Structuri de date – Curs 1

Prof. univ. dr. Cristian CIUREA
Departamentul de Informatică și Cibernetică Economică
Academia de Studii Economice din București
cristian.ciurea@ie.ase.ro

Agendă

- Structură curs
- Structură evaluare
- Bibliografie recomandată
- Necesitatea disciplinei
- Clasificare structuri de date
- Memoria
- Pointerii
- Pointeri la funcții
- Masive unidimensionale
- Masive bidimensionale

Structură curs

- **Obiective**:
  - Prezentarea tuturor structurilor de date cu proprietățile, avantajele și dezavantajele utilizării acestora, în vederea utilizării lor eficiente.
  - Asimilarea conceptelor de proiectare şi implementare a structurilor de date în procesul de dezvoltare software.
  - Alegerea structurilor de date adecvate fiecărei aplicaţii informatice pe baza criteriilor de ordin practic.

Structură evaluare

Condiții de promovare:

- Test practic la calculator 20% – evaluarea se face cu note 1-10. Absența se evaluează cu nota 0.
- Evaluări la latitudinea profesorului 20% – fiecare element/temă se evaluează cu nota 1-10. Absența se evaluează cu nota 0.
- Examen oral la calculator 60%, evaluat cu nota 1-10. Condiția de promovare este minim nota 5.

Bibliografie recomandată

- Marius Popa, Cristian Ciurea, Mihai Doinea, Alin Zamfiroiu – *Structuri de date: teorie şi practică*, Editura ASE, Bucureşti, 2023, 280 pg.
- Ion Ivan, Marius Popa, Paul Pocatilu (coordonatori) – *Structuri de date*, Editura ASE, Bucureşti, 2008.
- Ion Smeureanu, Marian Dârdală – *Programarea în limbajul C/C++*, Editura CISON, Bucureşti, 2001.
- Bjarne Stroustrup – *The C++ Programming Language – 3rd Edition*, Editura Addison-Wesley, http://www.research.att.com/~bs/3rd.html

Necesitatea disciplinei

- Cunoașterea proprietăților fiecărei structuri de date;
- Alegerea celei mai adecvate structuri de date;
- Efortul de programare să fie cât mai mic;
- Programul să conducă la durate de execuție cât mai reduse;
- Programul să fie cât mai ușor de întreținut;
- Depanarea să necesite eforturi mici;

Clasificare

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

Memoria

- Organizarea memoriei la execuția unui proces

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

Pointerii

- Pointer:
  - variabilă care conține adresa unei alte variabile;
  - referă o variabilă cunoscută prin adresa zonei de memorie alocată acesteia.
- Definire:

```
tip_data * nume_pointer;
```

- Inițializare:

```
nume_pointer = & nume_variabila;
```

- Utilizare:

```
nume_variabila = * nume_pointer;
```

Pointerii

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

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

Pointerii

Aritmetica pointerilor:

- pentru un pointer de tip T*, operatorii --/++ asigură deplasarea înapoi/înainte cu sizeof(T) octeți;
- pentru un pointer de tip T* pt, expresia pt + k sau pt - k este echivalentă cu deplasarea peste k * sizeof(T) octeți;
- diferența dintre 2 pointeri din interiorul aceluiași șir de valori reprezintă numărul de elemente dintre cele două adrese;
- adunarea dintre 2 pointeri nu este acceptată;

Pointeri la funcții

- Definire:

  tip_return (*den_pointer) (lista_parametri);

- Inițializare:

  den_pointer = den_functie;

- Apel funcție prin pointer:

  den_pointer (lista_parametri);

Pointeri la funcții

- 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

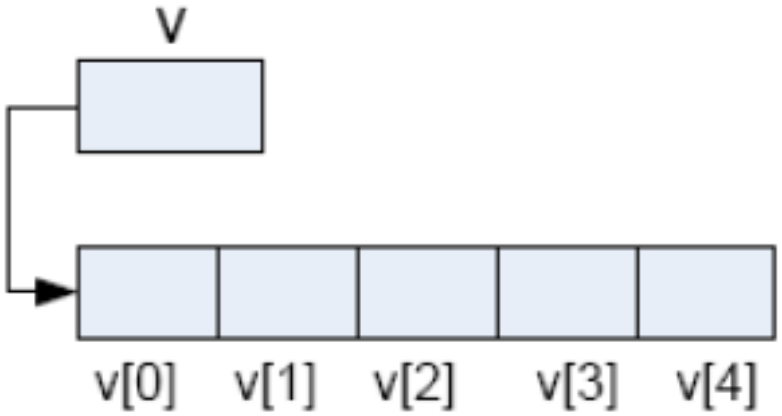
- Masivele:
  - structuri de date omogene;
  - număr finit și cunoscut de elemente;
  - ocupă un spațiu contiguu de memorie.
- Caracteristici:
  - denumirea;
  - tipul de date asociat;
  - numărul de dimensiuni;
  - numărul de elemente pentru fiecare dimensiune.

Masive unidimensionale

- Sintaxa de declarare:

  tip nume[n];

- Definire:
  - în varianta statica: int vec[100];
  - în varianta dinamica: int *vec;
  - *Denumirea variabilei vector este pointer către adresa primului element!*



Masive unidimensionale

- Inițializarea vectorului la declarare:

  tip nume[ ] = { lista_valori };

- Exemplu inițializare fără precizarea numărului de elemente (dedus de compilator):
  - int v1[ ] = {1, 2, 3, 4, 5};

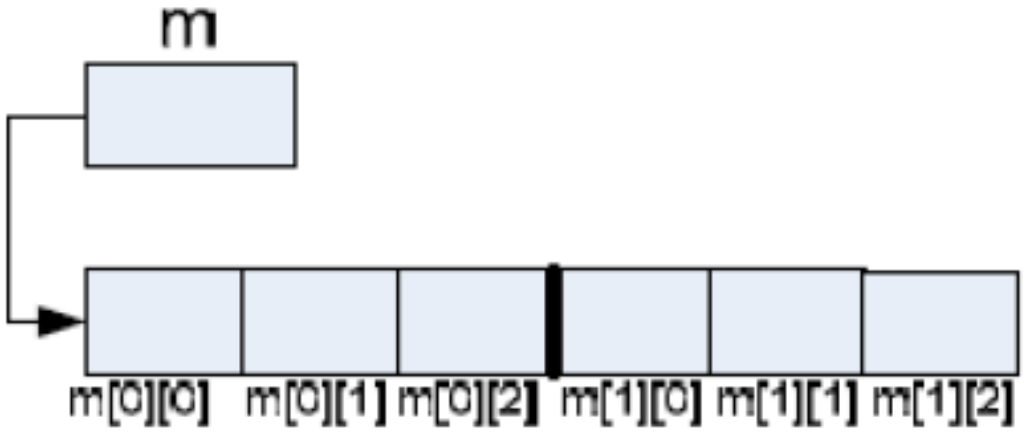
- Exemplu inițializare parțială:
  - int v2[5] = {7, 6, 5};

Masive bidimensionale

- Sintaxa de declarare:

  tip nume[lin][col];

- Definire:
  - în varianta statica: int mat[10][10];
  - în varianta dinamica: int **mat;
    - *Denumirea variabilei matrice este pointer către adresa primului element!*



Masive bidimensionale

- Inițializarea matricei la declarare:

  tip nume[ ][n] = {{lista1_val}, {lista2_val}…, {listam_val}};

- Exemplu inițializare fără precizarea numărului de elemente (dedus de compilator):
  - int m1[ ][2] = {{1, 2}, {3, 4}, {5, 6}};

- Exemplu inițializare parțială:
  - int m2[2][5] = {{7, 6}, {5}};

Masive unidimensionale în HEAP

Caracteristici:

- Pointer către tipul de date asociat unui element al masivului;
- Rezervare memorie heap la momentul execuției (nu la compilare) și inițializarea variabilei pointer cu adresa primului element;
- Posibilitatea schimbării adresei în timpul execuției programului (rezervarea la compilare nu permite acest lucru).

Masive unidimensionale în HEAP

- Sintaxa de declarare a unui vector alocat în heap:

```
tip *vect;
```

- Rezervarea de memorie heap: funcția **malloc**;

```
vect = (tip*)malloc(n * sizeof(tip));
```

- Referirea unui element din vectorul alocat în heap: pe baza deplasamentului de element cu baza în **vect**: ***(vect+i)** echivalent **vect[i]**;
- Dezalocarea de memorie heap: funcția **free**;

```
free(vect);
```

Masive unidimensionale în HEAP

- Semnificații ale zonelor de memorie accesate prin deplasamente:
  - **vect** – pointerul către primul element al vectorului; adresa primului element al vectorului; adresa de început a zonei de stocare a elementelor vectorului;
  - **vect+i** – adresa elementului cu deplasamentul **i** față de adresa de început a vectorului; adresa elementului **i+1**;
  - ***(vect+i)** – conținut de la adresa **vect+i**; valoarea elementului cu deplasamentul **i**; valoarea elementului **i+1**;

Masive bidimensionale în HEAP

Caracteristici:

- Pointer către un vector de pointeri (adrese ale liniilor); adresa de linie: pointer către tipul de date asociat unui element al matricei;

- Rezervare memorie heap la momentul execuției (nu la compilare) și inițializarea variabilei pointer cu adresa primului element: adresa primei linii din matrice;

Masive bidimensionale în HEAP

- Sintaxa de declarare a unei matrice alocate in heap:

```
tip **mat;
```

- Rezervarea de memorie heap: funcția **malloc**;
- Rezervarea de memorie heap pentru vectorul de pointeri către liniile matricei:

```
mat = (tip**)malloc(nrLin * sizeof(tip*));
```

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

```
for(int i=0; i<nrLin; i++)
        *(mat+i) = (tip*)malloc(nrCol * sizeof(tip);
```

Masive bidimensionale în HEAP

- Referirea unui element din matricea alocată în heap: pe baza deplasamentului adresei de linie și a deplasamentului de element cu baza în ***(mat+i)**: ***(*(mat+i)+j)** echivalent **mat[i][j]**;
- Dezalocarea matricei din memoria heap: funcția **free**; pași (invers alocării):

1. Dezalocarea liniilor;
2. Dezalocarea vectorului de pointeri:

```
for(int i=0; i<nrLin; i++)
        free(mat[i]);
free(mat);
```

Masive bidimensionale în HEAP

- Semnificații ale zonelor de memorie accesate prin deplasamente (aritmetica de pointeri):
  - **mat** – pointer către primul element din vectorul de pointeri către linii (adresa liniei 1);
  - **mat+i** – adresa elementului cu deplasamentul **i** în vectorul de pointeri (adresa unde se află pointerul către linia i+1);
  - ***(mat+i)** – conținut de la adresa **mat+i**: pointerul către linia cu deplasamentul i (adresa de început a liniei i+1);
  - ***(mat+i)+j** – adresa elementului matricei poziționat pe linia i+1, coloana j+1;
  - ***(*(mat+i)+j)** – elementul matricei poziționat pe linia i+1, coloana j+1;

Bibliografie

- Marius Popa, Cristian Ciurea, Mihai Doinea, Alin Zamfiroiu – *Structuri de date: teorie şi practică*, Editura ASE, Bucureşti, 2023, 280 pg.
  - Cap. 2. Reprezentarea internă a datelor
  - Cap. 3. Masive de date