Introducere în Programarea calculatoarelor
Laborator 5

Structuri statice si dinamice

Obiectiv: familiarizarea cu structurile statice şi dinamice asociate cu limbajul C

Activităţi:
- prezentarea tablourilor (unidimensionale şi multidimensionale) şi a constantelor de tip tablou;
- prezentarea pointer-ilor;
- alocarea dinamică de memorie, operatorul *sizeof*;
- prezentarea relaţiei dintre tablouri şi pointeri, aritmetica pointerilor;

Tablouri

Un *tablou* de elemente reprezintă o structură de date alcătuită din mai multe variabile din acelaşi tip de date.

Tablouri unidimensionale

Tablourile unidimensionale sunt alcătuite dintr-un grup de elemente de acelaşi tip (numit *tip de bază*) şi sunt referite printr-un nume comun. Tablourile sunt stocate în memorie la locaţii consecutive, un tablou ocupând o zonă contiguă de memorie, cu primul element al tabloului aflat la adresa cea mai mică.

Variabilele de tip tablou se definesc astfel:

```
nume_tip nume_var[dimensiune];
```

Exemplu de declaraţie de tablou:

```
int v[4];          // declararea tabloului v, care are 4 elemente
```

Iniţializarea elementelor unui tablou se poate face în următoarele moduri:

```
v[0] = 21;
v[1] = 23;
v[2] = 27;
v[3] = 12;
```

sau:

```
int v[4] = { 21, 23, 27, 12 };
```

Introducere în Programarea calculatoarelor
Laborator 5

În limbajul C, un tablou se poate iniţializa şi fără dimensiune:

```
int d[] = { 1, 4, 6, 3 }; // tabloul va avea dimensiunea 4
```

Dacă nu se specifică dimensiunea unui tablou atunci când se declară, compilatorul va aloca suficientă memorie pentru a păstra elementele matricei respective.

Observaţie: în limbajul C numerotarea elementelor unui tablou începe cu poziţia 0. Astfel, daca avem definiţia:

```
int tablou[10];
```

atunci primul element al tabloului este *tablou[0]*, iar ultimul element al tabloului este *tablou[9]*.

Accesarea unui element al unui tabloului se face folosind ca index poziţia elementului. Astfel, tablou[3] va referi al 4-lea element al tabloului *tablou*.

Cantitatea de memorie necesară pentru memorarea unui tablou este determinată de tipul şi numărul elementelor tabloului. Pentru un tablou unidimensional, cantitatea de memorie ocupată se calculează astfel: mem_ocupată = dimensiune_octeţi * mărime_tablou.

Atenţie! o problemă legată de tablouri este că în C nu se face nici o verificare legată de "marginile" tabloului, astfel încât se pot accesa greşit elemente din afara tabloului. De exemplu, pentru definiţia:

```
int tablou[10];
```

dacă se încearcă accesarea elementului tablou[15] nu se va semnala nici o eroare, returnându-se valoarea de la o locaţie de memorie aflată la o distanţă de 5 locaţii faţă de sfârşitul tabloului, fapt ce poate duce la comportări "bizare" ale programului.

Aceeaşi situaţie, dar faţă de începutul tabloului, se întâmplă la încercarea de a accesa elementul tablou[-5].

Exemplu: Afișarea unui vector cu n elemente citite de la tastatură

```
#include<stdio.h>

int main(void)
{
	int n, v[10], i;
	printf("Dati numarul de elemente: ");
	scanf("%d", &n);
	//Citirea elementelor vectorului
	for (i = 0; i < n; i++)
	{
		printf("v[%d]=", i);
		scanf("%d", &v[i]);
	}
	//Afisarea elementelor vectorului
	for (i = 0; i < n; i++)
		printf("\nv[%d]=%d", i, v[i]);

	return 0;
}
```

Introducere în Programarea calculatoarelor
Laborator 5

Tablouri multidimensionale

Limbajul C oferă posibilitatea folosirii tablourilor multidimensionale. Din punct de vedere semantic, tablourile multidimensionale sunt "tablouri de tablouri". Un tablou multidimensional se declară în modul următor:

```
nume_tip nume_tablou[dim_1][dim_2]…[dim_n]
```

Exemplu de declarare de tablou multidimensional (bidimensional, în cazul de faţă):

```
int matrice[3][3]     // declararea unui tablou 3 x 3
```

Valorile elementelor unui tablou multidimensional se pot declara atunci când se declară tabloul respectiv:

```
int matrice[3]3] = {{1, 3, 5},{2, 4, 6},{3, 6, 9} };
```

Se poate folosi şi următoarea reprezentare/declarare, pentru uşurarea vizualizării conţinutului tabloului:

```
int matrice[3][3] = {{1, 3, 5},{2, 4, 6}, {3, 6, 9}}
```

Exemplu: Afișarea unei matrici cu n linii și m coloane cu elementele citite de la tastatură

```
#include<stdio.h>

int main()
{
	int a[10][10], i, j, n, m;

	printf("Dati numarul de linii: ");
	scanf("%d", &n);
	printf("Dati numarul de coloane: ");
	scanf("%d", &m);
	//Citirea de la tastatura a elementelor matricii cu n linii si m coloane
	for (i = 0; i < n; i++)
		for (j = 0; j < m; j++)
		{
			printf("a[%d][%d]=", i, j);
			scanf("%d", &a[i][j]);
		}

	//Afisarea elementelor matricii
	for (i = 0; i < n; i++)
	{
		printf("\n"); //trece la rand nou, pentru a afisarea in forma de matrice
		for (j = 0; j < m; j++)
```

```
                printf("%d ", a[i][j]);
        }

        return 0;
}
```

Constante de tip tablou

Constantele de tip tablou sunt constante în care se memorează valorile dintr-un tablou de elemente. Forma generală a declaraţiei unui tablou de constante este:

```
const [nume_tip] nume_tablou[dim_1][dim_n] = { val_1, val_2,…,val_n
};
```

Exemplu de constantă de tip tablou unidimensional:

```
  const int c[3] = { 34, 42, 22 };
```

Exemplu de constantă de tip tablou multidimensional:

```
  const int d[2][3] = { {34, 42, 12}, {22, 35, 53} };
```

Exemplu: Următorul program citeşte de la tastatură elementele unui tablou de numere întregi si determină maximul dintre ele:

```
#include <stdio.h>
#define MAX 100
 int main
()
{
      int tab [MAX];
      int N, i, max;

      printf ("Câte numere doriţi? ");
      scanf ("%d", &N);

      for (i = 0; i < N; i++)
      {
            printf ("Introduceţi elementul %d: ", i);
            scanf ("%d", &tab[i]);
      }

      max = tab [0];

      for (i = 1; i < N; i++)
      {
            if (tab [i] > max)
                  max = tab[i];
      }

      printf ("Maximul este: %d \n", max);
      return 0;
}
```

Pointeri

Dupa cum s-a văzut, limbajul C este un limbaj de nivel mediu - conţine atât elemente de nivel înalt (funcţii, instrucţiuni decizionale şi repetitive şamd), cât şi elemente de nivel scazut, apropiat de nivelul hardware. Una dintre aceste caracteristici de nivel scăzut o reprezintă posibilitatea de a accesa orice adresă din memorie, prin intermediul pointerilor.

Un pointer este o variabilă care conţine adresa unei alte variabile. Pointerii sunt foarte mult utilizaţi în C, parte pentru că uneori sunt singura cale de rezolvare a unei anumite probleme, parte pentru ca folosirea lor duce la alcătuirea unui cod mai compact şi mai eficient decât altul obţinut în alt mod.

Un pointer este o variabilă care conţine (memorează) o adresă din memorie, de obicei adresa unei alte variabile.
Dacă o variabilă conţine adresa alteia, se spune că prima este ”un pointer la cea de-a doua” (sau că indică spre cea de-a doua).

Declararea unei variabile de tip pointer are forma generală:

```
      tip *nume;
```

Tipul de bază defineşte tipul de variabilă către care indică pointer-ul.

Exemplu de declaraţie de pointer:

```
int *a;
float *pf;
```

Prima declaraţie de mai sus se citeşte "*a este un pointer către o valoare de tip întreg*".

Operatorii folosiţi în lucrul cu pointeri

Operatorii folosiţi în lucrul cu pointeri sunt & şi *. *Operatorul &* este folosit pentru returnarea adresei operandului său. Din moment ce un pointer susţine adresa unui obiect, este posibilă adresarea acelui obiect "indirect" prin intermediul pointerului. Să presupunem ca **x** este o variabilă, de tip *int* şi că *px* este un pointer creat într-un mod neprecizat. Operatorul **&** dă adresa unui obiect, astfel încât instrucţiunea

```
px = &x
```

asignează variabilei px adresa lui x, px înseamnă "pointează pe x". Operatorul & poate fi aplicat numai variabilelor şi elementelor unui tablou, construcţii ca *&(x+1) si &3 sunt interzise*. Este deasemenea interzisă păstrarea adresei unei variabile registru.

```
int a, *b;
a = 23; b
= &a;
```

Adresa:	**0**	**1**	**2**
Continut:		23 (valoarea lui a)	1 (adresa lui a)
		a	b

Variabila a va avea valoarea "adresa variabilei b". Această adresă NU are nici o legătură cu valoarea variabilei a. Instrucţiunea anterioară de atribuire se citeşte "*b primeşte adresa lui a*".

Atenţie! Operatorul & poate fi aplicat doar variabilelor sau elementelor unui tablou, construcţii ca &(x+1) şi &3 fiind interzise.

Operatorul * este complementarul operatorului &. Acest operator este folosit pentru a returna conţinutul adresei de memorie către care indică pointerul care îi urmează. De exemplu, prin instrucţiunea:

```
int a, *c;
a = *c;
```

variabilei a i se atribuie valoarea de la adresa de memorie către care indică c. Această instrucţiune se citeşte *"a primeşte valoarea din locaţia de memorie către care indică c"*.

Atenţie! uneori se poate face confuzie între operatorul * şi operatorul pentru înmulţire (tot *), precum şi între operatorul & şi operatorul la nivel de bit AND (notat tot &). Aceşti operatori nu au nici o legătură între ei, atât & cât şi * având prioritate faţă de toţi operatorii aritmetici.

Pentru a înţelege mai bine complementarea celor doi operatori, considerăm secvenţa:

```
int a, b, *p;
p = &b; a =
*p;
```

Această secvenţă este echivalentă cu atribuirea:

```
a = b;
```

Atenţie! Este sarcina programatorului să se asigure că la adresa indicată de un pointer se află o variabilă de tipul dorit, atunci când se accesează valoarea respectivei locaţii. De exemplu, atunci când se declară un pointer ca fiind de tip int, compilatorul se aşteaptă ca la adresa pe care o conţine pointerul respectiv se află o variabilă de tip întreg, chiar dacă acest lucru nu este adevărat.

Exemplu de program:

```
int main ()
{
      float x, y;
      int *p;

        p = &x;  // atribuire incorectă – x este float, p este int

        y = *p;  //  nu va funcţiona aşa cum se doreşte

      return 0;
}
```

Afişarea adresei unei variabile se poate face folosind şirul de formatare *%p*:

```
int main ()
{
      int x, *p;

      scanf (”%d”, &x);
      p = &x;

      printf (“Adresa variabilei x este %p”, p);
      return 0;
}
```

Erori întâlnite frecvent în folosirea pointer-ilor

Există câteva erori frecvent întâlnite în folosirea pointer-ilor, care pot duce la comportări nedorite a programelor. Principala problemă întâlnită în folosirea pointer-ilor este *folosirea pointerilor neinițializați*. Considerăm următorul exemplu:

```
int  main ()
{
      int x, *p;

      *p = 23;
      return 0;
}
```

În exemplul de mai sus se atribuie valoarea 23 unei locații de memorie necunoscute.

Presupunerea incorectă asupra amplasării variabilelor în memorie este o altă eroare destul de întâlnită în lucrul cu pointeri. În general nu se poate cunoaşte cu exactitate amplasarea datelor în memoria calculatorului sau dacă vor fi amplasate în aceeaşi ordine sau locaţie după fiecare executare a unui program.

Exemplu:

```
#include <stdio.h>
int main ()
{
      int x, y;
      int *a, *b;

      a = &x;
      b = &y;

      //compararea adreselor variabilelor x si y, nu a valorilor
      if (a < b)
      {
            ...          // nu va funcţiona întotdeauna corect,
      }
      return 0;
}
```

Introducere în Programarea calculatoarelor
Laborator 5

Alocarea dinamică de memorie

În practică pot exista multe situaţii în care nu se poate determina cantitatea de memorie necesară unui program în momentul compilării acestuia, ci doar în momentul execuţiei (ex. numărul de elemente ale unui tablou care reprezintă o bază de date). În aceste situaţii, se poate apela la mecanismele alocării dinamice de memorie, în timpul execuţiei programelor, doar atunci când este necesar şi când se cunoaşte dimensiunea zonei de memorie dorită.

Operator sizeof ()

Funcţiile pentru alocarea dinamică de memorie necesită specificarea dimensii în octeţi a zonei care se doreşte alocată,. Pentru a simplifica utilizarea acestora, limbajul C pune la dispoziţia programatorilor operatorul *sizeof*, care determină dimensiunea *în octeţi* a unei variabile sau tip de date.

Exemplu de folosire al operatorului *sizeof ():*

```
int dim; float
x;

dim = sizeof (x);
      /* echivalent cu */
dim = sizeof (float);
```

Dimensiunile în octeţi ale principalelor tipuri de date sunt:

Nr. crt	Tip de date	Dimensiunea în octeţi
1	char	1
2	int	2
3	long	4
4	float	4
5	double	8

Spre exemplu, declaraţia:

```
double a;
```

determină alocarea unei zone de memorie de 8 octeţi, pentru memorarea valorii variabilei a.

Funcţii folosite în alocarea dinamică de memorie

Cele mai importante funcţii folosite pentru alocarea dinamică de memorie sunt *malloc ()* şi *free ()*. Pentru folosirea acestor funcţii e nevoie de includerea fişierului header *stdlib.h*

Funcţia *malloc ()* este folosită pentru alocarea dinamică a memoriei. Prototipul funcţiei este:

```
      void *malloc (size_t nr_oct)
```

size_t este un tip de date folosit pentru exprimarea dimensiunilor zonelor de memorie, echivalent cu un *unsigned int.*

Funcţia malloc () returnează un pointer (de tip void) către primul octet al regiunii de memorie alocate. Dacă nu există suficientă memorie disponibilă pentru a putea fi alocată, funcţia malloc () va returna valoarea 0 (sau NULL – macrodefiniţie echivalentă cu un pointer care indică spre adresa zero, rezervată de obicei sistemului de operare). **Exemplu** de alocare de memorie:

```
int *ex;
ex = malloc (1024);  /* aloca 1024 de octeti */
```

Funcţia *free ()* este complementară funcţiei malloc (), rolul ei fiind de a elibera memoria care a fost alocată (prin folosirea funcţiei malloc ()).

Prototipul funcţiei free () este:

```
        void free (void *p);
```

Pointerul p indică spre o zonă de memorie care a fost alocată anterior folosind funcţia malloc (). Funcţia *free ()* nu trebuie apelată cu un argument care nu a fost returnat de funcţia *malloc (),* deoarece această operaţie poate duce la distrugerea listei de memorie disponibilă. **Exemplu** de folosire a funcţiilor *malloc ()* şi *free ()*:

```
int main ()
{
      int *p, i;
      p = malloc (10 * sizeof (int));
      free (p);
      return 0;
}
```

Relaţia dintre tablouri şi pointeri. Aritmetica pointerilor

În limbajul C există o legătură foarte strânsă între tablouri şi pointeri. Datorită acestei relaţii se pot scrie programe mult mai eficiente şi se pot accesa şi folosi mult mai eficient tablourile.

- Numele unei variabile de tip tablou reprezintă un pointer către primul element al tabloului:

```
int a[10]; int
*p;

*a = 123; /* echivalent cu:  a [0] = 123; */ p =
a;    /* atribuirea este corecta si permisa */
```

- Orice zonă de memorie adresată printr-un pointer permite utilizarea indexării pentru accesarea elementelor:

```
int *b; b = malloc (10 *
sizeof (int));
/* accesarea celui de-al treilea element */ b
[3] = 23;
```

În limbajul C există două operaţii aritmetice care se pot efectua cu pointeri: adunare şi scădere.

Prin incrementarea unui pointer, se avansează pointerul spre o adresa superioară în memorie, avansându-se cu dimensiunea în octeţi a tipului de baza al pointerului respectiv. Pentru decrementare se procedează în mod analog.

```
int *p; int
tab [10];

p = tab;

printf ("Primul element este la adresa %p \n", p);
 p = p + 1; /* SAU p++
*/

printf ("Urmatorul element este la adresa %p \n ", p);
```

Se observă că cele două adrese afişate diferă prin doi octeţi, dimensiunea tipului int.

Exemplu: Aritmetica pointerilor se poate utiliza pentru a creşte viteza de execuţie a unui program:

```
int tab [10], i;

for (i = 0; i < 10; i++)          tab[i] = 0;
```

La fiecare accesare a unui element din tablou, procesorul face următoarele operaţii:

```
*(tab + i * sizeof (int)) = 0;
```

Înmulţirea care apare în paranteze este o operaţie costisitoare ca timp de execuţie, astfel încât dacă se accesează succesiv elementele unui tablou, e mai rapidă metoda:

```
int tab [10], *p, *sfarsit;

sfarsit = &tab[9];

for (p = tab; p <= sfarsit; p++)
     *p = 0;
```

Câştigul de viteză provine din faptul că incrementarea unei valori (adresa indicată de p) este mult mai rapidă decât o adunare şi o înmulţire.

Compararea pointerilor

Folosind o expresie relaţională se pot compara doi pointeri, pentru a se verifica adresa de memorie indicată de cei doi pointeri:

Exemplu:

```
int *a;
int *b;
if (a < b)
 printf ("a indică spre o adresă mai mică decat b");
```

Exemplu: Programul următor utilizează alocarea dinamică de memorie şi aritmetica pointerilor pentru a determina suma unui şir de numere dat de la tastatură:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
	int *tab, *crt, *sfarsit;
	int N, suma;

	printf("Cate numere doriti ? ");
	scanf("%d", &N);

	tab = (int *)malloc(N * sizeof (int));
	sfarsit = tab + N;
	printf("Introduceti numerele: \n");

	for (crt = tab; crt < sfarsit; crt++)
		scanf("%d", crt);
	suma = 0;
	for (crt = tab; crt < sfarsit; crt++)
		suma = suma + *crt;

	printf("Suma este: %d \n", suma);

	free(tab);
	getch(); return 0;
}
```

Exemplu: Programul următor citeşte un şir de numere de la tastatură şi le afişează ordonate crescător, utilizând algoritmul *bubblesort*:

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

int main()
{
	int tab[MAX];
	int k, N, aux, ordonat;

	printf("Cate numere doriti? ");
	scanf("%d", &N);

	printf("Introduceti numerele: \n");

	for (k = 0; k < N; k++)
		scanf("%d", &tab[k]);

	do
	{
		ordonat = 1;

		for (k = 1; k < N; k++)
		{
			if (tab[k - 1] > tab[k])
			{
				aux = tab[k];
				tab[k] = tab[k - 1];
				tab[k - 1] = aux;
				ordonat = 0;
			}
		}
	} while (!ordonat);

	printf("Sirul ordonat crescator est: \n");

	for (k = 0; k < N; k++)
		printf("%d \n", tab[k]);

	return 0;
}
```

6.4 Probleme propuse

1. Să se scrie un program în C care să citească de la tastatură două matrice pătratice de numere întregi, de dimensiune specificată de utilizator şi să afișeze suma celor două matrice.
2. Să se scrie un program în C care, folosind un meniu interactive să conțină următoarele opțiuni:
    1. Citirea unui vector cu n elemente
    2. Afișarea vectorului
    3. Afișarea elementelor de pe poziițile pare
    4. Afișarea produsului elementelor impare
    5. Ieșire
3. Să se calculeze suma elementelor de pe diagonala principală a unei matrice pătratice citite de la tastatură.
4. Să se calculeze suma elementelor dintr-un vector aflate pe poziţii impare.
5. Să se scrie un program în C care să citească de la tastatură o matrice pătratică. Să se creeze un meniu interactive cu următoarele opțiuni:
    1. Afișarea matricii pătratice
    2. Să se afișeze suma numerelor pare deasupra diagonalei principale
    3. Să se afișeze produsul elementelor impare de pe diagonal secundară
    4. Să se afișeze elementele prime din matrice
    5. Iesire