Laborator 11 – Pointeri

Un pointer este o variabilă care are ca valori adrese. Pointerii se utilizează pentru a face referire la date cunoscute prin adresele lor. Forma generală de declarare a unei variabile pointer este:

tip *nume;

tip reprezintă tipul de bază al pointerului şi poate fi orice tip de variabilă din limbajul C; nume reprezintă numele variabilei pointer.

Observaţii:
Menţionăm că în legătură cu noţiunea de pointer, în limba română se utilizează şi alte denumiri, ca de exemplu:
-referinţă;
-localizator;
-reper;
-indicator de adresă, etc.
În lucrarea de faţă se păstrează denumirea în limba engleză.

Așadar, dacă un pointer se declară prin:

tip*nume;

ceea ce înseamnă că nume este un pointer care pointează spre o zonă de memorie ce conţine o dată de tipul tip, iar prin declaraţia

tip nume;
se declară variabila nume de tipul tip, putem considera că:

tip *
dintr-o declaraţie de pointeri reprezintă

tip

dintr-o declaraţie obişnuită. De aceea, construcţia:

tip*

se spune că reprezintă tipul pointer. Acest tip se spune că este tipul pointer spre tip.

Exemple

int * i_point;

i_point este declarată de tip pointer şi conţine adrese de memorie în care se păstrează date de tip int. *i_point reprezintă conţinutul zonei de memorie spre care pointează i_point, conţinutul având tipul int.

În următorul exemplu,

double *d_point;

este un pointer către un tip double, iar în următorul:

char *sir;

sir este un pointer către un caracter.

Dacă p este o variabilă de tip pointer care are ca valoare adresa lui x, atunci
*p
reprezintă chiar valoarea lui x.
Fie de exemplu:
int x,z;
atunci dacă p are ca valoare adresa lui x, atribuirea:
y=x+100
este identică cu:
y=*p+100

Observaţie:
În cazul în care se doreşte ca un pointer să fie utilizat cu mai multe tipuri de date, la declararea acestuia nu se poate specifica un tip ci se face folosind cuvântul void. Utilizarea acestui tip asigură o flexibilitate mare în folosirea pointerilor.

Exemplu

void *NoType;

Lui NoType i se pot atribui adrese de memorie care pot conţine date de tipuri diferite: int, float, char etc. La utilizarea pointerilor de tip void este necesar să se facă conversii explicite prin expresii de tip cast, pentru a putea preciza tipul datei spre care pointează un astfel de pointer. Astfel vom avea:

(int*) NoType =15;

Valoarea lui NoType a fost convertită spre tipul " pointer spre tipul int". Astfel NoType are ca valoare o adresă a unei zone de memorie în care se păstrează întregi de tip int.

Se recomandă utilizarea cu prudenţă a acestui tip void fiindcă poate să conducă şi la erori. De exemplu dacă se utilizează expresia (int*) NoType într-un moment în care NoType are ca valoare adresa unei zone de memorie care conţine o dată dublă precizie, rezultatul va fi imprevizibil.

Pointerii sunt una dintre cele mai puternice caracteristici ale limbajului C. În limbajul C necesitatea de iniţializare a pointerilor este mult mai pregnantă decât iniţializarea variabilelor la declararea lor. Utilizarea de pointeri neiniţializaţi poate conduce la comportări surprinzătoare ale programului sau chiar la căderea sistemului.
Există o convenţie pe care majoritatea programatorilor în limbajul C o respectă atunci când folosesc pointeri: unui pointer care nu conţine o locaţie de memorie valabilă i se atribuie valoarea nulă. Acest pointer nu conţine nici o adresă şi nu trebuie folosit.

Totuşi simpla atribuire a valorii nule unui pointer nu oferă nici o siguranţă, existând riscul de a bloca programul dacă se foloseşte acest pointer nul în membrul stâng al instrucţiunii de atribuire.

Exemplu pointer neiniţializat:

```
void main(void)
{
double x,*p;
x=11.32;
*p=x;
}
```

În acest exemplu lucrăm cu un pointer neiniţializat p. Acesta conţine o valoare necunoscută în momentul atribuirii *p=x. Această atribuire determină scrierea valorii lui x într-o locaţie necunoscută de memorie. Cu cât programul este mai complex este mai probabil ca p să conţină o adresă vitală care ar duce la oprirea programului. De aceea este obligatoriu ca pointerul să conţină o adresă corectă înainte de a fi folosit.

Operatori pentru pointeri

Există doi operatori speciali pentru pointeri: & și *.

Operatorul & este un operator unar care returnează adresa de memorie a operandului său. El este denumit operator adresă sau de referenţiere.

Astfel, dacă dorim ca p să pointeze spre x (să aibă ca valoare adresa lui x), atunci putem utiliza atribuirea:
p=&x

Să presupunem că variabila x are valoarea 100 şi foloseşte locaţia de memorie 1500 pentru a-şi stoca valoarea. Conform instrucţiunii de atribuire de mai sus p va avea valoarea 1500.

Operatorul unar * se va numi operator de indirectare sau de dereferenţiere.
Ultima denumire a operatorului unar * decurge din efectul invers al acestuia faţă de operatorul unar &, adică * este complementul lui & şi este un operator unar care returnează valoarea situată la adresa care reprezintă operandul. Într-adevăr, expresia:
*&x
are aceeaşi valoare ca şi operandul x.

Dacă scriem:
q=*x
q primeşte valoarea situată la adresa x, care este valoarea variabilei control de 100, fiindcă 100 este stocat la locaţia 1500 care reprezintă adresa de memorie care era stocată în x.

Observaţie:
Trebuie să existe întotdeauna certitudinea că variabilele pointer indică spre date de tip corespunzător. De exemplu, dacă se declară un pointer ca fiind de tip int, compilatorul C

presupune că la orice adresă stocată de acesta se află o variabilă de tip int, chiar dacă în realitate este altfel.

Dacă avem declaraţiile:
int x;
int*p;
float y;
atunci atribuirea
p=&x
este corectă, în timp ce
p=&y
nu este corectă, deoarece p poate conţine numai adrese de zone de memorie în care se păstrează date de tip int.
Dacă există declaraţia:
float*q;
atunci se poate folosi atribuirea
q=&y

Expresii cu pointeri

În general, expresiile care conţin pointeri se conformează acelorași reguli ca și celelalte expresii din C. În acest paragraf se vor discuta câteva aspecte speciale ale expresiilor cu pointeri.

Atribuirea unui pointer

Ca și în cazul unei variabile oarecare, un pointer poate fi folosit în membrul drept al unei instrucţiuni de atribuire care îi atribuie valoarea unui alt pointer.

Exemplu

```
/*P06_01.CPP*/
/*Atribuirea unui pointer*/
#include <stdio.h>
void main(void)
{
double x=23.45;
int *p1, *p2;

p1=&x;
p2=p1;
printf("   %d", p2); //scrie adresa lui x si nu valoarea acestuia
}
```

Rezultatul execuţiei programului este:
4585572
reprezentând adresa din memorie la care este memorată valoarea de tip double a variabilei x (convertită în întreg). Acum, atât p1 cât şi p2 conţin adresa variabilei x. Adresa lui x este afişată folosind specificatorul de format %p care cere funcţiei printf() să afişeze o adresă în formatul folosit de către calculatorul gazdă.

Operaţii de incrementare şi decrementare a pointerilor

Operatorii ++ şi -- se pot aplica la operanzi de tip pointer. Ei se execută altfel decât asupra datelor care nu sunt pointeri.

Operatorul de incrementare (++) aplicat unui operand de tip pointer spre tipul t, mărește adresa, care este valoarea operandului, cu numărul de octeţi necesari pentru a păstra o dată de tip t.
Operatorul de decrementare (--) are un efect similar, doar că în acest caz valoarea operandului se micşorează cu numărul de octeţi necesari pentru a păstra o dată de tipul spre care pointează operandul.

De exemplu, dacă avem declaraţia:
int*p;
atunci expresiile:
++p;
şi
p++;
măresc valoarea lui p cu 4, deoarece o dată de tip int se păstrează pe 4 octeţi.

În mod analog, expresiile:
--p;
şi
p--;
micşorează valoarea lui p cu 4.
Aceste operaţii sunt utile când se au în vedere prelucrări de date de tip tablou.
6.2.3 Adunarea și scăderea unui întreg dintr-un pointer
Dacă p este un pointer spre tipul t şi n un întreg, atunci se pot utiliza expresiile:
p+n
și
p-n
Expresia:
p+n
are ca valoare, valoarea lui p mărită cu produsul r*n, unde prin r se notează numărul de octeţi necesari pentru a păstra în memorie o dată de tipul t.
În mod analog, valoarea expresiei
p-n
este valoarea lui p micşorată cu produsul r*n.

Expresia:
p=p+2;
are următorul efect: p primeşte adresa celui de-al doilea element de tipul lui p situat după acesta.

Diferenţa a doi pointeri

Operaţia de scădere a unui pointer din altul se face cu scopul de a găsi numărul de obiecte de acelaşi tip care separă cei doi pointeri.
Pot fi scăzuţi de exemplu doi pointeri care pointează spre numele unui tablou.

Fie pointerul p1 spre elementul a[i] al unui tablou şi p2 pointerul spre elementul a[i+m].
Atunci p2-p1=m
Observaţie:
Nu este permisă înmulţirea sau împărţirea pointerilor, aplicarea operatorilor bit cu bit, adunarea sau scăderea variabilelor de tip float sau double la sau din pointeri.

Compararea pointerilor

Doi pointeri care pointează spre elementele aceluiaşi tablou pot fi comparaţi folosind operatorii de relaţie şi de egalitate. Astfel, dacă x pointează spre elementul t[i] al tabloului t (adică are ca valoare adresa elementului t[i]) şi y spre elementul t[j] al aceluiaşi tablou, atunci expresiile din tabelul 6.1 sunt legale.
Tab. 6.1.

Relaţii	Condiţii pentru valoarea de adevărat	Condiţii pentru valoarea de fals
x<y	i<j	i□j
x<=y	i<=j	i>j
x>y	i>j	i□j
x>=y	i>=j	i<j
x==y	i==j	i!=j
x!=j	i!=j	i==j

Operatorii de egalitate (== şi !=) pot fi folosiţi pentru a compara pointerii cu o constantă specială NULL. Aceasta este definită în fişierul stdio.h astfel:

#define NULL 0

Ea reprezintă aşa numitul pointer nul.
Dacă p este un pointer spre orice tip, atunci se pot face comparaţii de forma:
p==NULL
şi
p!=NULL
Este posibil să comparăm un pointer cu valoarea 0 (se recomandă mai ales în limbajul C++), rezultând valorile logice de adevărat sau fals.

Pointeri și tablouri

Între aceste elemente există o legătură foarte strânsă. Numele unui tablou reprezintă un pointer, fiindcă are ca valoare adresa primului element. Totuşi, spre deosebire de o variabilă de tip pointer căreia i se atribuie valori la execuţie, numele unui tablou este un pointer constant care conţine adresa primului său element.

Fie tab un tablou unidimensional de tip t. Atunci tab este un pointer (constant) şi deci o expresie de forma:
tab+n;
este corectă. Conform celor spuse mai sus, rezultă că această expresie este chiar adresa elementului tab[n]. Într-adevăr, tab este adresa lui tab[0]. Atunci tab+1 are ca valoare adresa

lui tab[0] mărită cu r, r fiind numărul de octeţi alocaţi pentru o dată de tip t, adică numărul de octeţi ocupaţi de elementul tab[0].
Deci tab+1 are ca valoare adresa elementului tab[1].
În general, tab+n va avea ca valoare adresa elementului tab[n].

În secvenţa de program:

```
char sir[40], *p;
p=sir;
```

p conţine adresa primului element din tabloul sir. Pentru a accesa al şaselea element al tabloului trebuie să se scrie sir[5] sau *(p+5). Pentru tabloul sir expresiile &sir[0] şi sir sunt echivalente, reprezentând valoarea primului element al şirului.

Dacă considerăm o matrice mat expresiile &mat[0][0] şi mat sunt echivalente, fiecare reprezentând adresa primului element al matricei.

Formula de alocare a memoriei în cazul unui tablou multidimensional este următoarea:
Avem declaraţia de tablou:
tip nume_tablou[dim1][dim2]…[dimn];
Adresa unui element al tabloului nume_tablou[i1][i2]…[in] se calculează cu formula:

Observaţie:
În formula de alocare a memoriei nu apare limita superioară a dimensiunii dim1 a tabloului nume_tablou. Datorită acestui fapt la iniţializarea tabloului este permis ca limita superioară a primului indice să nu fie precizată. De asemenea, când un tablou este folosit ca argument al unei funcţii, este admis ca valoarea dim1 să nu fie specificată.

De exemplu o funcţie care are care ca parametru o matrice de tip int a[10][10] va fi declarată astfel:

```
/*functia are parametrul o matrice*/
func(double x[][150])
{
…
}
```

Indiferent de situaţie, compilatorul trebuie să cunoască dimensiunea parametrului din dreapta pentru a putea opera în cadrul funcţiei cu expresii de genul a[3][4].

Limbajul C asigură două metode de a accesa elementele unui tablou: operaţii cu pointeri şi aplicarea de indecşi tablourilor. Operaţiile cu pointeri sunt mai rapide şi de aceea programatorii în C folosesc adesea pointerii pentru accesarea elementelor dintr-un tablou.

Exemple

1. Să se citească un şir de numere întregi de la tastatură. Să se verifice dacă şirul este simetric şi anume dacă primul element este egal cu ultimul, al doilea cu penultimul, etc. În cazul în care nu există egalitate să se schimbe valorile elementelor în 0 şi să se afişeze noul şir.

```
/*P06_02.CPP*/
/*Verificarea simetriei elementelor unui sir. Utilizarea pointerilor*/
#include <stdio.h>
#define MAX 100
void main(void)
{
int sir[MAX];
int n,i;
printf("Introduceti dimensiunea sirului n=");
scanf("%d",&n);
for(i=0;i<n;i++){
printf("sir[%d]=",i);
scanf("%d",&sir[i]);
}
for(i=0;(n%2==0)?i<n/2:i<(n-1)/2;i++)
        if(*(sir+i)!=sir[n-i-1])
        {
                *(sir+i)=0; *(sir+n-i-1)=0;
        }
printf("\nNoul sir este:\n");
for(i=0;i<n;i++)
printf("sir[%d]=%d\n",i,sir[i]);
}
```

2. Se citeşte de la tastatură un număr natural. Să se afişeze cel mai mare număr care se poate forma cu cifrele numărului dat.
Ex. Pentru numărul 29363, mulţimea cifrelor este {2, 3, 3, 6, 9}, iar numărul cerut este 96332.

```
/*P06_03.CPP*/
/*Formarea celui mai mare numar natural cu cifrele numarului n*/
#include <stdio.h>
#include <math.h>
#define MAX 100
void main(void)
{
int a[MAX];
int n,i=0,j,k, n1;
printf("Introduceti numarul n=");
scanf("%d",&n1);
n=n1;
while (n>1)
{
        *(a+i)=n%10;
        n=floor((double)n/10);
        i++;
}
for (j = 0; j < i; j++){
        for (k = j+1; k < i; k++){
                if (*(a+j) > *(a+k))
```

```
                {
                        *(a+j)=*(a+j)+*(a+k);
                        *(a+k)=*(a+j)-*(a+k);
                        *(a+j)=*(a+j)-*(a+k);
                }
        }
}
n=0;
for(j=0;j<i;j++)
        n=n+*(a+j)*pow(10.0,j);
printf("\nCel mai mare numar natural care se poate forma cu cifrele numarului %d este:
%d\n", n1, n);
}
```

3. Se citeşte de la tastatură un tablou unidimensional cu n elemente numere întregi. Să se afişeze elementul care apare de cele mai multe ori în tablou. Dacă există mai multe astfel de elemente, se vor afişa toate.
Ex. Pentru n=8 şi elementele (23, 7, 11, 7, 19, 7, 11, 11) se vor afişa elementele 7 şi 11, care apar fiecare de câte 3 ori.

```
/*P06_04.CPP*/
/*Afisarea elementului care apare de cele mai multe ori intr-un tablou de n elemente*/
#include <stdio.h>
#include <math.h>
#define MAX 100
void main(void)
{
int a[MAX], imax[MAX];
int n,i,j,k,v=0,u=0;
printf("Introduceti numarul de elemente ale sirului n=");
scanf("%d",&n);
for(i=0;i<n;i++){
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
}
for(i=0;i<n;i++){
        k=0;
        for(j=i+1;j<n;j++){
                if(*(a+i) == *(a+j)){
                        k++;
                }
        }
        if(k>u&&k!=0){
                *(imax+v)=i;
                u=k;
        }
        else
                if(k==u&&k!=0){
                        v++;
```

```
                        *(imax+v)=i;
                        u=k;
                }
}
printf("\Numerele cu cele mai multe aparitii in sirul a sunt:");
for(k=0;k<=v;k++){
        printf("\n %d   ", *(*(imax+k)+a));
}
}
```

4. Să se scrie un program care alocă în mod dinamic spaţiu pentru un şir de caractere şi scrie şirul.

```
/*P06_06.CPP*/
/* Siruri de caractere alocate dinamic */
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
#include<string.h>
void main(void)
{
char *sir;
register int i;
sir=(char*)malloc(60);
if(sir==0) {
        printf("Nu mai este memorie disponibila\n");
        exit(1);
            }
printf("Introduceti sirul:");
gets(sir);
for(i=0;i<=strlen(sir)-1;i++)
      putchar(sir[i]);
      putchar('\n');
free(sir);
}
```