Programarea Calculatoarelor
Cursul 6

Pointeri;
Legătură pointeri și tablouri

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Obiective

- Ce este un pointer?
- De ce avem nevoie de pointeri?
- Ce legătură există între un pointer și un tablou?

Model simplu memorie

- Codul și datele (variabilele) programului sunt stocate în memorie
- Unitatea de bază este un byte (octet)
  - 8 biți = două cifre hexazecimale
- Adresa unei variabile este poziția primului byte în memorie
  - Adresele se schimbă la fiecare rulare
  - Nu respectă ordinea în care au fost declarate variabilele
- Byte-ul cel mai puțin semnificativ este stocat la adresa cea mai mică (little endian)
  - depinde de sistemul de operare

Model memorie - exemplu

int x = 1000000007;
= 0011 1011 1001 1010
1100 1010 0000 0111$_{(2)}$
= 0x 3B 9A CA 07

short s = -3000;
= 1111 0100 0100 1000$_{(2)}$
= 0x F448

char c = ‘a’;
= 97
= 0110 0001$_{(2)}$
= 0x 61

Adresă	Conținut
...	
304	0000 0111 = 07
305	1100 1010 = CA
306	1001 1010 = 9A
307	0011 1011 = 3B
...	
412	0110 0001 = 61
...	
512	0100 1000 = 48
513	1111 0100 = F4

Pointeri - introducere

- Definiție
  - Un pointer este menit să conțină adresa din memorie a unei variabile
  - Este un număr întreg care reprezintă poziția în memorie
    - În general este un segment și offset
  - Este legat la un tip anume
  - Nu este controlat sau verificat
- Utilitate
  - Modificarea variabilelor în funcții
  - Evitarea copierii variabilelor mari
    - tablouri, șiruri de caractere, structuri
    - se trimite doar adresa de început
- Periculos
  - Putem avea acces la zone de memorie interzise
  - Putem să interpretăm în mod greșit ce se află la o adresă

Pointeri – declarare

- Caracterul * urmează după tipul pointer-ului

```
tip* identificator
```

- Exemple

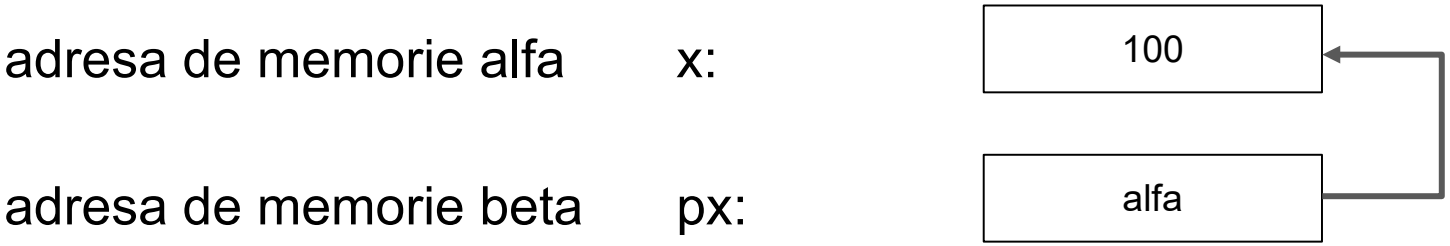
```
int* px;
char* pc;
long long int* pll;
```

- Se citește: px este pointer la un int
- Se recomandă denumirea sugestivă a pointerilor
  - De exemplu, numele lor să înceapă cu litera p
- Inițializarea se va face cu o adresă de memorie validă
- Există un pointer 0 care se cheamă `NULL`

Pointeri – operatorul &

- Adresa unei variabile se poate obține utilizând operatorul unar referință &
- Dacă x are tipul T atunci &x are tipul pointer la T
- Adresele se afișează cu specificatorul de format %p
  - Arată poziția din memorie ca un număr hexazecimal

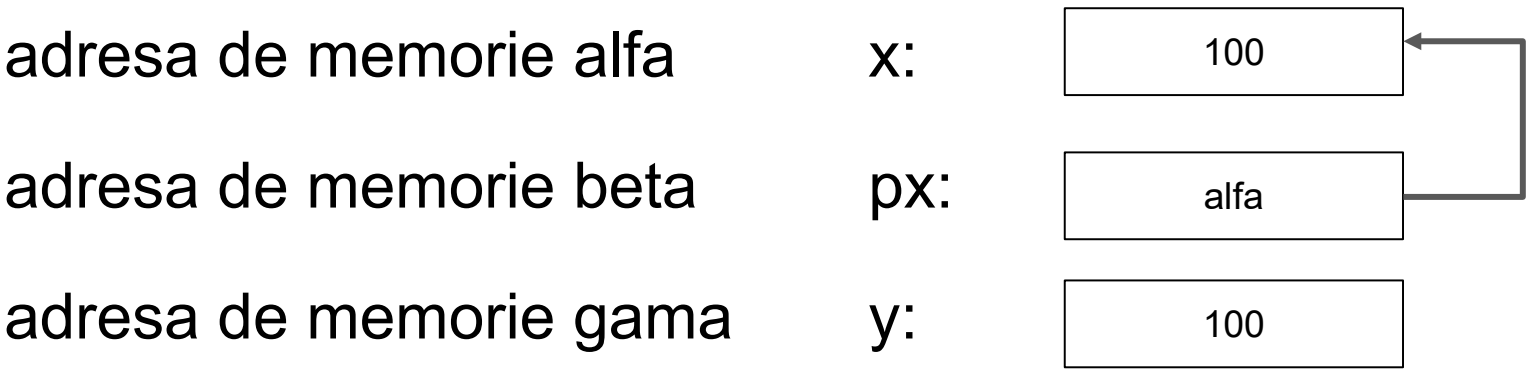
```
int x = 100;
int* px = &x;
printf("%p\n", px); //alfa
```



Pointeri – operatorul *

- Valoarea stocată la adresa de memorie referită de un pointer se poate determina utilizând operatorul unar dereferențiere *

```
int x = 100;
int* px = &x;
int y = *px;
```



Pointeri – operatorul *

- Atenție!
- Operatorul * și modificatorul * de la declararea unui pointer au roluri total diferite
  - Când este după un tip specifică faptul că avem un pointer la tipul respectiv
  - Când apare în fața unei variabile se referă la operatorul de dereferențiere

`int*` = tipul unui pointer la int

*x este conținutul de la adresa stocată în x

Erori frecvente cu pointeri

- Modificatorul * care apare în declarația unui pointer înainte de numele identificatorului nu este distributiv

```
int* pa, b; //pa este un pointer la un int, b este un int
```

- După declarare un pointer local este neinițializat

```
int* p;   //ca orice variabila locala, p este aleator
```

- Accesare zonă de memorie interzisă - eroare la rulare

```
int* p;

scanf("%d", p);

//pointerul p probabil că arată către o zonă de memorie interzisă
```

Pointeri - exemplu simplu

```c
#include <stdio.h>

int main(){
    int x = 100;
    int* px = &x;
    int* px2 = &x;
    *px2 = 50;
    printf("%d\n", x);
    int y = *px;
    y /= 2;
    printf("%d\n", x);
    *px /= 2;
    printf("%d\n", x);
    px = NULL;
    *px /= 2;
    printf("%d\n", x);
    return 0;
}
```

- px arată către zona de memorie lui x
- px2 la fel
- modificăm valoarea de la adresa din px2
- 50 - se modifică valoarea lui x
- y este valoarea stocată la adresa px
- se modifică valoarea lui y
- 50 - variabila x este neafectată
- se modifică conținutul de la adresa px
- 25 - se modifică valoarea lui x
- setăm pointerul px la pointerul special NULL
- eroare la rulare, nu avem voie să modificăm conținutul de la adresa respectivă

Pointeri ca parametri și valoare returnată

- Declararea unei funcții care așteaptă un pointer la int

```
void f(int* px)
```

- Declararea unei funcții care returnează un pointer la char

```
char* f(void)
```

- Pointerii se transmit în mod normal (prin valoare) la funcții
- Dar se poate folosi adresa trimisă pentru a modifica valoarea unei variabile
- Nu returnați un pointer la o variabilă locală
  - Această variabilă este alocată pe stivă, iar la terminarea execuției corpului funcției nu se mai garantează accesul valid la zona respectivă de memorie

Pointeri - exemplu simplu

```c
#include <stdio.h>

void inter(int* pa, int* pb){
    int* t = pa;
    pa = pb;
    pb = t;
}

int main(){
    int a = 1, b = 2;
    int* pa = &a;
    int* pb = &b;
    printf("%d %d\n", a, b);
    inter(pa, pb);
    printf("%d %d\n", a, b);
    return 0;
}
```

- funcția NU interschimbă
- se modifică doar argumentele pa și pb care sunt copii locale

- se preiau adresele variablelor

- 1 2
- apel la interschimbare prin intermediul pointerilor
- 1 2

Pointeri - exemplu simplu

```c
#include <stdio.h>

void inter(int* pa, int* pb){
    int t = *pa;
    *pa = *pb;
    *pb = t;
}

int main(){
    int a = 1, b = 2;
    int* pa = &a;
    int* pb = &b;
    printf("%d %d\n", a, b);
    inter(pa, pb);
    printf("%d %d\n", a, b);
    return 0;
}
```

- funcția interschimbă două variabile
- este posibilă prin intermediul pointerilor
- se modifică conținuturile la care arată pointerii

- se preiau adresele variablelor

- 1 2
- apel la interschimbare prin intermediul pointerilor
- 2 1

Pointeri și tablouri

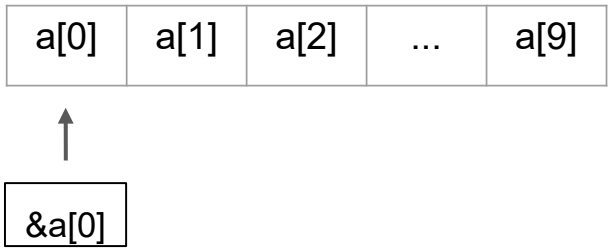
- Numele tabloului este automat convertit la un pointer constant la adresa primului element
- Nu se poate schimba unde arată acest pointer

```
int a[10];

int* p = a;

int a0 = *p;

a = p;        //nu se poate
```



Pointeri și tablouri

- Tablourile se trimit la funcții prin copierea adresei primului element
- Se pierde informația despre numărul de elemente din tablou
- Următoarele anteturi sunt echivalente
  - varianta 1 arată că a este tablou - dar nu se știe de câte elemente
    `void f(int a[])` sau `void f(int a[10])`
  - varianta 2 arată că se trimite o adresa - asta se întâmplă în realitate
    `void f(int* a)`
- Pentru tablouri bidimensionale numele tabloului este automat convertit la un pointer la un tablou unidimensional

```
int a[2][2];
int (*pa)[2] = a;
```

- pa este un pointer la un tablou cu 2 elemente

Operații cu pointeri

- Incrementare/decrementare ++/--
  - se modifică pointerul astfel încât să arate la adresa următoare, respectiv adresa anterioară
  - se ține cont de tipul pointerului
  - se modifică pointerul cu multipli de `sizeof(element)`

```
double a[100];
double* p;              // sizeof(double) = 8
p=&a[10];
printf("%p\n", p);  // 0028fc38 adresa elementului 10
p++;
printf("%p\n", p);  // 0028fc40 = 0028fc38 + 8 în hexa
```

Operații cu pointeri

- Adunare/scădere număr întreg *n*
  - rezultă în incrementarea respectiv decrementarea valorii pointerului cu *n* x numărul de octeți necesari pentru a memora o valoare de tipul la care arată pointerul

```
double a[100];
double* p;              // sizeof(double) = 8
p=&a[1];
printf("%p\n", p);  // 0028fc38  - adresa elementului 1
p = p - 1;
printf("%p\n", p);  // 0028fc30 = 0028fc38 - 8
p = p + 11;
printf("%p\n", p);  // 0028fc88 = 0028fc30 + 8*11
```

Operații cu pointeri

- Diferența a doi pointeri
  - Dacă ambii conțin adresele unor elemente din același tablou atunci diferența lor este este egală cu diferența pozițiilor
  - În general, este diferența dintre adrese `/ sizeof(element)`
- Compararea a doi pointeri
  - rezultatul este cel de la compararea adreselor

```
double a[100];
double* p = a+8;    // p referă la elementul a[8]
double* q = a+10;   // p referă la elementul a[10]
int dif = q-p;
printf("%p;%p;%d\n", p, q, dif); //0028fc20;0028fc30;2
printf("%d;%d\n", p > q, p+2 == q); //0;1
```

Indexare cu pointeri

- Folosind operațiile definite pe pointeri se pot accesa elementele dintr-un tablou
  - sintaxă alternativă și echivalentă indexării cu directe cu []
  - există situații când se preferă una din cele două modalități

`a[i]` echivalent cu `*(a+i)`

`&a[i]` echivalent cu `a+i`

Indexare cu pointeri - exemplu

```c
#include <stdio.h>

int sum(int* a, int n){
    int s = 0;
    while(n--){
        s += *a;
        a++;
    }
    return s;
}

int main(){
    int a[] = {1, 2, 3};
    int n = sizeof(a) / sizeof(a[0]);
    int s = sum(a, n);
    printf("%d %d\n", s, *(a+n-1));
    return 0;
}
```

- funcția însumează valorile din tablou
- buclă pentru parcurgerea lui *a*
- adunăm conținutul de la adresa *a* la *s*
- trecem la următorul element
- se declară și se inițializează tabloul
- dimensiunea se poate obține cu sizeof
- la funcție trebuie să trimitem dimensiunea
- 6 3 - suma și ultimul element

Pointeri constanți

- Declararea unui pointer constant

```
tip * const nume = init;
```

- Pointerul nume arată către o zonă de memorie și pointerul nu se poate schimba
  - numele unui tablou este pointer constant
- Se pot schimba valorile de la adresa la care arată

Pointeri la valori constante

- Declararea unui pointer la constante

```
const tip * nume = init;
```

- Pointerul nume arată către adresa unei variabile de tip care nu se poate schimba
  - șiruri de caractere declarate cu pointer și inițializator sunt pointeri la constante
- Se poate schimba la ce adresă arată
- Există și pointeri constanți la valori constante

```
const tip * const nume = init;
```

Pointeri la void

- Se utilizează când un pointer poate referi la orice tip

```
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

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

- Atribuiri corecte: `p = &x; p = &y;`
- Pentru dereferențiere trebuie mai întâi convertit la `(tip*)p` și apoi efectuată dereferențierea

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

Exemplu – conversie int* in char*

```
#include <stdio.h>

int main(){
    int x = ('c' << 24) + ('p' << 16)
          + ('3' << 8 ) + '<';
    char* pc = (char*) &x;
    printf("%c%c%c%c\n",
        pc[0], pc[1], pc[2], pc[3]);
    return 0;
}
```

- un int este reprezentat pe 4 bytes
- un char este reprezentat pe 1 byte
- interpretăm variabila x ca 4 caractere
- afișăm caracter cu caracter conținutul
- ordine little-endian - prima dată avem byte-ul cel mai puțin semnificativ

- `<3pc`