Перегрузка операторов (окончание)

7 июня 2017 г.

Перегрузка глобальной функцией

1. Функция для унарной операции получает один явный аргумент.
2. Функция для бинарной операции получает два явных аргумента.

унарный -

- ClassA operator-(const ClassA&);

бинарный -

- ClassA operator-(const ClassA&, const ClassA&);

Перегрузка глобальной функцией

3. Объявляется и реализуется вне класса.
4. Перед словом operator не надо писать имя класса (~~*MyClass::*~~operator+).
5. Обращение к полям класса осуществляется через аксессоры.

```
return-type operator op( arg );
```

Арифметические бинарные операторы: пример

```
class Digit{
      int x_;
public:
      explicit Digit(const int x) : x_(x) { }
      int getDigit( ) const { return x_; }
      void setDigit( const int x ) { x_ = x; }
};

Digit  operator+(const Digit& a, const Digit& b){
      Digit tmp;
      tmp.setDigit(a.getDigit() + b.getDigit());
      return tmp;
}
```

Бинарные операторы: пример

```
void f ( ){
    Digit a(5);
    Digit b(7);
    Digit c(0);

    c = a + b;

    Digit d = operator+(a, b);
}
```

Унарные операторы: пример

```
class Digit{
      int x_;
public:
       explicit Digit(const int x) : x_(x) { }
      int getDigit( ) const { return x_; }
      void setDigit( const int x ) { x_ = x; }
};

Digit  operator-(const Digit& a){
      Digit tmp = a;
      tmp.setDigit( -a.getDigit( ) );
      return tmp;
}
```

```
Digit a(10);
Digit b = -a;
b = operator-(a);
```

Операторы >> и <<

```
std::ostream& operator<<( std::ostream&, const Digit&);

std::istream& operator>>( std::istream&, Digit&);
```

- cin – объект класса istream (оператор >>)
  cout – объект класса ostream (оператор <<)
- всегда не член класса
- возвращают ссылку на поток
- принимают два аргумента (ссылка на поток и пользовательский объект)

Операторы << и >>: пример

```
class Digit{ …};

std::ostream& operator<<(std::ostream& os, const Digit& a){
        os << a.getDigit( );
        return os;
}

std::istream& operator>>(std::istream& is, Digit& a){
        int tmp;
        is >> tmp;
        a.setDigit( tmp);
        return is;
}
```

```
std::cout << a << b;
std::cin >> a >> b;

((std::cout << a) << b);
((std::cin >> a) >> b);
```

Дружественные функции: общие сведения

1. Имеют доступ к закрытой части класса.
2. Объявляются в классе с ключевым словом friend.
3. Функция не будет методом класса.
4. Реализацию можно писать как в классе (inline), так и вне класса.
5. При реализации не надо писать :: и friend.
6. Объекты надо передавать как аргументы (не получает this).

Дружественные функции: общие сведения

7. Функция независима от спецификаторов доступа (private, protected, public).
8. Метод другого класса можно объявить как дружественный.
9. Одна функция может быть другом нескольких классов (нужно предварительное объявление класса – class B;).

Дружественные функции

```
class A {
    int x_;
public:
    …
    friend void set(A& a, const int value);
    friend int get(const A& a);
};

void set(A& a, const int value){ a.x_ = value; }

int get(const A& a){ return a.x_; }

    …
A obj(15);
std::cout  << get(obj) << std::endl;
```

Перегрузка дружественной функцией

1. Функция для унарной операции получает один явный аргумент.
2. Функция для бинарной операции получает два явных аргумента.

унарный -

- friend ClassA operator-(const ClassA&);

бинарный -

- friend ClassA operator-(const ClassA&, const ClassA&);

Арифметические бинарные операторы: пример

```
class Digit{
    int x_;
public:
    …
    explicit Digit(const int x) : x_(x) { }
    friend Digit  operator+(const Digit& a, const Digit& b){
            Digit tmp;
            tmp.x_ = a.x_+ b.x_;
            return tmp;
    }
};
```

Унарные операторы: пример

```
class Digit{
    int x_;
public:
    …
    explicit Digit(const int x) : x_(x) { }
    friend operator-(const Digit& a){
            Digit tmp = a;
            tmp.x_ = -a.x_;
            return tmp;
    }
};
```

Дружественные функции vs. методы

1. Если оператор создает новый объект, то оператор лучше вынести из класса.
2. Если оператор изменяет исходный объект, то оператор должен быть методом класса.
3. Если первый аргумент – встроенный тип, то оператор не может быть методом.
4. Только методы класса:

   = [ ] ( ) -> new/new[] delete/delete[]

   преобразования типов

Какие операторы лучше реализовать как методы класса?

=	%	!
*=	>>	!=
+(MyClass&, int)	>	delete[]
+(MyClass&, int)	( )	[ ]
++	new	==
–>	int	<<
–	+=	char

Вопросы?