Предефиниране на оператори

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

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

29 март 2009 г.

Съдържание

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:

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

Пример: Операции с вектори

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

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

Пример: Операции с вектори

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

```
1 Point& Point::add(const Point& p) {
2   x_+=p.x_;
3   y_+=p.y_;
4   return *this;
5 }
```

- 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.

```
1 Point& Point::sub(const Point& p) {
2   x_-=p.x_;
3   y_-=p.y_;
4   return *this;
5 }
```

- 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.

```
1 Point& Point::mul(double alpha) {
2   x_*=alpha;
3   y_*=alpha;
4   return *this;
5 }
```

- 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);
```

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

```
1   Point& add(const Point& p);
2   Point& sub(const Point& p);
3   Point& mul(double a);
4 };
5
6 Point& Point::add(const Point& p) {
7   x_+=p.x_;
8   y_+=p.y_;
9   return *this;
10 }
11 Point& Point::sub(const Point& p) {
12   x_-=p.x_;
13   y_-=p.y_;
14   return *this;
15 }
```

Пример: Операции с вектори

```
1  Point& Point::mul(double alpha) {
2    x_*=alpha;
3    y_*=alpha;
4    return *this;
5  }
6
7  int main(void) {
8    Point p1(1.0,1.0);
9    Point p2(2.0,2.0);
10   Point p3(3.0,3.0);
```

Пример: Операции с вектори

```
1   p3.add(p2).sub(p1).mul(10.0);
2
3   cout<<"p3=("
4       <<p3.get_x()<<",␣"
5       <<p3.get_y()<<")"<<endl;
6   return 0;
7 }
```

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

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.

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.

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)
```

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

```
1   double get_x() const {return x_;}
2   double get_y() const {return y_;}
3   Point operator+(const Point& p) const;
4 };
5 Point Point::operator+(const Point& p) const {
6   Point result(get_x()+p.get_x(),
7                get_y()+p.get_y());
8   return result;
9 }
```

Бинарни оператори

```
1 int main(void) {
2   Point p1(1.0,1.0), p2(2.0,2.0), p3;
3
4   p3=p1+p2;
5   cout<<"p3=("
6       <<p3.get_x()<<",␣"
7       <<p3.get_y()<<")"<<endl;
8   return 0;
9 }
```

- Изразът в ред 4 е еквивалентен на следното:

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

Бинарни оператори

- Резултатът от изпълнението на тази програма е:

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

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

```
1  Point operator+(const Point& p1, const Point& p2) {
2    Point result(p1.get_x()+p2.get_x(),
3                 p1.get_y()+p2.get_y());
4    return result;
5  }
6  int main(void) {
7    Point p1(1.0,1.0), p2(2.0,2.0), p3;
8    p3=p1+p2;
9
10   cout<<"p3=("
11       <<p3.get_x()<<",␣"
12       <<p3.get_y()<<")"<<endl;
13   return 0;
14 }
```

Бинарни оператори

- Изразът в ред 8 е еквивалентен на следното:

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

- Резултатът от изпълнението на тази програма е:

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

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)
```

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

```
1   double get_x() const {return x_;}
2   double get_y() const {return y_;}
3   Point operator-(void) const;
4 };
5 Point Point::operator-() const {
6   Point result(-get_x(),-get_y());
7   return result;
8 }
```

Унарни оператори

```
1 int main(void) {
2   Point p1(1.0,1.0), p2;
3
4   p2=-p1;
5   cout<<"p2=("
6       <<p2.get_x()<<",␣"
7       <<p2.get_y()<<")"<<endl;
8   return 0;
9 }
```

Унарни оператори

- Изразът в ред 4 е еквивалентен на следното:

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

- Резултатът от изпълнението на тази програма е:

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

Унарни оператори

- Нека разгледаме втория вариант за предефиниране на унарен 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

```
1  Point operator-(const Point& p) {
2    Point result(-p.get_x(),-p.get_y());
3    return result;
4  }
5  int main(void) {
6    Point p1(1.0,1.0), p2;
7
8    p2=-p1;
9    cout<<"p2=("
10       <<p2.get_x()<<",␣"
11       <<p2.get_y()<<")"<<endl;
12   return 0;
13 }
```

Унарни оператори

- Изразът в ред 8 е еквивалентен на следното:

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

- Резултатът от изпълнението на тази програма е:

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

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 унарен 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 унарен.
  - Операторът -, обаче, може да бъде предефиниран като унарен и като бинарен оператор, тъй като в спецификацията на езика са дефинирани и двата варианта на оператора.

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 }
```

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

```
1  Point& Point::operator+=(const Point& p) {
2    x_+=p.get_x();
3    y_+=p.get_y();
4    return *this;
5  }
6  Point& Point::operator-=(const Point& p) {
7    x_-=p.get_x();
8    y_-=p.get_y();
9    return *this;
10 }
11 Point& Point::operator*=(double alpha) {
12   x_*=alpha;
13   y_*=alpha;
14   return *this;
15 }
```

Пример: векторна аритметика

```
1 Point operator+(const Point& p1, const Point& p2) {
2   Point result=p1;
3   result+=p2;
4   return result;
5 }
6 Point operator-(const Point& p1, const Point& p2) {
7   Point result=p1;
8   result-=p2;
9   return result;
10 }
```

Пример: векторна аритметика

```
1  Point operator*(const Point& p, double alpha) {
2    Point result=p;
3    result*=alpha;
4    return result;
5  }
6  Point operator*(double alpha, const Point& p) {
7    return p*alpha;
8  }
9  ostream& operator<<(ostream& out, const Point& p) {
10   out << "point(" << p.get_x() << ",␣"
11       << p.get_y() << ")";
12   return out;
13 }
```

Пример: векторна аритметика

```
1  int main(void) {
2    Point p1(1.0,1.0), p2(2.0,2.0), p3;
3    p3=p1+p2;
4    cout<< "p3=" << p3 << endl;
5    p3+=p1+p2;
6    cout<< "p3=" << p3 << endl;
7    p3=10.0*p1;
8    cout<< "p3=" << p3 << endl;
9    p3=p2*10.0;
10   cout<< "p3=" << p3 << endl;
11   return 0;
12 }
```

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

```
1   int& element(unsigned int index) {
2     if(index>=size_) {
3       cerr << "index␣out␣of␣bounds..." << endl;
4       throw exception();
5     }
6     return data_[index];
7   }
8   unsigned size() const {
9     return size_;
10  }
11 };
```

Пример: масив с проверка на границите

```
1  int main(void) {
2    Array v(3);
3
4    for(int i=0;i<3;++i) {
5      v.element(i)=i;
6    }
7    for(int i=0;i<3;i++) {
8      cout << "v[i]=" << v.element(i) << endl;
9    }
10
11   return 0;
12 }
```

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

```
1   int& operator[](unsigned int index) {
2     if(index>=size_) {
3       cerr << "index␣out␣of␣bounds..." << endl;
4       throw exception();
5     }
6     return data_[index];
7   }
8   unsigned size() const {
9     return size_;
10  }
11 };
```

Пример: масив с проверка на границите

```
1  int main(void) {
2    Array v(3);
3    for(int i=0;i<3;++i) {
4      v[i]=i;
5    }
6    for(int i=0;i<3;i++) {
7      cout << "v[i]=" << v[i] << endl;
8    }
9    try {
10     v[3]=5;
11   } catch(const exception& e) {
12     cout << "exception␣catched..." << endl;
13   }
14   return 0;
15 }
```

Пример: масив с проверка на границите

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