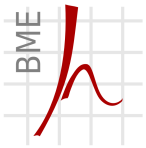
Struktúrák. Mutatók

A programozás alapjai I.



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

2020. október 12.

Tartalom

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 ← 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;
```

- Hallgatói adatok tárolása

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

- 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

- 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 */
```

- `[<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>;`
- 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  }
```

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 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;
```

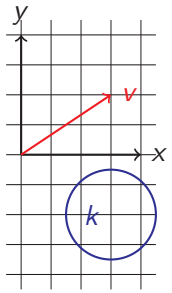

Vektoros feladat typedeffel

```
1  typedef struct { /* ekkor a címke lehagyható */
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 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;
```



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
```

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

[1] általánosabban 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    */
```

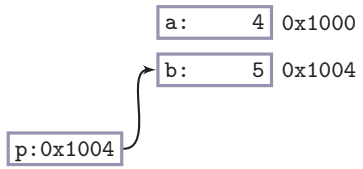
másként tördelve is ugyanaz

```
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”
- 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

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

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

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

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


Alkalmazás – Függvény két változó cseréjére

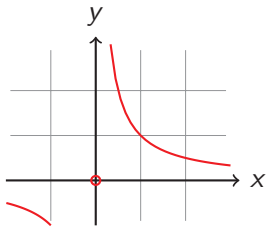
```
void xchg(int x, int y) {
  int tmp = x;
  x = y;
  y = tmp;
}

void xchgp(int *px, int *py) {
  int tmp = *px;
  *px = *py;
  *py = tmp;
}

int main(void) {
  int a = 2, b = 3;
  xchg(a, b);   /* nem cserél*/
  xchgp(&a, &b);/* cserél */
  return 0;
}
```


Alkalmazás – paraméterlistán visszaadott értékek

- Ha egy függvénynek több adatot kell kiszámolnia, akkor...

  ...alkalmazhatunk struktúrákat, de ez sokszor erőltetett. Inkább...

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




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


Alkalmazás – paraméterlistán visszaadott értékek

- Most már értjük, mit jelent az, hogy

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