Программирование на языке C++
Лекция 5

Перегрузка операторов

Александр Смаль

Основные операторы

- Арифметические
  - Унарные: префиксные + - ++ --, постфиксные ++ --
  - Бинарные: + - * / % += -= *= /= %=
- Битовые
  - Унарные: ~.
  - Бинарные: & | ^ &= |= ^= >> <<.
- Логические
  - Унарные: !.
  - Бинарные: && ||.
  - Сравнения: == != > < >= <=

Другие операторы

1. Оператор присваивания: `=`
2. Специальные:
   - префиксные `*` `&`,
   - постфиксные `->` `->*`,
   - особые `,` `.` `::`
3. Скобки: `[]` `()`
4. Оператор приведения `(type)`
5. Тернарный оператор: `x ? y : z`
6. Работа с памятью: `new` `new[]` `delete` `delete[]`

- Нельзя перегружать операторы `.` `.:` и тернарный оператор.

Перегрузка операторов

```
Vector operator-(Vector const& v) {
    return Vector(-v.x, -v.y)
}

Vector operator+(Vector const& v,
                 Vector const& w) {
    return Vector(v.x + w.x, v.y + w.y);
}

Vector operator*(Vector const& v, double d) {
    return Vector(v.x * d, v.y * d);
}

Vector operator*(double d, Vector const& v) {
    return v * d;
}
```

Перегрузка операторов внутри классов

- **NB:** Обязательно для `(type)` `[]` `()` `->` `->*` `=`

```
struct Vector {
  Vector operator-() const { return Vector(-x, -y); }
  Vector operator-(Vector const& p) const {
      return Vector(x - p.x, y - p.y);
    }
  Vector operator*=(double d) {
      x *= d;
      y *= d;
      return *this;
    }
  double operator[](size_t i) const {
      return (i == 0) ? x : y;
    }
  bool   operator()(double d)   const   { ... }
  void   operator()(double a, double b) { ... }
    double x, y;
};
```

Перегрузка инкремента и декремента

```
struct BigNum {
    BigNum & operator++() { //prefix
        //increment
        ...
        return *this;
    }

    BigNum operator++(int) { //postfix
        BigNum tmp(*this);
        ++(*this);
        return tmp;
    }
    ...
};
```

Переопределение операторов ввода-вывода

```
#include <iostream>

struct Vector { ... };

std::istream& operator>>(std::istream & is,
                         Vector & p) {
  is >> p.x >> p.y;
  return is;
}

std::ostream& operator<<(std::ostream &os,
                         Vector const& p) {
  os << p.x << ' ' << p.y;
  return os;
}
```

Умный указатель

Реализует принцип: *“Получение ресурса есть инициализация”*
Resource Acquisition Is Initialization (RAII)

```
struct SmartPtr {
  Data & operator*()  const {return *data_;}
  Data * operator->() const {return  data_;}
    Data * get()      const {return  data_;}
    ...
private:
    Data * data_;
};

bool operator==(SmartPtr const& p1,
                SmartPtr const& p2) {
    return p1.get() == p2.get();
}
```

Оператор приведения

```
struct String {
    operator bool() const {
        return size_ != 0;
    }

    operator char const *() const {
        if (*this)
            return data_;
        return "";
    }

private:
    char * data_;
    size_t size_;
};
```

Операторы с особым порядком вычисления

```
int main() {
    int a = 0;
    int b = 5;
    (a != 0) && (b = b / a);
    (a == 0) || (b = b / a);

    foo() && bar();
    foo() || bar();
    foo(), bar();
}

// no lazy semantics
Tribool operator&&(Tribool const& b1,
                   Tribool const& b2) {
    ...
}
```