POINTERI

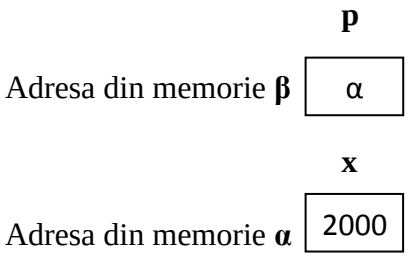
1. Conţinutul lucrării

În lucrare se prezintă tipul pointer, operaţiile permise asupra pointerilor, modul de alocare şi eliberare dinamică a memoriei.

2. Consideraţii teoretice

2.1. Tipul pointer

Un pointer este o variabilă care are ca valori adrese. Dacă pointerul **p** are ca valoare adresa de memorie a variabilei **x**, se spune că p pointează spre **x.**



Un pointer este legat de un tip. Dacă **x** este de tipul **int**, pointerul **p** este legat de tipul **int**.

Declararea unui pointer se face la fel ca declararea unei variabile, cu deosebirea că numele pointerului este precedat de caracterul ***** :

tip *nume;

Exemplu:

int *p;

Adresa unei variabile se obţine cu ajutorul operatorului unar **&**, numit **operator de referenţiere.**

Exemplu: Fie declaraţiile:

int x;
int *p;

atunci **p=&x**; are ca efect atribuirea ca valoare pentru **p** a adresei variabilei **x**. În desenul de mai sus, variabila **x** fiind localizată la adresa **α**, valoarea lui **p** va fi **α**.

Furnizarea valorii din zona de memorie a cărei adresă este conţinută în **p** se face cu ajutorul operatorului unar *****, numit **operator de dereferenţiere**.

Exemplu:

a) instrucţiunea **x=y** este echivalentă cu una din secvenţele:

p=&x;	sau	**p=&y;**
***p=y;**		**x=*p;**

b) instrucţiunea **x++** este echivalentă cu secvenţa:

```
p=&x;
(*p)++;
```

În aceste exemple **p** trebuia să fie legat de tipul lui **x** sau **y**. De exemplu:

```
int x, y;
int *p;
```

Există cazuri când un pointer trebuie să nu fie legat de un tip de date. În acest caz, se foloseşte declaraţia următoare:

```
void *nume;
```

În acest caz, dacă avem declaraţiile:

```
int x;
float y;
void *p;
```

atunci este corectă oricare din instrucţiunile:

```
p=&x;
p=&y;
```

însă este necesară folosirea expresiilor de schimbare explicită de tip (*cast*), pentru a preciza tipul datei spre care pointează **p**:

```
(tip *)p
```

Observaţie: este necesară cunoaşterea în fiecare moment a tipului valorii ce se găseşte la adresa atribuită pointerului de tip **void ***. Neţinând seama de acest lucru se ajunge la erori.

Exemplu de utilizare a unui pointer de tip **void**:

```
int x;
void *p;
```

Instrucţiunea **x=10** este echivalentă cu secvenţa :

```
p=&x;
*(int *)p=10;
```

În esenţă, dacă **p** este pointer declarat ca **void *p**, nu poate fi folosită dereferenţierea ***p** fără a preciza tipul datei referite printr-o expresie de tipul *cast*.

2.2. Legătura dintre pointeri şi tablouri

Numele unui tablou are drept valoare adresa primului său element. Ca urmare, se spune că numele unui tablou este un pointer constant, neputând fi modificat în timpul execuţiei.

Exemplu:

```
int tab[100];
int *p;
int x;
...
p=tab;  /* p primeşte ca valoare adresa elementului tab[0] */
…
```

În acest exemplu, atribuirea **x=tab[0]** este echivalentă cu **x=*p**;

Ca urmare a celor prezentate, rezultă că dacă un parametru efectiv este un tablou unidimensional, atunci parametrul formal corespunzător poate fi declarat în două moduri:

a) ca tablou: **tip nume_parametru_formal[]**;
b) ca pointer: **tip *nume_parametru_formal;**

Cele două declaraţii ale parametrului formal sunt echivalente, fiind corectă utilizarea în corpul funcţiei a construcţiei de variabilă indexată:

nume_parametru_formal [indice];

Acest lucru este ilustrat în exemplul de mai jos, care are drept scop găsirea maximului şi minimului dintre elementele unui şir.

```
/* Programul L7Ex1 */

/* Programul exemplifica transmiterea parametrului formal tablou prin pointer */
 #include <stdio.h>

 void max_min1(int n,int a[],int *max,int* min)
 {
        int i;
        *max=a[0];
        *min=a[0];
        for (i=1;i<n;i++)
          {
                if (a[i]>*max) *max=a[i];
                else if (a[i]< *min) *min=a[i];
          }
  }

  void max_min2(int n,int *a,int *max,int *min)
  {
        int i;
        *max=a[0];
        *min=a[0];
        for (i=1;i<n;i++)
          {
```

```
            if (a[i]>*max) *max=a[i];
            else if (a[i]< *min) *min=a[i];
        }
}

int main()
{
        int i,n,maxim,minim;
        int x[100];
        /* Introducerea datelor */
        printf("\nNumarul elementelor tabloului n=");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                {
                 printf("\nx[%d]=",i);
                 scanf("%d",&x[i]);
                };
         /* Apelul primei functii */
         max_min1(n,x,&maxim,&minim);
         printf("\nLa apelul functiei max_min1 rezulta:\
                  maximul=%d minimul=%d\n",maxim,minim);
         /* Apelul celei de-a doua functii */
         max_min2(n,x,&maxim,&minim);
         printf("\nLa apelul functiei max_min2 rezulta:\
                  maximul=%d  minimul=%d\n",maxim,minim);
         return 0;
}
```

2.3. Operaţii asupra pointerilor

Asupra pointerilor sunt permise următoarele operaţii:

a) Incrementare/decrementare cu 1. În acest caz valoarea pointerului este incrementată/decrementată cu numărul de octeţi necesari pentru a păstra o dată de tipul de care este legat pointerul.
Operatorii folosiţi sunt ++ şi -- .

De exemplu:

```
int tab[100];
int *p;
……..
p=&tab[10];
p++;   /* Valoarea lui p este incrementată cu 4 octeți,
          având adresa elementului  tab[11] */
```

b) Adunarea şi scăderea unui întreg dintr-un pointer.

Operaţia **p±n** are drept efect creşterea, respectiv scăderea din valoarea **p** a **n*numărul de octeţi** necesari pentru a păstra o dată de tipul de care este legat pointerul.

Pentru exemplul de mai sus, dacă **x** este de tipul **int**, atunci:

x=tab[i];

este echivalentă cu:

x=*(tab+i);

c) Diferenţa a doi pointeri.

Dacă doi pointeri **p** şi **q** pointează spre elementele **i** şi **j** ale aceluiaşi tablou (**j**>**i**), adică **p=&tab[i]** şi **q=&tab[j],** atunci **q-p = j–i** .

d) Compararea a doi pointeri.

Doi pointeri care pointează spre elementele aceluiaşi tablou pot fi comparaţi folosind operatorii de relaţie şi de egalitate:

< <= > >= == !=

Mai jos este prezentat programul de la paragraful precedent, folosind operaţii asupra pointerilor.

```
/* Programul L7Ex2 */

/* Programul exemplifica folosirea operatiilor asupra pointerilor */

 #include <stdio.h>

 void max_min1(int n,int a[],int *max,int* min)
{
        int i;
        *max=a[0];
        *min=a[0];
        for (i=1;i<n;i++)
          {
                if (a[i]>*max) *max=a[i];
                else if (a[i]< *min) *min=a[i];
          }
 }

 void max_min2(int n,int *a,int *max,int *min)
 {
        int i;
        *max=*a;
        *min=*a;
        for (i=1;i<n;i++)
          {
                if (*(a+i)>*max) *max=*(a+i);
                else if (*(a+i)< *min) *min=*(a+i);
          }
 }
```

```
int main()
{
        int i,n,maxim,minim;
        int x[100];
        /* Introducerea datelor */
        printf("\nNumarul elementelor tabloului n=");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                {
                  printf("\nx[%d]=",i);
                  scanf("%d",&x[i]);
                };
         /* Apelul primei functii */
         max_min1(n,x,&maxim,&minim);
         printf("\nLa apelul functiei max_min1 rezulta:\
                        maximul=%d  minimul=%d\n",maxim,minim);
         /* Apelul celei de-a doua functii */
         max_min2(n,x,&maxim,&minim);
         printf("\nLa apelul functiei max_min2 rezulta:\
                        maximul=%d   minimul=%d\n",maxim,minim);
         return 0;
        }
```

2.4. Alocarea/eliberarea dinamică a memoriei *heap*

Alocarea memoriei pentru variabilele globale şi statice este statică, adică alocarea rămâne până la terminarea programului.

Alocarea memoriei pentru variabilele automatice este dinamică, în sensul că stiva este "curăţată" la terminarea funcţiei.

Memoria *heap* este o zonă de memorie dinamică, specială, distinctă de stivă. Ea poate fi gestionată prin funcţii, care au prototipurile în fişierul **stdlib.h** .

Alocarea unei zone de memorie *heap* se poate realiza cu ajutorul funcţiilor de prototip:

```
void *malloc (unsigned n);
void *calloc (unsigned nr_elem, unsigned dim);
```

Funcţia **malloc** alocă în *heap* o zonă contiguă de **n octeţi**. Funcţia **calloc** alocă în *heap* o zonă contiguă de **nr_elem * dim** octeţi, inițializați cu valoarea zero.

Funcţiile returnează:

- în caz de succes, adresa de început a zonei alocate (pointerul fiind de tip void, este necesară conversia spre tipul dorit);
- în caz de insucces, returnează zero (pointerul NULL).

Eliberarea unei zone alocate cu **malloc** sau **calloc** se face cu ajutorul funcţiei de prototip:

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

Mai jos se prezintă un exemplu de utilizare a funcţiilor prezentate în acest paragraf.

```
/*  Programul L7Ex3  */

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

int main()
{
  char *str1,*str2;

  /* Aloca memorie pentru primul sir de caractere */
  if ((str1 = (char *) malloc(100)) == NULL)
  {
    printf("Memorie insuficienta\n");
    exit(1);
  }
  printf("\nIntroduceti primul sir de caractere terminat cu ENTER\n");
  gets(str1);
  printf("\nSirul de caractere introdus este\n%s\n",str1);

  /* Aloca memorie pentru al doilea sir de caractere */
  if ((str2 = (char *) calloc(100,sizeof(char))) == NULL)
  {
    printf("Memorie insuficienta\n");
    exit(2);
  }
  printf("\nIntroduceti al doilea sir de caractere terminat cu ENTER\n");
  gets(str2);
  printf("\nSirul de caractere introdus este\n%s\n",str2);
  printf("\nUrmeaza eliberarea memoriei. Apasati o tasta\n");
  getch();
  /* Eliberarea memoriei */
  free(str1);
  free(str2);
  return 0;
}
```

2.5. Funcții cu parametri de tip funcție

Numele unei funcţii este un pointer spre funcţia respectivă. De aceea, numele unei funcţii poate fi folosit ca parametru efectiv la apelul unei funcţii.

Fie **f** o funcţie care va fi transmisă ca parametru efectiv, având antetul:

tipf f(lista_parametrilor_formali_f);

În acest caz, parametrul formal al unei funcţii **g** care va fi apelată cu parametrul efectiv **f,** este prezentat în antetul funcţiei **g**:

tipg g(…, tipf(*p)(lista_parametrilor_formali_f), …)

Apelul se va face astfel:
g(…, f, …);

Programul următor prezintă un exemplu în acest sens. Este vorba de integrarea unei funcţii prin metoda trapezului.

$$\int_a^b f(x)dx = h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(a+i*h)\right)$$

unde, **n** este numărul de subintervale în care s-a împărţit intervalul [**a**, **b**], dimensiunea unui subinterval fiind **h**.

```
/* Programul L7Ex4 */

/* Programul exemplifica modul de folosire a unei functii ca parametru */
#include <stdio.h>
#include <conio.h>
#include <math.h>

double f(double x) {
    return (3*x*x +1);
}

/* Calculul integralei prin metoda trapezelor */
double integrala(double a,double b,int n,double(*p)(double x)) {
    int i;
    double h,s;
    h=(b-a)/n;
    s=((*p)(a)+(*p)(b))/2.0;
    for(i=1;i<n;i++)
       s=s+(*p)(a+i*h);
    s=s*h;
    return s;
}

int main()
{
   double a,b;
   int n;
   char ch;
   /* Citirea intervalului de integrare */
   printf("\na=");scanf("%lf",&a);
   printf("\nb=");scanf("%lf",&b);
   ch='D';
```

```
   while (ch=='D' || ch=='d')  {
      printf("\nn="); scanf("%d",&n);
      printf("\nPentru n=%d Valoarea integralei este %f",n, integrala(a,b,n,f));
      printf("\nApasati o tasta\n");getch();
      printf("\nIntroduceti alt n? DA=D/d NU=alt caracter ");
      ch=getch();
   }
   return 0;
}
```

3. Mersul lucrării

3.1. Se vor executa exemplele din lucrare şi din materialul de curs. Se vor analiza următoarele lucruri:

- folosirea unei variabile indexate, atunci când numele tabloului a fost definit ca pointer în antetul funcţiei (L7Ex1);
- operaţiile permise asupra pointerilor (L7Ex2);
- alocarea memoriei în *heap* – spaţiul de memorie pentru variabile dinamice (L7Ex3);
- avantajul transmiterii funcţiilor ca parametru efectiv (L7Ex4).

3.2. Folosind numai pointeri şi expresii cu pointeri să se scrie funcţii de citire, afişare şi înmulţire a două matrice.

3.3. Folosind numai pointeri şi expresii cu pointeri să se scrie funcţii de sortare a unui vector cu elemente reale.

3.4. Folosind numai pointeri şi expresii cu pointeri să se scrie o funcţie de interclasare a doi vectori, care conţin elemente de tip real ordonate crescător.

3.5. Să se scrie o funcţie care sortează în ordine crescătoare n şiruri de caractere.

3.6. Să se scrie o funcţie care determină rădăcina unei funcţii f(x), în intervalul [a, b], ştiind că admite o singură rădăcină în acest interval. Funcţia f va fi transmisă ca parametru efectiv.

3.7. Să se scrie o funcție pentru calculul derivatei unui polinom P(x) de grad n, într-un punct dat $x=x_0$. Gradul și coeficienții polinomului precum și punctul x_0 se for trimite ca și argumente la apelul funcției.

3.8. Să se scrie o funcție pentru calculul produsului a două polinoame. Se vor utiliza numai pointeri și expresii cu pointeri.

3.9. Să se scrie o funcție pentru calculul matricei A^k, unde A este o matrice pătratică. Se vor utiliza pointeri și expresii cu pointeri pentru accesul la elementele matricei. Matricea inițială și cea rezultată vor fi ulterior afișate în format natural (sub formă de linii și coloane).

3.10. Să se scrie o funcție pentru calculul matricei transpuse A^T a unei matrice A. Se vor utiliza pointeri și expresii cu pointeri pentru accesul la elementele matricei. Matricea inițială și cea rezultată vor fi ulterior afișate în format natural (sub formă de linii și coloane).