

Предефиниране на оператори (Rev: 1.1)

Любомир Чорбаджиев¹
lchorbadjiev@elsys-bg.org

22 февруари 2007 г.

Съдържание

Съдържание

1	Операции с вектори	1
1.1	Дефиниции	1
1.2	Примерна реализация	2
1.3	Пълен листинг	4
2	Предефиниране на оператори	6
2.1	Бинарни и унарни оператори	6
2.2	Бинарни оператори	7
2.3	Бинарен оператор като член-функция	7
2.4	Бинарен оператор като функция	8
2.5	Унарни оператори	9
2.6	Унарен оператор като член-функция	10
2.7	Унарен оператор като функция	11
2.8	Общи правила	12
2.9	Предефиниране на оператора за изход	12
2.10	Пълен листинг	13

1. Операции с вектори

1.1. Дефиниции

Пример: Операции с вектори

– Основните операции, които могат да се извършват с вектори са *събиране*, *изваждане* и *умножение по число*.

– Нека разгледаме вектори, дефинирани в равнината. Всеки вектор може да се представи като двойка числа $\vec{a} = (a_x, a_y)$, където a_x и a_y са съответно x и y -координатата на вектора \vec{a} .

Пример: Операции с вектори

– Нека са дадени два вектора $\vec{a} = (a_x, a_y)$ и $\vec{b} = (b_x, b_y)$. Операцията *събиране на вектори* дава нов вектор $\vec{c} = (c_x, c_y)$, такъв че:

$$c_x = a_x + b_x, c_y = a_y + b_y$$

– Нека са дадени два вектора $\vec{a} = (a_x, a_y)$ и $\vec{b} = (b_x, b_y)$. Операцията *изваждане на вектори* дава нов вектор $\vec{c} = (c_x, c_y)$, такъв че:

$$c_x = a_x - b_x, c_y = a_y - b_y$$

– Нека се дадени вектор $\vec{a} = (a_x, a_y)$ и число α . Операцията *умножение на вектор по число* дава нов вектор $\vec{b} = (b_x, b_y)$, такъв че:

$$b_x = \alpha a_x, b_y = \alpha a_y$$

1.2. Примерна реализация

Пример: Операции с вектори

– Нека дефинираме клас Point, който представя *вектор в равнината*.

```
4 class Point {
5     double x_, y_;
6 public:
7     Point(double x=0, double y=0)
8         : x_(x), y_(y)
9     {}
10    double get_x() const {return x_;}
11    double get_y() const {return y_;}
12
13    Point& add(const Point& p);
14    Point& sub(const Point& p);
15    Point& mul(double a);
16 };
```

Пример: Операции с вектори

- Методът `Point& add(const Point& p)` реализира операцията събиране на вектори.

```
18 Point& Point::add(const Point& p) {
19     x_+=p.x_;
20     y_+=p.y_;
21     return *this;
22 }
```

- Нека са дадени два вектора \vec{p}_1 и \vec{p}_2 . Операцията $\vec{p}_1 = \vec{p}_1 + \vec{p}_2$ може да се изпълни по следния начин:

```
Point p1, p2;
//....
p1.add(p2);
```

Пример: Операции с вектори

- Методът `Point& sub(const Point& p)` реализира операцията изваждане на вектори.

```
23 Point& Point::sub(const Point& p) {
24     x_-=p.x_;
25     y_-=p.y_;
26     return *this;
27 }
```

- Нека са дадени два вектора \vec{p}_1 и \vec{p}_2 . Операцията $\vec{p}_1 = \vec{p}_1 - \vec{p}_2$ може да се изпълни по следния начин:

```
Point p1, p2;
//....
p1.sub(p2);
```

Пример: Операции с вектори

- Методът `Point& mul(double alpha)` реализира операцията умножение на вектор по число.

```
28 Point& Point::mul(double alpha) {
29     x_*=alpha;
30     y_*=alpha;
31     return *this;
32 }
```

- Нека е даден вектор \vec{p} и числото α . Операцията $\vec{p} = \alpha\vec{p}$ може да се изпълни по следния начин:

```
Point p;
double alpha;
//....
p.mul(alpha);
```

Пример: Операции с вектори

- И трите разгледани метода връщат препратка към `Point`, като тази препратка препраща към обекта, върху който се изпълнява операцията (`*this`).
- Това позволява тези операции да се прилагат последователно (каскадно) върху даден обект:

```
1 Point p1, p2, p3;
2 //...
3 p1.add(p2).sub(p3).mul(10.0);
```

- Ред 3 е еквивалентен на следния код:

```
1 p1.add(p2);
2 p1.sub(p3);
3 p1.mul(10.0);
```

1.3. Пълен листинг

Пример: Операции с вектори

```
1 #include <iostream>
2 using namespace std;
3
4 class Point {
5     double x_, y_;
6 public:
7     Point(double x=0, double y=0)
8         : x_(x), y_(y)
9     {}
10    double get_x() const {return x_;}
11    double get_y() const {return y_;}
```

Пример: Операции с вектори

```
13 Point& add(const Point& p);
14 Point& sub(const Point& p);
15 Point& mul(double a);
16 };
17
18 Point& Point::add(const Point& p) {
19     x_+=p.x_;
20     y_+=p.y_;
21     return *this;
22 }
23 Point& Point::sub(const Point& p) {
24     x_-=p.x_;
25     y_-=p.y_;
26     return *this;
27 }
```

Пример: Операции с вектори

```
28 Point& Point::mul(double alpha) {
29     x_*=alpha;
30     y_*=alpha;
31     return *this;
32 }
33
34 int main(void) {
35     Point p1(1.0,1.0);
36     Point p2(2.0,2.0);
37     Point p3(3.0,3.0);
```

Пример: Операции с вектори

```
39 p3.add(p2).sub(p1).mul(10.0);
40
41 cout<<"p3=("
42     <<p3.get_x()<<" ,□"
43     <<p3.get_y()<<" "<<endl;
44     return 0;
45 }
```

```
lubo@kid:~/school/notes> ./a.out
p3=(40, 40)
```

2. Предефиниране на оператори

Предефиниране на оператори

- Представената реализация на векторна аритметика е удобна, но щеше да бъде много по удобна, ако можехме да използваме естествените математически оператори +, -, *, +=, -=, *=. Например:

```
1 Point p1, p2, p3;
2 //...
3 p1=p2+p3;
4 p1*=10.0;
5 p3-=p3;
```

- Една от важните концепции при създаването на езика C++ е, че класовете трябва да бъдат равноправни на вградените (примитивни) типове.

Предефиниране на оператори

- В езика C++ е предвидена възможност операторите да бъдат дефинирани за потребителските типове.
- Има само няколко оператора, които не могат да се предефинират от потребителя:
 - :: – оператор за избор на област на видимост;
 - . – оператор за избор на член;
 - .* – оператор за избор на член чрез указател към член;
 - sizeof – оператор за размер на обект;
 - typeid – оператор за идентификация на типа;
 - ?: – оператора за условен избор;
- Всички останали оператори могат да се предефинират.

2.1. Бинарни и унарни оператори

Бинарни и унарни оператори

- *Бинарен* оператор се нарича оператор, който действа върху два аргумента. *Унарен* е оператор, който действа върху един аргумент.
- Примери за бинарни оператори са операторите + (a+b), * (a*b), - (a-b), / (a/b) и т.н.

- Примери за унарни оператори са операторите `- (-a)`, `! (!a)`, `~ (~a)`, `++ (a++)` и т.н.
- Видът на оператора определя начините, по които той може да бъде предефиниран.

2.2. Бинарни оператори

Бинарни оператори

- Бинарните оператори могат да се дефинират по два начина:
 - Като нестатична член-функция на класа, която приема един аргумент – например:

```
Point Point::operator+(const Point& p)
```

- Като функция, която не е член на класа и приема два аргумента – например:

```
Point operator+(const Point& p1, const Point& p2)
```

2.3. Бинарен оператор като член-функция

Бинарни оператори

- Нека разгледаме първия вариант за предефиниране на бинарен оператор. За пример ще използваме класа `Point` и бинарния оператор за събиране:

```
1 #include <iostream>
2 using namespace std;
3
4 class Point {
5     double x_, y_;
6 public:
7     Point(double x=0, double y=0)
8         : x_(x), y_(y)
9     {}
```

Бинарни оператори

```
10 double get_x() const {return x_;}
11 double get_y() const {return y_;}
12 Point operator+(const Point& p) const;
13 };
```

```
14 Point Point::operator+(const Point& p) const {
15     Point result(get_x()+p.get_x(),
16                 get_y()+p.get_y());
17     return result;
18 }
```

Бинарни оператори

```
19 int main(void) {
20     Point p1(1.0,1.0), p2(2.0,2.0), p3;
21
22     p3=p1+p2;
23     cout<<"p3=("
24         <<p3.get_x()<<" ,□"
25         <<p3.get_y()<<" )"<<endl;
26     return 0;
27 }
```

Бинарни оператори

- Резултатът от изпълнението на тази програма е:

```
lubo@kid:~/school/notes> ./a.out
p3=(3, 3)
```

- Изразът в ред `??` е еквивалентен на следното:

```
p3=p1.operator+(p2);
```

2.4. Бинарен оператор като функция

Бинарни оператори

- Нека разгледаме втория вариант за предефиниране на бинарен оператор. Като пример отново използваме класа `Point`:

```
1 #include <iostream>
2 using namespace std;
3 class Point {
4     double x_, y_;
5 public:
6     Point(double x=0, double y=0)
7         : x_(x), y_(y)
8     {}
9     double get_x() const {return x_;}
10    double get_y() const {return y_;}
11 };
```

Бинарни оператори

```
12 Point operator+(const Point& p1, const Point& p2) {
13     Point result(p1.get_x()+p2.get_x(),
14                 p1.get_y()+p2.get_y());
15     return result;
16 }
17 int main(void) {
18     Point p1(1.0,1.0), p2(2.0,2.0), p3;
19     p3=p1+p2;
20
21     cout<<"p3=("
22           <<p3.get_x()<<" ,□"
23           <<p3.get_y()<<" )"<<endl;
24     return 0;
25 }
```

Бинарни оператори

- Резултатът от изпълнението на тази програма е:

```
lubo@kid:~/school/notes> ./a.out
p3=(3, 3)
```

- Изразът в ред ?? е еквивалентен на следното:

```
p3=operator+(p1, p2);
```

2.5. Унарни оператори

Унарни оператори

- Унарните оператори могат да се дефинират по два начина:
 - Като нестатична член-функция на класа, която не приема аргументи – например:

```
Point Point::operator-(void)
```

- Като функция, която не е член на класа и приема един аргумент – например:

```
Point operator-(const Point& p)
```

2.6. Унарен оператор като член-функция

Унарни оператори

- Нека разгледаме първия вариант за предефиниране на унарен оператор. За пример ще използваме класа Point и унарния оператор -:

```
1 #include <iostream>
2 using namespace std;
3
4 class Point {
5     double x_, y_;
6 public:
7     Point(double x=0, double y=0)
8         : x_(x), y_(y)
9     {}
```

Унарни оператори

```
10 double get_x() const {return x_;}
11 double get_y() const {return y_;}
12 Point operator-(void) const;
13 };
14 Point Point::operator-() const {
15     Point result(-get_x(),-get_y());
16     return result;
17 }
```

Унарни оператори

```
18 int main(void) {
19     Point p1(1.0,1.0), p2;
20
21     p2=-p1;
22     cout<<"p2=("
23           <<p2.get_x()<<" ,□"
24           <<p2.get_y()<<" )"<<endl;
25     return 0;
26 }
```

Унарни оператори

- Резултатът от изпълнението на тази програма е:

```
lubo@kid:~/school/notes> ./a.out
p2=(-1, -1)
```

– Изразът в ред ?? е еквивалентен на следното:

```
p2=p1.operator - ();
```

2.7. Унарнен оператор като функция

Унарни оператори

– Нека разгледаме втория вариант за предефиниране на унарнен оператор. Като пример отново ще използваме класа `Point` и унарния оператор `-`:

```
1 #include <iostream>
2 using namespace std;
3 class Point {
4     double x_, y_;
5 public:
6     Point(double x=0, double y=0)
7         : x_(x), y_(y)
8     {}
9     double get_x() const {return x_;}
10    double get_y() const {return y_;}
11 };
```

Унарни оператори

```
12 Point operator-(const Point& p) {
13     Point result(-p.get_x(), -p.get_y());
14     return result;
15 }
16 int main(void) {
17     Point p1(1.0, 1.0), p2;
18
19     p2=-p1;
20     cout<<"p2=("
21         <<p2.get_x()<<" ,□"
22         <<p2.get_y()<<" "<<endl;
23     return 0;
24 }
```

Унарни оператори

– Резултатът от изпълнението на тази програма е:

```
lubo@kid:~/school/notes> ./a.out
p2=(-1, -1)
```

– Изразът в ред ?? е еквивалентен на следното:

```
p2=operator -(p1);
```

2.8. Общи правила

Предефиниране на оператори

– Всеки оператор може да се дефинира само за синтаксиса, който е определен за него в спецификацията на езика. Например:

- Не може да се дефинира унарнен оператор за делене `/`, тъй като в спецификацията на езика този оператор е дефиниран като бинарен.
- Не може да се дефинира бинарен оператор за логическо отрицание `!`, тъй като в спецификацията на езика този оператор е дефиниран като унарнен.
- Операторът `-`, обаче, може да бъде предефиниран като унарнен и като бинарен оператор, тъй като в спецификацията на езика са дефинирани и двата варианта на оператора.

2.9. Предефиниране на оператора за изход

Предефиниране на оператора за изход `<<`

- Операторът за изход `<<` е бинарен оператор. Първият аргумент на оператора за изход задължително трябва да бъде от типа `ostream`.
- Типичният начин за предефиниране на оператора за изход е той да бъде дефиниран като функция извън рамките на класа по следният начин:

```
ostream& operator<<(ostream& out,
                    const Point& p);
```

Предефиниране на оператора за изход <<

```
1 ostream& operator<<(ostream& out,  
2     const Point& p) {  
3     out << "point(" << p.get_x() << ",␣"  
4     << p.get_y() << ")";  
5     return out;  
6 }
```

2.10. Пълен листинг

Пример: векторна аритметика

```
1 #include <iostream>  
2 using namespace std;  
3 class Point {  
4     double x_, y_;  
5 public:  
6     Point(double x=0, double y=0)  
7         : x_(x), y_(y)  
8     {}  
9     double get_x() const {return x_;}  
10    double get_y() const {return y_;}  
11    Point& operator+=(const Point& p);  
12    Point& operator-=(const Point& p);  
13    Point& operator*=(double alpha);  
14 };
```

Пример: векторна аритметика

```
15 Point& Point::operator+=(const Point& p) {  
16     x_+=p.get_x();  
17     y_+=p.get_y();  
18     return *this;  
19 }  
20 Point& Point::operator-=(const Point& p) {  
21     x_-=p.get_x();  
22     y_-=p.get_y();  
23     return *this;  
24 }  
25 Point& Point::operator*=(double alpha) {  
26     x_*=alpha;  
27     y_*=alpha;  
28     return *this;  
29 }
```

Пример: векторна аритметика

```
30 Point operator+(const Point& p1, const Point& p2) {  
31     Point result=p1;  
32     result+=p2;  
33     return result;  
34 }  
35 Point operator-(const Point& p1, const Point& p2) {  
36     Point result=p1;  
37     result-=p2;  
38     return result;  
39 }
```

Пример: векторна аритметика

```
40 Point operator*(const Point& p, double alpha) {  
41     Point result=p;  
42     result*=alpha;  
43     return result;  
44 }  
45 Point operator*(double alpha, const Point& p) {  
46     return p*alpha;  
47 }  
48 ostream& operator<<(ostream& out, const Point& p) {  
49     out << "point(" << p.get_x() << ",␣"  
50     << p.get_y() << ")";  
51     return out;  
52 }
```

Пример: векторна аритметика

```
53 int main(void) {  
54     Point p1(1.0,1.0), p2(2.0,2.0), p3;  
55     p3=p1+p2;  
56     cout<< "p3=" << p3 << endl;  
57     p3+=p1+p2;  
58     cout<< "p3=" << p3 << endl;  
59     p3=10.0*p1;  
60     cout<< "p3=" << p3 << endl;  
61     p3=p2*10.0;  
62     cout<< "p3=" << p3 << endl;  
63     return 0;  
64 }
```

Пример: векторна аритметика

```
lubo@kid:~/school/notes> ./a.out
p3=point(3, 3)
p3=point(6, 6)
p3=point(10, 10)
p3=point(20, 20)
```

Пример: масив с проверка на границите

```
1 #include <iostream>
2 #include <exception>
3
4 using namespace std;
5 class Array {
6     int* data_;
7     unsigned int size_;
8 public:
9     Array(unsigned int size=10)
10        : size_(size), data_(new int[size])
11    {}
12    ~Array(void) {
13        delete data_;
14    }
```

Пример: масив с проверка на границите

```
15 int& element(unsigned int index) {
16     if(index<0 || index>=size_) {
17         cerr << "index out of bounds..." << endl;
18         throw exception();
19     }
20     return data_[index];
21 }
22 unsigned size() const {
23     return size_;
24 }
25 };
```

Пример: масив с проверка на границите

```
26 int main(void) {
27     Array v(3);
28 }
```

```
29 for(int i=0;i<3;++i) {
30     v.element(i)=i;
31 }
32 for(int i=0;i<3;i++) {
33     cout << "v[i]=" << v.element(i) << endl;
34 }
35
36 return 0;
37 }
```

Пример: масив с проверка на границите

```
lubo@kid:~/school/notes> ./a.out
v[i]=0
v[i]=1
v[i]=2
```

Пример: масив с проверка на границите

```
1 #include <iostream>
2 #include <exception>
3 using namespace std;
4
5 class Array {
6     unsigned int size_;
7     int* data_;
8 public:
9     Array(unsigned int size=10)
10        : size_(size), data_(new int[size])
11    {}
12    ~Array(void) {
13        delete [] data_;
14    }
```

Пример: масив с проверка на границите

```
15 int& operator[](unsigned int index) {
16     if(index<0 || index>=size_) {
17         cerr << "index out of bounds..." << endl;
18         throw exception();
19     }
20     return data_[index];
21 }
```



```
22 unsigned size() const {
23     return size_;
24 }
25 };
```

Пример: масив с проверка на границите

```
26 int main(void) {
27     Array v(3);
28     for(int i=0;i<3;++i) {
29         v[i]=i;
30     }
31     for(int i=0;i<3;i++) {
32         cout << "v[i]=" << v[i] << endl;
33     }
34     try {
35         v[3]=5;
36     } catch(const exception& e) {
37         cout << "exception caught..." << endl;
38     }
39     return 0;
40 }
```

Пример: масив с проверка на границите

```
lubo@kid:~/school/notes> ./a.out
v[i]=0
v[i]=1
v[i]=2
```