POINTERI. ALOCAREA DINAMICĂ A MEMORIEI

1. Pointeri

Un pointer este o variabilă care conține adresa unei zone de memorie care conține o variabilă sau o funcție (dacă este inițializat cu adresa variabilei sau a funcției) sau adresa unei zone de memorie (obținută în urma unei alocări dinamice de memorie).

Definirea unei variabile pointer:

```
int *pi;        // pi este pointer la întreg - poate conține adresa unui întreg
double *pd;     // pd este pointer la double - poate conține adresa unui double
```

Aflarea adresei unei variabile și extragerea conținutului unei adrese

```
int x=3, y;
int *pi = &x;   // pi conține adresa lui x
y = *pi;        // extrage conținutul de la adresa onținută în pi
                // (adica valoarea lui y este tot 3)
*pi = 5;        // x devine 5, y rămâne la valoarea 3
```

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);
}
```

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 valorea 0).

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

– funcția redimensionează (prim mărire 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 *p = 0;
int n = 3;
/*
 * Alte definiri de date și instrucțiuni
 */
.....
p = (int *)malloc(n * sizeof(int));
if(p == 0)
{
        fprintf(stderr, "Memorie insuficienta\n");
        exit(EXIT_FAILURE);
}

/*
 * Prelucrari de date
 */

.....
```

```
/*
 * Eliberarea memoriei
 */
free(p);
p = 0;          // Obligatoriu pentru a invalida pointerul
```

TEMA

Problema nr. 1.1

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

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

  void citireVectorP(FILE *f, int *a, int n);
- ✓ funcție pentru afișarea elementelor unui vector de întregi, cu prototipul:

  void afisareVectorP(int *a, int n);

Problema nr. 1.2

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

  void afisareVectorP(int *a, int n);

Problema nr. 1.3

Să se citească de la tastatură elementele a două tablouri unidimensionale (vectori) de **n** numere reale **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 citirea unui vector de întregi cu **n** elemente. Funcția trebuie să aibă următorul prototip:

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

  void afisareVectorP(double *a, int n);
- ✓ funcție pentru calculul vectorului sumă a doi vectori, cu prototipul:

  double *sumaVectori(double *x, double *y, int 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.

Problema nr. 1.5

Se citește de la tastatură un număr întreg (care 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:

- ✓ transformarea numărului în șir de caractere. În acest caz funcția are prototipul:

```
char *transformareToAscii(long int);
```