Programarea Calculatoarelor

Cursul 6: Pointeri (I). Declarare. Pointeri constanți. Pointeri și tablouri. Operații cu pointeri. Pointeri ca argument și valoare returnată

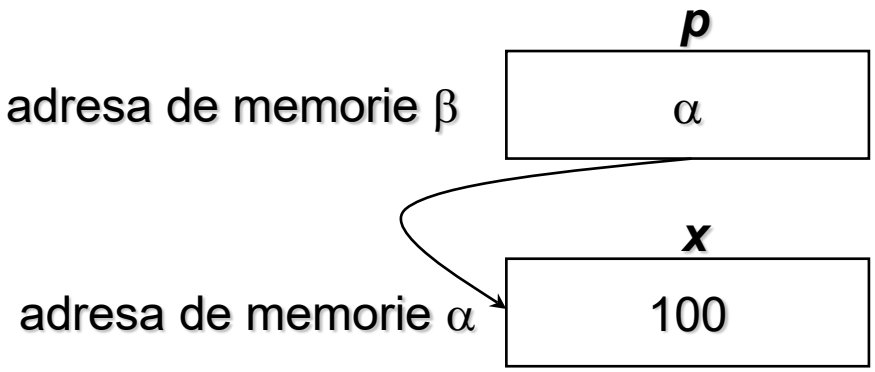
Ion Giosan

Universitatea Tehnică din Cluj-Napoca

Departamentul Calculatoare

Pointeri

- Un **pointer** este o variabilă care conține adresa unei alte variabile
- Exemplu



Cod în C:

```
int x = 100;
int *p = &x;
```

- Orice pointer are un anumit **tip**
- În exemplul de mai sus
  - **x** este o variabilă de tip **int** având valoarea 100
  - **p** este un pointer de tip **int** având valoarea α
- Un pointer (o adresă de memorie) ocupă în memorie
  - 4 octeți, dacă programul rezultat este o aplicație pe 32 de biți (*în întregul curs de Programarea Calculatoarelor considerăm acest caz*)
  - 8 octeți, dacă programul rezultat este o aplicație pe 64 de biți

Declararea pointerilor. Asignarea adresei unei variabile

- Caracterul * precede identificatorul pointer-ului

```
tip *identificator
```

- Exemplu

```
int *p;
```

- Adresa unei variabile se poate obține utilizând **operatorul referință &**
- Asignarea adresei variabilei întregi **x** la un pointer **p**:
  - Declarații

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

  - Asignarea adresei

```
p=&x;
```

Determinarea valorii stocate la o adresă de memorie

- Valoarea stocată la adresa de memorie referită de pointerul **p** se poate determina utilizând **operatorul de dereferențiere ***

- Exemplu

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

  - Asignarea **x=y** se poate face utilizând una din următoarele două secvențe de cod

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

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

Erori frecvente cu pointeri

- Notația `*` care apare în declarația unui pointer înainte de numele identificatorului nu este distributivă

```
int *a, b;
```

  - `a` este un pointer la un întreg
  - `b` este un întreg
- După declararea unui pointer, acesta este neinițializat
  - Exemplu – declarația unui pointer la `int`; acesta nu referă o zonă de memorie alocată

```
int *a;
```

  - Eroare frecventă

```
int *a;
scanf("%d", a);
```

    - Nu putem citi un întreg într-o zonă de memorie nealocată în prealabil!

Erori frecvente cu pointeri

- După declararea unui pointer, acesta referă aproape sigur o zonă de memorie nealocată
  - Nu putem scrie o valoare la acea adresă întrucât nu a fost alocat spațiu de memorie!

```
int* p; /* declara pointer-ul, insa acesta nu
           refera o zona de memorie alocata */

*p = 10; /* dereferentierea unei zone de memorie
            nealocate cauzeaza o eroare serioasa
            in momentul executiei programului */
```

  - Pointerii trebuie inițializați înainte de a fi utilizați!

Pointeri la *void*

- Când un pointer nu trebuie să aibă un anumit tip se utilizează

```
void *identifier;
```

- Utilizarea pointerilor la `void` asigură genericitate implementărilor
  - Un pointer la `void` poate primi valoarea unui pointer de orice tip
- Un pointer la `void` nu poate fi dereferențiat
- Exemplu

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

- Atribuiri corecte: `p = &x; p = &y;`
- Pentru dereferențiere trebuie specificat mai întâi tipul (operație de cast) `(tip *)p` și apoi efectuată dereferențierea

```
p = &y;   float z = *((float *)p);
```

Pointeri la *void*

- La fiecare moment de timp, programatorul trebuie să știe tipul valorii memorate la adresa referită de către un pointer la **`void`**
- Exemplu

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

Atribuirea **`x=10;`** este echivalentă cu utilizarea următoarei secțiuni de cod

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

Dereferențierea directă a lui **`p`** nu se poate face fără a face mai întâi conversia de tip la **`int *`**

Eroare:

```
*p = 10;
```

Pointeri constanți

- Declararea unui pointer constant

```
tip* const identificator=valoare;
```

  - Pointer-ul `identificator` este un pointer constant la o zonă de memorie care conține o valoare de tipul `tip`
  - Pointer-ului respectiv nu i se va putea schimba adresa pe care o referă

- Exemplu

```
double z = 4.52;
double* const x = &z;
double y = 3.89;
```

Atribuire permisă (valoarea de la adresa `x` se schimbă)

```
*x = y;
```

Atribuire imposibilă (adresa referită de `x` este constantă!)

```
x = &y;
```

Pointeri la valori constante

- Declararea unui pointer la o valoare constantă

```
const tip* identificator=valoare;
tip const* identificator=valoare;
```

  - Pointer-ul `identificator` este un pointer la o zonă de memorie care conține o valoare constantă de tipul `tip`
  - Valoarea stocată la adresa respectivă nu se va putea schimba

- Exemplu

```
double z = 4.52;
const double* x = &z;
double y = 3.89;
```

Atribuire imposibilă (valoarea stocată la adresa `x` nu se poate schimba!)

```
*x = y;
```

Atribuire permisă (adresa `x` poate fi schimbată cu adresa lui `y`)

```
x = &y;
```

Pointeri constanți la valori constante

- Declararea unui pointer constant la o valoare constantă

```
const tip* const identificator=valoare;
```

  - Pointer-ul `identificator` este un pointer constant la o zonă de memorie care conține o valoare constantă de tipul `tip`
  - Nici pointer-ul, nici valoarea stocată la adresa respectivă nu se vor putea schimba

- Exemplu

```
double z = 4.52;
const double* const x = &z;
double y = 3.89;
```

Atribuire imposibilă (valoarea de la adresa lui `x` nu se poate schimba!)

```
*x = y;
```

Atribuire imposibilă (adresa `x` nu poate fi schimbată cu adresa lui `y`!)

```
x = &y;
```

Pointeri constanți vs. Pointeri la constante

- Diferențele constau în poziționarea cuvântului **`const`** înainte sau după caracterul **`*`**
  - Pointer constant: **`tip* const identificator;`**
  - Pointer la o constantă: **`tip const *identificator;`**

- În declarația unui parametru formal al unei funcții

  `const tip *parametru_formal`

  - Face ca valorile din zona de memorie referită de parametrul actual corespondent să nu poată fi modificate!

Pointeri și tablouri

- Numele unui tablou reprezintă valoarea adresei de memorie a primului său element
- Numele unui tablou este un pointer constant la primul său element – nu poate schimbat de-a lungul execuției programului
- Exemplu

```
int a[100], b[100];
int *p;
int x;
...
p=a; // p reține adresa lui a[0]
b=a; // gresit! -> b este pointer constant
```

`x=a[0]` este echivalentă cu `x=*p;`

`x=a[10]` este echivalentă cu `x=*(p+10);`

```
  // al unsprezece-lea element din tablou
     (elementul de pe pozitia numarul 10)
```

Operații cu pointeri

- Incrementarea/decrementarea cu 1
  - Se pot utiliza operatorii **++** și **--**
- Exemplu

```
double a[100];
double *p;
p=&a[10];
printf("%p\n", p); // 0028fc38
p++; /* p refera acum elementul a[11].
        Valoarea lui p este incrementata cu
        dimensiunea unui double - 8 octeti*/
printf("%p\n", p); // 0028fc40
```

Operații cu pointeri

- Adunarea/scăderea unui întreg la/dintr-un pointer
  - Operațiile **`p+n`** și **`p-n`** rezultă în incrementarea respectiv decrementarea valorii lui **`p`** cu (**n x numărul de octeți**) necesari pentru a memora o valoare de tipul lui **`p`**
- Exemplu

```
double a[100];
double *p = a+4; /* p este adresa de inceput a
                    tabloului a plus 32 octeti
                    (4 elemente x 8 octeti),
                    p referă pe elementul a[4]*/
printf("%p;%p\n", a, p); // 0028fbe0;0028fc00

 /* urmatoarele trei instructiuni sunt
    echivalente, x fiind valoarea lui a[4] */

 double x = a[4];
 double x = *p;
 double x = *(a+4);
```

Operații cu pointeri

- Diferența a doi pointeri
  - Dacă doi pointeri `p` și `q` referă elementele de pe pozițiile `i` și `j` dintr-un tablou `a`, adică `p=&a[i]` și `q=&a[j]` atunci `q-p=j-i`
  - Diferența a doi pointeri reprezintă numărul de octeți dintre cele două adrese de memorie împărțit la numărul de octeți pe care este reprezentat tipul de date al pointerilor respectivi
- Exemplu

```
double a[100];
double *p = a+8; /* p referă elementul a[8] */
double *q = a+10; /* p referă elementul a[10] */
int dif = q-p;
printf("%p;%p;%d\n",p,q,dif); //0028fc20;0028fc30;2
```

Operații cu pointeri

- Compararea a doi pointeri
  - Se poate face cu ajutorul operatorilor `<  <=  >  >=  ==  !=`
  - Rezultatul este dat de compararea valorilor adreselor de memorie pe care le conțin acei pointeri
- Exemplu

```
double a[100];
double *p = a+8; /* p referă elementul a[8] */
double *q = a+10; /* p referă elementul a[10] */
double *r = &a[8]; /* r referă elementul a[8] */
printf("%d\n", p>q);  // 0
printf("%d\n", p==r); // 1
printf("%d\n", q!=r); // 1
printf("%d\n", p<=q); // 1
```

Pointeri – exemplul 1

```
#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);
    }
 }
```

Pointeri – exemplul 1

```
int main()
{
   int i, n, maximum, minimum, x[100];
   printf("Dimensiunea tabloului este:");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
       printf("x[%d]=", i);
       scanf("%d", &x[i]);
   }

   /* Apelul primei functii */
   max_min1(n, x, &maximum, &minimum);
   printf("max_min1: maximum=%d si minimum=%d\n", maximum, minimum);

   /* Apelul celei de-a doua functii -> utilizand operatii cu pointeri*/
   max_min2(n, x, &maximum, &minimum);
   printf("max_min2: maximum=%d si minimum=%d\n", maximum, minimum);

   return 0;
}
```

Pointeri – exemplul 2

```
int array[10]; /* Tablou de intregi -> date de intrare */

int main() {

   int *data_ptr; /* Pointer la date */
   int value; /* O valoare numar intreg */
   data_ptr = &array[0];/* Pointer la primul element din tablou*/

   value = *data_ptr++; /* Obtine elementul de pe pozitia #0,
                           data_ptr refera elementul #1 */

   value = *++data_ptr; /* Obtine elementul de pe pozitia #2,
                           data_ptr refera elementul #2 */

   value = ++*data_ptr; /* Incrementeaza elementul #2,
                           se returneaza valoarea lui,
                           data_ptr ramane neschimbat */
}
```

Pointeri ca argument și valoare returnată

- Trimiterea unui pointer ca și argument la apelul unei funcții se face, în C, prin valoare (ca și orice alt argument de alt tip)

- O funcție poate returna un pointer (ca și orice altă valoare de alt tip)

  Atenție!

  - Nu returnați un pointer la o variabilă locală (automată, alocată pe stivă) funcției respective!
  - Această variabilă este alocată pe stivă, iar la terminarea execuției corpului funcției nu se mai garantează accesul valid la zona respectivă de memorie
  - Este posibil ca zona respectivă de memorie să fie eliberată imediat după ieșirea din funcție!

Pointeri ca argument și valoare returnată

```
int* f(int* a, int* b) {
    int *c=(*a<*b)?a:b;
    return c;
}
int* g(int* a, int* b) {
    int val=(*a<*b)?*a:*b;
    int *c=&val;
    return c; // adresa unei variabile locale automate (alocate pe stiva)!
}
int main() {
    int x=30;
    int y=60;
    int z=*(f(&x,&y));
    printf("%d\n",z); // 30 -> cu siguranta!
    int t=*(g(&x,&y)); /* este posibil ca zona de memorie unde este alocat
                          "val" sa fie eliberata imediat dupa apelul lui "g" */
    printf("%d\n",t); /* 30 -> numai in cazul in care dereferentierea din
                           instructiunea precedenta nu a esuat! */
    return 0;
}
```