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
```


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.

Конструктор по подразбиране

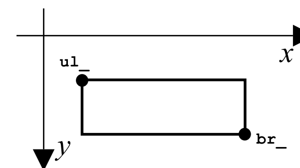
- **Подразбиращ се конструктор (конструктор по подразбиране)** се нарича конструктор, който може да се извика без да му се предават аргументи.
- Ако програмистът не дефинира никакъв конструктор на класа, то компилаторът при необходимост ще генерира подразбиращ се конструктор.
- Генерираният от компилатора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

Конструктори: пример

```
class Point {
  double x_, y_;
public:
  Point(double x, double y) {
    x_=x;
    y_=y;
  }
};
```

```
class Point {
  double x_, y_;
public:
  Point(double x, double y)
    :x_(x), y_(y)
  {}
};
```


Конструктори: пример




```
class Rectangle {
  Point ul_, br_;
public:
  Rectangle(double x,double y,double w,double h)
    : ul_(x,y), br_(x+w,y+h)
  {}
};
```

```
class Rectangle {
  Point ul_, br_;
public:
  Rectangle(double x,double y,double w,double h){
    ul_.set_x(x).set_y(y);
    br_.set_x(x+w).set_y(y+h);
  }
};
```


Копиращ конструктор

- Обектите могат да бъдат копирани.

```
Point p(1.0,1.0);
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 класа `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.

Дефиниране на член-функции

```
class Point {
  double x_, y_;
public:
  Point(double x, double y);
  Point& set_x(double x);
  Point& set_y(double y);
};

Point::Point(double x, double y)
  : x_(x), y_(y)
{}

Point& Point::set_x(double x) {
  x_=x;
  return *this;
}
Point& Point::set_y(double y) {
  y_=y;
  return *this;
}
```


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.

```
class DeepThought {
  static int ANS;
  //...
};
int DeepThought::ANS=-1;
```


Статични член-функции

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

```
class DeepThought {
  int foo_;
public:
  static void find_the_answer(void);
};
```

- 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.

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


Статични член-функции

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

```
class DeepThought {
  int foo_;
public:
  static void find_the_answer(void) {
    DeepThought some_thoughts;
    some_thoughts.foo_=42; // OK!
    ...
  }
};
```

- 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.

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

Статични членове

```
#include <iostream>
using namespace std;
class DeepThought {
public:
  static int ANSWER;
  static void find_the_answer(void);
};

int DeepThought::ANSWER=-1;

void DeepThought::find_the_answer(void) {
  // some deep calculations
  ANSWER=42;
}

int main(void) {
  DeepThought::find_the_answer();
  cout << "The answer is: "
       << DeepThought::ANSWER << endl;
  return 0;
}
```


Константни член-функции

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

```
class Foo {
  int bar_;
public:
  int get_bar(void) const { return bar_; }
};
```

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.

```
class Foo {
  int bar_;
public:
  void bar(void) const { bar_++; }// грешка
};
```


Константни член-функции

```
class Point {
  double x_, y_;
public:
  Point(double x, double y);
  double get_x(void) const;
  double get_y(void) const;
  Point& set_x(double x);
  Point& set_y(double y);
};

void fun(Point& rp, const Point& crp) {
  rp.get_x();
  crp.get_y();
  rp.set_x(1.0);
  crp.set_y(1.0); // грешка!
}
```

```
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
```


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);
};
```

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 де се превърне тя във вградена член-функция трябва да се използва ключовата дума **inline**.

```
class Foo {
  int bar_;
public:
  int get_bar(void) const;
};
inline int Foo::get_bar(void) const {
  return bar_;
}
```


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.

```
class Foo {
public:
  class Bar {/*...*/};
private:
  Bar bar_;
  //...
};
Foo::Bar bar;
```