

Лекция 7

*ПЕРЕГРУЗКА ФУНКЦИЙ.
ПЕРЕГРУЗКА ОПЕРАТОРОВ.*

Перегрузка функций

Статический полиморфизм реализуется с помощью перегрузки функций и операций.

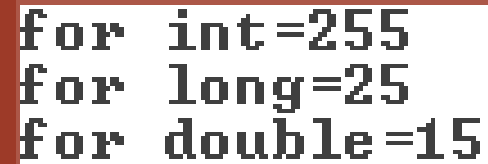
Под **перегрузкой функций** в C++ понимается описание в одной области видимости нескольких функций с *одним и тем же именем*.

Перегрузить **можно** только функции, которые отличаются либо типом, либо количеством аргументов. Перегрузить функции, отличающиеся только типом возвращаемого значения **нельзя**.

Перегрузка функций

Пример:

```
int abs(int n);  
long abs(long l);  
double abs(double n);  
main(){  
    int n=255; long l=-25L; double d=-15.0L;  
    cout<<"for int="<< abs(n)<<endl;  
    cout<<"for long="<<abs(l)<<endl;  
    cout<<"for double="<<abs(d)<<endl;  
}  
int abs(int n){ return n<0 ? -n : n; }  
long abs(long l){ return l<0 ? -l : l; }  
double abs(double d){ return d<0 ? -d : d;}
```



```
for int=255  
for long=25  
for double=15
```

Перегрузка функций

Компилятор автоматически выберет правильный вариант вызова на основании числа и типа используемых аргументов. В данном примере осуществляется перегрузка функции **abs()**. В зависимости от переданного аргумента вызывается нужный вариант функции.

Рекомендация: не путайте **переопределение** функции с **перегрузкой**: если в производном классе объявлена функция с тем же именем, что и функция в базовом классе, функция из производного класса скрывает функцию из базового класса вместо того, чтобы осуществлять перегрузку (так как их области видимости различны), то говорят о **переопределении** функции.

Мы уже рассматривали перегруженные функции — это конструкторы (при описании класса использовались несколько видов конструкторов: по умолчанию и с параметрами, т.е. функции с одним именем, но разными параметрами).

Перегрузка операций (операторов)

Перегрузка операций (операторов) дает возможность использовать собственные типы данных точно так же, как стандартные (т.е. можно над собственными данными определить собственные операции).

О перегрузке операций в C++ говорят в том случае, если в некоторой области видимости появляется описание функции с именем **operator<операция>**.

Обозначения собственных операций вводить **нельзя**.

Можно перегружать любые операции, существующие в C++, за исключением:

- . – выбор элемента;
- .* – выбор элемента по указателю;
- :: – оператор расширения области видимости;
- ?: – оператор условия;
- # – препроцессорный символ;
- ## – препроцессорный символ.

Перегрузка операций

Перегрузка операций подчиняется следующим правилам:

- при перегрузке операций сохраняются количество аргументов, приоритеты операций;
- для стандартных типов данных **нельзя** переопределять операции;
- функции-операции **не могут** иметь аргументов по умолчанию;
- функции-операции **наследуются** (за исключением =);
- функции-операции не могут определяться как **static**.

Функцию-операцию можно определить тремя способами, она должна быть *либо методом класса, либо дружественной функцией класса, либо обычной функцией*. В двух последних случаях функция должна принимать хотя бы один аргумент, имеющий тип класса, указателя или ссылки на класс.

Перегрузка операций

Функция-операция содержит ключевое слово **operator**, за которым следует знак переопределяемой операции:

тип operator операция (список параметров)

{ тело функции }

где *тип* — это тип возвращаемого операцией значения,
а *операция* — символьное обозначение перегруженной операции.

Функции-операторы могут вызываться таким же образом, как и любая другая функция. Использование операции — это лишь сокращенная запись явного вызова функции-оператора, например, для комплексных чисел сокращенная запись **c = a + b**; эквивалентна вызову функции **c = operator+(a, b)**;

Перегрузка операций

В отношении действий, выполняемых перегружаемыми операциями, C++ не накладывает никаких ограничений. Можно, например, перегрузить операцию "+" для выполнения вычитания комплексных чисел, а "-" — для их сложения, но подобное использование механизма перегрузки операций **не рекомендуется** с точки зрения здравого смысла.

При *перегрузке операции с помощью метода* число формальных параметров **на единицу меньше** числа фактических операндов операции. В этом случае первый операнд операции соответствует объекту типа класса, в котором перегружается операция. В случае *бинарной операции* входной параметр соответствует второму операнду перегружаемой операции.

При перегрузке операции с помощью *функции-друга* число формальных параметров **совпадает** с числом операндов операции, так как в этом случае операнды операции, представленные формальными параметрами, являются внешними объектами для такой функции.

Перегрузка операций

Унарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект, например:

```
class data {                                // класс дата  
    int day, month, year;  
    public:  
        data() {cin>>day>>month>>year;} //конструктор без параметров  
        data(int x, int y, int z)    // констр. с параметрами  
            { day=x; month=y; year=z;}  
        ~data() {day=month=year=0;}    // деструктор  
  
        //...  
  
    };
```

Перегрузка унарных операций

// функция перегрузки унарного оператора !

```
bool data::operator!() {    // проверка на равенство нулю  
    if ((day==0)|| (month==0)|| (year==0)) return true;  
    return false;  
}
```

bool operator !(); // объявление внутри класса **data**

// Вызов перегруженной функции !

```
int main() { // основная программа
```

```
    data d1;    // создание объекта d1
```

```
    // ...
```

```
/* if (!d1) */    if (d1.operator!())
```

```
    cout << "d1 содержит поле == 0";
```

```
    else cout << "d1 не содержит поле ==0";
```

```
//...
```

Перегрузка унарных операций

// функция перегрузки унарного оператора -

```
data data::operator-() {  
    year-=3; month-=3; day-=3;  
    return *this;  
}
```

data operator -(); // объявление внутри класса **data**

// Вызов перегруженной функции -

```
int main() { // основная программа
```

```
data d1, d2; // создание объектов d1, d2
```

...

```
-d1;           // 1 ВЫЗОВ      // d1.operator-();
```

```
d2=-d1;       // 2 ВЫЗОВ      // d2=d1.operator-();
```

...

Указатель **this**

Каждый объект содержит свой экземпляр полей класса.

Методы места в классе **не занимают и не дублируются** для каждого объекта. Единственный экземпляр метода используется всеми объектами совместно, поэтому каждый нестатический метод класса должен «знать», для какого объекта он вызван. Для этого, при вызове каждого нестатического метода класса, ему неявно передается указатель на объект, вызвавший его **<class> * const this**.

Выражение ***this** представляет собой разыменованное указание указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (**return *this;**).

Указатель **this** используют для модификации объекта и для замены его другим объектом того же типа.

Перегрузка унарных операций

Если функция определяется вне класса (**friend**), она должна иметь один параметр типа класса. Пример:

```
// функция перегрузки унарного оператора ++
```

```
void operator ++ (data d){  
    d.year+=3; d.month+=3; d.day+=3;  
    cout<<d.year<<" "<<d.month<<" "<<d.day;  
}
```

```
// объявление функции ++
```

```
friend void operator ++ (data d);
```

Вызов:

```
++d1;                // увеличение d1 на 3
```

```
// operator++(d1);
```

Операции постфиксного инкремента и декремента должны иметь первый параметр типа **int**. Он используется только для того, чтобы отличить их от префиксной формы.

Перегрузка унарных операций

Пример: функций перегрузки унарного оператора --

```
void data:: operator--(){  
    cout<<"префиксный декремент";  
}  
void data:: operator--(int){  
    cout<<"постфиксный декремент";  
}
```

// объявление функции ++

```
void operator--();  
void operator--(int);
```

ВЫЗОВ:

```
--d1;
```

```
d1--;
```

```
// d1.operator--();
```

```
// d1.operator--(4);
```

Перегрузка бинарных операций

Бинарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода с параметрами, при этом вызвавший ее объект считается первым операндом.

// функция перегрузки бинарного оператора +

```
int data::operator+ (data d) {  
    return( year+d.year);  
}
```

// объявление +

```
int operator +(data d);
```

// ВЫЗОВ

```
cout<<d1+d2;
```

```
// cout<<d1.operator+(d2);
```

// вычисление суммы годов

Перегрузка бинарных операций

Если функция определяется вне класса (friend), она должна иметь два параметра типа класса.

```
// функция перегрузки бинарного оператора ==  
int operator == (data d1, data d2){  
    if ((d1.day==d2.day) && (d1.month==d2.month) &&  
        (d1.year==d2.year)) return 1;  
    return 0;  
}
```

```
// объявление ==  
friend int operator == (data d1, data d2);
```

//ВЫЗОВ:

```
if (d1==d2)    // if (operator==(d1,d2))  
    cout << "\n d1 = d2";  
    else cout << "\n d1 != d2";
```


Пример перегрузки операторов ввода/вывода

```
// функция перегрузки оператора вывода в поток >>
ostream & operator<<(ostream &out, data &d){
    out << "data:\n";
    out << d.day << d.month << d.year << endl;
    return out;
}

// функция перегрузки оператора ввода из потока >>
istream & operator>>(istream &in, data &d){
    out << "Введите день, месяц, год:\n";
    in >> d.day >> d.month >> d.year;
    return in;
}

// объявление
friend ostream & operator<< (ostream &out, data &d);
friend istream & operator>> (istream &in, data &d);

// ВЫЗОВ:
    cout << "Введите data:\n";  cin >> d1;
    cout << "data:";    cout<<d1;
```

Перегрузка операции присваивания

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого.

Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания. Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.

Возврат из функции указателя на объект делает возможной цепочку операций присваивания:

```
main(){ //...  
    vector v1,v2,v3; v1.vvod(); v3=v2=v1;  
    //...
```

Операцию присваивания можно определять *только как метод класса*. Она *не наследуется*.

Пример перегрузки операции присваивания

```
class vector{
int* p; int size;
public:
vector(){size=10; p=new int[size];}
~vector(){delete []p;}
vvod(){ for(int i=0; i<size; i++) cin>>p[i];}
vector& operator=(const vector& v1);
};

vector& vector::operator=(const vector& v1){
if (size != v1.size)
    { delete []p;    size = v1.size;    p = new int[size];
    }
for (int i = 0; i < size; i++)
    p[i] = v1.p[i];
return *this; // возвращается ссылка на текущий объект
}
```