Перегрузка операторов

Александр Смаль

Академический университет
8 ноября 2013
Санкт-Петербург

Операторы

Арифметические

1. Унарные префиксные: `+ - ++ --`, постфиксные `++ --`
2. Бинарные: `+ - * / % += -= *= /= %=`

Битовые

Унарные: `~`. Бинарные: `& | ^ &= |= ^= >> <<`

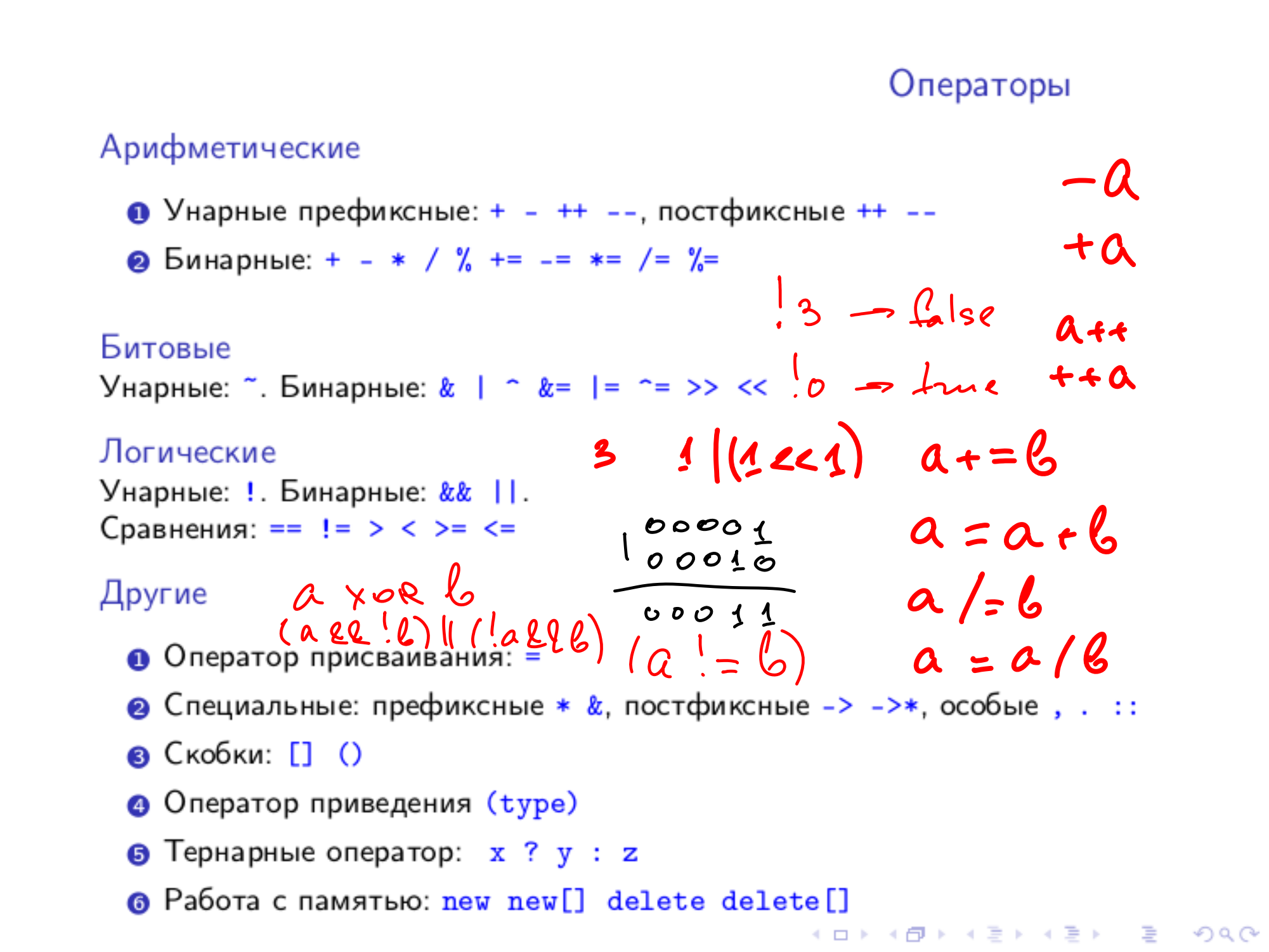
Логические

Унарные: `!`. Бинарные: `&& ||`.

Сравнения: `== != > < >= <=`

Другие

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



Операторы

Арифметические

1. Унарные префиксные: `+ - ++ --`, постфиксные `++ --`
2. Бинарные: `+ - * / % += -= *= /= %=`

Битовые

Унарные: `~`. Бинарные: `& | ^ &= |= ^= >> <<`

Логические

Унарные: `!`. Бинарные: `&& ||`.
Сравнения: `== != > < >= <=`

f(a, 6)

x = a+=6, c+=d;

Другие

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

Перегрузка операторов

Point2 p(3,4);

p = -p;

```
Point2 operator-(Point2 const& p) {
    return Point2(-p.x, -p.y)
}
Point2 operator+(Point2 const& p1, Point2 const& p2) {
    return Point2(p1.x + p2.x, p1.y + p2.y);
}
Point2 operator*(Point2 const& p, double d) {
    return Point2(p.x * d, p.y * d);
}
```

Перегрузка операторов внутри классов

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

```
struct Point2 {
    Point2 operator-() const {
        return Point2(-p.x, -p.y)
    }
    Point2 operator*=(double d) {
        p.x *= d;
        p.y *= d;
        return *this;
    }
    Point2 operator-(Point2 const& p) const {
        return Point2(x - p.x, y - p.y);
    }
    //only one argument
    double operator[](size_t i) const {
        return (i == 0) ? x : y;
    }
    // any number of arguments
    void     operator()()                              const { ... }
    bool     operator()(Point2 const& p, double d)     { ... }
};
```

m[2,3]

Point2 p;
p();

Перегрузка инкремента и декремента

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

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

```
int a = 3;
int b = a++;
```

Переопределение операторов ввода-вывода

```
#include <iostream>

struct Point2 {
    double x;
    double y;
};

std::istream& operator>>(std::istream & is, Point2 & p)
{
    is >> p.x >> p.y;
    return is;
}

std::ostream& operator<<(std::ostream &os, Point2 const& p)
{
    os << p.x << ' ' <<  p.y;
    return os;
}
```

Умный указатель

operator = ^ ^ = ()

expr2 := expr2 binop expr2

:= unop expr2

p->;

delete data_; &p

} return &data_; }

```
struct Point2Ptr {
  Point2** operator2&()
    Point2Ptr & operator*()  const { return *data_; }
    Point2Ptr * operator->() const { return  data_; }
    Point2Ptr * get()        const { return  data_; }
    ...
private:
    Point2Ptr * data_;
};

bool operator==(Point2Ptr const& p1, Point2Ptr const& p2) {
    return p1.get() == p2.get();
}
```

Point2Ptr p = new Point2(3,4);

p->x = 5;

Point *q = p->, p.operator2->()

p->x

(p.operator2->())->x

RAII

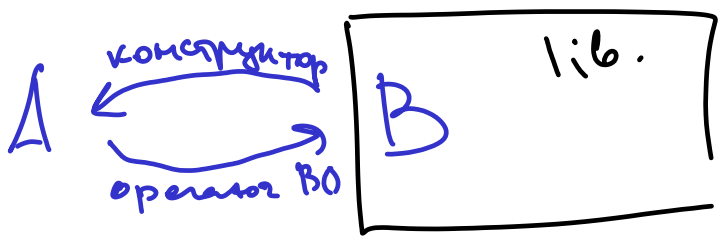
Resource
Acquisition
Is
Initialization

Оператор приведения

```
struct String {
    operator bool() const {
        return !empty();
    }

    operator char const *() const {
        return data_;
    }

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



f -> * "Hello";

String s = "";

if (s)

safe Bool

int k = 5;

void f(char const *s)

f(s)

"Правильное" переопределение арифметических операторов

```
struct String {
    String( char const * cstr ) {
        ...
    }

    String & operator+=(String const& s) {
        ...
        return *this;
    }

    // wouldn't work with C-strings
    //String operator+(String const& s2) const;
};

String operator+(String s1, String const& s2) {
    return s1 += s2;
}
```

String s = "Hello";

s = "Hey!" + s;

s = s + "Jane";

const &

String + (s1);
return + += s2;

“Правильное” переопределение операторов сравнения

```
bool operator==(Point2 const& a, Point2 const& b) {
    return a.x == b.x && a.y == b.y;
}
bool operator!=(Point2 const& a, Point2 const& b) {
    return !(a == b);
}
bool operator<(Point2 const& a, Point2 const& b) {
    return ....
}
bool operator>(Point2 const& a, Point2 const& b) {
    return b < a;
}
bool operator<=(Point2 const& a, Point2 const& b) {
    return !(b < a);
}
bool operator>=(Point2 const& a, Point2 const& b) {
    return !(a < b);
}
```

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
BigNum operator&&(BigNum const& b1, BigNum const& b2) {
    ...
}
```

f(foo(), bar());

О чём стоит помнить?

```
Point3   p;
+ - *
z = p^q + q^2;
```

- Стандартная семантика операторов.
- Приоритет операторов.
- Хотя бы один из параметров должен быть пользовательским.

```
++i++
```