

Перегрузка операций

2013

- ▶ Иногда возникает потребность описывать и применять к созданным программистом типам данных операции, по смыслу эквивалентные уже имеющимся в языке

- ▶ Перегрузка операций предполагает введение в язык двух взаимосвязанных особенностей:
 - возможности объявлять в одной области видимости несколько процедур или функций с одинаковыми именами и
 - возможности описывать собственные реализации операторов (то есть знаков операций, обычно записываемых в инфиксной нотации, между операндами)

- ▶ Перегружать операции можно только для определенных пользователем типов, т.е. для классов (операции для стандартных типов перегружать нельзя)
- ▶ Определение перегруженной операции очень похоже на определение функции с именем `operator@`, где `@` – идентификатор перегружаемой операции

Методы класса

Complex operator+(const Complex & c);

Использование

Complex a,b,c;

...

c = a.operator+(b); // обычный вызов метода

c = a + b; // что дает перегрузка

Дружественные функции

```
friend Complex operator+(const Complex& c1,  
                          const Complex& c2);
```

Использование

```
Complex a,b,c;
```

```
...
```

```
c = operator+(a,b); // обычный вызов метода
```

```
c = a + b;         // что дает перегрузка
```

Пример

```
Array Array::operator + (const Array &B){  
    if (n != B.n) return *this;  
    Array C(n);    //можно: Array C(n,0);  
    for (int i=0; i<n; i++)  
        C.A[i]=A[i]+B.A[i];  
    return C;  
}
```

▶ Проверяем

Array A(5,1), B(5,3), C;

print (A); // 1 1 1 1 1

print(B); // 3 3 3 3 3

C = A+B;

print(C); // мусор из памяти – почему?

// нужна перегрузка операции =


```
Array& Array::operator = (const Array &B) {  
    if (this != &B) { // здесь это опасней чем в equal  
        delete[ ] A;  
        n=B.n;  
        A=new int[n];  
        for (int i=0; i<n; i++)  
            A[i]=B.A[i];  
    }  
    return *this;  
}
```

Правила перегрузки

- ▶ Только для пользовательских типов (классов)
- ▶ Нельзя менять количество операндов (арность операции)
- ▶ Невозможно изменить приоритет
- ▶ Нельзя создавать операции, которых нет в синтаксисе языка
- ▶ Для некоторых операций перегрузка запрещена (`sizeof`, `.`, `::`, `?:` и пр.)
- ▶ Не рекомендуется сильно изменять семантику (операция `<<` – плохой пример)

Правила перегрузки

- ▶ Если операция унарная, то ее можно перегрузить или методом класса без параметров, или дружественной функцией с одним параметром
- ▶ например, унарный минус

Complex operator-();

friend Complex operator – (const Complex& c1);

Правила перегрузки

- ▶ Если операция бинарная, то ее можно перегрузить или методом класса с одним параметром, или дружественной функцией с двумя параметрами
- ▶ например, сложение

```
Complex operator+(const Complex & c);
```

```
friend Complex operator+(const Complex& c1,  
                          const Complex& c2);
```

Правила перегрузки

- ▶ Следующие операции можно перегружать только в классе (методом класса)

присваивание	=
вызов функции	()
индексация	[]
доступ через указатель	->

Правила перегрузки

- ▶ Только дружественной функцией можно перегружать бинарные операции, у которых левый операнд не является объектом данного класса
- ▶ Например

```
friend Complex operator+(double c1,  
                          const Complex& c2);
```

Правила для отдельных операций

- ▶ операция присваивания (всегда метод класса)
- ▶ по умолчанию реализуется побитовым копированием
- ▶ требуется перегрузка для динамических объектов

```
Array& Array::operator = (const Array &B) {  
    if (this != &B) {  
        delete[ ] A;  
        n=B.n;  
        A=new int[n];  
        for (int i=0; i<n; i++)  
            A[i]=B.A[i];  
    }  
    return *this;  
}
```


Правила для отдельных операций

- ▶ операция индексирования (всегда метод класса)
- ▶ должна возвращать ссылку, чтобы можно было использовать и для доступа по индексу и для изменения по индексу

```
int& Array::operator [ ] ( int i ) {  
    return A [ i ];  
}
```

```
Array mass(2);  
mass[0] = 10; cin >> mass[1];  
cout << mass[0] << " " << mass[1];
```

Правила для отдельных операций

- ▶ При перегрузке операции инкремента (декремента) для получения возможности использования и префиксной, и постфиксной форм, каждая из этих двух перегруженных функций-операций должна иметь разную сигнатуру

++ для класса Timer

- ▶ Когда компилятор встречает выражение с префиксным инкрементом ++t генерируется вызов функции-элемента `t.operator++()`;
- ▶ объявление должно иметь вид:

```
Timer operator++();
```

++ для класса Timer

- ▶ По соглашению, принятому в C++, когда компилятор встречает выражение постфиксной формы инкремента

`t++`

он генерирует вызов функции

`t.operator++(0);`

- ▶ объявлением является:

`Time operator++(int);`

++ для класса Timer

```
Timer Timer :: operator++( ) {  
    Increment();  
    return *this;  
}
```

```
Timer Timer :: operator++( int ) {  
    // фиктивный целый параметр не имеет имени  
    Timer temp (*this); //сохраняем для возврата  
    Increment();  
    return temp;  
}
```

```
// Функция-утилита увеличения времени на 1
// секунду. Может быть private
void Time::Increment() {
    second++;
    if (second==60) {
        second=0;
        minute++;
    }
    if (minute==60) {
        minute=0;
        hour++;
    }
    if (hour==24) hour=0;
}
```

Операция << для вывода в поток

```
ostream & operator<<
    (ostream & out, const Array &mas)
{
    for (int i=0; i<mas.n; i++)
        out << mas.A[i] << ' ';
    return out;
}
```

```
Array vect(5);
cout<<"элементы массива ="<<vect<<endl;
```

Преобразование типов

- ▶ Операции преобразования между встроенными типами и типами, определенными пользователями, или только между типами, определенными пользователями, требуют определения

Преобразование типов

- ▶ Конструктор с единственным аргументом может быть использован для преобразования объектов разных типов (включая встроенные типы) в объекты данного класса

```
Timer ( int h) :
```

```
    hour(h) { minute=0; second=0;}
```

```
Timer t=12; // преобразуем
```

- ▶ *Операция преобразования* (называемая также *операцией приведения*) может быть использована для преобразования объекта одного класса в объект другого класса или в объект встроенного типа.
- ▶ Операция преобразования этого вида не может быть дружественной функцией.
- ▶ Перегруженная функция–операция приведения не указывает тип возвращаемой величины, потому что им является тип, к которому преобразован объект.

```
operator int() const;
```

```
Timer : : operator int( ) const {  
    return ( hour*60+minute) * 60 + second;  
}
```

```
Timer less (12,10,54);
```

```
int timeSec = less; // timeSec= (int) less;
```

- ▶ Одной из особенностей операций приведения и конструкторов преобразований является то, что при необходимости компилятор может вызывать эти функции автоматически для создания временных объектов

Операция вызова функции

```
Array Array :: operator( ) (int pos , int len)
{
  //проверка
  assert( pos >= 0 && pos < n &&
          len>0 && pos+len <n);
  Array subAr (len+1);
  for (int i=0; i<len; ++i)
    subAr.A[i] = A[pos+i];
  return subAr;
}
```

```
Array a1(10);  
for (int i=0; i<10 ++i)  
    a1[ i ] = i;  
Array a2(5);  
a2=a1(3,5);  
cout <<a2<<endl; //4 5 6 7 8
```