Введение в язык C++

Углич П. С.[1]

[1]Южный федеральный университет

Лекция 5

Outline

Унарные операции

Ключевое слово operator предоставляет объявление функции, указывающей что operator-symbol означает при его применении к экземплярам класса. Это предоставляет оператору многозначность или «перегружает» его. Компилятор различает разные виды одного и того же оператора, изучая типы его операндов.

```
type operator operator-symbol ( parameter-list )
```

Можно переопределять большинство встроенных операторов глобально или в классе. Перегруженные операторы реализуются в виде функции.
Перегруженный оператор имеет имя operatorx, где x означает оператор из следующей таблицы. Например, для сложения/присвоения += необходимо определить функцию operator+=.

Унарные операции

Оператор присваивания (=) является, строго говоря, бинарным оператором.Его объявление идентично объявлению любого другого бинарного оператора, со следующими исключениями:

- Он должен быть нестатической функцией-членом.Никакой оператор operator= не может быть объявлен как функция, не являющаяся членом.
- Он не наследуется производными классами.
- Компилятор может создавать для типов классов функции operator= по умолчанию, если они не существуют.

Унарные операции

```
// assignment.cpp
class Point
{
public:
   Point &operator=( Point & );
   // Right side is the argument.
   int _x, _y;
};
// Define assignment operator.
Point &Point::operator=( Point &ptRHS )
{  _x = ptRHS._x; _y = ptRHS._y;
   return *this;  // Assignment operator returns left side.
}
int main()
{}
```

Унарные операции

Оператор возвращает объект для сохранения поведения оператора присваивания, который после завершения присваивания возвращает значение левой части.Это позволяет писать операторы следующего вида:

```
pt1 = pt2 = pt3;
```

Перегрузка равенства

```
//fraction.h
class fraction {
int numerator;
unsigned int denumerator;
....};
```

Унарные операции

```
// subscripting.cpp
// compile with: /EHsc
#include <iostream>

using namespace std;
class IntVector {
public:
   IntVector( int cElements );
   ~IntVector() { delete [] _iElements; }
   int& operator[]( int nSubscript );
private:
   int *_iElements;
   int _iUpperBound;
};
```

Унарные операции

```
// Construct an IntVector.
IntVector::IntVector( int cElements ) {
    _iElements = new int[cElements];
    _iUpperBound = cElements;
}
```

Унарные операции

```
// Subscript operator for IntVector.
   int& IntVector::operator[]( int nSubscript ) {
   static int iErr = -1;
   if( nSubscript >= 0 && nSubscript < _iUpperBound )
      return _iElements[nSubscript];
   else {
      clog << "Array bounds violation." << endl;
      return iErr;
   }
}
```

Унарные операции

Объявление функций:

```
{
........
fraction& operator+=(const fraction &other);
fraction operator+(const fraction &other);
.........
};
```

Унарные операции

Реализация функций

```
//fraction.cpp
...
fraction fraction::operator+(const fraction & other)
{
fraction a;
a.numerator = numerator*other.denumerator +
                                    denumerator*other.numerator;
a.denumerator = denumerator*other.denumerator;
a.reduce();
return a;
}
....;
```

Унарные операции

Реализация функций

```
void fraction::operator+=(const fraction & other)
{
*this = *this + other;
reduce();
}
```

Унарные операции

Отношения сравнения

```
{
........
bool operator==(fraction const &) const;
bool operator<(fraction const &) const;
bool operator<=(fraction const &) const;
.........
};
```

Унарные операции

Реализация функций

```
bool fraction::operator==(fraction const & other) const
{
return !(numerator - other.numerator
&& denumerator - other.denumerator);
}

bool fraction::operator<(fraction const & other) const
{
return numerator*other.denumerator <
denumerator*other.numerator;
}
```

Унарные операции

```
bool fraction::operator<=(fraction const & other) const
{
return *this<other || *this==other;
}
```

Унарные операции

Примеры

```
class bookshop {
string _name;
vector<book> _books;
public:
bookshop();
bookshop(string name) : _name(name)
{
_books = vector<book>();
};
```

Унарные операции

Примеры

```
string name() const;
bookshop & operator+=(book const & new_book) {
 _books.push_back(new_book); return *this; }

int quantity() const;

friend ostream& operator<<
(ostream& os, const bookshop& bsh);

book TheCheapest();
};
```

Бинарные операторы

Чтобы объявить функцию бинарного оператора как нестатический член, необходимо объявить ее в виде

```
возвращаемый-тип оператор op ( arg )
```

где ret-type — возвращаемое значение, op — один из операторов, а arg — аргумент любого типа.
Чтобы объявить функцию бинарного оператора как глобальную функцию, необходимо объявить ее в виде

```
ret-type operator op( arg1, arg2 )
```

где ret-type и op — элементы, описанные для функций операторов членов, а arg1 и arg2 — аргументы.Хотя бы один из аргументов должен принадлежать типу класса.

```
friend fraction & operator + (const int &i, fraction &f);
```

Бинарные операторы

Внешним функциям запрещен доступ напрямую к полям класса, объявленным как private.
Способ решения проблемы — объявить такую операцию дружественной для класса. Дружественными по отношению к рассматриваемому классу могут быть внешние функции, или функции другого класса, или другой класс.
Для того чтобы внешнюю функцию сделать дружественной классу, нужно внести в интерфейс класса заголовок функции, добавив впереди ключевое слово friend.

```
friend fraction & operator + (const int &i, fraction &f);
friend ostream & operator<<(ostream& os, const fraction &f)
friend istream & operator>>(istream& is, fraction &f);
```

Бинарные операторы

Реализация

```
ostream & operator<<(ostream& os, const fraction &f)
{
// TODO: вставьте здесь оператор return
os << f.numerator << '/' << f.denumerator;
return os;
}

istream & operator >> (istream & is, fraction & f){
is >> f.numerator;
is >> f.denumerator;
return is;
}
```

Бинарные операторы

```
fraction & operator+(const int & i, fraction & f)
{
// TODO: вставьте здесь оператор return
return f + i;
}
```

Другой пример

```
ostream & operator<<(ostream & os, const bookshop & bsh)
{
// TODO: insert return statement here
os << "Name: " << bsh._name << "\n";
for (unsigned int i = 0; i < bsh._books.size(); i++)
os << bsh._books[i] << "\n";
return os;
}
```

Бинарные операторы

Функция бинарного оператора как глобальная функция

```
fraction & operator-(const int & i, fraction & f){
// TODO: вставьте здесь оператор return
 fraction res(f.Numerator() - i*f.Denumerator(),
 f.Denumerator());
 return res;
}
```

Бинарные операторы

Префиксная и постфиксная операции инкремента

```
fraction& operator++();       // Prefix increment operator.
fraction operator++(int);     // Postfix increment operator

fraction & fraction::operator++() {
          numerator += denumerator;
          return *this;
}
fraction fraction::operator++(int){
           fraction frac = *this;
           ++*this;
           return frac;
}
```