

OO Programozás

Osztályok

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM085.git

2023. szeptember 3.

Struktúrák C-ben:

- + Logikailag összetartozó *adatok* egységbe zárása
- Függvények nem lehetnek tagok (de függvényeket címző mutatók igen)
- Minden adat mindenki számára hozzáférhető
- A *tagok* inicializálhatók, de el lehet feledkezni róla

Hogyan valósítanánk meg téglalapok kerületének és területének kiszámítását a régi eszközökkel?

Rectangle01.cpp

```
1 #include <iostream>
2
3 struct Rectangle { // type of data related to a concept
4     double width;
5     double height;
6 };
7
8 void initRect(Rectangle &r, double w, double h) {
9     r.width = w;
10    r.height = h;
11 }
```

Rectangle01.cpp

```
13 // weak connection between data and actions
14 double getRectArea(const Rectangle& r) {
15     return r.width * r.height;
16 }
17
18 double getRectPerimeter(const Rectangle& r) {
19     return 2. * (r.width + r.height);
20 }
21
22 void printRect(const Rectangle& r) {
23     std::cout << "Rectangle[dimensions:_"
24               << r.width << "_x_" << r.height << "]"
25               << std::endl;
26 }
```

Rectangle01.cpp

```
28 int main() {  
29     Rectangle r1;  
30     printRect(r1); // r1 is not initialized  
31     initRect(r1, 5., 3.);  
32     printRect(r1);  
33     r1.width = 4.5; // data is not hidden  
34     printRect(r1);  
35     std::cout << "Area: " << getRectArea(r1) << std::endl;  
36     std::cout << "Perimeter: " << getRectPerimeter(r1) << std::endl;  
37 }
```

Kimenet

```
Rectangle[dimensions: 6.95335e-310 x 0]  
Rectangle[dimensions: 5 x 3]  
Rectangle[dimensions: 4.5 x 3]  
Area: 13.5  
Perimeter: 15
```


Rectangle02.cpp

```
1 #include <iostream>
2
3 struct Rectangle {
4     private:
5         double mWidth; // m -> member, indicating private data
6         double mHeight;
7     public:
8         // it is not necessary to refer to r
9         void init(double width, double height) {
10             mWidth = width;
11             mHeight = height;
12         }
```

Rectangle02.cpp

```
14 // member function's name does not contain the name of the
15 // type (Rect[angle]); it is obvious because of *encapsulation*
16 // const -> the function cannot modify any data
17 double getArea() const {
18     // error: cannot assign to non-static data member within
19     // const member function 'getArea'
20     // mWidth = 1.;
21     return mWidth * mHeight;
22 }
23
24 double getPerimeter() const {
25     return 2. * (mWidth + mHeight);
26 }
```


Rectangle02.cpp

```
35 int main() {
36     Rectangle r1;
37     r1.init(5., 3.); // calling *member functions*
38     r1.print();
39     // error: 'mWidth' is a private member of 'Rectangle'
40     // r1.mWidth = 4.5; // data IS hidden
41     std::cout << "Area: " << r1.getArea() << std::endl;
42     std::cout << "Perimeter: " << r1.getPerimeter() << std::endl;
43 }
```

Kimenet

```
Rectangle[dimensions: 5 x 3]
Area: 15
Perimeter: 16
```

Osztályok (class) C++-ban:

- Új, felhasználói adattípust hoz létre
- Adatok és rajtuk végezhető műveletek *egységbe zárása* (encapsulation)
- Szigorúbb *adatrejtés*: `private` (alapértelmezett), `public` → hibák elkerülése
- Adatszerkezet és a megvalósítás részletei változtathatók, ameddig az interfész (nyilvános függvények) változatlan
- *Adattagok* és *tagfüggvények* tekintetében névteret képez

Osztály: „terv”, **nincs memórafoglalás!**

Objektum (példány): saját memóriaterület az adatokhoz, de osztoznak a függvényeken (mindig az aktuális példány adatain dolgoznak → `this`)

További lehetőségek:

- inline függvények: ha lehet, a fordító a kódot többször is beágyazza annak hívása helyett
- Tagfüggvények az osztályon belül (implicit inline) és kívül is definiálhatók

Rectangle03.cpp

```
3  class Rectangle {
4      // private:
5      // this is the default access modifier in case of a 'class'
6      double mWidth;
7      double mHeight;
8      public:
9      // member function defined outside the class;
10     // prototype, no parameter name needed
11     void init(double, double);
12
13     // member function defined implicitly 'inline'
14     double getArea() const {
15         return mWidth * mHeight;
16     }
```

Rectangle03.cpp

```
18      // explicit 'inline' member function -> see definition
19      double getPerimeter() const;
20
21      void print() const;
22  };
23
24  // type, scope resolution (::), member function name
25  void Rectangle::init(double width, double height) {
26      mWidth = width;
27      mHeight = height;
28  }
```

Rectangle03.cpp

```
30  // making a function *inline* is only a request, not a command
31  inline double Rectangle::getPerimeter() const {
32      return 2. * (mWidth + mHeight);
33  }
34
35  void Rectangle::print() const {
36      std::cout << "Rectangle[dimensions:_"
37                  << mWidth << "_x_" << mHeight << "]"
38                  << std::endl;
39  }
```

Rectangle03.cpp

```
41 int main() {  
42     Rectangle r1;  
43     r1.init(5., 3.);  
44     r1.print();  
45     std::cout << "Area:_" << r1.getArea() << std::endl;  
46     std::cout << "Perimeter:_" << r1.getPerimeter() << std::endl;  
47 }
```

Az inline függvények alternatívája C-ben: paraméteres makrók (ld. ctype).

macro-vs-inline.cpp

```
3 #define SQUARE(x) ((x)*(x))
4 #define ADD(x,y) ((x)+(y))
5
6 inline int add(int x, int y) {
7     return x + y;
8 }
9
10 int main(void) {
11     std::cout << "3^2=" << SQUARE(3) << std::endl;
12     std::cout << "(3+1)^2=" << SQUARE(3+1) << std::endl;
13     std::cout << "1+2=" << ADD(1, 2) << std::endl;           // ((1)+(2))=3
14     std::cout << "4*(1+2)=" << 4*ADD(1, 2) << std::endl;      // 4*((1)+(2))=12
15     std::cout << "4*(1+2)=" << 4*add(1, 2) << std::endl;      // 4*(1+2)=12
16     return 0;
17 }
```


Szempont	Inline fv.	Makro
Felhasználható	Rövid, gyakran	futó kódoknál
Feldolgozza	fordító	előfeldolgozó
Paraméterek ki- értékelése	egyszer	minden „hívásnál”
Kifejtés	nem garantált	garantált
Adattagok elérése	közvetlenül	csak közvetetten (nem tagja osztálynak)
Nyomkövetés	egyszerű	nehézkés
Típusok	fordító kikényszeríti a megfelelő típusú paramétereket; biztonság- os, de rugalmatlan → felültöl- tés, sablonok	nincs típusellenőrzés; rugalmas, de nem biztonságos

Hatókör feloldó operátor (::)

- Adott névtér elemeinek eléréséhez (pl. `std::cout`)
- Adott osztály tagfüggvényének külső definiálásakor (pl. `void Rectangle::print() {...}`)
- Azonos nevű változó által elfedett globális változó eléréséhez
- Statikus adattagok eléréséhez (ld. később)
- És még néhány további esetben

scope-resolution.cpp

```
1 #include <iostream>
2
3 int i = 1; // global 'i'
4
5 namespace A {
6     int i = 2; // global 'i' in namespace 'A'
7 }
8
9 int main() {
10     int i = 3; // local 'i'
11     // access global variable
12     std::cout << "global_i=" << ::i << std::endl;
13     // access global variable of namespace 'A'
14     std::cout << "A/global_i=" << A::i << std::endl;
15     std::cout << "local_i=" << i << std::endl;
16 }
```

Kimenet

```
global i=1
A/global i=2
local i=3
```

Nagyobb programoknál célszerű osztályonként két fájlt létrehozni:

- fejfájlt (.h, .hpp) a deklarációknak, inline függvényeknek, majd védeni többszörös beszerkesztés ellen
- forrásfájlt (.cpp, .cc, .cxx) a definícióknak

Gyorsabb fordítás, kisebb binárisok.

Rectangle04.h

```
1 #ifndef RECTANGLE_H
2 #define RECTANGLE_H
3 // https://en.cppreference.com/w/cpp/preprocessor/impl#23pragma\_once
4 // #pragma once
5
6 #include <iostream>
```

Rectangle04.h

```
8  class Rectangle {
9      double mWidth;
10     double mHeight;
11     public:
12         void init(double , double);
13
14         double getArea() const {
15             return mWidth * mHeight;
16         }
17
18         double getPerimeter() const;
19
20         void print() const;
21     };
```

Rectangle04.h

```
23 // inline functions are not included in the object file
24 // compiled from 'Rectangle.cpp'
25 inline double Rectangle::getPerimeter() const {
26     return 2. * (mWidth + mHeight);
27 }
28
29 #endif
```

Rectangle04.cpp

```
1  #include "Rectangle04.h"
2
3  void Rectangle::init(double width, double height) {
4      mWidth = width;
5      mHeight = height;
6  }
7
8  void Rectangle::print() const {
9      std::cout << "Rectangle[dimensions:_"
10                 << mWidth << "_x_" << mHeight << "]"
11                 << std::endl;
12 }
```

main04.cpp

```
1 #include "Rectangle04.h"
2 int main() {
3     Rectangle r1;
4     r1.init(5., 3.);
5     r1.print();
6     std::cout << "Area:_" << r1.getArea() << std::endl;
7     std::cout << "Perimeter:_" << r1.getPerimeter() << std::endl;
8 }
```


Régi probléma: inicializálás elfelejthető, megkerülhető → konstruktor
Neve egyezik az osztályéval, visszatérési érték/típus nincs.

Rectangle05.h

```
6  class Rectangle {  
7      double mWidth;  
8      double mHeight;  
9      public:  
10     Rectangle(double, double); // constructor
```

Rectangle05.cpp

```
3  Rectangle::Rectangle(double width, double height) {  
4      mWidth = width;  
5      mHeight = height;  
6  }
```

main05.cpp

```
2  int main() {  
3      // error: no matching constructor for initialization of 'Rectangle'  
4      // Rectangle r1;  
5      Rectangle r1(5., 3.);  
6      r1.print();  
7      std::cout << "Area:␣" << r1.getArea() << std::endl;  
8      std::cout << "Perimeter:␣" << r1.getPerimeter() << std::endl;  
9  }
```

Alapértelmezett (default) konstruktor:

- nem kapnak paramétereket, vagy mindegyiknek van alapértelmezett értéke
- adattagok alapértelmezett értékekkel feltöltésére, többnyire nullázásra
- néha létre *kell* hozni (pl. sok objektum dinamikus létrehozásához)
- ha nem készítünk konstruktort egy osztályhoz, akkor a fordító létrehoz egy alapértelmezettet (többnyire nem hasznos)
- hívásakor ne használjunk zárójelpárt (→ függvény, ami az osztály példányával tér vissza)

A konstruktorok felüldefiniálhatók.

Két konstruktor egy osztályban.

Rectangle06.h

```
6  class Rectangle {  
7      double mWidth;  
8      double mHeight;  
9  public:  
10     Rectangle(); // default constructor  
11     Rectangle(double, double);
```

Rectangle06.cpp

```
3 Rectangle::Rectangle() {  
4     mWidth = mHeight = 0.;  
5 }  
6  
7 Rectangle::Rectangle(double width, double height) {  
8     mWidth = width;  
9     mHeight = height;  
10 }
```

main06.cpp

```
2  int main() {
3      Rectangle r1(5., 3.);
4      r1.print();
5      std::cout << "Area:␣" << r1.getArea() << std::endl;
6      std::cout << "Perimeter:␣" << r1.getPerimeter() << std::endl;
7
8      Rectangle r2(-3.5, 10.); r2.print();
9  }
```

- 1 A konstruktor *még* nem ellenőrzi a megadott adatok helyességét.
- 2 A két konstruktor összevonható.

Rectangle07.h

```
6  class Rectangle {
7      double mWidth;
8      double mHeight;
9  public:
10     // default parameters
11     Rectangle(double=0., double=0.);
```

Rectangle07.cpp

```
4  Rectangle::Rectangle(double width, double height) {
5      // negative values are not allowed
6      mWidth = std::max(0., width);
7      mHeight = std::max(0., height);
8  }
```

main07.cpp

```
2  int main() {  
3      Rectangle r1(5., 3.);  
4      r1.print();  
5      std::cout << "Area: " << r1.getArea() << std::endl;  
6      std::cout << "Perimeter: " << r1.getPerimeter()  
7          << std::endl;  
8  
9      Rectangle r2(-3.5, 10.); r2.print();  
10     Rectangle r3(5.); r3.print();  
11     Rectangle r4; r4.print();  
12 }
```

Kimenet

```
Rectangle[dimensions: 5 x 3]  
Area: 15  
Perimeter: 16  
Rectangle[dimensions: 0 x 10]  
Rectangle[dimensions: 5 x 0]  
Rectangle[dimensions: 0 x 0]
```


Bővítések:

- adatok lekérdezése és ellenőrzött beállítása (getter, setter)
- azonos nevű paraméter elrejtí az adattagot → `this`

Rectangle08.h

```
7  class Rectangle {
8      double mWidth;
9      double mHeight;
10     public:

13         double getWidth() {
14             return mWidth;
15         }

16
17         double getHeight() {
18             // legal but useless
19             return this->mHeight;
20         }
```

Rectangle08.h

```
22     void setWidth(double width) {
23         mWidth = std::max(0., width);
24     }
25
26     // unusual and dangerous naming
27     void setHeight(double mHeight) {
28         // 'this' points to the current object
29         this->mHeight = std::max(0., mHeight);
30     }
```

main08.cpp

```
1 #include "Rectangle08.h"
2 int main() {
3     Rectangle r1;
4     r1.setWidth(5.);
5     r1.setHeight(3.);
6     std::cout << "Width:_" << r1.getWidth() << std::endl;
7     std::cout << "Height:_" << r1.getHeight() << std::endl;
8 }
```

Destruktorok:

- objektum megsemmisülésekor kerül meghívásra
- tipikusan a lefoglalt erőforrások felszabadítására használják
- neve `~` jellel kezdődik, az osztálynevével folytatódik
- nincs visszatérési értéke/típusa
- nem fogadhat paramétereket

Mikor hívják a konstruktorokat és destruktorokat?

Jelleg	Konstruktor	Destruktor
globális obj.	main előtt	main után
lokális obj.	vezérlés az obj. definíciójára kerül	vezérlés elhagyja a definiáló blokkot
lokális statikus obj.	vezérlés az obj. definíciójára kerül	program végén
dinamikus obj.	new hatására	delete hatására

Statikus tagok

- minden objektum osztozik ugyanazon az adaton
- `static` kulcsszóval hozható létre
- változó definiálása, inicializálása az osztályon kívül történik
- az osztály nevével és hatókör feloldó operátorral érhető el
- vagy egy statikus tagfüggvénnyel, ami csak statikus adattagokat érhet el és statikus tagfüggvényeket hívhat

Rectangle09.h

```
7  class Rectangle {
8      double mWidth;
9      double mHeight;
10     // 'static' variable is shared among objects;
11     // defined inside, initialized outside
12     static int count;
13     public:
14
40     static int getCount();
41
42     ~Rectangle();
43 };
```

Rectangle09.cpp

```
4  Rectangle::Rectangle(double width, double height) {
5      count++;
6      std::cout << "Rectangle_#" << count << "_created.\n";
7      mWidth = std::max(0., width);
8      mHeight = std::max(0., height);
9  }
10
11 Rectangle::~~Rectangle() {
12     std::cout << "Rectangle_#" << count << "_freed.\n";
13     count--;
14 }
15
16
17
18
19 // initializing static variable must be in source, not in header!
20 int Rectangle::count = 0;
21
22
23 int Rectangle::getCount() {
24     return count;
25 }
```


main09.cpp

```
1 #include "Rectangle09.h"
2 int main() {
3     Rectangle r1, r2, r3;
4     std::cout << "Number of rectangles: " << Rectangle::getCount() << std::endl;
5 }
```

Kimenet

```
Rectangle #1 created.
Rectangle #2 created.
Rectangle #3 created.
Number of rectangles: 3
Rectangle #3 freed.
Rectangle #2 freed.
Rectangle #1 freed.
```

Objektumtömbök létrehozása, memórfoglalás/felszabadítás

message.cpp

```
4  class Message {
5      private:
6          char* pStr;
7      public:
8          Message() {
9              pStr = new char( '\0' );
10             std::cout << "Created_[" << this << "]\n";
11         }
12
13         Message(const char* s) {
14             pStr = new char[ strlen(s) + 1 ];
15             strcpy( pStr, s );
16             std::cout << "Created_[" << this << ",_" << pStr << "]\n";
17         }
```

message.cpp

```
19 ~Message() {
20     std::cout << "Freed " << this << ", " << pStr << "]\n";
21     delete[] pStr;
22 }
23
24 void print() {
25     std::cout << pStr;
26 }
27
28 void setMessage(const char* s) {
29     delete[] pStr;
30     pStr = new char[strlen(s) + 1];
31     strcpy(pStr, s);
32 }
33 };
```

message.cpp

```
35  int main() {  
36      Message m1;  
37      m1.setMessage("Hello_C++_world!\n");  
38      m1.print();  
39  
40      Message* pm1 = new Message("Object_on_heap.");  
41      delete pm1;  
42  
43      Message aMessages[] = {"alpha", "beta"};  
44  
45      // using default constructor  
46      Message* pMessages = new Message[3];  
47      delete [] pMessages;  
48  }
```

Kimenet

```
Created [0x7ffe101e7718]
Hello C++ world!
Created [0xbfa2e0, Object on heap.]
Freed [0xbfa2e0, Object on heap.]
Created [0x7ffe101e7720, alpha]
Created [0x7ffe101e7728, beta]
Created [0xbfa328]
Created [0xbfa330]
Created [0xbfa338]
Freed [0xbfa338, ]
Freed [0xbfa330, ]
Freed [0xbfa328, ]
Freed [0x7ffe101e7728, beta]
Freed [0x7ffe101e7720, alpha]
Freed [0x7ffe101e7718, Hello C++ world!]
]
```