

Язык C++

Перегрузка функций, методов и операторов

Операторы

1. Арифметические ·
 - а. Унарные: префиксные + - ++ --,
 - б. постфиксные ++ -- ·
 - с. Бинарные: + - * / % += -= *= /= %=
2. Битовые ·
 - а. Унарные: ~. ·
 - б. Бинарные: & | ^ &= |= ^= >> << >>= <<=.
3. Логические ·
 - а. Унарные: !. ·
 - б. Бинарные: && ||. ·
 - с. Сравнения: == != > < >= <=

Операторы

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

Перегрузка операторов

1. **Не должна противоречить здравой логике**
2. Как член класса или как глобальная функция
3. [], (), ->, = - всегда члены класса
4. Ввод (>>) и вывод(<<) всегда глобальные функции
5. Операторы ::, .*, ., ?: перегружать нельзя
6. Новые операторы сделать нельзя

Оператор	Как член класса	Не как член класса
@a	(a).operator@ ()	operator@ (a)
a@b	(a).operator@ (b)	operator@ (a, b)
a=b	(a).operator= (b)	
a(b...)	(a).operator()(b...)	
a[b]	(a).operator[](b)	
a->	(a).operator-> ()	
a@	(a).operator@ (0)	operator@ (a, 0)

CRational

```
/*   Класс из первой лекции   */

class CRational {
public:

    int numerator() const {
        return numerator_;
    }

    unsigned denominator() const {
        return denominator_;
    }
private:
    int      numerator_;
    unsigned denominator_;
};
```

CRational

```
CRational& operator=(const CRational& other) {  
    if(&other == this) {  
        return *this;  
    }  
  
    numerator_ = other.numerator_;  
    denominator_ = other.denominator_;  
  
    return *this;  
}
```

CRational

Операторы ввода\вывода:

- operator<< & operator>>
- Не как члены класса

```
std::ostream& operator<<(std::ostream& stream, const CRational& value) {  
    stream << value.numerator() << '/' << value.denominator();  
    return stream;  
}
```


CRational

```
std::istream& operator>>(std::istream& stream, CRational& r) {  
    int a;  
    unsigned b;  
    char ch;  
    stream >> a;  
  
    if (stream.get() != '/')  
        stream.setstate(std::ios::failbit);  
    stream >> b;  
  
    r = CRational(a,b);  
    return stream;  
}
```

CRational

```
bool operator==(const CRational& lhs, const CRational& rhs) {  
    return lhs.numerator() * rhs.denomonator() ==  
           rhs.numerator() * lhs.denomonator();  
}  
  
bool operator!=(const CRational& lhs, const CRational& rhs) {  
    return !operator==(lhs, rhs);  
}
```

CRational

```
bool operator<(const CRational& lhs, const CRational& rhs) {  
    return lhs.numerator() * rhs.denomonator() < rhs.numerator() *  
    lhs.denomonator();  
}  
  
bool operator>(const CRational& lhs, const CRational& rhs) {  
    return (rhs < lhs);  
}  
  
bool operator<=(const CRational& lhs, const CRational& rhs) {  
    return !operator>(lhs, rhs);  
}
```

CRational

```
// prefix operator
CRational& operator++() {
    numerator_ += denominator_;
    return *this;
}

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

CRational

```
explicit operator float() const {  
    return numerator_/denominator_;  
}
```

friend

```
class CRational {
public:
    friend std::ostream& operator<<(std::ostream& stream, const CRational& value) ;
};

std::ostream& operator<<(std::ostream& stream, const CRational& value) {
    stream << value.numerator_ << '/' << value.denominator_;
    return stream;
}
```

CIntArray

```
class CIntArray {  
public:  
    // ...  
    int& operator[](size_t idx) {  
        return data_[idx];  
    }  
  
private:  
    int* data_;  
    size_t size_;  
};
```

operator()

```
class CMult {  
public:  
    explicit CMult(int mult)  
        :mult_(mult)  
    {}  
  
    int operator()(int value) {  
        return mult_ * value;  
    }  
  
private:  
    int mult_;  
};
```


operator()

```
int main() {  
    CMult m(2);  
    std::cout << m(3);  
  
    std::vector<int> data = {1, 2, 3, 4, 5};  
    std::transform(data.begin(), data.end(), data.begin(), CMult(5));  
  
    return 0;  
}
```

RAII (*Resource Acquisition Is Initialization*)

```
class CFileDescriptor {
public:
    explicit CFileDescriptor(const char* path, const char* mode) {
        file_ = fopen(path, mode);
        if(file_ == nullptr) {
            // throw some exception (see next lection)
        }
    }
    operator FILE*() {
        return file_;
    }
    ~CFileDescriptor(){
        if(file_ != nullptr)
            fclose(file_);
    }
private:
    FILE* file_;
};
```

operator->, operator*

```
class Foo {  
public:  
    int foo() const {  
        return value_;  
    }  
private:  
    int value_ = 2022;  
};
```

RAII (*Resource Acquisition Is Initialization*)

```
class FooPtr {  
public:  
    explicit FooPtr(Foo* ptr = nullptr)  
        : ptr_(ptr)  
    {}  
  
    ~FooPtr() {  
        delete ptr_;  
    }  
  
private:  
    Foo* ptr_;  
};
```

operator->, operator*

```
class FooPtr {  
public:  
    Foo& operator*() {  
        return *ptr_;  
    }  
  
    Foo* operator->() {  
        return ptr_;  
    }  
  
private:  
    Foo* ptr_;  
};
```

operator->, operator*

```
int main() {  
    FooPtr p {new Foo()};  
    if(p->foo() == 2021)  
        return 1;  
  
    return 0;  
}
```