Pointeri

Pointerii reprezintă cel mai complex și mai dificil subiect al limbajului C/C++ datorită libertăților oferite de limbaj în utilizarea lor. Dacă pointerii nu sunt inițializați corect sau dacă conțin valori incorecte conduc la erori greu de depistat și pot determina blocarea aplicației.

O variabilă pointer (pe scurt vom spune un **pointer**) este o variabilă care păstrează **adresa unde este stocata o data**, nu valoarea datei.

O locație de memorie are o adresă și un conținut. Pe de altă parte, o variabilă este o locație de memorie care are asociat un nume și care poate stoca o valoare de un tip particular. În mod normal, fiecare adresă poate păstra 8 biți (1 octet) de date. Un întreg reprezentat pe 4 octeți ocupă 4 locații de memorie. Un sistem „pe 32 de biți" folosește în mod normal adrese pe 32 de biți. Pentru a stoca adrese pe 32 de biți sunt necesare 4 locații de memorie.

Declararea pointerilor

Ca orice variabilă, pointerii trebuie declarați înainte de a putea fi utilizați.

În sintaxa declarării unui pointer se folosește caracterul * înaintea numelui pointerului. Declararea unei variabile de tip pointer include declararea tipului datelor (sau funcției) la care se referă acel pointer. Sintaxa declarării unui pointer la o valoare de tipul "tip" este:

```
tip * nume;  // sau
tip* nume;   // sau
tip *nume;
```

unde

- `tip` reprezintă tipul de baza al pointerului, este tipul variabilei (obiectului) pe care pointerul îl referă (indica)
- `nume` este numele variabilei de tip pointer
- `* - asterix -` operator de indirectare

Exemplu de pointeri:

```
int * p1;   // variabila de tip pointer ce poate memora adresa
            // unei variabile de tip intreg

char * p2;  //variabila de tip pointer ce poate memora adresa
            // unei variabile de tip caracter

float * p3;  //variabila de tip pointer ce poate memora adresa
            // unei variabile de tip float
```

Inițializarea pointerilor, operația de referențiere (&)

Este foarte important ca pointerii sa fie initializati înainte de folosire pentru a evita situatiile in care se acceseaza zone de memorie pentru care programul rulat nu are acces. Accesarea unor zone de memorie nepermise poate cauza erori în program.

Putem inițializa un pointer fie cu o adresă de memorie folosind operatorul de referențiere - de luare a adresei - (&) (a), fie cu pointerul NULL (b).

```
(a)
int var;     // variabila de tip intreg
int * p;     // variabila de tip pointer ce poate memora adresa
             // unei variabile de tip intreg
p = &var;      // in p se memoreaza adresa de inceput a zonei de
              //memorie alocata variabilei var
```

Sunt situatii in care variabila de tip pointer trebuie inițializata cu valoarea 0 (de ex. pentru a fi folosita in instructiuni iterative). Acest lucru se realizeaza astfel:

```
(b)
int * p;     // variabila de tip pointer ce poate memora adresa
             // unei variabile de tip intreg
p = NULL;  // NULL este o constanta predefinita egala cu 0.
```

Observatie: indiferent de tipul variabilei pointer, valoarea atribuita in acest mod este intotdeauna 0 (zero).

Maparea memoriei

Daca un pointer numit `p` contine adreasa unei variabile `var` de tip intreg se spune ca `p` indica (pointeaza) catre un intreg. Dacă prima locatie de memorie pentru variabila `var` este `0x66FEE0`, atunci `p` are valoarea `0x66FEE0`.

Nume variabila	**Valoare variabila**	**Adresa**
var	100	0x66FEE0
		0x66FEE1
		0x66FEE2
		0x66FEE3

p	0x66FEE0	0x87FFF0

Indirectarea sau operația de dereferențiere (*)

Conținutul unei adrese de memorie (stocată într-un pointer) se obține cu operatorul de dereferențiere (operatorul unar * sau operatorul de indirectare). Acest operator returnează valoarea memorata la adresa indicată de pointer.

Daca `var` este o variabila de tip intreg și avem un pointer `p` ce retine adresa de inceput a zonei de memorie alocata variabilei `var` putem retine valoarea lui `var` in alta variabila de tip intreg astfel:

```
(1) int var = 100;    // variabila de tip intreg cu valoarea 100
(2) int * p;          // variabila de tip pointer la intreg

(3) p = &var;        // in p se memoreaza adresa de inceput a zonei
                     // de memorie alocata variabilei var

(4) int nr = *p; // dacă p este adresa lui var, nr devine egal cu
                 // cu valoarea retinuta la adresa retinuta in p,
                 // echivalent nr = var

(5) *p = - 9;    //  atribuie o valoare ce va fi stocata la adresa
                 //  indicata de pointer, NU in variabila pointer!
                 //  echivalent var = -9
```

	Nume variabila	Valoare variabila	Adresa
(1)	var	100	0x66FEE0 0x66FEE1 0x66FEE2 0x66FEE3
(2+3)	p	0x66FEE0	0x87FFF0
(4)	nr	100	0x66FFF0 0x66FFF1 0x66FFF2 0x66FFF3
(5)	var	- 9	0x66FEE0 0x66FEE1 0x66FEE2 0x66FEE3

Observatie: Simbolul * are înțelesuri diferite. Atunci când este folosit într-o declarație (int *p), el denotă că numele care îi urmează este o variabilă de tip pointer. În timp ce, atunci când este folosit într-o expresie/instrucțiune (ex. *p = 99; printf("%d\n", *p);), se referă la valoarea indicată de variabila pointer.

Rețineți:

- Operatorul unar & („adresa lui”) aplicat unei variabile are ca rezultat adresa variabilei respective.
- Operatorul unar * („de la adresa”) are ca rezultat valoarea (continutul) de la adresa pe care o indica.

Aritmetica pointerilor

Deși pointerii pot fi folosiți în expresii aritmetice, aritmetica acestui tip de date este diferită de aritmetica numerelor întregi. Asupra pointerilor pot fi realizate operații de incrementare (++), decrementare (--), adăugare a unui întreg (+ sau +=), scădere a unui întreg (- sau -=) și scădere a unui pointer din alt pointer. Semnificațiile sunt diferite in funcție de tipul de date către care indică (*pointează*), permițând obținerea unor rezultate diferite, în funcție de dimensiunea tipurilor referite. După cum știm, tipurile de date fundamentale din C au dimensiuni diferite. Astfel, de obicei, char este reprezentat pe 1 octet, int pe 2 sau 4 etc. P

Presupunem că o variabila de tip char reste memorata pe un octet iar variabila de tip intreg (int) pe 4 octeți. În aceste condiții, presupunem că am avea urmatoarele declaratii și intializari:

```
(1) char *pc;        // variabila de tip pointer la char
(2) int *pi;         // variabila de tip pointer la intreg

(3) char ch = 'M';   // variabila de tip caracter, retine caracterul
'M'
(4) int nr = 100;    // variabila de tip intreg cu valoarea 100


(5) pc = &ch;        // in pc se memoreaza adresa de inceput a zonei
                     // de memorie alocata variabilei ch
(6) pi = &nr;        // in pi se memoreaza adresa de inceput a zonei
                     // de memorie alocata variabilei nr
```

	Nume variabila	Valoare variabila	Adresa
(3)	ch	M	0x66FFF0
(4)	nr	100	0x66FEE0
			0x66FEE1
			0x66FEE2
			0x66FEE3
(5)	pc	0x66FFF0	0x87DEA0
(6)	pi	0x66FEE0	0x99AF00

Atat operațiile pc++, cat și pi++ sunt permise, insă, efectul obținut nu este același în fiecare caz. Adunarea sau scăderea unui întreg la un pointer, incrementarea sau decrementarea unui pointer se face în multipli de dimensiunea tipului de date la care pointerii se referă, pentru a permite accesul la memorie ca într-un vector.

```
(7) pi++;

/* Aduna la adresa initiala dimensiunea tipului de date referit
   de pointer (pe sizeof(int)), dand acces  la urmatorul intreg
   care ar fi stocat dacă zona aceea de memorie ar fi organizata
   sub forma unui vector
  */

(8) pc++;

(9) pi = pi + 5;    // Incrementeaza adresa cu 5 * sizeof(int)
```

În (7) variabila pi va conține valoarea `0x66FEE4`, in (8), variabila pc va conține valoarea `0x66FFF1`, Explicația este că, în cazul pointerilor, incrementarea se face cu numărul de octeți pe care este reprezentat tipul de bază, pentru ca, în urma acestei operații, ei să indice către elementul următor de același tip.

De fiecare dată când este incrementat/decrementat un pointer, el va indica spre locația din memorie a elementului următor/precedent, element de același tip cu tipul lui de bază. Astfel se aplica regula următoare:

```
tip *pointer;

atunci pointer ± i => adresa_veche ± i*sizeof(tip).
```

Un plus de atenție este necesar când folosim pointeri și diferite tipuri de operatori pentru construirea unor expresii. Astfel, în expresia `*p++`; nu vom obține acces la valoarea aflată la adresa consecutivă adresei indicate de `p`, pentru că operatorul `++`, deși are prioritatea mai mare decât cel de dereferențiere, fiind postfixat, își face efectul după evaluarea expresiei.

Există patru posibile combinații între operatorul de dereferențiere și operatorii ++, respectiv -- și anume:

```
*p++  // incrementează pointerul, dar dereferențiază adresa
      //neincrementată

(*p)++ //face dereferențierea și apoi incrementează pointerul
```

```
*++p //incrementează pointerul și, apoi, face dereferențierea

++*p //face dereferențierea și apoi incrementează pointerul
```

Relația Pointer - Vector

O variabilă vector conține adresa de început a vectorului (adresa primei componente a vectorului), și de aceea este echivalentă cu un pointer. Această echivalență este exploatată, de obicei, în argumentele de tip vector și în lucrul cu vectori alocați dinamic. De exemplu, pentru declararea unei funcții care primește un vector de întregi și dimensiunea lui, avem două posibilități:

```
void printVec(int v[], int n);

sau

void printVec(int *v, int n);
```

În interiorul funcției ne putem referi la elementele vectorului a fie prin indici, fie prin indirectare, indiferent de felul cum a fost declarat parametrul vector `v`:

```
void printVec(int v[], int n);
{
  int i;
  for (i = 0; i < n; i++)
    printf("%d", v[i]); // Indexare
}
```

sau

```
void printVec(int *v, int n);
{
  int i;
  for (i = 0; i < n; i++)
    printf("%d", *(v+i)); // Indirectare
}
```

Astfel, există următoarele echivalențe de notații pentru un vector `v`:

```
v[0] <==> *v
v[1] <==> *(v + 1)
v[k] <==> *(v + k)
&v[0] <==> v
```

```
&v[1] <==> v + 1
&v[k] <==> v + k
```

Diferența dintre o variabilă pointer și un nume de vector este aceea că un nume de vector este un pointer constant (adresa sa este alocată de către compilatorul C și nu mai poate fi modificată la execuție), deci nu poate apărea în stânga unei atribuiri, în timp ce o variabilă pointer are un conținut modificabil prin atribuire sau prin operații aritmetice. De exemplu:

```
int v[100], *p;
p = v; ++p; //corect
v = p; ++v; //EROARE
```

De asemenea, o variabilă de tip vector conține și informații legate de lungimea vectorului și dimensiunea totală ocupată în memorie, în timp ce un pointer doar descrie o poziție în memorie (e o valoarea punctuală). Operatorul `sizeof(v)` pentru un vector `v[N]` de tipul `T` va fi `N * sizeof(T)`, în timp ce `sizeof(v)` pentru o variabila `v` de tipul `T *` va fi `sizeof(T *)`, adică dimensiunea unui pointer.

Ca o ultimă notă, este importat de remarcat că o funcție poate avea ca rezultat un pointer, dar nu poate avea ca rezultat un vector.