POINTERI

O variabilă de tip pointer reţine o adresă din memorie. Prin abuz de limbaj numim o variabilă de tip pointer doar „pointer”. Declararea unui pointer are forma următoare:

```
tip *nume;
```

Pointerul `nume` poate reţine adresa unei variabile de tipul `tip`. Tipul variabilei `nume` este `tip *`. În declaraţia de mai jos, pointerul `p` reţine adresa variabilei `a` de tip `int` .

```
int a = 10;
int *p = &a;
```

Operatorul adresă `&` (numit şi operatorul de referenţiere) indică adresa variabilei în faţa căreia este plasat. În exemplul de mai sus, prin `&a` înţelegem *adresa din memorie a variabilei* `a`. Tipul lui `&a` este `*int` . Tipul lui `&p` este `**int` . Pentru a accesa valoarea aflată la o adresă reţinută de un pointer, putem folosi operatorul de indirectare `*` (numit şi operatorul de dereferenţiere). Dacă pointerul `p` este cel de mai sus, atunci după următoarea atribuire, variabila `b` reţine valoarea `10`.

```
int b = *p;
```

Operatorul `*` este complementar operatorului `&`. Instrucţiunea

```
b = *(&a);
```

este echivalentă cu instrucţiunea

```
b = a;
```

Ambii operatori (`*` şi `&`) sunt unari, adică au un singur operand. Pentru a atribui adresa reţinută de un pointer `p_1` unui pointer `p_2`, trebuie ca ambii pointeri să reţină adrese de acelaşi tip:

```
float d = 1.0;
float *p_1 = &d;
float *p_2 = p_1;
```

Atribuirea `p_2 = p_1` din exemplul următor NU este corectă:

```
int a = 10;
int *p_1 = &a;
float *p_2 = p_1;
```

Un pointer de tip `void *` poate reţine adrese de orice tip.

```
void *p_3 = p_1;
p_3 = p_2;
```

Pointeri către structuri

```
struct nod{
  int info;
  nod *next;
} *pnod;
```

Pointerul `pnod` poate reţine adresa variabilelor de tip `nod`.

```
nod n;
n.info = 10;
n.next = 0;
pnod = &n;
```

În exemplul de mai sus, s-a declarat o variabilă `n` de tipul `nod`. Prin operatorul `.` (de selecţie) s-au accesat pe rând câmpurile variabilei `n` şi au fost iniţializate. În ultima instrucţiune, pointerului `pnod` i s-a atribuit adresa variabilei `n`. Pentru a modifica cele două câmpuri, `info` şi `next`, din structura de tip `nod` aflată acum la adresa reţinută de `pnod` putem folosi două variante. Prima variantă utilizează operatorul `*` de dereferenţiere:

```
(*pnod).info++;
```

Astfel, operatorul de selecţie `.` se aplică variabilei de tip `nod` ce se află la adresa indicată de `pnod`. Parantezele sunt necesare, deoarece operatorul de selecţie are o prioritate mai mare decât cel de dereferenţiere. A doua variantă foloseşte operatorul de selecţie indirectă `->`:

```
pnod->info++;
```

Alocarea dinamică a memoriei

Prin alocarea dinamică a memoriei se alocă spaţiu în memorie în timpul execuţiei programului (dinamic). Operatorul `new` este folosit pentru alocarea de spaţiu pentru o variabilă. Spaţiul alocat are o dimensiune egală cu numărul de bytes necesari reţinerii unei date de tipul asociat variabilei. Adresa de la care acesta începe poate fi reţinută de un pointer către date de acelaşi tip.

```
pointer = new tip;
```

Durata de viaţă a unei variabile alocate dinamic este fie până când spaţiul este eliberat prin utilizarea operatorului `delete` , fie până când execuţia programului se încheie.

```
delete pointer;
```

Aritmetica pointerilor

Considerăm un pointer `p` care reţine adrese de tipul `tip`. Expresia `p + n`, unde `n` este un număr întreg, reprezintă adresa din memorie obţinută din suma dintre adresa reţinută de `p` şi `n` înmulţit cu numărul de bytes ale tipului `tip`. Numărul de bytes necesari reţinerii unei variabile de tipul `tip` poate fi obţinut prin `sizeof (tip)`. Spre exemplu, fie un tablou unidimensional (vector):

```
int a[4] = {0, 1, 2, 3};
```

Variabila `a` este un pointer cu tipul `int *`, care reţine adresa `&a[0]`, a primului element din vector. Expresia `*a` este echivalentă cu `a[0]`. Expresia `*(a+2)` este echivalentă cu `a[2]`, deoarece `(a+2)` este adresa obţinută adăugând la adresa din `a`, $2 \cdot$ numărul de bytes pe care se reprezintă o dată de tip `int` , adică un total de 8 bytes. În general, `*(a+i)` reprezintă `a[i]`. Parcurgerea vectorului `a` cu dimensiunea `int n = 4` folosind un pointer `pa` pentru traversare:

```
int *pa = a;
while (pa < a + n) {
  cout << *pa;
  pa++;
}
```

Într-un tablou bidimensional `int b[m][n]`, pentru a accesa elementul `b[i][j]` folosind pointeri, putem scrie:

```
*(*(b + i) + j)
```

Expresia `*(b + i)` este echivalentă cu `b[i]`, care este un tablou de dimensiune `n` şi deci un pointer de tip `int *`.

TRANSMITEREA PARAMETRILOR

Transmiterea parametrilor unei funcţii se poate face prin valoare sau prin referinţă. Dacă transmiterea unui parametru se face prin valoare, atunci modificările realizate în interiorul funcţiei asupra parametrului respectiv, nu vor avea vizibilitate în afara funcţiei, modificările sunt locale funcţiei (temporare). În schimb, dacă parametrul este transmis prin referinţă, atunci este transmisă de fapt adresa variabilei, iar modificările se fac la această adresă, fiind deci permanente şi vizibile şi după revenirea din funcţie. Există posibilitatea de a modifica valoarea unor variabile transmise prin valoare, dacă se lucrează cu pointeri (acesta este mecanismul care permite modificarea datelor dintr-un tablou/matrice transmis ca parametru unei funcţii):

```
void swap( int *a, int *b ) {
  int aux = *a;
  *a = *b;
  *b = aux;
}

void main(){
  int a = 2, b = 5;
  swap(&a, &b);
}
```

Transmiterea prin referinţă:

```
void swap( int &a, int &b ) {
  int aux = a;
  a = b;
  b = aux;
}
void main(){
  int a = 2, b = 5;
  swap(a, b);
}
```

La transmiterea prin valoare se pasează de fapt o copie a variabilei transmise ca parametru. Această variabilă-copie se află la o altă adresă din memorie, motiv pentru care modificarea directă a ei nu va avea vizibilitate în afara funcţiei. În primul exemplu, valoarea transmisă însă este adresa variabilelor `a` şi `b`, iar în funcţia `swap` se creează copii a căror valoare este adresa aceasta. O atribuire de tipul `a = null ;` în funcţia `swap` va face ca adresa variabilei-copie care reţine `&a` să devină `null` , nu adresa `&a`, adică valoarea reţinută de ea.

La transmiterea prin referinţă nu se mai efectuează acest mecanism de copiere, ci se transmite o variabilă „alias" (cu un alt nume, chiar dacă poate fi acelaşi nume), dar care se află la aceeaşi adresă cu variabila transmisă. În al doilea exemplu, au fost transmise efectiv variabilele `a` şi `b` funcţiei `swap`.

Transmiterea parametrilor prin valoare - exemple

Modificare eşuată a valorii transmise

```
f( char a1 ) {
   a1++;
}
void main(){
   char a = 1;
   f(a);
}
```

&a1	0x01	1	a1
&a	0x00	1	a
	⇓		
&a1	0x01	2	a1
&a	0x00	1	a

Modificare cu succes a valorii transmise prin pointer

```
f1( char *pa1 ) {
   (*pa1)++;
}
void main(){
   char a = 1;
   char *pa = &a;
   f1(pa);
}
```

&pa1	0x02	0x00	pa1
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&pa1	0x02	0x00	pa1
&pa	0x01	0x00	pa
&a	0x00	2	a

Modificare cu succes a valorii transmise prin pointer la pointer

```
f2( char **ppa1 ) {
   (*(*ppa1))++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f2(ppa);
}
```

&ppa1	0x03	0x01	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa1	0x03	0x01	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x00	pa
&a	0x00	2	a

Modificare eşuată a pointerului transmis

```
f( char *pa1 ) {
   pa1++;
}
void main(){
   char a = 1;
   char *pa = &a;
   f(pa);
}
```

&pa1	0x02	0x00	pa1
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&pa1	0x02	0x02	pa1
&pa	0x01	0x00	pa
&a	0x00	2	a

Modificare cu succes a pointerului transmis prin pointer

```
f1( char **ppa1 ) {
   (*ppa1)++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f1(ppa);
}
```

&ppa1	0x03	0x01	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa1	0x03	0x01	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x01	pa
&a	0x00	2	a

Modificare eşuată a pointerului la pointer transmis

```
f( char **ppa1 ) {
   ppa1++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f(ppa);
}
```

&ppa1	0x03	0x01	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa1	0x03	0x02	ppa1
&ppa	0x02	0x01	ppa
&pa	0x01	0x00	pa
&a	0x00	2	a

Transmiterea parametrilor prin referinţă - exemple

Modificare cu succes a valorii transmise

```
f( char &a1 ) {
   a1++;
}
void main(){
   char a = 1;
   f(a);
}
```

&a/&a1	0x00	1	a/a1
	⇓		
&a/&a1	0x00	2	a/a1

Modificare cu succes a valorii transmise prin pointer

```
f1( char *&pa1 ) {
   (*pa1)++;
}
void main(){
   char a = 1;
   char *pa = &a;
   f1(pa);
}
```

&pa/&pa1	0x01	0x00	pa/pa
&a	0x00	1	a
	⇓		
&pa/&pa1	0x01	0x00	pa/pa
&a	0x00	2	a

Modificare cu succes a valorii transmise prin pointer la pointer

```
f2( char **&ppa1 ) {
   (*(*ppa1))++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f2(ppa);
}
```

&ppa/&ppa1	0x02	0x01	ppa/ppa1
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa/&ppa1	0x02	0x01	ppa/ppa1
&pa	0x01	0x00	pa
&a	0x00	1	a

Modificare cu succes a pointerului transmis

```
f( char *&pa1 ) {
   pa1++;
}
void main(){
   char a = 1;
   char *pa = &a;
   f(pa);
}
```

&pa/&pa1	0x01	0x00	pa/pa1
&a	0x00	1	a
	⇓		
&pa/&pa1	0x01	0x01	pa/pa1
&a	0x00	1	a

Modificare cu succes a pointerului transmis prin pointer

```
f1( char **&ppa1 ) {
   (*ppa1)++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f1(ppa);
}
```

&ppa/&ppa1	0x02	0x01	ppa/ppa1
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa/&ppa1	0x02	0x01	ppa/ppa1
&pa	0x01	0x01	pa
&a	0x00	1	a

Modificare cu succes a pointerului la pointer transmis

```
f( char **&ppa1 ) {
   ppa1++;
}
void main(){
   char a = 1;
   char *pa = &a;
   char *ppa = &pa;
   f(ppa);
}
```

&ppa/&ppa1	0x02	0x01	ppa/ppa1
&pa	0x01	0x00	pa
&a	0x00	1	a
	⇓		
&ppa/&ppa1	0x02	0x02	ppa/ppa1
&pa	0x01	0x00	pa
&a	0x00	1	a