

POINTERI. ALOCAREA DINAMICĂ A MEMORIEI

1. Pointeri

Un pointer este o variabilă care conține adresa altei variabile, a unei funcții dacă este inițializat cu această adresă.

2. Alocarea dinamică a memoriei

void *malloc(size_t n);

- alocă un bloc de memorie de **n** octeți în memoria heap;
- în caz de succes, returnează un adresa zonei de memorie alocate (căreia nu-i schimbă conținutul)
- returnează valoarea 0 (zero) dacă nu s-a putut face alocarea de memorie (pentru că nu există spațiu liber cu mărimea solicitată sau dacă se apelează cu $n = 0$).

Funcția returnează un pointer generic (de tip void), a cărui valoare poate fi asignată unui pointer de orice tip folosind o conversie explicită.

Întotdeauna trebuie testată valoarea returnată (dacă este sau nu 0 (zero)) pentru că în cazul nealocării memoriei programul nu trebuie să continue.

Exemplu:

```
int *p = 0;
int n = 5;
p = (int *)malloc(n * sizeof(int));
if(p == 0)
{
    fprintf(stderr, "Memorie insuficienta\n");
    exit(EXIT_FAILURE);
}
```

Pentru că această testare trebuie să o facem ori de câte ori alocăm memorie pentru un pointer se poate scrie o funcție care să conțină această testare. Prototipul acestei funcții (care nu se găsește în nici o bibliotecă a limbajului) este:

void *aloca1d(size_t nrOcteti);

(**size_t** este definit în **stdio.h** ca sinonim pentru **unsigned int**)

Definiția acestei funcții este:

```
void * aloca1d(size_t nrOcteti)
{
    void *p;
    p = malloc(nrOcteti);
    if(p == 0)
    {
        fprintf(stderr, "Memorie insuficientă\n");
        exit(EXIT_FAILURE);
    }
    return p;
}
```

În funcția în care se face alocarea dinamică (funcția apelantă) secvența de cod este:

```
int *a = 0;
int n = 4;
.....
a = (int *)aloca1d(n * sizeof(int);    // aloca spațiul necesar pentru a stoca
                                       // n elemente de tip întreg
.....
```

void *calloc(size_t nrElemente, size_t dimElement);

– alocă un bloc de memorie de mărime nrElemente * dimElement (care nu trebuie să depășească 64 Ko octeți din memoria heap), conținutul blocului fiind resetat (se scrie 0);

– în caz de succes, returnează adresa blocului de memorie alocat;

– returnează 0 dacă nu există spațiu liber de mărimea solicitată (sau dacă se apelează cu valoarea 0).

void *realloc(void* block, size_t marime);

– funcția redimensionează (prim mărime sau micșorare) un bloc de memorie (alocat dinamic anterior) la numărul de octeți specificați de parametrul **marime**;

– block trebuie să indice un bloc de memorie obținut prin apelarea funcțiilor malloc() / calloc() / realloc() (altfel rezultatul este imprevizibil);

– dacă block este 0 (zero), lucrează exact ca malloc();

– funcția ajustează mărimea blocului alocat la **marime**, copiind (dacă este cazul) conținutul său la o nouă adresă;

– returnează adresa blocului realocat (poate diferi de block) sau 0 (zero), dacă nu se poate face realocarea sau marime = 0 (în acest ultim caz funcția lucrează ca și funcția free()).

void free(void *block);

– eliberează un bloc de memorie alocat anterior de către malloc() / calloc() / realloc());

– rezultatul este dezastruos dacă parametrul nu este rezultatul unei alocări dinamice anterioare sau dacă se apelează de două ori la rând cu același parametru.

– după folosirea funcției **free** se recomandă invalidarea pointerului (pentru a nu mai putea fi folosit accidental în continuare) prin atribuirea valorii 0 (zero) (vezi exemplul următor).

Exemplu:

```
int *a = 0;
int n = 3;
/*
 * Alte definiri de date și instrucțiuni
 */
.....

a = (int *)aloca1d(n * sizeof(int));    // alocă spațiul necesar pentru a stoca
                                         // n elemente de tip întreg

/*
 * Prelucrări de date
 */
.....
/*
 * Eliberarea memoriei
 */
free(a);
a = 0;                                // Obligatoriu pentru a invalida pointerul
```

TEMA 1**Problema nr. 1.1**

Să se citească de la tastatură elementele unui vector cu **n** elemente numere întregi și să se afișeze pe monitor. Vor trebui scrise două funcții:

- ✓ funcție pentru citirea unui vector de întregi cu **n** elemente. Funcția trebuie să aibă următorul prototip:


```
int *citireVectorP(size_t n);
```
- ✓ funcție pentru afișarea elementelor unui vector de întregi, cu prototipul:


```
void afisareVectorP(int *a, size_t n);
```

Problema nr. 1.2

Să se citească de la tastatură elementele unui vector cu **n** elemente numere întregi din fișierul **doi.txt** și să se afișeze pe monitor. Vor trebui scrise două funcții:

- ✓ funcție pentru citirea unui vector de întregi cu **n** elemente. Funcția trebuie să aibă următorul prototip:


```
void citireVectorP(int *a, size_t n);
```

- ✓ funcție pentru afișarea elementelor unui vector de întregi, cu prototipul:
`void afisareVectorP(int *a, size_t n);`

Problema nr. 1.3

Să se citească de la tastatură elementele a două tablouri unidimensionale (vectori) de **n** numere întregi **a** și **b** alocate dinamic. Să se calculeze vectorul sumă și să se afișeze.

Rezolvarea problemei presupune construirea unui proiect (cu fișierul header corespunzător) și scrierea următoarelor funcții:

- ✓ funcție pentru: alocarea cu verificarea alocării (funcția **aloca1d** din curs);
- ✓ funcție pentru citirea unui vector de întregi cu **n** elemente. Funcția trebuie să aibă următorul prototip:
`int *citireVectorP(size_t n);`
- ✓ funcție pentru afișarea elementelor unui vector de întregi, cu prototipul:
`void afisareVectorP(int *a, size_t n);`
- ✓ funcție pentru calculul vectorului sumă a doi vectori, cu prototipul:
`int *sumaVectori(int *x, int *y, size_t n);`

Problema nr. 1.4

Se citește de la tastatură un număr natural **n** și un vector de numere naturale **v** pentru care se face alocare dinamică de memorie.

Se formează un nou vector **w** cu **n** elemente, de asemenea alocat dinamic, în care valoarea fiecărui element este suma cifrelor elementului corespunzător din vectorul **v**.

Să se afișeze elementul din vectorul **v** care are cea mai mare sumă a cifrelor sale.

Se vor scrie următoarele funcții:

- ✓ Funcție pentru citirea unui vector de numere naturale care are ca parametru numărul de elemente și returnează un pointer.
- ✓ Funcție pentru afișarea vectorului de numere naturale sub forma

`A = (23, 543, 912)`

Funcția are ca parametri vectorul de afișat (exprimat ca un pointer) și numărul de elemente.

✓ Funcție pentru calculul sumei cifrelor unui număr natural. Funcția are ca parametru un număr natural (numărul pentru care se calculează suma cifrelor) și returnează un număr natural (suma calculată).

✓ Funcție pentru determinarea elementelor vectorului **w**. Funcția are ca parametri un pointer și un număr natural și returnează un pointer la un număr natural. Această funcție face apel, pentru calculul sumei cifrelor unui număr natural, la funcția definită la punctul anterior.

✓ Funcție pentru determinarea maximului dintr-un șir de numere. Funcția primește ca parametri un pointer la un număr natural și un întreg și returnează indexul elementului cu valoarea maximă.

TEMA 2

Problema nr. 2.1

Se citește de la tastatură un număr întreg (poate fi pozitiv sau negativ). Citirea se face cu `scanf`. Se determină numărul de cifre ale numărului (prin logaritmare în baza 10).

Se face alocare dinamică de memorie pentru un șir de caractere capabil să memoreze cifrele numărului și eventualul semn. Numărul citit se transformă în șir de caractere, fără a folosi funcții de bibliotecă. Transformarea se face astfel încât fiecare cifră să se plaseze direct pe locul ei.

Se afișează șirul de caractere rezultat.

Rezolvarea problemei presupune realizarea unui proiect care, pe lângă funcția **main**, să cuprindă și funcții pentru:

- ✓ alocare dinamică cu verificare (funcția **aloca1d** din curs)
- ✓ transformarea numărului în șir de caractere. În acest caz funcția are prototipul:
`char *transformareToAscii(long int);`

Problema nr. 2.2

Să se definească tipul de dată **MULTIME** ca o structură care cuprinde:

- cardinalul mulțimii (numărul de elemente din mulțime) care este un număr întreg fără semn;
- elementele mulțimii (de tip real), stocate prin intermediul unui pointer

Să se scrie un program care pentru două mulțimi

1. citește de la tastatură cardinalul și elementele mulțimii și le stochează într-o structură de tip **MULTIME**.

2. afișează cele două mulțimi sub forma

$A = \{3.14, 5.12, 3.00, 4.39\}$

$B = \{34.29, 15.14, 3.14\}$

(se presupune că numele celor două mulțimi sunt **A** și **B**, iar numerele reale se vor afișa cu două zecimale și vor fi separate de o virgulă și un spațiu).

Observație (referitoare la punctele 1 și 2):

Se scriu funcții pentru o singură mulțime. Funcțiile se apelează de mai multe ori.

3. afișează un meniu care dă posibilitatea utilizatorului să aleagă una din următoarele prelucrări:

a) determinarea mulțimii intersecție a celor două mulțimi (**A&B**) și cardinalul acestei mulțimi.

b) determinarea mulțimii diferență simetrică a celor două mulțimi (**A-B**) (mulțimea diferență simetrică include elementele mulțimii **A** care nu sunt în mulțimea **B** și elementele mulțimii **B** care nu sunt în mulțimea **A**) și cardinalul acestei mulțimi.

c) determinarea mulțimii reuniune a celor două mulțimi (**A+B**) și cardinalul acestei mulțimi.

4. Realizează prelucrarea dorită și afișează mulțimea care rezultă în urma prelucrării.

Programul poate face o singură prelucrare în funcție de opțiunea utilizatorului și va trebui scris astfel încât să permită prelucrarea mai multor seturi de date.

Observații:

1) Se va citi cu atenție tabelul următor care descrie în detaliu modul de rezolvare a problemei și descrierea prototipurilor funcțiilor care trebuie scrise.

2) Dacă nu se folosesc funcțiile indicate codul nu se punctează.

3) Punctajul maxim se acordă pentru rezolvarea **CORECTĂ** a fiecărei subprobleme.

1. Construirea structurii MULTIME (funcția primește ca parametru numele mulțimii și returnează o structură de tip MULTIME)	1,0
2. Alocarea corectă (cu verificare) de spațiu de memorie pentru pointeri	0,5
3. Citirea valorilor elementelor mulțimii (într-o funcție care primește ca parametru un număr natural și returnează un pointer la real)	1,0
4. Afișarea valorilor vectorului din structură conform modelului dat în problemă (funcția are un parametru – o structură de tip MULTIME și nu returnează nimic)	1,0
5. Scrierea meniului de prelucrare (funcția nu are nici un parametru și returnează un număr care reprezintă numărul opțiunii de prelucrare)	0,75
6. posibilitatea de reluare a programului (prelucrarea mai multor seturi de date)	0,5
7. Folosire proiect (corect)	0,5
8. Fișier header (corect și complet)	0,5
9. Funcția main (complet – inclusiv eliberarea corectă a zonelor de memorie folosite)	0,75
10. Funcție care stabilește dacă o valoare aparține unei mulțimi. Funcția are 2 parametri: valoarea și mulțimea (reprezentată prin structură) și returnează 1/0 după cum valoarea se găsește sau nu în mulțime.	0,5
11. Calculul mulțimii intersecție (funcția primește ca parametri două structuri de tip MULTIME și returnează o structură de tip MULTIME) – inclusiv alocarea corectă de memorie.	1,0
12. Calculul mulțimii diferență (funcția primește ca parametri două structuri de tip MULTIME și returnează o structură de tip MULTIME) – inclusiv alocarea corectă de memorie.	1,0
13. Calculul mulțimii reuniune (funcția primește ca parametri doi pointeri la date de tip MULTIME și returnează o structură de tip MULTIME) – inclusiv alocarea corectă de memorie.	1,0
TOTAL	10 p