. Toate matricele constau din locații de memorie contigue.

Sintaxă declarare

```
tip identificator [mărime];
```

- `tip` declară tipul de bază al matricei, care este tipul fiecărui element al său;
- `mărime` indică ce număr de elemente va conține matricea;
- ☞ Toate matricele au 0 ca indice pentru primul element.

Matrice unidimensionale

- Cantitatea de memorie necesară pentru înregistrarea unei matrice este direct proporțională cu tipul și mărimea sa.

Mărimea totală în octeți pentru matrice unidimensională

```
total octeți = sizeof(tip de baza) × mărimea matricei;
```

- ☞ Limbajul C/C++ nu controlează limitele unei matrice. Puteți depăși ambele margini ale unei matrice și scrie în locul altor variabile sau peste codul programului.

Sintaxă inițializare

```
tip identificator [mărime] = {listă_de_valori};
```

- `lista_de_valori` este o listă separată prin virgule de un tip compatibil cu tipul matricei. Prima constantă este plasată în prima poziție a matricei, a doua constantă în a două poziție și așa mai departe.

Șiruri de caractere

Semantică

Un șir de caractere este definit ca o matrice de caractere ASCII (*"American Standard Code for Information Interchange"*) care se termină cu un caracter nul. Un caracter nul este specificat prin '`\0`' (codul ASCII 0) și are valoarea 0.

- Declararea unui șir de caractere se realizează cu un caracter mai mult decât cel mai mare șir pe care îi este permis să îl conțină;
- Deși C nu specifică date de tip șir, el permite constante șir. O constantă șir este o listă de caractere închisă între ghilimele simple. Nu este necesar să introduceți manual caracterul nul la sfârșitul șirului de constante, compilatorul o face automat.

Matrice multidimensionale

- Limbajul C permite matrice cu mai mult de o dimensiune:

Sintaxă declarare

```
tip identificator[dimensiune1],...,[dimensiuneN];
```

- Cea mai simplă formă de matrice multidimensională este matricea bidimensională. O matrice bidimensională este o matrice de matrice unidimensionale;
- Matricele bidimensionale sunt stocate în forma rând-coloană (convenție `"rând-major"`), unde primul indice indică rândul iar al doilea precizează coloana;
- Pentru o matrice bidimensională de tip caracter, mărimea indicelui din stânga determină numărul de șiruri, iar mărimea indicelui din dreapta specifică mărimea maximă a fiecărui șir.

Matrice multidimensionale

Mărimea totală în octeți pentru matrice multidimensională

```
total octeți = dimensiune1 x,..., x dimensiuneN x
sizeof(tip de bază);
```

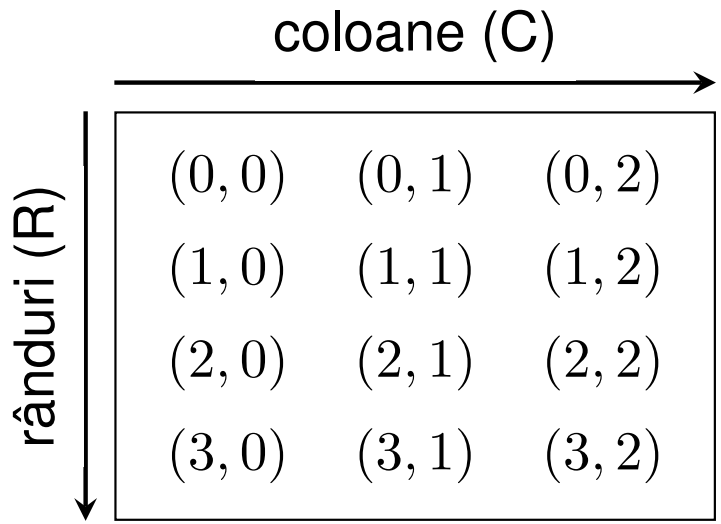

Figura: Indexare matrice bidimensională

Matrice multidimensionale

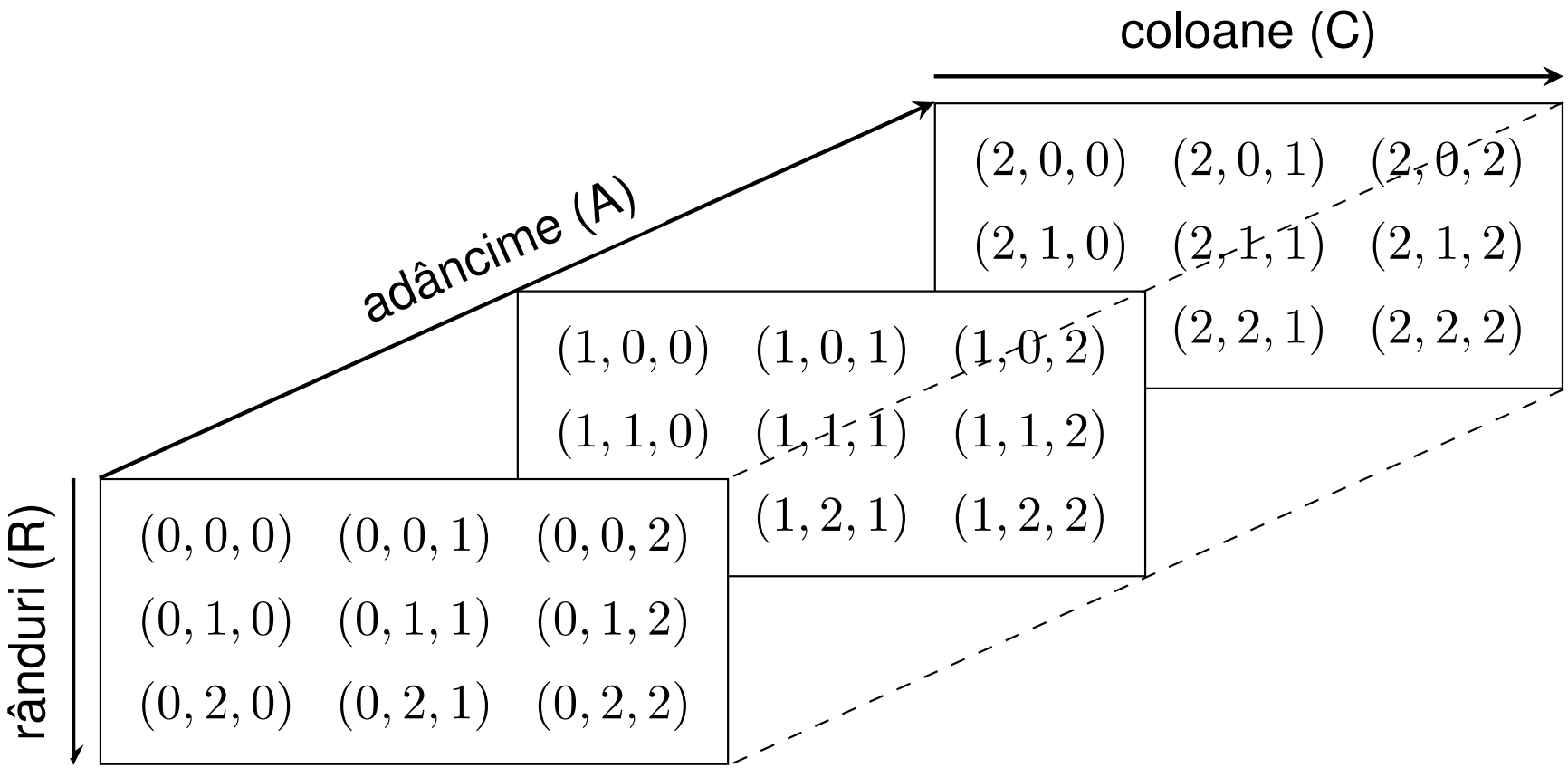

Figura: Inidexare matrice tridimensională $A \times R \times C$

Accesare matrice

- Limbajul C furnizează două metode de acces la elementele matricei:
  - aritmetica pointerilor;
    - ☞ Există doar două operații aritmetice care să pot efectua cu pointeri: adunarea și scăderea. Toți ceilalți operatori aritmetici sunt interzișii.
  - indecșii matricelor.
    - ☞ Accesarea elementelor unei matrice folosind indecși, poate fi mai lentă decât prin folosirea aritmeticii pointerilor.
  - Identificatorul unei matrice fără un indice generează un pointer la primul element al matricei. Invers, un pointer poate să aibă un indice ca și cum ar fi fost declarat ca o matrice.

Pointeri și matrice

- O matrice bidimensională poate fi redusă la un pointer la o matrice unidimensională;
- O matrice cu trei dimensiuni poate fi redusă la un pointer la o matrice bidimensională, care poate fi redusă la un pointer la o matrice unidimensională;
- O matrice n-dimensională poate fi redusă la un pointer la o matrice cu $(n-1)$ dimensiuni. Această nouă matrice poate fi redusă cu același metodă. Procesul se încheie până când se ajunge la o matrice unidimensională.

malloc()

Există numeroase cazuri în care memoria alocată static pentru o matice multidimensională nu este suficientă. Astfel, există un mecanism prin care memoria poate fi alocată dinamic în timpul rulării programului. Funcțiile care se ocupă de alocarea dinamică de memorie se regăsesc în fișierul antet `<stdlib.h>` și sunt următoarele:

- `malloc()`–alocă un singur bloc de memorie de dimensiune precizată și returnează un pointer de tip void către începutul blocului. Conținutul blocului de memorie este neinițializat, fiind populat cu valori nedeterminate.

Prototip `malloc()`

```
void* malloc (size_t numar_de_octeti);
```

```
1  /* Alocă memorie pentru un vector de 10 elemente de tip întreg */
2  #include <stdlib.h>
3  int main(){
4      int *p = (int*) malloc (10*sizeof(int));
5      return 0;
6  }
```

calloc()

- `calloc()`–alocă un bloc de memorie pentru un vector de n elemente de dimensiune precizată și returnează un pointer de tip `void` către începutul blocului. Spre deosebire de funcția `malloc()` unde valorile inițiale sunt nedeterminate, prin intermediul funcției `calloc()` toți biții alocați sunt inițializați cu valoarea 0.

Prototip `calloc()`

```
void* calloc (size_t numar_de_elemente, size_t numar_de_octeti);
```

```c
/* Alocă memorie pentru un vector de 10 elemente de tip întreg și le
inițializa pe toate cu valoarea 0 */

#include <stdlib.h>
int main(){
    int* p = (int*) calloc (10, sizeof(int));
    return 0;
}
```

realloc()

- `realloc()`—schimbă dimensiunea blocului de memorie către care indică un pointer și returnează un pointer de tip `void` către începutul noului bloc. Această funcție mută complet blocul de memorie către o nouă adresă.

Prototip `realloc`

```
void* realloc (void* pointer, size_t
numar_de_octeti);
```

```
1  /* Aloca memorie pentru un vector de 10 elemente de tip întreg, folosind un
2  pointer p, și îi mărește dimensiunea cu 10 prin realocarea memoriei */
3
4  #include <stdlib.h>
5  int main(){
6      int *p = (int*)malloc(10*sizeof(int));
7      p = (int*)realloc(p, 20*sizeof(int));
8
9      return 0;
10 }
```

free()

- `free()` -–eliberează blocul de memorie către care indică pointerul transmis ca argument, alocat anterior cu una dintre funcțiile malloc, calloc sau realloc.

Prototip `free()`

```
void free (void* pointer);
```

```
1  /* Aloca memorie pentru un vector de 10 elemente de tip întreg, folosind un
2  pointer p, și apoi eliberează memoria lui p */
3
4  #include <stdlib.h>
5  int main(){
6      int *p = (int*)malloc(10*sizeof(int));
7      free(p);
8
9      return 0;
10 }
```

Exemplu traversare matrice folosind indecși

```c
#include<stdio.h>
/* Inițializează de la tastatură o matrice bidimensională de dimensiuni maxime
10 x 10, citeste elementele de la tastatura, si le afiseaza in consola folosind
o parcurgere prin indecși.
*/
int main() {
  int i, j, randuri, coloane, M[10][10];
  printf("Introduceti numarul de randuri si coloane: ");
  scanf("%d %d", &randuri, &coloane);

  for (i = 0; i < randuri; i++) {
    for (j = 0; j < coloane; j++) {
        printf("M[%d][%d] = ", i, j);
        scanf("%d", &M[i][j]);
    }
  }
  for (i = 0; i < randuri; i++) {
    for (j = 0; j < coloane; j++) {
      printf("%d\t", M[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

Exemple alocare de memorie matrice unidimensională

```c
#include <stdio.h>
#include <stdlib.h>
/* Alocă memorie pentru un vector de dimensiune arbitrară, citește elementele
   vectorului de la tastatură și afișează sum tuturor elementelor folosind
   aritmetica pointerilor. */
int main() {
  int n, i, *ptr, suma = 0;
  printf("Introduceti dimensiunea vectorului: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));

  for(i = 0; i < n; ++i)
    scanf("%d", ptr + i);

  for(i = 0; i < n; ++i)
    printf("%d ", *(ptr + i));

  free(ptr);
  return 0;
}
```

Exemple alocare de memorie matrice bidimensională

```c
#include<stdio.h>
#include<stdlib.h>
/* Aloca memorie pentru o matrice bidimensională de dimensiuni arbitrare,
   citeste elementele de la tastatura, si le afiseaza in consola folosind
   aritmetica pointerilor */
int main() {
  int i, j, randuri, coloane, *M;
  printf("Introduceti numarul de randuri si coloane: ");
  scanf("%d %d", &randuri, &coloane);
  M = (int * ) malloc(randuri * coloane * sizeof(int));
  for (i = 0; i < randuri; i++) {
    for (j = 0; j < coloane; j++) {
        printf("M[%d][%d] = ", i, j);
        scanf("%d", (M + i * coloane + j));
    }
  }
  for (i = 0; i < randuri; i++) {
    for (j = 0; j < coloane; j++) {
      printf("%d\t", *(M + i * coloane + j));
    }
    printf("\n");
  }
  free(M);
  return 0;
}
```