Класове в C++ (продължение) (Rev: 775)

Любомир Чорбаджиев[1]
lchorbadjiev@elsys-bg.org

[1]Технологическо училище "Електронни системи"
Технически университет, София

22 ноември 2006 г.

Съдържание

1. **Константни член-функции и константни член-променливи**
   - Константи
   - Константни член-функции
   - Константни член-променливи
2. **Указател this**
3. **Статични членове на класа**
   - Статични член-променливи
   - Статични член-функции
4. **Вградени (inline) член-функции**
5. **Вложени класове**
6. **Примери**

Константи

- Към дефиницията на всяка променлива може да се прилага модификаторът **const**.
- Указва, че обекта не може да се променя.
- Води до грешка при компилация, в случай че се опитаме да променим **const** обект.
- Константите задължително трябва да се инициализират.
- Пример:

```
1 const Point origin(0.0,0.0);
2 origin.set_x(2.0); // грешка!
3 origin.get_x(); // грешка???
```

Указатели и константи

- При операциите с указатели участват два обекта – самият указател и обекта, към който сочи указателя.
- Когато ключовата дума **const** се постави пред дефиницията на указателя, това означава че константен е обекта към който сочи указателя.
- За де се декларира, че самият указател е константен, се използва ***const**, вместо *.

Указатели и константи

```
1  char str1[]="hello";
2  char str2[]="hell";
3
4  const char* pc=str1;
5  pc[2]='a';   // грешка!
6  pc=str2;     // OK!
7
8  char *const cp=str1;
9  cp[2]='a';   // OK!
10 cp=str2;     // грешка!
11
12 const char *const cpc=str1;
13 cpc[2]='a'; // грешка!
14 cpc=str2;   // грешка!
```

Препратки и константи

```
1 int a=6;
2 int& ra=a;
3 const int& cra=a;
4
5 const int ca=5;
6 const int& crca=ca; // OK
7 int& rca=ca; // грешка!!
```

Константни член-функции

- За да може една член-функция да се прилага към константен обект, компилатора трябва да е информиран, че тази член-функция не променя състоянието на обекта.
- За тази цел се използват **const** член-функции.
- Когато една член-функция е **const**, то в нейната дефиниция не може да се променя състоянието на обекта.
- За дефиниране на една член-функция като константна се използва ключовата дума **const**:
  - В прототипа на функция след списъка от параметри.
  - В дефиницията на функцията преди тялото на функцията.

Константни член-функции

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

```
1  class Point {
2    double x_, y_;
3  public:
4    ...
5    double get_x() const {
6      return x_;
7    }
8    double get_y() const;
9    ...
10 };
11 double Point::get_y() const {
12   return y_;
13 }
```

Константни член-функции

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

```
class Point {
  double x_,y_;
public:
  ...
  void set_x(double x) const {
    x_=x; // грешка!!
  }
  ...
};
```

Пример: Константни член-функции

```
1  class Point {
2    double x_, y_;
3  public:
4    Point(double x=0.0, double y=0.0)
5      : x_(x), y_(y)
6    {}
7    double get_x(void) const {return x_;}
8    double get_y(void) const {return y_;}
9    void set_x(double x) {x_=x;}
10   void set_y(double y) {y_=y;}
11 };
```

Пример: Константни член-функции

```
1  const Point origin;
2  ...
3  origin.get_x(); // OK!
4  origin.set_x(2.0); // грешка!!
5
6  void fun(Point& p1, const Point& p2) {
7    p1.get_x();
8    p2.get_y();
9    p1.set_x(1.0);
10   p2.set_y(1.0); // грешка!
11 }
```

Пример: Константни член-променливи

```
1  class Increment {
2    int count_;
3    const int step_;
4  public:
5    Increment(int c=0, int s=1)
6      : count_(c), step_(s)
7    {}
8    void step() {
9      count_+=step_;
10   }
11   void print() const {
12     cout << "count=" << count_
13          << ", step=" << step_ << endl;
14   }
15 };
```

Пример: Константни член-променливи

```
1 int main() {
2   Increment counter(7,7);
3   for(int i=0;i<5;i++) {
4     counter.step();
5     cout << "i=" << i << ", ";
6     counter.print();
7   }
8   return 0;
9 }
```

Пример: Константни член-променливи

```
lubo@kid ~/school/cpp/notes $ ./a.out
i=0, count=14, step=7
i=1, count=21, step=7
i=2, count=28, step=7
i=3, count=35, step=7
i=4, count=42, step=7
```

Указател this

- Всяка член-функция има достъп до допълнителен параметър — указателят **this**.
- Указателят **this** не е част от самия обект. Всяка нестатична член-функция получава този указател като допълнителен параметър.
- Типът на указателя **this** в различни член-функции е различен.
    - В константни член-функции **this** е константен указател към константен обект.
    - В неконстантни член-функции **this** е константен указател към неконстантен обекта.

Указател this

- В константната член-функция `get_x()` указателят **this** е от типа: **const** `Point*` **const**.
- В член-функцията `set_x()` указателят **this** е от типа: `Point*` **const**.

```
1 class Point {
2   ...
3 public:
4   double get_x() const {...}
5   void set_x() {...}
6   ...
7 };
```

Указател this

- Всяка член-функция притежава указател, който е насочен към обекта, за който тази член-функция е извикана. Това е указателят **this**.

```
class Point {
public:
  double get_x() {return this->x_;}
  ...
};
Point p1, p2;
p1.get_x();
p2.get_x();
```

- Указателя **this** може да се използва за обръщане към член-променливите, но това е излишно.

Указател this

- Има случаи, в които използването на указателя **this** е необходимо.
- Когато трябва да се реализира *каскадно* извикване на функции, използването на указателя **this** става наложително.

```
1 Point p;
2 p.set_x(1.0).set_y(1.0);
```

```
(p.set_x(1.0)).set_y(1.0);
```

- За да е възможно подобно поведение е необходимо методът `set_x()` да връща препратка към обекта, чрез който е извикан.

Указател this

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

Указател this

```
1 ...
2 Point p;
3 p.set_x(1.0).set_y(1.0);
4 ...
```

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;
```

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 };
```

Статични член-функции

- 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 }
```

- 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 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!
    ...
  }
};
```

Пример: 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 };
```

Пример: статични членове

```
1  int DeepThought::ANSWER=-1;
2  void DeepThought::find_the_answer(void) {
3    // some deep calculations
4    ANSWER=42;
5  }
6
7  int main(void) {
8    DeepThought::find_the_answer();
9    cout << "The answer is: "
10        << DeepThought::ANSWER << endl;
11   return 0;
12 }
```

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** .

Вградени (inline) член-функции

```
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 }
```

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 вложения клас се вижда в областта на действие на обграждащия клас и не се вижда в другите области на действие.

Вложени класове

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

Point.hpp

```
1 #ifndef POINT_HPP__
2 #define POINT_HPP__
3
4 class Point {
5   double x_, y_;
6 public:
7   Point(double x=0.0, double y=0.0)
8     : x_(x), y_(y)
9   {}
10  double get_x(void) const {return x_;}
11  double get_y(void) const {return y_;}
```

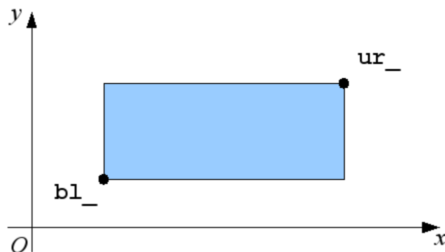
Point.hpp

```
12   Point& set_x(double x) {
13     x_=x;
14     return *this;
15   }
16   Point& set_y(double y) {
17     y_=y;
18     return *this;
19   }
20   void print() const;
21 };
22
23 #endif
```

Point.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 #include "Point.hpp"
5
6 void Point::print() const {
7   cout << "(" << x_ << ",␣" << y_ << ")";
8 }
```

Rectangle.hpp



Rectangle.hpp

```
1 #ifndef RECTANGLE_HPP_
2 #define RECTANGLE_HPP_
3
4 #include "Point.hpp"
5
6 class Rectangle {
7   Point bl_, ur_;
8
9   static double max(double a, double b) {
10    return a>b?a:b;
11  }
12  static double min(double a, double b) {
13    return a<b?a:b;
14  }
```

Rectangle.hpp

```
16 public:
17   Rectangle(const Point& p1,
18             const Point& p2);
19   double get_width() const;
20   double get_height() const;
21   double get_x() const;
22   double get_y() const;
23   const Point& get_ur() const;
24   const Point& get_bl() const;
25   void print() const;
26 };
27 #endif
```

Rectangle.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 #include "Rectangle.hpp"
5
6 Rectangle::Rectangle(const Point& p1,
7                      const Point& p2)
8   : bl_(min(p1.get_x(),p2.get_x()),
9         min(p1.get_y(),p2.get_y())),
10    ur_(max(p1.get_x(),p2.get_x()),
11        max(p1.get_y(),p2.get_y()))
12 {}
```

Rectangle.cpp

```
13
14 double Rectangle::get_x() const {
15   return bl_.get_x();
16 }
17 double Rectangle::get_y() const {
18   return bl_.get_y();
19 }
20 const Point& Rectangle::get_bl() const {
21   return bl_;
22 }
23 const Point& Rectangle::get_ur() const {
24   return ur_;
25 }
```

Rectangle.cpp

```
26 double Rectangle::get_width() const {
27   return ur_.get_x()-bl_.get_x();
28 }
29 double Rectangle::get_height() const {
30   return ur_.get_y()-bl_.get_y();
31 }
32 void Rectangle::print() const {
33   cout << "R{";
34   bl_.print();
35   cout << "; ";
36   ur_.print();
37   cout << "}";
38 }
```

Сечение и обединение на Rectangle