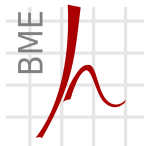


Struktúrák. Mutatók

A programozás alapjai I.



Hálózati Rendszerek és Szolgáltatások Tanszék
Farkas Balázs, Fiala Péter, Vitéz András, Zsóka Zoltán

2020. október 12.

Tartalom

1 Struktúrák

- Motiváció
- Definíció
- Értékadás

2 Típusnév-hozzárendelés

3 Mutatók

- Mutatók definíciója
- Cím szerinti paraméterátadás

1. fejezet

Struktúrák

Felhasználói típusok

A C nyelv beépített típusai nem, vagy csak körülményesen felelnek meg a bonyolultabb, összetettebb adatok tárolására.

Felhasználó (programozó) által bevezetett típusok

- Felsorolás
- Struktúra
- Bitmezők
- Unió

Felhasználói típusok

A C nyelv beépített típusai nem, vagy csak körülményesen felelnek meg a bonyolultabb, összetettebb adatok tárolására.

Felhasználó (programozó) által bevezetett típusok

- **Felsorolás** ← múlt héten erről volt szó
- **Struktúra** ← ma erről lesz szó
- Bitmezők
- Unió

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;  
2  int ho;  
3  int nap;
```

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;  
2  int ho;  
3  int nap;
```

■ Hallgatói adatok tárolása

```
1  char neptun[6];  
2  unsigned int kiszhpont;  
3  unsigned int hanyzasok;
```

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;
2  int ho;
3  int nap;
```



■ Hallgatói adatok tárolása

```
1  char neptun[6];
2  unsigned int kiszhpont;
3  unsigned int hanyzasok;
```

- Sakkjátzsma adatai
(világos játékos, sötét játékos,
mikor, hol, lépések, eredmény)

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;
2  int ho;
3  int nap;
```



■ Hallgatói adatok tárolása

```
1  char neptun[6];
2  unsigned int kiszhpont;
3  unsigned int hanyzasok;
```

- Sakkjátszma adatai
(világos játékos, sötét játékos,
mikor, hol, lépések, eredmény)

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;
2  int ho;
3  int nap;
```



■ Hallgatói adatok tárolása

```
1  char neptun[6];
2  unsigned int kiszhpont;
3  unsigned int hanyzasok;
```

- Sakkjátszma adatai
(világos játékos, sötét játékos,
mikor, hol, lépések, eredmény)
- Egy lépés adatai
(figura, honnan, hová)

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;
2  int ho;
3  int nap;
```



■ Hallgatói adatok tárolása

```
1  char neptun[6];
2  unsigned int kiszhpont;
3  unsigned int hanyzasok;
```

- Sakkjátszma adatai
(világos játékos, sötét játékos,
mikor, hol, **lépések**, eredmény)
- Egy lépés adatai
(figura, **honnan**, hová)

Logikailag összetartozó adatok

■ Dátum eltárolása

```
1  int ev;
2  int ho;
3  int nap;
```



■ Hallgatói adatok tárolása

```
1  char neptun[6];
2  unsigned int kiszhpont;
3  unsigned int hanyzasok;
```

- Sakkjátszma adatai
(világos játékos, sötét játékos,
mikor, hol, lépések, eredmény)
- Egy lépés adatai
(figura, honnan, hová)
- A tábla egy mezőjének adatai
(oszlop, sor)

Összetartozó adatok átadása

- Írjunk függvényt, amely 2D vektorok skalárszorzatát számítja!

```
1 double v_skalarszorzat(double x1, double y1,  
2                        double x2, double y2)  
3 {  
4     return x1*x2 + y1*y2;  
5 }
```

Hogyan adjuk át az összetartozó adatokat (x_1, y_1)?

A függvényparaméterek száma túl sok lehet

Összetartozó adatok átadása

- Írjunk függvényt, amely 2D vektorok skalárszorzatát számítja!

```
1 double v_skalarszorzat(double x1, double y1,  
2                        double x2, double y2)  
3 {  
4     return x1*x2 + y1*y2;  
5 }
```

Hogyan adjuk át az összetartozó adatokat (x_1, y_1) ?

A függvényparaméterek száma túl sok lehet

- Írjunk függvényt, amely két vektor különbségét számítja!

```
1 ?????? v_kulonbseg(double x1, double y1,  
2                  double x2, double y2)  
3 {  
4     ...  
5 }
```

Hogyan kapjuk vissza az összetartozó adatokat?

Egységbezárás (encapsulation)

Struktúra

logikailag egy egységet alkotó, akár különböző típusú adatokból álló, összetett adattípus

hallgató

neptun

kis zh pontok

hiányzások

- A részadatokat mezőknek vagy tagoknak hívjuk
- Egyetlen értékadással másolható
- Lehet függvény paramétere
- Lehet függvény visszatérési értéke
- A C nyelv leghatékonyabb típusa

Struktúrák C-ben

```
1 struct vektor { /* struktúra típusdefiníció */
2     double x; double y;
3 };
4
5 struct vektor v_kulonbseg(struct vektor a,
6                           struct vektor b) {
7     struct vektor c;
8     c.x = a.x - b.x;
9     c.y = a.y - b.y;
10    return c;
11 }
12
13 int main(void) {
14     struct vector v1, v2, v3;
15     v1.x = 1.0; v1.y = 2.0;
16     v2 = v1;
17     v3 = v_kulonbseg(v1, v2);
18     return 0;
19 }
```


Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1  /* egy dátumot tároló struktúra típus */  
2  struct datum {  
3      int ev;  
4      int ho;  
5      int nap;  
6  } d1, d2; /* két változó példány */
```

Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1  /* egy dátumot tároló struktúra típus */  
2  struct datum {  
3      int ev;  
4      int ho;  
5      int nap;  
6  } d1, d2; /* két változó példány */
```

Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1  /* egy dátumot tároló struktúra típus */  
2  struct datum {  
3      int ev;  
4      int ho;  
5      int nap;  
6  } d1, d2; /* két változó példány */
```

Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1  /* egy dátumot tároló struktúra típus */  
2  struct datum {  
3      int ev;  
4      int ho;  
5      int nap;  
6  } d1, d2; /* két változó példány */
```

Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1  /* egy dátumot tároló struktúra típus */  
2  struct datum {  
3      int ev;  
4      int ho;  
5      int nap;  
6  } d1, d2; /* két változó példány */
```

■ [<struktúra címke>]_{opt}

elhagyható olyan esetekben, amikor később nem hivatkozunk rá

Struktúrák szintaxisa

Struktúra definíciója

```
struct [<struktúra címke>]opt  
{<struktúra tag deklarációk>}  
[<változó azonosítók>]opt;
```

```
1 /* egy dátumot tároló struktúra típus */  
2 struct datum {  
3     int ev;  
4     int ho;  
5     int nap;  
6 } d1, d2; /* két változó példány */
```

- [<struktúra címke>]_{opt}
elhagyható olyan esetekben, amikor később nem hivatkozunk rá
- [<változó azonosítók>]_{opt}
struktúra típusú változók deklarációja

Struktúrák szintaxisa

Struktúra típus használata

■ Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

■ Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```


Struktúrák szintaxisa

Struktúra típus használata

■ Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

- Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

- Struktúra tagok elérése

<struktúra azonosító>.<tag azonosító>

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

- Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

- Struktúra tagok elérése

<struktúra azonosító>.<tag azonosító>

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

- Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

- Struktúra tagok elérése

<struktúra azonosító>.<tag azonosító>

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

■ Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

■ Struktúra tagok elérése

<struktúra azonosító>.<tag azonosító>

- Struktúrataggal mindaz megtehető, ami különálló változóval

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

Struktúrák szintaxisa

Struktúra típus használata

- Változók deklarációja

`struct` <struktúra címke> <változó azonosítók>;

- Struktúra tagok elérése

<struktúra azonosító>.<tag azonosító>

- Struktúrataggal mindaz megtehető, ami különálló változóval

```
1 struct datum d1, d2;  
2 d1.ev = 2012;  
3 d2.ev = d1.ev;  
4 scanf("%d", &d2.ho);
```

- A tömbökhöz hasonlóan struktúráknál is lehetséges a kezdetiérték-adás:

```
1 struct datum d3 = {2011, 5, 2};
```

Értékadás struktúráknál

- Struktúra típusú változó értéke (összes tagja) frissíthető **egyetlen** értékadással.

```
1 struct datum d3 = {2013, 10, 8}, d4;  
2 d4 = d3;
```

2. fejezet

Típusnév-hozzárendelés

Definíció

■ C-ben átkeresztelhetjük típusneveinket

```
1 typedef int cica;  
2  
3 cica main() {  
4     cica i = 3;  
5     int b = 2;  
6     return i;  
7 }
```

Definíció

- C-ben átkeresztelhetjük típusneveinket

```
1 typedef int cica;  
2  
3 cica main() {  
4     cica i = 3;  
5     int b = 2;  
6     return i;  
7 }
```

Típusnév-hozzárendelés

- A `typedef` egy álnevet rendel az adott típushoz.
- Nem hoz létre új típust, az álnévvel bevezetett változók típusa az eredeti típus marad.

Mire jó?

- Beszédesebb forráskód, jobban átlátható

```
1 typedef double voltage;  
2  
3 voltage V1 = 1.0;  
4 double c = 2.0;  
5 voltage V2 = c * V1;
```

Mire jó?

■ Beszédesebb forráskód, jobban átlátható

```
1 typedef long double voltage; /* pontosabb kell */  
2  
3 voltage V1 = 1.0;  
4 double c = 2.0;  
5 voltage V2 = c * V1;
```

■ Könnyen karbantartható

Mire jó?

■ Beszédesebb forráskód, jobban átlátható

```
1 typedef float voltage; /* kisebb kell */  
2  
3 voltage V1 = 1.0;  
4 double c = 2.0;  
5 voltage V2 = c * V1;
```

■ Könnyen karbantartható

Mire jó?

- Beszédesebb forráskód, jobban átlátható

```
1 typedef float voltage; /* kisebb kell */  
2  
3 voltage V1 = 1.0;  
4 double c = 2.0;  
5 voltage V2 = c * V1;
```

- Könnyen karbantartható
- Megszabadulhatunk többszavas típusneveinktől

```
1 typedef struct vektor vektor;
```

Vektoros feladat typedeffel

```
1 struct vektor { /* új struktúra típus */
2     double x; double y;
3 };
4 typedef struct vektor vektor; /* átnevezés */
5
6 vektor v_kulonbseg(vektor a, vektor b) {
7     vektor c;
8     c.x = a.x - b.x;
9     c.y = a.y - b.y;
10    return c;
11 }
12
13 int main(void) {
14     vektor v1, v2, v3;
15     v1.x = 1.0; v1.y = 2.0;
16     v2 = v1;
17     v3 = v_kulonbseg(v1, v2);
18     return 0;
19 }
```

Vektoros feladat typedeffel

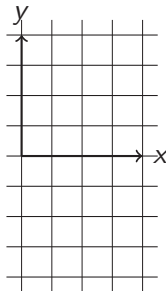
```
1  typedef struct vektor { /* egyszerre is megy */
2      double x; double y;
3  } vektor;
4
5
6  vektor v_kulonbseg(vektor a, vektor b) {
7      vektor c;
8      c.x = a.x - b.x;
9      c.y = a.y - b.y;
10     return c;
11 }
12
13 int main(void) {
14     vektor v1, v2, v3;
15     v1.x = 1.0;  v1.y = 2.0;
16     v2 = v1;
17     v3 = v_kulonbseg(v1, v2);
18     return 0;
19 }
```


Vektoros feladat typedeffel

```
1  typedef struct { /* ekkor a címke leghagyható */
2      double x; double y;
3  } vektor;
4
5
6  vektor v_kulonbseg(vektor a, vektor b) {
7      vektor c;
8      c.x = a.x - b.x;
9      c.y = a.y - b.y;
10     return c;
11 }
12
13 int main(void) {
14     vektor v1, v2, v3;
15     v1.x = 1.0; v1.y = 2.0;
16     v2 = v1;
17     v3 = v_kulonbseg(v1, v2);
18     return 0;
19 }
```

Egy összetettebb struktúra

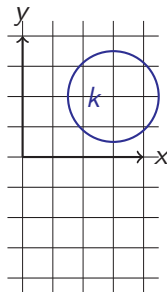
```
1 typedef struct {  
2     double x;  
3     double y;  
4 } vektor;  
5  
6 typedef struct {  
7     vektor kozep pont;  
8     double sugar;  
9 } kor;  
  
1 kor k = {{3.0, 2.0}, 1.5};  
2 vektor v = k.kozep pont;  
3 k.kozep pont.y = -2.0;
```



Egy összetettebb struktúra

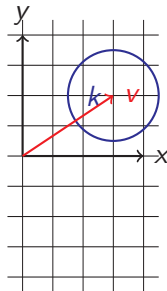
```
1 typedef struct {  
2     double x;  
3     double y;  
4 } vektor;  
5  
6 typedef struct {  
7     vektor kozeppont;  
8     double sugar;  
9 } kor;
```

```
1 kor k = {{3.0, 2.0}, 1.5};  
2 vektor v = k.kozeppont;  
3 k.kozeppont.y = -2.0;
```



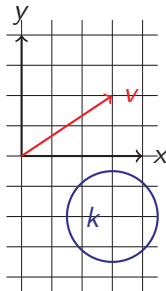
Egy összetettebb struktúra

```
1 typedef struct {  
2     double x;  
3     double y;  
4 } vektor;  
5  
6 typedef struct {  
7     vektor kozep pont;  
8     double sugar;  
9 } kor;  
  
1 kor k = {{3.0, 2.0}, 1.5};  
2 vektor v = k.kozep pont;  
3 k.kozep pont.y = -2.0;
```



Egy összetettebb struktúra

```
1 typedef struct {
2     double x;
3     double y;
4 } vektor;
5
6 typedef struct {
7     vektor kozep pont;
8     double sugar;
9 } kor;
10
11 kor k = {{3.0, 2.0}, 1.5};
12 vektor v = k.kozep pont;
13 k.kozep pont.y = -2.0;
```



3. fejezet

Mutatók

Fundamental Theorem of Software Engineering (FTSE)

*„We can solve any problem
by introducing an extra level of indirection.”*

Andrew Koenig

Hol vannak a változók?

Írjunk programot, mely kilistázza változók címét és értékét

```
1 int a = 2;  
2 double b = 8.0;  
3 printf("a címe: %p, értéke: %d\n", &a, a);  
4 printf("b címe: %p, értéke: %f\n", &b, b);
```

```
a címe: 0x7fffa3a4225c, értéke: 2  
b címe: 0x7fffa3a42250, értéke: 8.000000
```

¹általánosabban balérték

Hol vannak a változók?

Írjunk programot, mely kilistázza változók címét és értékét

```
1 int a = 2;  
2 double b = 8.0;  
3 printf("a címe: %p, értéke: %d\n", &a, a);  
4 printf("b címe: %p, értéke: %f\n", &b, b);
```

```
a címe: 0x7fffa3a4225c, értéke: 2  
b címe: 0x7fffa3a42250, értéke: 8.000000
```

- változó címe: a változót tartalmazó „memóriarekesz” kezdőcíme bájtokban mérve
- a címképzés operátorával tetszőleges változó ¹ címe képezhető &<balérték> formában

¹általánosabban balérték

A mutató típus

memóriacímek tárolására való

Mutató (pointer) deklarációja

```
<mutatott típus> * <azonosító>;
```

```
1 int*    p; /* p egy int adat címét tárolja */
2 double* q; /* q egy double adat címét tárolja */
3 char*   r; /* r egy char adat címét tárolja */
```

A mutató típus

memóriacímek tárolására való

Mutató (pointer) deklarációja

```
<mutatott típus> * <azonosító>;
```

```
1 int*    p; /* p egy int adat címét tárolja */
2 double* q; /* q egy double adat címét tárolja */
3 char*   r; /* r egy char adat címét tárolja */
```

másként tördelve is ugyanaz

```
1 int     *p; /* p egy int adat címét tárolja */
2 double *q; /* q egy double adat címét tárolja */
3 char   *r; /* r egy char adat címét tárolja */
```

Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”

Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: ?? 0x1000

b: ?? 0x1004

p: ????

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: ?? 0x1000

b: ?? 0x1004

p: ????

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: 2 0x1000

b: ?? 0x1004

p: ????

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: 2 0x1000

b: ?? 0x1004

p: ????

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: 2 0x1000

b: 3 0x1004

p: ????

Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```

a: 2 0x1000

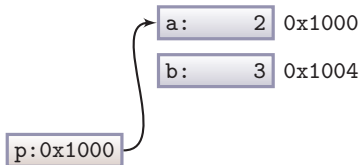
b: 3 0x1004

p: ????

Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

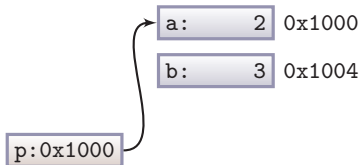
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a p mutató az a változó címét tartalmazza, akkor p „a-ra mutat”
- Ha p a-ra mutat, akkor az a változó *p-ként elérhető. Itt * az indirekció operátora (dereferencia operátor).

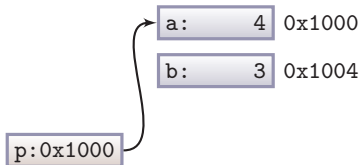
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

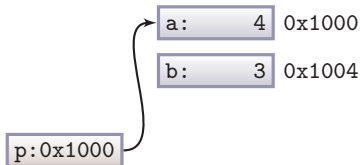
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

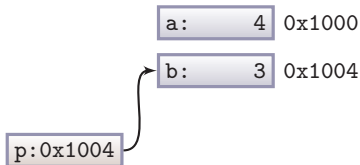
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

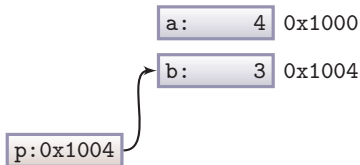
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

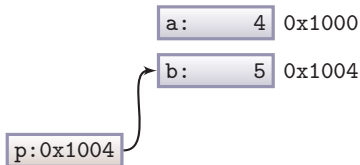
```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Az indirekció operátora

- Ha a `p` mutató az a változó címét tartalmazza, akkor `p` „a-ra mutat”
- Ha `p` a-ra mutat, akkor az a változó `*p`-ként elérhető. Itt `*` az indirekció operátora (dereferencia operátor).

```
1 int a, b;  
2 int *p; /* int pointer */  
3  
4 a = 2;  
5 b = 3;  
6 p = &a; /* p a-ra mutat */  
7 *p = 4; /* a = 4 */  
8 p = &b; /* p b-re mutat */  
9 *p = 5; /* b = 5 */
```



Címképzés és indirekció – összefoglalás

operátor	művelet	leírás
&	címképzés	változóhoz a címét rendeli
*	indirekció	címhez a változót rendeli

Címképzés és indirekció – összefoglalás

operátor	művelet	leírás
&	címképzés	változóhoz a címét rendeli
*	indirekció	címhez a változót rendeli

- Deklaráció értelmezése: `*p` `int` típusú

```
1 int *p;      /* ezt szokjuk meg */
```

Címképzés és indirekció – összefoglalás

operátor	művelet	leírás
&	címképzés	változóhoz a címét rendeli
*	indirekció	címhez a változót rendeli

- Deklaráció értelmezése: `*p int` típusú

```
1 int *p;      /* ezt szokjuk meg */
```

- Többszörös deklaráció: `a, *p` és `*q int` típusúak

```
1 int a, *p, *q; /* már csak ezért is */
```

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél */  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél */  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

b 0x1FFC:

3

a 0x2000:

2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

0x1FF0:	15
0x1FF4:	2
0x1FF8:	3
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }
6
7 void xchgp(int *px, int *py) {
8     int tmp = *px;
9     *px = *py;
10    *py = tmp;
11 }
12
13 int main(void) {
14     int a = 2, b = 3;
15     xchg(a, b);    /* nem cserél */
16     xchgp(&a, &b); /* cserél */
17     return 0;
18 }
```

0x1FF0:	15
x 0x1FF4:	2
y 0x1FF8:	3
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

tmp 0x1FEC:	2
0x1FF0:	15
x 0x1FF4:	2
y 0x1FF8:	3
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

tmp 0x1FEC:	2
0x1FF0:	15
x 0x1FF4:	3
y 0x1FF8:	3
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```

1 void xchg(int x, int y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }
6
7 void xchgp(int *px, int *py) {
8     int tmp = *px;
9     *px = *py;
10    *py = tmp;
11 }
12
13 int main(void) {
14     int a = 2, b = 3;
15     xchg(a, b);    /* nem cserél */
16     xchgp(&a, &b); /* cserél */
17     return 0;
18 }

```

tmp 0x1FEC:	2
0x1FF0:	15
x 0x1FF4:	3
y 0x1FF8:	2
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

0x1FF0:	15
x 0x1FF4:	3
y 0x1FF8:	2
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

0x1FF0:	15
0x1FF4:	3
0x1FF8:	2
b 0x1FFC:	3
a 0x2000:	2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél */  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

b 0x1FFC:

3

a 0x2000:

2

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

b 0x1FFC:

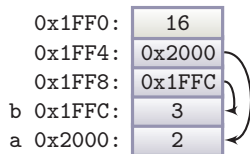
3

a 0x2000:

2

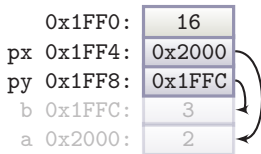
Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```



Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél */  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

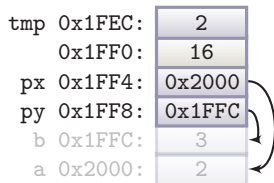


Alkalmazás – Függvény két változó cseréjére

```

1 void xchg(int x, int y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }
6
7 void xchgp(int *px, int *py) {
8     int tmp = *px;
9     *px = *py;
10    *py = tmp;
11 }
12
13 int main(void) {
14     int a = 2, b = 3;
15     xchg(a, b);    /* nem cserél */
16     xchgp(&a, &b); /* cserél */
17     return 0;
18 }

```

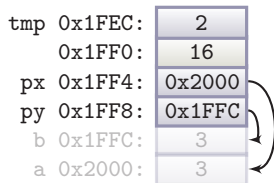


Alkalmazás – Függvény két változó cseréjére

```

1 void xchg(int x, int y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }
6
7 void xchgp(int *px, int *py) {
8     int tmp = *px;
9     *px = *py;
10    *py = tmp;
11 }
12
13 int main(void) {
14     int a = 2, b = 3;
15     xchg(a, b);    /* nem cserél */
16     xchgp(&a, &b); /* cserél */
17     return 0;
18 }

```

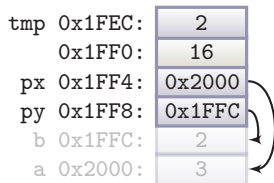


Alkalmazás – Függvény két változó cseréjére

```

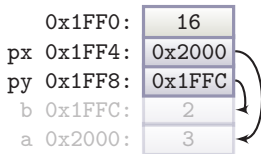
1 void xchg(int x, int y) {
2     int tmp = x;
3     x = y;
4     y = tmp;
5 }
6
7 void xchgp(int *px, int *py) {
8     int tmp = *px;
9     *px = *py;
10    *py = tmp;
11 }
12
13 int main(void) {
14     int a = 2, b = 3;
15     xchg(a, b);    /* nem cserél */
16     xchgp(&a, &b); /* cserél */
17     return 0;
18 }

```



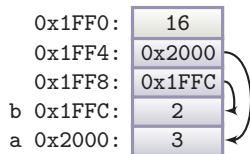
Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```



Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```



Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél*/  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

b 0x1FFC:

2

a 0x2000:

3

Alkalmazás – Függvény két változó cseréjére

```
1 void xchg(int x, int y) {  
2     int tmp = x;  
3     x = y;  
4     y = tmp;  
5 }  
6  
7 void xchgp(int *px, int *py) {  
8     int tmp = *px;  
9     *px = *py;  
10    *py = tmp;  
11 }  
12  
13 int main(void) {  
14     int a = 2, b = 3;  
15     xchg(a, b);    /* nem cserél */  
16     xchgp(&a, &b); /* cserél */  
17     return 0;  
18 }
```

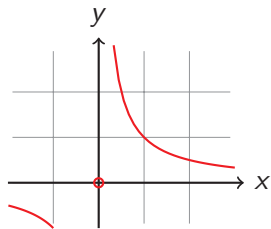

Alkalmazás – paraméterlistán visszaadott értékek

- Ha egy függvénynek több adatot kell kiszámolnia, akkor...
...alkalmazhatunk struktúrákat, de ez sokszor erőltetett.

Alkalmazás – paraméterlistán visszaadott értékek

- Ha egy függvénynek több adatot kell kiszámolnia, akkor...
...alkalmazhatunk struktúrákat, de ez sokszor erőltetett.
Inkább...

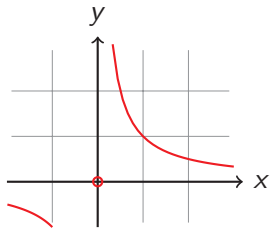
```
1 int inverse(double x, double *py)
2 {
3     if (abs(x) < 1e-10) return 0;
4     *py = 1.0 / x;
5     return 1;
6 }
```

[link](#)

Alkalmazás – paraméterlistán visszaadott értékek

- Ha egy függvénynek több adatot kell kiszámolnia, akkor...
...alkalmazhatunk struktúrákat, de ez sokszor erőltetett.
Inkább...

```
1 int inverse(double x, double *py)
2 {
3     if (abs(x) < 1e-10) return 0;
4     *py = 1.0 / x;
5     return 1;
6 }
```

[link](#)

```
1 double y;          /* helyfoglalás az eredménynek */
2 int success = inverse(5.0, &y);
3 if (success)
4     printf("%f reciproka %f\n", 5.0, y);
5 else
6     printf("Nem képezhető a reciprok");
```

[link](#)

Alkalmazás – paraméterlistán visszaadott értékek

- Most már értjük, mit jelent az, hogy

```
1 int n, p;  
2 /* paraméterlistán visszaadott értékek */  
3 scanf("%d%d", &n, &p); /* a címeket adjuk át */
```

Köszönöm a figyelmet.