ПРО ПЕРЕГРУЗКУ ОПЕРАТОРОВ

#include <iostream>

using namespace std;

class Counter

{

public:

Counter(int sec)

{

seconds = sec;

}

void display()

{

cout << seconds << " seconds" << endl;

}

int seconds;

};

Counter operator + (Counter c1, Counter c2)

{

return Counter(c1.seconds + c2.seconds);

}

int main()

{

Counter c1(20);

Counter c2(10);

Counter c3 = c1 + c2;

c3.display(); // 30 seconds

return 0;

}

ФУНКЦИЯ ОПЕРАТОРА ОПРЕДЕЛЕНА КАК ЧЛЕН КЛАССА

#include <iostream>

using namespace std;

class Counter

{

public:

Counter(int sec)

{

seconds = sec;

}

void display()

{

cout << seconds << " seconds" << endl;

}

Counter operator + (Counter c2)

{

return Counter(this->seconds + c2.seconds);

}

int operator + (int s)

{

return this->seconds + s;

}

int seconds;

};

int main()

{

Counter c1(20);

Counter c2(10);

Counter c3 = c1 + c2;

c3.display(); // 30 seconds

int seconds = c1 + 25; // 45

return 0;

}

class Integer

{

private:

int value;

public:

Integer(int i) : value(i)

{}

const Integer operator+(const Integer& rv) const {

return (value + rv.value);

}

};

УНАРНЫЕ ОПЕРАТОРЫ

#include <iostream>

using namespace std;

class Integer

{

private:

int value;

public:

Integer(int i) : value(i)

{}

//унарный +

friend const Integer& operator+(const Integer& i);

//унарный -

friend const Integer operator-(const Integer& i);

//префиксный инкремент

friend const Integer& operator++(Integer& i);

//постфиксный инкремент

friend const Integer operator++(Integer& i, int);

//префиксный декремент

friend const Integer& operator--(Integer& i);

//постфиксный декремент

friend const Integer operator--(Integer& i, int);

};

//унарный плюс ничего не делает.

const Integer& operator+(const Integer& i) {

return i.value;

}

const Integer operator-(const Integer& i) {

return Integer(-i.value);

}

//префиксная версия возвращает значение после инкремента

const Integer& operator++(Integer& i) {

i.value++;

return i;

}

//постфиксная версия возвращает значение до инкремента

const Integer operator++(Integer& i, int) {

Integer oldValue(i.value);

i.value++;

return oldValue;

}

//префиксная версия возвращает значение после декремента

const Integer& operator--(Integer& i) {

i.value--;

return i;

}

//постфиксная версия возвращает значение до декремента

const Integer operator--(Integer& i, int) {

Integer oldValue(i.value);

i.value--;

return oldValue;

}

БИНАРНЫЕ ОПЕРАТОРЫ

#include <iostream>

using namespace std;

class Integer

{

private:

int value;

public:

Integer(int i) : value(i)

{}

friend const Integer operator+(const Integer& left, const Integer& right);

friend Integer& operator+=(Integer& left, const Integer& right);

friend bool operator==(const Integer& left, const Integer& right);

};

const Integer operator+(const Integer& left, const Integer& right) {

return Integer(left.value + right.value);

}

Integer& operator+=(Integer& left, const Integer& right) {

left.value += right.value;

return left;

}

bool operator==(const Integer& left, const Integer& right) {

return left.value == right.value;

}

**Перегрузка оператора через дружественную функцию**

Рассмотрим класс Point (точка на координатной оси). Определим для нее операцию сложения, и попробуем реализовать ее, используя перегрузку оператора сложения

class Point

{

public:

Point(int x, int y) : m\_x(x), m\_y(y) {};

friend Point operator+(const Point& left, const Point& rigth);

private:

int m\_x;

int m\_y;

};

Point operator+(const Point& left, const Point& rigth)

{

return Point(left.m\_x + rigth.m\_y, left.m\_y + rigth.m\_y);

}

**Перегрузка оператора через функцию**

Попробуем решить ту же задачу, используя обычные функции. Тогда для обращения к приватным полям m\_x и m\_y нам понадобится написать гетеры

class Point {

public:

Point(int x, int y) : m\_x(x), m\_y(y) {};

int get\_x() const { return m\_x; }

int get\_y() const { return m\_y; }

private:

int m\_x;

int m\_y;

};

Point operator+(const Point& left, const Point& rigth) {

return Point(left.get\_x() + rigth.get\_x(), left.get\_x() + rigth.get\_y());

}

**Перегрузка оператора через метод класса**

Также, есть способ реализовать оператор, как метод класса. Давайте рассмотрим на примере оператора +=

class Point {

public:

Point(int x, int y) : m\_x(x), m\_y(y) {};

int get\_x() const { return m\_x; }

int get\_y() const { return m\_y; }

Point operator+=(const Point& other);

private:

int m\_x;

int m\_y;

};

Point Point::operator+=(const Point& other) {

this->m\_x += other.get\_x();

this->m\_y += other.get\_y();

return \*this;

}