Перегрузка операций, наследование

Артамонов Ю.Н.

Филиал «Котельники» университета «Дубна»

5 декабря 2018 г.

Содержание

Основные принципы перегрузки операций

C++ дает программисту возможность перегружать большинство операций, так чтобы они были чувствительны к контексту, в котором они используются. Например, операции $>>$, $<<$ в обычном режиме означают побитовый сдвиг вправо, влево соответственно. Однако в C++ они перегружены для работы с потоками ввода, вывода. Несмотря на то, что C++ не позволяет создавать новые операции, он позволяет перегружать существующие операции и, когда они применяются к объектам класса, операции приобретают смысл, соответствующий новым типам. Это очень сильная сторона C++ и важная причина его популярности.

Основные принципы перегрузки операций

Операции перегружаются посредством написания обычного определения функции (с заголовком и телом) за исключением того, что именем функции становится ключевое слово operator с последующим символом перегружаемой операции. Например, имя функции operator+ можно использовать для перегрузки операции сложения. При этом действует ряд ограничений:

- перегрузить можно только имеющиеся операции, нельзя ввести новую операцию;
- для перегруженной операции можно использовать только тот же самый графический символ;
- перегрузить операцию можно только на новый тип, нельзя, скажем перегрузить сложение целых чисел.
- нельзя изменить правило применения или приоритет операции;
- нельзя перезагружать условную операцию, операцию разрешения видимости и операцию взятия члена структуры - точка.

Пример перегрузки операции сложения

Рассмотрим пример перегрузки операции сложения целых чисел для объектов класса

```
#include <iostream>
using namespace std;
class number{
public:
  int x;
  int operator+ (number);
};
int number::operator+ (number b){  return this->x-b.x;}
int main(){
  number a,b;
  a.x=9;
  b.x=2;
  cout<<a+b<<endl;
  return 0;
}
```

Видно, что здесь вместо сложения будет выполняться вычитание.

Запреты, разрешения на перезагрузку операций

Таблица 1: Операции, которые нельзя перегружать

.	.*	::	?:	sizeof

При перезагрузке ( ), [ ], ->, = перегружающая функция должна быть методом класса. Для других операций перегружающие функции могут быть друзьями.

Перезагрузка операции присваивания и операции суммирования с целью разрешить такие операции, как

object2 = object1+object2;

не означает, что автоматически будет перегружена операция +=. Однако такого поведения можно добиться, если явно перегрузить операцию +=.

Дружественные функции или методы класса

Функции-операции могут как быть методами класса, так и не быть ими. Во втором случае они обычно являются друзьями класса. Методы класса используют неявный указатель this, чтобы получить один из аргументов-объектов своего класса. В противном случае такой аргумент должен быть получен явным образом.
Когда функция-операция объявляется в качестве метода класса, левый операнд должен быть объектом, или ссылкой на объект, принадлежащий классу. Если необходимо, чтобы левый операнд был объектом другого класса, эта функция операция должна объявляться как функция, не являющаяся элементом класса. В этом случае такой функции-операции нужно быть другом класса, если она должна получать прямой доступ к закрытым или защищенным элементам класса.

Дружественные функции или методы класса

Перегруженная операция $<<$ должна иметь в качестве левого операнда тип $\mathrm{ostream}\ \&$ (такой, как cout в выражении $\mathrm{cout} << \mathrm{classObject}$), так что она должна быть функцией, не являющейся методом класса.

Аналогично операция $>>$ должна иметь в качестве левого операнда тип $\mathrm{istream}\ \&$(такой, как cin в выражении $\mathrm{cin} >> \mathrm{classObject}$) и не являться методом класса. Кроме того, каждая из таких перегружающих функций должна иметь доступ к закрытым элемента класса, который должен выводиться или вводиться. Поэтому такие перегружающие функции удобно сделать дружественными функциями класса.

Пример перегрузки операций передачи в поток и извлечения из потока

В C++ имеется возможность ввода и вывода стандартных типов данных с использованием операций извлечения из потока $>>$ и передачи в поток $<<$. Эти операции являются перегруженными и могут обрабатывать любой стандартный тип данных, включая строки и адреса памяти. Кроме этого, операции передачи и извлечения данных из потока могут быть перегружены, чтобы выполнять ввод и вывод типов, определяемых пользователем. Рассмотрим пример такого рода. Определим класс телефонного номера, который будем вводить и выводить с использованием стандартных потоков. Например, два объекта могут вводиться следующим образом:

```
cin >> phone1 >> phone2;
```

Пример перегрузки операций передачи в поток и извлечения из потока (продолжение)

```
#include <iostream>
using namespace std;
class phonenumber
{
  friend ostream &operator<<(ostream &, phonenumber &);
  friend istream &operator>>(istream &, phonenumber &);

private:
  char first[2]; //цифра 8 и null
  char prefix[4];
  char two[4];
  char three[3];
  char four[3];
};
```

Пример перегрузки операций передачи в поток и извлечения из потока (продолжение)

```
ostream &operator<<(ostream &output, phonenumber &num)
{
  output << num.first << "-"<<"(" << num.prefix <<")"<<"-"<<num.two
  <<"-"<<num.three<<"-"<<num.four;
  return output;
}
istream & operator>>(istream &input, phonenumber &num)
{
  char s[18];
  input.getline(s, 18);  num.first[0]=s[0];  num.first[1]='\0';
  num.prefix[0]=s[3];  num.prefix[1]=s[4];  num.prefix[2]=s[5];  num
  .prefix[3]='\0';
  num.two[0]=s[8];  num.two[1]=s[9];  num.two[2]=s[10];  num.two[3]=
  '\0';
  num.three[0]=s[12];  num.three[1]=s[13];  num.three[2]='\0';
  num.four[0]=s[15];  num.four[1]=s[16];  num.four[2]='\0';
  return input;
}
```

Пример перегрузки операций передачи в поток и извлечения из потока (продолжение)

```
int main()
{
  phonenumber phone;
  cout<<"введите телефонный номер в формате 8-(926)-345-12-23"<<endl;
  cin >> phone;
  cout<<"Итак, Вы ввели телефонный номер:"<<phone<<endl;
  return 0;
}
```

Пример работы с массивом с перегруженными операциями ++, +=, =, ==, < <

Рассмотрим здесь еще один пример работы с классом массива целых чисел. В конструкторе будем задавать массив с определенной размерностью.
Перегрузим для класса массива операции:

- Операции инкремента (постинкремент, прединкремент);
- Операцию += увеличивает все элементы массива на одно и тоже число;
- Операцию = присваивает один массив другому массиву;
- Операцию == сравнивает один массив с другим;
- Операцию < < выводит элементы массива в стандартный поток вывода.

Работа с классом массива

```
#include<iostream>
#include<time.h>
#include<stdlib.h>
using namespace std;
class array{
  friend ostream & operator<< (ostream &, array &);
  friend istream & operator>> (istream &, array &);
  friend array & operator+(array &, array &);
public:
  int & operator [] (int);
  array & operator++(); // Это прединкремент
  array & operator++(int); //Постинкремент требует фиктивного переменного
  array & operator +=(int);
  array & operator=(array  &);
  bool operator ==(array &);
  void filling_random(void);
  const  int get_size();
```

Работа с классом массива

```
array(int);
  array(array &); //Конструктор копирования
  ~array();
private:
  int size;
  int *mass;
};

array current(1);

array::array(int n)
{
  size = n;
  mass = new int [size];
  for (int i = 0; i<size; i++) mass[i]=0;
}
```

Работа с классом массива

```
array & operator+(array &ar1, array &ar2)
{
  current=((ar1.get_size()<ar2.get_size())? ar2:ar1);
  for (int i=0; i<current.get_size(); i++)
    {
      if ((i<ar1.get_size()) && (i<ar2.get_size()))
current[i] = ar1[i]+ar2[2];
      if ((i<ar1.get_size()) && (i>=ar2.get_size()))
current[i] = ar1[i];
      if ((i>=ar1.get_size()) && (i<ar2.get_size()))
current[i] = ar2[2];
    }
  return current;
}
```

Работа с классом массива

```
array::array(array & copy){
  size = copy.get_size();
  mass = new int [size];
  for (int i = 0; i<size; i++) mass[i]=copy[i];
}
bool array::operator ==(array & ar2){
  if (size == ar2.get_size()){
      bool pr = true;
      for (int i=0; i<ar2.get_size();i++)
if ((mass[i]) != (ar2[i])){
    pr = false;
    break;
  }
      return pr;
    }
  else
    return false;
}
```

Работа с классом массива

```
array & array::operator= (array  & ar)
{
  size = ar.get_size();
  delete [] mass;
  mass = new int [size];
  for (int i = 0; i<this->size; i++) this->mass[i]=ar[i];
  return *this;
}

array::~array(void)
{
  delete [] this->mass;
}

const int array::get_size()
{
  return this->size;
}
```

Работа с классом массива

```
int & array::operator[](int n){
  if (n < size)
    return mass[n];
  else{
      if (n<0)
return mass[0];
      else
return mass[size-1];
    }
}

ostream & operator<<(ostream & output, array & arr){
  for (int i = 0; i<arr.size; i++)
    output << (arr[i])<<"; ";
  output<<endl;
}
```

Работа с классом массива

```
istream & operator>>(istream & input, array &arr){
  cout<<"array size: ";
  cin>>arr.size;
  delete [] arr.mass;
  arr.mass = new int [arr.size];
  for (int i=0; i<arr.size;i++)
    {cout<<"element "<<i<<": ";
    cin>>arr[i];
    }
}

array& array::operator++()
{
  for (int i = 0; i<size;i++)
    mass[i]++;
  return *this;
}
```

Работа с классом массива

```
array & array::operator++(int)
{
  current = *this;
  for (int i = 0; i<size;i++)
    mass[i]++;
  return current;
}
array & array::operator+=(int value)
{
  for (int i = 0;i<size; i++)
    mass[i]+=value;
  return *this;
}
void array::filling_random()
{
  for (int i = 0; i<size; i++) mass[i] = rand()&100+1;
}
```