

Класове в C++ (продължение)

Любомир Чорбаджиев
Технологическо училище "Електронни системи"
Технически университет, София
lchorbadjiev@elsys-bg.org
Revision : 1.4

12 октомври 2004 г.

Конструктори

- Член-променливите не могат да се инициализират при тяхната дефиниция. За инициализиране на обектите от даден клас се използва специализирана член-функция, която се нарича **конструктор**.
- Името на конструктора съвпада с името на самият клас.

```
1 class Point {  
2     double x_, y_;  
3 public:  
4     Point(double x, double y); // конструктор  
5     // ...  
6 };
```

Конструктори

- Когато се създават обекти от даден клас, то за всеки създаден обект се извиква конструктор на класа, който инициализира обекта.
- Ако конструкторът има аргументи, то те трябва да се предадат при извикването му.

```
1 Point p1 = Point(1.0,1.0);
2 Point p2(2.0,2.0);
3 Point p3; // грешка
4 Point p4(4.0); // грешка
```

2

Конструктори

- Има възможност за един клас да се дефинират няколко конструктора.

```
1 class Point {
2 public:
3     Point(double x, double y);
4     Point(void);
5 };
```

- Кой конструктор ще се извика в даден конкретен случай се решава според аргументите, които се предадени при извикването на конструктора.

```
1 Point p1(1.0,1.0);
2 Point p2;
```

Забележка: В C++ е допустимо да се дефинират няколко функции с едно и също име, които се различават по броя и типа на аргументите. За да реши коя точно функция трябва се извика, компилаторът използва списъка на предадените аргументи.

3

Конструктор по подразбиране

- **Подразбиращ се конструктор (конструктор по подразбиране)** се нарича конструктор, който може да се извика без да му се предават аргументи.
- Ако програмистът не дефинира никакъв конструктор на класа, то компилаторът при необходимост ще генерира подразбиращ се конструктор.
- Генерираният от компилатора подразбиращ се конструктор извиква конструкторите по подразбиране на всички член-променливи на класа.

4

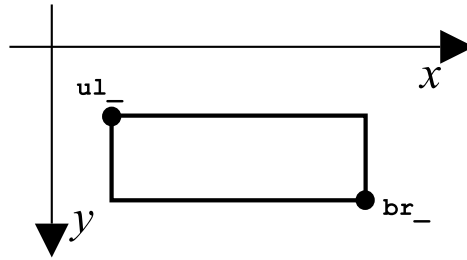
Конструктори: пример

```
1 class Point {  
2     double x_, y_;  
3 public:  
4     Point(double x, double y) {  
5         x_=x;  
6         y_=y;  
7     }  
8 };
```

```
1 class Point {  
2     double x_, y_;  
3 public:  
4     Point(double x, double y)  
5         :x_(x), y_(y)  
6     {}  
7 };
```

5

Конструктори: пример



```
1 class Rectangle {
2     Point ul_, br_;
3 public:
4     Rectangle(double x, double y, double w, double h)
5         : ul_(x, y), br_(x+w, y+h)
6     {}
7 };
```

```
1 class Rectangle {
2     Point ul_, br_;
3 public:
4     Rectangle(double x, double y, double w, double h){
5         ul_.set_x(x).set_y(y);
6         br_.set_x(x+w).set_y(y+h);
7     }
8 };
```

6

Копиращ конструктор

- Обектите могат да бъдат копирани.

```
1 Point p(1.0, 1.0);
2 Point p1=p; // копиране
```

- При създаване на обекта p1 се извиква специален конструктор (копиращ конструктор), който инициализира обекта p1 като копие на обекта p.
- За даден клас X, копиращият конструктор има вида X(const X&). За класа Point, копиращият конструктор има вида Point(const Point& p).
- Когато в даден клас програмистът не е дефинирал копиращ конструктор, компилаторът ще генерира такъв. Поведението на генерираният от компилатора копиращ конструктор е да копира всички член-променливи на класа.

7

Дефиниране на член-функции

```
1 class Point {
2     double x_, y_;
3 public:
4     Point(double x, double y);
5     Point& set_x(double x);
6     Point& set_y(double y);
7 };
8
9 Point::Point(double x, double y)
10    : x_(x), y_(y)
11 {}
12
13 Point& Point::set_x(double x) {
14     x_=x;
15     return *this;
16 }
17 Point& Point::set_y(double y) {
18     y_=y;
19     return *this;
20 }
```

8

Статични член-променливи

- Променлива, която е част от класа, но не е част от обектите на класа се нарича **статична** член-променлива.
- Статичните член-променливи имат само по **едно** копие, за разлика от нестатичните член-променливи.
- Статичните член-променливи на класа съществуват, независимо от това дали са създадени инстанции на класа. Поради това тяхната инициализация се различава от инициализацията на нестатичните член-променливи.

```
1 class DeepThought {
2     static int ANS;
3     //...
4 };
5 int DeepThought::ANS=-1;
```

9

Статични член-функции

- Член-функция, която не се свързва с обектите на класа, се нарича **статична член-функция**.

```
1 class DeepThought {
2     int foo_;
3 public:
4     static void find_the_answer(void);
5 };
```

- При извикването на статична член-функция, тя не се свързва с конкретна инстанция на класа. Поради това не може директно да се използват нестатични член-променливи.

```
1 void DeepThought::find_the_answer(void) {
2     foo_=8; // грешка!
3     ...
4 }
```

10

Статични член-функции

- Статичните член-функции имат пълен достъп до членовете на класа.

```
1 class DeepThought {
2     int foo_;
3 public:
4     static void find_the_answer(void) {
5         DeepThought some_thoughts;
6         some_thoughts.foo_=42; // OK!
7         ...
8     }
9 };
```

- За да се извика статична член-функция не е необходим обект от класа. Статичните член-функции могат да се викат директно, чрез името на класа в който са дефинирани.

```
1 DeepThought::find_the_answer();
```

11

Статични членове

```
1 #include <iostream>
2 using namespace std;
3 class DeepThought {
4 public:
5     static int ANSWER;
6     static void find_the_answer(void);
7 };
8
9 int DeepThought::ANSWER=-1;
10
11 void DeepThought::find_the_answer(void) {
12     // some deep calculations
13     ANSWER=42;
14 }
15
16 int main(void) {
17     DeepThought::find_the_answer();
18     cout << "The answer is:"
19          << DeepThought::ANSWER << endl;
20     return 0;
21 }
```

12

Константни член-функции

- За дефиниране на член-функция като константна се използва ключовата дума **const**.

```
1 class Foo {
2     int bar_;
3 public:
4     int get_bar(void) const { return bar_; }
5 };
```

Член-функцията `get_bar()`, дефинирана в ред 4, е константна член-функция.

- Когато една член-функция е дефинирана като константна, тя не може да променя състоянието на обекта, за който е извикана.

```
1 class Foo {
2     int bar_;
3 public:
4     void bar(void) const { bar_++; } // грешка
5 };
```

13

Константни член-функции

```
1 class Point {
2     double x_, y_;
3 public:
4     Point(double x, double y);
5     double get_x(void) const;
6     double get_y(void) const;
7     Point& set_x(double x);
8     Point& set_y(double y);
9 };
10
11 void fun(Point& rp, const Point& crp) {
12     rp.get_x();
13     crp.get_y();
14     rp.set_x(1.0);
15     crp.set_y(1.0); // грешка!
16 }
```

```
lubo@dobby:~/school/cpp/notes> g++ -c code/lecture04-point02.cpp
code/lecture04-point02.cpp: In function 'void fun(Point&, const Point&)':
code/lecture04-point02.cpp:15: error: passing 'const Point' as 'this' argument
of 'Point& Point::set_y(double)' discards qualifiers
```

14

Структури и класове

- В езика C++ структурите и класовете са тясно свързани. Съгласно определението структурата е клас, за който по подразбиране всички членове са публични. Следните две дефиниции са еквивалентни:

```
class s {
public:
    //...
};
```

```
struct s {
    //...
};
```

- В структурите е възможно да се дефинират член-функции, конструктори и т.н.

```
class Foo1 {
    int bar_;
public:
    Foo1(int bar);
    int get_bar(void);
};
```

```
struct Foo2 {
private:
    int bar_;
public:
    Foo2(int bar);
    int get_bar(void);
};
```

15

Вградени (inline) член-функции

- Когато една член-функция е дефинирана в тялото на класа, то тя се превръща във вградена (inline) член-функция — при обръщане към такава функция, в точката на извикване се вгражда дефиницията на функцията.
- Когато дефиницията на дадена член-функция е извън тялото на класа, за да се превърне тя във вградена член-функция трябва да се използва ключовата дума **inline**.

```
1 class Foo {
2     int bar_;
3 public:
4     int get_bar(void) const;
5 };
6 inline int Foo::get_bar(void) const {
7     return bar_;
8 }
```

16

Вложени класове

- Клас може да бъде дефиниран в рамките на друг клас. Такъв клас се нарича **вложен** клас.
- Дефиницията на вложен клас може да бъде направена в публичната, скритата или защитената секция на обграждащия клас.
- Името на вложеният клас се вижда в областта на действие на обграждащия клас и не се вижда в другите области на действие.

```
1 class Foo {
2 public:
3     class Bar { /* ... */ };
4 private:
5     Bar bar_;
6     // ...
7 };
8 Foo::Bar bar;
```

17