Objektumok és függvények

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2023 október 10.

Konstruktorok

A **konstruktorok** olyan függvények, amelyek az objektum létrehozásakor elsőként lefutnak és inicializálják az elemeket

- a konstruktorok jellemzője, hogy a nevük megegyezik az objektum nevével
- a konstruktoroknak nincs visszatérési érték típusa
- egy objektumnak több konstruktora is lehet
- a konstruktort pl használhatjuk arra, hogy csak megengedett értékekkel inicializálhassunk egy objektumot
- inicializálatlan objektum lehetséges hibaforrás!

Példa konstruktorokra I

Default konstruktor

```
struct smallmatrix
 {
   double a11, a12, a21, a22;
   smallmatrix():a11{0.0}, a12{0.0},
                 a21{0.0}, a22{0.0}
     {
     }
   };
```

- `smallmatrix()`: a konstruktor függvény neve megegyezik az objektum nevével
- `smallmatrix()`: a konstruktornak most nincs argumentuma
- `:a11{0.0}, ...` inicializáló lista
- a konstruktor függvény teste most üres

Példa konstruktorokra I

Default konstruktor

```
struct smallmatrix
 {
   double a11, a12, a21, a22;
   smallmatrix():a11{0.0}, a12{0.0},
                 a21{0.0}, a22{0.0}
     {
     }
   };
```

- `smallmatrix()`: a konstruktor függvény neve megegyezik az objektum nevével
- `smallmatrix()`: a konstruktornak most nincs argumentuma
- `:a11{0.0}, ...` inicializáló lista
- a konstruktor függvény teste most üres

- A `default konstruktor` akkor hívódik meg, ha semmit nem írunk az objektum létrehozásakor

```
smallmatrix mtr1;
```

Példa konstruktorokra

Nem default konstruktor:

```
struct smallmatrix
 {
   double a11, a12, a21, a22;
   smallmatrix(double a11c, double a12c,
               double a21c, double a22c):
               a11{a11c}, a12{a12c}, a21{a12c}, a22{a22c}
     {
     }
   };
```

- `smallmatrix(...)` a konstruktor függvény argumentum listája most nem üres
- `:a11{a12c}, ...` inicializáló lista
- a konstruktor függvény teste most is üres, de elvileg itt is lehetnek utasítások
  - pl. lehet ellenőrzés arra, hogy megengedett értékekkel történik-e az inicializálás

Példa konstruktorokra

```
struct smallmatrix
 {
   double a11, a12, a21, a22;
   smallmatrix(double a11c, double a12c,
               double a21c, double a22c):
               a11{a11c}, a12{a12c}, a21{a12c}, a22{a22c}
     {
     }
 };
```

Hogyan használjuk?

```
smallmatrix mtr2 {0.0,0.0,0.0,0.0};
smallmatrix mtr2(0.0,0.0,0.0,0.0)

smallmatrix mtr3 {1.0,0.0,0.0,1.0};
smallmatrix mtr4 {0.0,1.0,1.0,0.0};
```

- az előbbi default konstruktor-os példával egyező működés
- az inicilazáló értékek megadhatóak `{..}` és `(..)` között is
- de lehet különböző értékekkel is inicializálni az objektumokat

Példa konstruktorokra

Egy objekumhoz akár több konstruktort is írhatunk:

```
struct smallmatrix
 {
   double a11, a12, a21, a22;

   // default konstruktor
   smallmatrix():a11{0.0}, a12{0.0}, a21{0.0}, a22{0.0}
   {}

   //konstruktor
   smallmatrix(double a11c, double a12c,
               double a21c, double a22c):
               a11{a11c}, a12{a12c}, a21{a12c}, a22{a22c}
     {
     }
   };
```

Objektumok tagfüggvényeinek helye a forráskódban

Eddig a következő példákat láttuk egy objektummal kapcsolatos függvényekre

- konstruktorok
- egyéb, az objektumhoz tartozó függvény (pl `En` a `particle` objektumban)

Ezeket a függvényeket az objektumon belül definiáltuk

Objektumok tagfüggvényeinek helye a forráskódban

Eddig a következő példákat láttuk egy objektummal kapcsolatos függvényekre

- konstruktorok
- egyéb, az objektumhoz tartozó függvény (pl `En` a `particle` objektumban)

Ezeket a függvényeket az objektumon belül definiáltuk

- a forráskód olvashatósága miatt azonban gyakran érdemes lehet egy objektum tagfüggvényeit az objektum törzsén kívül definiálni
- egyszerű szabály: ha egy függvény hosszabb, mint egy sor, akkor érdemes lehet máshol definiálni

Objektumok tagfüggvényeinek helye a forráskódban

Pl: egy `header` fájlban van a `smallmatrix` objektum definíciója:

```
struct smallmatrix
{
 double a11, a12, a21, a22;
 smallmatrix(double a11c, double a12c, double a21c, double a22c);
};
```

Hogyan jelezzük, hogy ehhez az objektumhoz tartozó `smallmatrix(...)` konstruktort akarunk definiálni valahol a program forráskódjában?

Objektumok tagfüggvényeinek helye a forráskódban

Pl: egy `header` fájlban van a `smallmatrix` objektum definíciója:

```
struct smallmatrix
{
 double a11, a12, a21, a22;
 smallmatrix(double a11c, double a12c, double a21c, double a22c);
};
```

Hogyan jelezzük, hogy ehhez az objektumhoz tartozó `smallmatrix(...)` konstruktort akarunk definiálni valahol a program forráskódjában?

```
smallmatrix::smallmatrix(double a11c, double a12c,
                         double a21c, double a22c):
                         a11{a11c}, a12{a12c}, a21{a12c}, a22{a22c}
    {
    }
```

- `smallmatrix::smallmatrix(...)` jelzi, hogy a `smallmatrix` objektumhoz tartozó `smallmatrix(...)` konstruktort definiáltuk
- hasonlóan lehet bármely tagfüggvény esetén eljárni, pl a `particle` objektum `En(...)` tagfüggvénye

Tagfüggvény és objektumot használó függvény

- eddig olyan függvényeket tekintettünk, amelyek szorosan hozzátartoznak az objektumhoz, pl konstruktorok
- előfordulhat, hogy egy adott objektumot használó függvényt az objektumtól függetlenül akarunk definiálni
- ilyenkor át kell neki adni az objektumot

Tekintsük a korábban már definiált `particle` objektumot!

```
struct particle
{
  double px, py;
  char szin;
  double En(double px, double py)
  { return px*px + py*py ;}
};
```

- Tfh az `En()` függvényt külön szeretnénk definiálni
- hogyan adhatjuk át neki a `struct`-t ?

Objektumot használó függvény

- definiáljuk az objektumot

```
struct particle
{
  double px, py;
  char szin;
};
```

- majd egy függvényt, amely az objektumot felhasználva kiszámolja a részecske energiáját

```
double En(const particle& prt)
{
  return prt.px*prt.px+prt.py*prt.py;
};
```

- `const` szó biztosítja, hogy az `En()` függvény véletlenül se írja felül az átadott objektum valamely tagját
- `particle& prt`: egy `particle` típusú objektumot adunk át referencia szerint
  - vagyis a függvényhíváshoz nem másoljuk át az objektumot a memória egyik részéből a másikba