

Structuri de date – Curs 1

Prof. univ. dr. Cristian CIUREA
Departmentul de Informatica si Cibernetica Economica
Academia de Studii Economice Bucuresti
cristian.ciurea@ie.ase.ro

Agenda

- ▶ Structura curs
- ▶ Evaluare
- ▶ Bibliografie recomandata
- ▶ Necesitatea disciplinei
- ▶ Clasificare structuri de date
- ▶ Memoria
- ▶ Pointeri
- ▶ Pointeri la functii
- ▶ Masive unidimensionale
- ▶ Masive bidimensionale

Structura curs

► Obiective:

- Prezentarea tuturor structurilor de date cu proprietatile, avantajele si dezavantajele utilizarii lor, in vederea utilizarii lor eficiente.
- Asimilarea conceptelor de proiectare și implementare a structurilor de date în procesul de dezvoltare software.
- Alegerea structurilor de date adecvate fiecărei aplicații informatice pe baza criteriilor de ordin practic.

Evaluare

Conditii de promovare:

- ▶ Teme si teste **online** scrise la calculator cu evaluare sincrona si asincrona – **40%**
- ▶ Test **online** practic scris la calculator – **60%**
- ▶ Criteriu minim promovare: obtinerea a minim 50% din punctajul examenului.

Bibliografie recomandata

- ▶ Ion Ivan, Marius Popa, Paul Pocatilu (coordonatori) – *Structuri de date*, Editura ASE, București, 2008.
 - Vol. I Tipologii de structuri de date
 - Vol. II Managementul structurilor de date
- ▶ Ion Smeureanu, Marian Dardala – *Programarea in limbajul C/C++*, Editura CISON, București, 2001.
- ▶ Bjarne Stroustrup – *The C++ Programming Language – 3rd Edition*, Editura Addison-Wesley,
<http://www.research.att.com/~bs/3rd.html>

Necesitatea disciplinei

- ▶ cunoasterea proprietatilor fiecărei structuri de date;
- ▶ alegerea celei mai adecvate structuri de date;
- ▶ efortul de programare sa fie cat mai mic;
- ▶ programul sa conduca la durate de executie cat mai reduse;
- ▶ programul sa fie cat mai usor de intretinut;
- ▶ depanarea sa necesite eforturi mici;

Clasificare

- ▶ Dupa criteriul alocarii memoriei, exista structuri de date:
 - statice (masive, articol, fisier);
 - dinamice (liste, stive, cozi, arbori).
- ▶ Dupa disciplina de parcurgere:
 - LIFO
 - FIFO
 - RSD
 - SRD
 - SDR

Memoria

- ▶ Organizarea memoriei la executia unui proces



- ▶ Tipuri de variabile:
 - ▶ automate – *auto*;
 - ▶ statice – *static*;
 - ▶ externe – *extern*;
 - ▶ registru – *register*.

Pointeri

▶ Pointer:

- variabila care contine adresa altei variabile;
- refera o variabila cunoscuta prin adresa zonei de memorie alocata acesteia.

▶ Definire:

```
tip_data * nume_pointer;
```

▶ Initializare:

```
nume_pointer = & nume_variabila;
```

▶ Utilizare:

```
nume_variabila = * nume_pointer;
```

Pointeri

▶ Exemple declarare:

- `int *px;` //pointer la int
- `char **ppx;` //pointer la pointer de char
- `int * vp[10];` //vector de 10 pointeri la int

▶ Exemple initializare:

- `int x=7, *px;`
- `px=&x; => *px=x;`
- `px=&768; => Eroare! Nu este permisa extragerea adresei unei constante!`

Pointeri

Aritmetica pointerilor:

- ▶ pentru un pointer de tip T^* , operatorii $--/++$ asigura deplasarea inapoi/inainte cu $\text{sizeof}(T)$ octeti;
- ▶ pentru un pointer de tip T^* pt , expresia $pt + k$ sau $pt - k$ este echivalenta cu deplasarea peste $k * \text{sizeof}(T)$ octeti;
- ▶ diferenta dintre 2 pointeri din interiorul aceluiasi sir de valori reprezinta numarul de elemente dintre cele doua adrese;
- ▶ adunarea dintre 2 pointeri nu este acceptata;

Pointeri la functii

- ▶ Definire:

`tip_return (*den_pointer) (lista_parametri);`

- ▶ Initializare:

`den_pointer = den_functie;`

- ▶ Apel functie prin pointer:

`den_pointer (lista_parametri);`

Pointeri la functii

- ▶ `float (*fp)(int*);` //pointer la functie ce primeste un pointer la `int` si ce returneaza un `float`
- ▶ `int *f(char*);` //functie ce primeste `char*` si returneaza un pointer la `int`
- ▶ `int *(*fp[5]) (char*);` //vector de 5 pointeri la functii ce primesc `char*` si returneaza un pointer la `int`

Masive

- ▶ Masivele:
 - structuri de date omogene;
 - numar finit si cunoscut de elemente;
 - ocupa un spatiu contiguu de memorie.
- ▶ Caracteristici:
 - denumirea;
 - tipul de date asociat;
 - numarul de dimensiuni;
 - numarul de elemente pentru fiecare dimensiune.

Masive unidimensionale

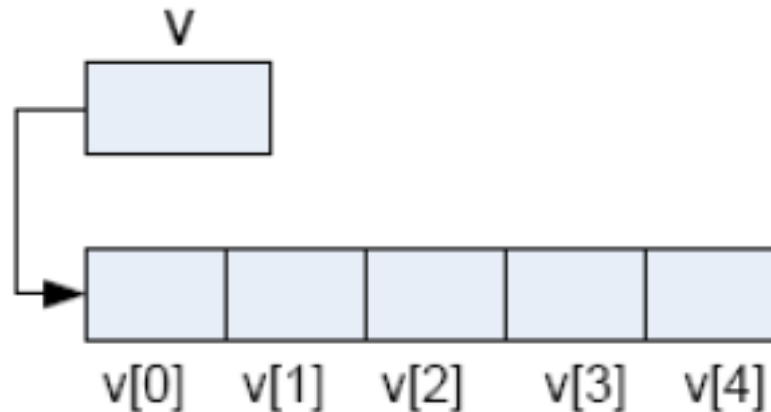
- ▶ Sintaxa de declarare:

tip nume[n];

- ▶ Definire:

- in varianta statica: **int vec[100];**
- in varianta dinamica: **int *vec;**

- *Denumirea variabilei vector este pointer catre adresa primului element!*



Masive unidimensionale

- ▶ Initializarea vectorului la declarare:

tip nume[] = { lista_valori };

- ▶ Exemplu initializare fara precizarea numarului de elemente (dedus de compilator):

- **int v1[] = {1, 2, 3, 4, 5};**

- ▶ Exemplu initializare partiala:

- **int v2[5] = {7, 6, 5};**

Masive bidimensionale

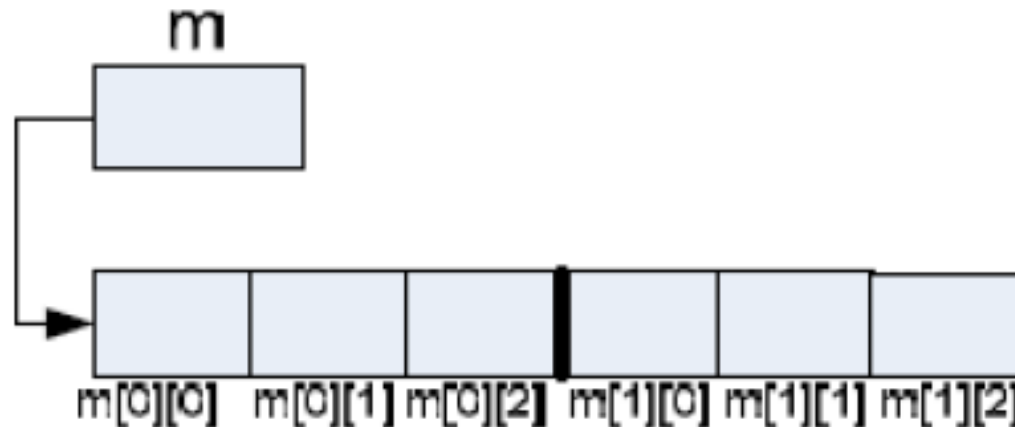
- ▶ Sintaxa de declarare:

tip nume[lin][col];

- ▶ Definire:

- in varianta statica: **int mat[10][10];**
- in varianta dinamica: **int **mat;**

- *Denumirea variabilei matrice este pointer catre adresa primului element!*



Masive bidimensionale

- ▶ Initializarea matricei la declarare:
`tip nume[][n] = {{lista1_val}, {lista2_val}..., {listam_val}};`
- ▶ Exemplu initializare fara precizarea numarului de elemente (dedus de compilator):
 - `int m1[][2] = {{1, 2}, {3, 4}, {5, 6}};`
- ▶ Exemplu initializare partiala:
 - `int m2[2][5] = {{7, 6}, {5}};`

Masive unidimensionale in HEAP

Caracteristici:

- ▶ Pointer catre tipul de date asociat unui element al masivului;
- ▶ Rezervare memorie heap la momentul executiei (nu la compilare) si initializarea variabilei pointer cu adresa primului element;
- ▶ Posibilitatea schimbarii adresei in timpul executiei programului (rezervarea la compilare nu permite acest lucru).

Masive unidimensionale in HEAP

- ▶ Sintaxa de declarare a unui vector alocat in heap:
tip *pnumev;
- ▶ Rezervarea de memorie heap: operator **new**;
pnumev = new tip[n];
- ▶ Referirea unui element din vectorul alocat in heap: pe baza deplasamentului de element cu baza in **pnumev**: ***(pnumev+i)** echivalent **pnumev[i];**
- ▶ Dezalocarea de memorie heap: operator **delete**;
delete [] pnumev;

Masive unidimensionale in HEAP

- ▶ Semnificatii ale zonelor de memorie accesate prin deplasamente:
 - **pnumev** – pointerul catre primul element al vectorului; adresa primului element al vectorului; adresa de inceput a zonei de stocare a elementelor vectorului;
 - **pnumev+i** – adresa elementului cu deplasamentul **i** fata de adresa de inceput a vectorului; adresa elementului **i+1**;
 - ***(pnumev+i)** – continut de la adresa **pnumev+i**; valoarea elementului cu deplasamentul **i**; valoarea elementului **i+1**;

Masive bidimensionale in HEAP

Caracteristici:

- ▶ Pointer catre un vector de pointeri (adrese ale liniilor); adresa de linie: pointer catre tipul de date asociat unui element al matricei;
- ▶ Rezervare memorie heap la momentul executiei (nu la compilare) si initializarea variabilei pointer cu adresa primului element: adresa primei linii din matrice;

Masive bidimensionale in HEAP

- ▶ Sintaxa de declarare a unei matrice alocate in heap:

tip **pnumem;

- ▶ Rezervarea de memorie heap: operator **new**;
- ▶ Rezervarea de memorie heap pentru vectorul de pointeri catre liniile matricei:

pnumem = new tip* [m];

- ▶ Rezervarea de memorie heap pentru liniile matricei cu elementele propriu-zise:

for(int i=0; i<m; i++)

***(pnumem+i) = new tip[n];**

Masive bidimensionale in HEAP

- ▶ Referirea unui element din matrice alocata in heap: pe baza deplasamentului adresei de linie si a deplasamentului de element cu baza in `*(pnumem+i):` `*(*(pnumem+i)+j)` echivalent `pnumem[i][j]`;
- ▶ Dezalocarea matricei din memoria heap: operator **delete**; pasi (invers alocarii):
 1. Dezalocarea liniilor;
 2. Dezalocarea vectorului de pointeri:

```
for(int i=0; i<m; i++)  
    delete [ ] *(pnumem+i);  
delete [ ] pnumem;
```


Masive bidimensionale in HEAP

- ▶ Semnificatii ale zonelor de memorie accesate prin deplasamente (aritmetica de pointeri):
 - **pnumem** – pointer catre primul element din vectorul de pointeri catre linii (adresa liniei **1**);
 - **pnumem+i** – adresa elementului cu deplasamentul **i** in vectorul de pointeri (adresa unde se afla pointerul catre linia **i+1**);
 - ***(pnumem+i)** – continut de la adresa **pnumem+i**: pointerul catre linia cu deplasamentul **i** (adresa de inceput a liniei **i+1**);
 - ***(pnumem+i)+j** – adresa elementului matricei pozitionat pe linia **i+1**, coloana **j+1**;
 - ***(*(pnumem+i)+j)** – elementul matricei pozitionat pe linia **i+1**, coloana **j+1**;

Bibliografie

- ▶ Ion Ivan, Marius Popa, Paul Pocatilu (coordonatori) – *Structuri de date*, Editura ASE, București, 2008.
 - Cap. 4. Masivele – structuri de date omogene și contigue
 - Cap. 5. Funcții de prelucrare cu masive
 - Cap. 8. Variabile pointer