Laborator 5 – Programarea Calculatoarelor Pointeri și Funcții

Un pointer este o variabilă care conţine (memorează) o adresă din memorie, de obicei adresa unei alte variabile.

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;
```

Operatorii folosiţi în lucrul cu pointeri

Operatorii folosiţi în lucrul cu pointeri sunt *&* şi ***. *Operatorul &* este folosit pentru returnarea adresei operandului său. 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
```

*asigneaz*ă *variabilei px adresa lui x,* px înseamnă "pointează pe x".

```
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"*.

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 pointerilor

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 neiniţializaţi*. Considerăm următorul exemplu:

```
int  main ()
{
  int x, *p;
  *p = 23;
  return 0;
}
```

În exemplul de mai sus se atribuie valoarea 23 unei locaţii 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;
}
```

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	4 (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.*

Exemplu de alocare de memorie:

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

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;
}
```

Funcții în limbajul C

Forma generală a unei funcţii este următoarea:

```
tip_returnat nume_functie (tip_date1 arg1, tip_date2 arg2, ..., tip_dateN argN );
        {
                .       .       .
                    corpul_functiei;
                .       .       .
        }
```

Exemplu de definire de funcţie care returnează o valoare de tip întreg:

```
int calcul (int a, int b)
{
          int c;

           c = a + b;
          return c;                    // valoarea returnată de funcţie
}
```

Exemplu de funcţie care nu primeşte nici un (argument) şi nu returnează nimic:

```
void afisare (void) // sau void afisare ()
       {
       printf ("Functie care nu returnează nici o valoare..."; }
```

Apelarea unei funcţii constă în folosirea propriu-zisă a funcţiei, într-o altă funcţie. Apelarea se poate face atât în funcţia principală (main), cât şi într-o altă funcţie.

Returnarea unei valori de către o funcţie se face folosind instrucţiunea *return* înainte de încheierea funcţiei respective**.** De asemenea, instrucţiunea *return* se poate folosi şi pentru încheierea forţată a execuţiei unei funcţii.

O funcţie poate returna orice valoare dintr-un tip de bază, cu excepţia unui tablou. Dacă funcţia nu returnează nici o valoare, atunci se consideră că returnează *void* (tipul vid).

Exemplu: definirea şi apelarea unei funcţii care returnează o valoare întreagă:

```
int calcul (int a, int b)
{
  int c;
  c = a + b;
  return c;
}
int main ()
```

```
{
  int x, y, z;
  x = calcul (y, z); // apelarea funcţiei
  printf ("Rezultatul este %d", x);
  return 0;
}
```

Exemplu: definirea şi apelarea unei funcţii care nu retunează nici o valoare:

```
void afisare (void)
{
  printf ("Functie care nu returnează nici o valoare...");
}
int main(void)
{
  int x, y, z;
  afisare ();      // apelarea funcţiei
  return 0;
}
```

Transmiterea parametrilor prin valoare, transmiterea prin adresă

În general, este necesară transmiterea valorilor calculate într-o funcţie, pentru a putea fi folosite într-o altă funcţie. Transmiterea valorilor se poate face în două moduri:

- prin valoare;
- prin adresă;

Transmiterea parametrilor prin valoare

Transmiterea parametrilor prin valoare constă în copierea valorii unui argument într-un parametru formal al unei funcţii. În cazul transmiterii prin valoare, modificările efectuate asupra parametrului formal NU au efect asupra argumentului funcţiei.

Transmiterea parametrilor prin valoare este cea mai folosită metodă de transfer din limbajul C.

Exemplu:

```
float medie (int a, int b)
{
    float c;
    c = (a + b) / 2;
    return c;
}
int main(void)
{
    int y, z;
    float x;
     x = medie (y, z);
    printf ("Rezultatul este %f", x);
    return 0;
}
```

Transmiterea matricelor ca argumente pentru o funcţie reprezintă o excepţie de la regula de transmitere prin valoare a parametrilor.

Dacă se foloseşte o matrice ca şi argument al unei funcţii, atunci funcţiei respective i se transmite adresa matricei respective (adresa primului element al matricei), iar funcţia respectivă poate acţiona asupra conţinutului matricei respective.

Exemplu de transformare a unui şir de caractere în majuscule:

```
void majuscule (char *sir)
{
  int i;
  for (i = 0; sir[i]; i++)
  {
    sir[i] = toupper (sir[i]);
    printf ("%c", sir[i]);
  }
}
int main(void)
{
 char sir1[50];
 gets(sir1);
 majuscule (sir1);    // apelarea funcţiei
 return 0;
}
```

Funcții - Transmiterea parametrilor prin adresă

În limbajul C **e**xistă şi instrumente prin care se poate simula un transfer prin referinţă. Pentru a transfera un obiect V ca parametru prin referinţă, se transferă un pointer către V; parametrul formal va fi un *pointer* către valoarea obiectului V.

Transmiterea parametrilor prin referinţă constă în copierea adresei unui argument într-un parametru. În funcţia în care se face apelarea, este folosită adresa respectivă pentru accesarea argumentului folosit efectiv la apelare.

În transmiterea prin adresă, modificările efectuate asupra unui parametru formal afectează argumentul cu care se face apelarea funcţiei (parametrul actual). Pentru a folosi transmiterea prin referinţă, argumentele funcţiei trebuie declarate ca pointeri.

Exemplu – interschimbarea a două numere:

```
void interschimba (int *a, int *b) //transmit prin adresă
{
  int c;
  c = *a;
  *a = *b;
  *b = c;
}
int main (void)
{
  int *x, *y;  // parametrii locali - pointeri
  int z, w;    // parametrii locali
  x = &z;      // z şi w sunt folosite pentru
   y = &w; // iniţializarea celor doi pointeri
   printf ("Introd. valoarea primului numar:
   "); scanf ("%d", x);
   printf ("Introd. valoarea pentru al doilea numar:
   "); scanf ("%d", y);
   interschimba (x, y); // apelarea funcţiei
   interschimba (&z, &w); // apel recomandat al funcţiei
   printf ("Nr. interschimbate sunt %d si %d", *x, *y);
   getch ();
   return 0;
}
```

Probleme rezolvate

1. modificarea unui număr întreg, transmis prin adresă, într-o funcţie:

```
#include <stdio.h>

void incrementare_număr (int *nr)
{
 (*nr)++;
}
int main (void)
{ int n=0;
  incrementare_număr (&n);
  printf ("Numarul este: %d", n);
 return 0;
}
```

2 - definirea şi folosirea unei funcţii care citeşte un şir de caractere şi îl returnează:

```
#include <stdio.h>

void citire_sir (char *sir)
{
 gets (sir);
}
int main(void)
{
  char sir1[50];
  citire_sir (sir1);
  printf ("Sirul introdus este: %s", sir1);
  return 0;
}
```

3. Să se citească un șir de la tastatură și apoi să se afișeze pe ecran caracter cu caracter. Elementele șirului vor fi accesate prin indecși și prin aritmetica pointerilor.

```
#include<stdio.h>
#include<string.>
#include<conio.h>

void afiseaza_prin_indici(char*sir)
{
        int k;
        printf("\n sirul afisat utilizandu-se tablou ci indice:");
          for (k = 0; sir[k]; ++k)
                putchar(sir[k]);

}

void afiseaza_prin_pointer(char *s){

        printf("\n sirul afisat utilizandu-se pointeri:");
        while (*s)
                putchar(*s++);

}

int main()
{
        char sirdat[20];
        printf("\n dati un sir de max 20 caractere:");
        gets(sirdat);

        /* apelarea celor doua functii pentru afisarea folosind indici si pointeri */

        afiseaza_prin_indici(sirdat);
        afiseaza_prin_pointer(sirdat);
        getch();
        return 0;
}
```

4. Să se realizeze un program care citește și afișează un vector cu ajutorul pointerilor.

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

int citire(int tab[])
{
	//citeste elementele lui tab - accesarea fiecarui element se
	//face printr-un pointer la el
	int *pi, i, N;
	printf("Introduceti nr de elemente: ");
	scanf("%d", &N);
	pi = tab;
	printf("Introduceti elementele tabloului:\n");
	for (i = 0; i<N; i++)
	{
		scanf("%d", pi);
		pi++;
	}
	return pi - tab;
}

void tiparire(int tab[], int n){
	// tipareste elementele lui tab prin accesare prîn
	pointeri int *pi;
	printf("Elementele tabloului:\n");
	for (pi = tab; pi<tab + n; pi++)
		printf("%d ", *pi);
}

int main()
{
	int tab[20], n;
	system("cls");
	n = citire(tab);
	tiparire(tab, n);
	getch();
	return 0;
}
```

Probleme propuse

1. Să se scrie un program în C care, folosind funcții și un meniu interactiv să conțină următoarele opțiuni:
   a) Citirea unui vector cu n elemente
   b) Afișarea vectorului
   c) Afișarea elementelor divizibile cu 5
   d) Afișare sumei elementelor de pe poziții impare
   e) Afișarea produsului elementelor impare
   f) Ieșire

2. Să se calculeze suma elementelor de pe diagonala secundare a unei matrice pătratice citite de la tastatură folosind funcții.

3. Să se scrie un program în C folosind funcții care să citească de la tastatură o matrice pătratică. Să se creeze un meniu interactive cu următoarele opțiuni:
   a) Afișarea matricii pătratice
   b) Să se afișeze suma numerelor pare deasupra diagonalei secundare
   c) Să se afișeze suma elementelor pare de sub diagonal principală
   d) Să se afișeze elementele prime din matrice
   e) Iesire

4. Să se scrie un program în C care, folosind funcții și un meniu interactiv să conțină următoarele opțiuni:

   a. Şterge dintr-un şir de caractere un subşir specificat prin poziţie şi lungime.
   b. Inserează într-un şir începând cu o poziţie dată un al şir.
   c. Citeşte doua cuvinte şi înlocuieşte într-un text introdus de la tastatură toate apariţiile primului cuvânt prin cel de-al doilea.

5. Să se realizeze un program care să determine dacă un număr citit de la tastatură este prim, utilizând o funcție definită de utilizator.