

Семинар 3 (переопределение операторов)

МГТУ им. Н.Э. Баумана

March 9, 2016

Операции над пользовательскими типами в C

Вектор (математический)

```
1 typedef struct {  
2     int32_t* data;  
3     uint32_t length;  
4 } Vector;
```

Вводим операции

```
1 Vector* sum(const Vector*, const Vector*);  
2 Vector* mult(const Vector*, const Vector*);  
3 Vector* mult_c(int32_t c, const Vector*);  
4 int32_t mult_scalar(const Vector*, const Vector*);
```

Использование

$$c \cdot A + B \times (C + D)$$

```
1 Vector* CD = sum(C, D);  
2 Vector* cA = mult_c(c, A);  
3 Vector* BCD = mult(B, CD);  
4 Vector* result = sum(cA, BCD);  
5 free(CD); free(cA); free(BCD);
```

Решение в C++: переопределение операций

```
1 class Vector;
2 Vector operator*(int32_t, const Vector&);
3 class Vector {
4     std::vector<int32_t> data;
5 public:
6     Vector operator+(const Vector&);
7     Vector operator*(const Vector&);
8     int32_t operator^(const Vector&); // for scalar multiplication
9     friend Vector operator*(int32_t, const Vector&);
10 };
```

Использование

$$c \cdot A + B \times (C + D)$$

```
1 Vector result = c*A + B*(C+D);
2 // all temporary variables are stack-allocated
3 // and destructed automatically
```

Подробнее про переопределение операторов

Нельзя перегружать операторы

```
1 :: . .* ?: sizeof typeid
```

Операторная функция для операции

```
1 X operator#(Y, Z);
```

Замечания

- Вводить новые знаки операций нельзя
- Приоритет операций сохраняется
- Операция должны быть объявлена в соответствии с ее стандартным синтаксисом (арность не изменить)
- вычисление аргументов всегда предшествует вычислению результата (вспомнить && и ||)

Переопределение бинарной операции

Метод класса

```
1 class X {  
2 public:  
3     void operator+(int);  
4 };
```

Использование

```
1 void f(X a) {  
2     a+1; // a.operator+(1)  
3 }
```

Глобальная функция с 2мя аргументами

```
1 class X {  
2 public:  
3     X(int);  
4 };  
5 void operator+(X,X);  
6 void operator+(X,double);
```

Использование

```
1 void f(X a) {  
2     1+a; // ::operator+(X(1),a)  
3     a+1.0; // ::operator+(a,1.0)  
4 }
```

Переопределение унарной операции

Метод класса или функция с одним аргументом

```
1 class X {  
2 public:  
3     X operator-();  
4 };  
5  
6 int* operator&(X); // address of
```

Использование

```
1 void f(X a) {  
2     int* b = &a; // ::operator&(a)  
3     X c = -a; // a.operator-()  
4 }
```

Ограничения

Должны быть нестатическими методами класса:

```
1 operator=()  
2 operator[]()  
3 operator()()
```

Переопределение унарной операции

Постфиксные унарные операции

Если определены постфиксные и префиксные варианты оператора, то постфиксные операторы при переопределении принимают фиктивный дополнительный параметр типа `int`

```
1 class X {  
2 public:  
3     X operator--(int); // postfix decrement  
4     X operator++(); // prefix increment  
5 };  
6 X operator++(X, int); //postfix increment  
7 X operator--(X); // prefix decrement
```

Использование

```
1 void f(X a) {  
2     a--; // a.operator--(1)  
3     --a; // ::operator--(a)  
4     a++; // ::operator++(a, 1)  
5     ++a; // a.operator++()  
6 }
```

Операции и пользовательские типы

```
1 class X {  
2 public:  
3     X operator+(int);  
4 };
```

Использование

```
1 void f(X a) {  
2     a = a + 1; // OK  
3     a = 1 + a; // ERROR!  
4 }
```


Операции и пользовательские типы

```
1 class X {  
2 public:  
3     X operator+(int);  
4 };
```

Использование

```
1 void f(X a) {  
2     a = a + 1; // OK  
3     a = 1 + a; // ERROR!  
4 }
```

Решение 1

```
1 class X {  
2 public:  
3     X(int);  
4 };  
1 a = 1 + a; // X(1) + a
```

Операции и пользовательские типы

```
1 class X {  
2 public:  
3     X operator+(int);  
4 };
```

Использование

```
1 void f(X a) {  
2     a = a + 1; // OK  
3     a = 1 + a; // ERROR!  
4 }
```

Решение 1

```
1 class X {  
2 public:  
3     X(int);  
4 };  
1 a = 1 + a; // X(1) + a
```

Решение 2

```
1 X operator+(int, X);  
1 a = 1 + a; // operator+(1, a)
```

Операции и пространства имен

```
1 namespace std {  
2 class string { /* ... */ };  
3 class ostream {  
4 ostream& operator<<(const char*);  
5 // ...  
6 };  
7 ostream& operator<<(ostream&, const string&);  
8 } // namespace std  
9 int main() {  
10     const char* p = "Hello";  
11     std::string s = "world";  
12     std::cout << p << ", " << s << "!\n"; // OK  
13 };
```

Операции и пространства имен

```
1 namespace std {  
2 class string { /* ... */ };  
3 class ostream {  
4 ostream& operator<<(const char*);  
5 // ...  
6 };  
7 ostream& operator<<(ostream&, const string&);  
8 } // namespace std  
9 int main() {  
10     const char* p = "Hello";  
11     std::string s = "world";  
12     std::cout << p << ", " << s << "!\n"; // OK  
13 };
```

Поиск оператора

$x \# y$, x is $X \in N$, y is $Y \in M$

- $X::\text{operator}\#(Y)$
- $::\text{operator}\#(X, Y)$
- $N::\text{operator}\#(X, Y)$
- $M::\text{operator}\#(X, Y)$