Limbaje de programare

Pointeri

19 noiembrie 2012

Un pointer e o adresă

Orice variabilă `x` de tipul *tip* are o *adresă* `&x` de tipul *tip* `*`
= adresa la care e memorată valoarea (conținutul) variabilei

Adresa e o valoare numerică, dar nu e un `int` / `unsigned` .
Se poate tipări cu `printf`, formatul %p

Adresele sunt nenule. Valoarea `NULL` (0) indică o adresă invalidă.

Discutăm:

1. Cum *declarăm* o variabilă pointer (de tip adresă)
2. Cum *obținem* o valoare de tip pointer (adresă)
3. Cum și la ce *folosim* un pointer (o adresă)

Declararea, inițializarea și atribuirea adreselor

Declararea de pointeri: *tip* **nume_var*;

$\Rightarrow$ *nume_var* poate conține adresa unei valori de tipul *tip*

Exemplu: `char *s;` `int *p;`

Când declarăm mai mulți pointeri, * apare la *fiecare nume*:

`int *p, *q;` declară doi pointeri la întregi

`int *p, q;` declară un pointer `p` și un întreg `q`

Obținerea unor valori pointer (adresă)

Numele unui tablou e un pointer: `int tab[10], *a = tab;`

sau: `int tab[10]; int *a; a = tab;`

Operatorul & produce un pointer: `int n, *p = &n;`

sau: `int n; int *p; p = &n;`

O *constantă șir* are tip pointer: `char *s = "test";`

sau: `char *s; s = "test";`

Ce valoare se află la o adresă?

Operatorul de dereferențiere (indirectare) `*` operator prefix

operand: pointer; rezultat: *obiectul* (variabila) indicat de pointer

`*p` poate fi folosit la dreapta unei atribuiri
sau la stânga (eng. *lvalue*), ca o variabilă (sau element de tablou)

dacă `p` e `&x`, atunci `*p` e obiectul de la adresa `p` (a lui `x`), deci `x`

```
int x, y, *p = &x;   y = *p;  /* y = x */  *p = y; // x = y
```

Operatorul `*` e *inversul* lui `&`:

`*&x` e chiar `x` (obiectul de la adresa lui x)

Declarație și dereferențiere

Putem citi declarația *tip* * *p*;

tip * p; p are tipul *tip* *

tip *p; *p are tipul *tip*

`char **s;` adresă de adr.de char

`char *t[8];` tab.de 8 adr.de char

Variabilă	Valoare	Adresă
`int x = 5;`	5	0x408
	...	
`int *p=&x;`	0x408	0x51C
	...	
`int **pp=&p;`	0x51C	0x9D0

ATENȚIE O *declarație* cu *inițializare* NU este o *atribuire* !

`int t[2] = { 3, 5 };` inițializează `t`. Greșit: ~~`t[2] = { 3, 5 };`~~

`int x, *p = &x;` e la fel ca `int x; int *p; p = &x;`
(e inițializat/atribuit `p`, NU `*p`). ~~`*p = &x`~~ e greșit ca tip!

`char *p = "sir";` e `char *p; p = "sir";` Greșit: ~~`*p = "sir;"`~~

Folosirea pointerilor: atribuiri în funcții

O funcție NU poate modifica o variabilă transmisă ca parametru.

Cu *adresa* `p` a unei variabile putem: să-i *folosim* valoarea: `...= *p;`
să *atribuim* valoarea: `*p =...;`

Primind o *adresă*, o funcție *poate scrie* la ea o valoare (ca `scanf`).

```
void swap (int *pa, int *pb) { // schimbă valori de la 2 adrese
  int tmp;  // var. temporară pentru valoarea schimbata prima
  tmp = *pa; *pa = *pb; *pb = tmp;   // atribuiri de intregi
}
```

Ex.: `int x = 3, y = 5; swap(&x, &y); // acum x = 5 şi y = 3`

Folosim *adrese ca parametri* de funcții:
- ca să transmitem *tablouri* (în C nu putem transmite *conținutul*)
- pentru a întoarce *mai multe valori* (`return` permite doar una)
- ex. minimul *și* maximul unui tablou; rezultat *și* cod de eroare