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

```
class Vector;
Vector operator*(int32_t, const Vector&);
class Vector {
  std::vector<int32_t> data;
public:
  Vector operator+(const Vector&);
  Vector operator*(const Vector&);
  int32_t operator^(const Vector&); // for scalar multiplication
  friend Vector operator*(int32_t, const Vector&);
};
```

Использование

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

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

Подробнее про переопределение операторов

Нельзя перегружать операторы

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

Операторная функция для операции

```
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

```
class X {
public:
  void operator+(int);
};
```

Использование

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

Глобальная функция с 2мя аргументами

```
class X {
public:
  X(int);
};
void operator+(X,X);
void operator+(X,double);
```

Использование

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

Переопределение унарной операции

Метод класса или функция с одним агрументом

```
class X {
public:
  X operator-();
};

int* operator&(X); // address of
```

Использование

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

Ограничения

Должны быть нестатическими методами класса:

```
operator=()
operator[]()
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

```
class X {
public:
  X operator--(int); // postfix decrement
  X operator++(); // prefix increment
};
X operator++(X, int); //postfix increment
X operator--(X); // prefix decrement
```

Использование

```
void f(X a) {
  a--; // a.operator--(1)
  --a; // ::operator--(a)
  a++; // ::operator++(a, 1)
  ++a; // a.operator++()
}
```

Операции и пользовательские типы

```
class X {
public:
  X operator+(int);
};
```

Использование

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

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

```
class X {
public:
  X operator+(int);
};
```

Использование

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

Решение 1

```
class X {
public:
  X(int);
};
```

```
a = 1 + a; // X(1) + a
```

Решение 2

```
X operator+(int, X);
```

```
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::operator#(Y)
- ::operator#(X, Y)
- N::operator#(X, Y)
- M::operator#(X, Y)