**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Дисциплина «**Объектно-Ориентированное Программирование**»

Отчет

Лабораторная работа №5

Выполнил: студент группы ПИ-2-21

Сатыбалдыев Нурислам

Проверил: Мусабаев Э. Б.

**Бишкек 2024**

**Лабораторная работа № 5**

**Тема: Перегрузка операторов**

**Задания:**

1. Типы данных полезны там, где ошибки могут быть вызваны арифметическим переполнением, которое не допустимо.

Создайте и откомпилируйте класс **Int**. Перегрузите четыре бинарных целочисленных арифметических операции (+, -, \*, /) и унарные операции постфиксной и префиксной форм инкремента с помощью внутренней операторной функции так, чтобы их можно было использовать для операций с объектами класса **Int**.

Если результат какой-либо из них выходит за границы типа **int** (в 32-битной системе), имеющее значения от **2 147 483 648** до **-2 147 483 648**, то операция должна послать сообщение об ошибке и завершить программу. Для выявления ошибки арифметического переполнения используйте концепцию **исключения**.

Для облегчения проверки переполнения выполняйте вычисления с использованием типа **long** **double**. При описании унарных операций используйте указатель **this**.

#include <iostream>

#include <limits>

#include <stdexcept>

class Int {

private:

int value;

public:

Int(int val = 0) : value(val) {}

Int operator+(const Int& other) const {

long double result = static\_cast<long double>(value) + other.value;

if (result > std::numeric\_limits<int>::max() || result < std::numeric\_limits<int>::min()) {

throw std::overflow\_error("Arithmetic overflow occurred.");

}

return Int(value + other.value);

}

Int operator-(const Int& other) const {

long double result = static\_cast<long double>(value) - other.value;

if (result > std::numeric\_limits<int>::max() || result < std::numeric\_limits<int>::min()) {

throw std::overflow\_error("Arithmetic overflow occurred.");

}

return Int(value - other.value);

}

Int operator\*(const Int& other) const {

long double result = static\_cast<long double>(value) \* other.value;

if (result > std::numeric\_limits<int>::max() || result < std::numeric\_limits<int>::min()) {

throw std::overflow\_error("Arithmetic overflow occurred.");

}

return Int(value \* other.value);

}

Int operator/(const Int& other) const {

if (other.value == 0) {

throw std::invalid\_argument("Division by zero.");

}

return Int(value / other.value);

}

Int& operator++() {

if (value == std::numeric\_limits<int>::max()) {

throw std::overflow\_error("Arithmetic overflow occurred.");

}

++value;

return \*this;

}

Int operator++(int) {

Int temp(\*this);

++(\*this);

return temp;

}

friend std::ostream& operator<<(std::ostream& out, const Int& obj) {

out << obj.value;

return out;

}

};

int main() {

// Пример использования класса Int

Int a(2), b(10), c;

try {

c = a + b ;

std::cout << "a + b = " << c << std::endl;

// Проверка арифметического переполнения

//c = Int(std::numeric\_limits<int>::max()) + Int(1); // Вызовет исключение

}

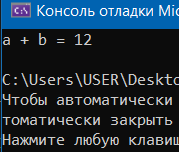
catch (const std::exception& e) {

std::cerr << "Exception caught: " << e.what() << std::endl;

}

return 0;

}



1. Опишите класс **fraction**, у которого есть одно закрытое целочисленное поле **chislo** типа **double**. Перегрузите для этого класса арифметические операции **сложения**, **вычитания**, **умножения** и **деления** так, чтобы они могли оперировать как с объектами класса, так и с числами (то есть выполнять, например, не только действие 3/4 +2/5, но и 1/2 + 4 или 2\* 5/6). Также перегрузите унарную операцию инкремента в префиксной или постфиксной форме увеличения дроби. Продемонстрируйте работу класса. Используйте конструктор по умолчанию и конструктор с одни аргументом для инициализации поля класса.

#include <iostream>

class Fraction {

private:

double chislo;

public:

double get() const {

return this->chislo;

}

Fraction() : chislo(0.0) {}

Fraction(double num) : chislo(num) {}

// Binary arithmetic operators

Fraction operator+(const Fraction& other) const {

return Fraction(chislo + other.chislo);

}

Fraction operator-(const Fraction& other) const {

return Fraction(chislo - other.chislo);

}

Fraction operator\*(const Fraction& other) const {

return Fraction(chislo \* other.chislo);

}

Fraction operator/(const Fraction& other) const {

if (other.chislo == 0.0) {

throw std::invalid\_argument("Division by zero");

}

return Fraction(chislo / other.chislo);

}

// Prefix increment operator

Fraction& operator++() {

++chislo;

return \*this;

}

// Postfix increment operator

Fraction operator++(int) {

Fraction temp = \*this;

++(\*this);

return temp;

}

void display() const {

std::cout << "Value: " << chislo << std::endl;

}

};

Fraction operator+( double a ,const Fraction& other) {

return Fraction( a+other.get());

}

Fraction operator-(double a, const Fraction& other) {

return Fraction(a - other.get());

}

Fraction operator/(double a, const Fraction& other) {

if (other.get() == 0.0) {

throw std::invalid\_argument("Division by zero");

}

return Fraction(a / other.get());

}

Fraction operator\*(double a, const Fraction& other) {

return Fraction(a \* other.get());

}

int main() {

Fraction a(0.5);

Fraction b(0.25);

Fraction c = 2 + a + b + 2;

c.display();

Fraction d = a - b;

d.display();

Fraction e = a \* b;

e.display();

Fraction f = a / b;

f.display();

++a;

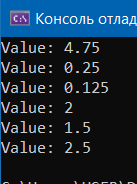
a.display();

a++;

a.display();

return 0;

}



1. Создать класс, в котором перегружается метод **rect\_area()**.

Этот метод возвращает площадь прямоугольника. В этой программе метод **rect\_area()** перегружается двумя способами. В первом — методу передаются оба размера фигуры. Эта версия используется для прямоугольника. Однако, в случае квадрата необходимо задавать только один аргумент, поэтому вызывается вторая версия метода **rect\_area().**

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double rect\_area() const {

return length \* width;

}

// Overloaded rect\_area() method for square

double rect\_area(double side) const {

return side \* side;

}

};

int main() {

Rectangle rectangle(3.0, 4.0);

std::cout << "Rectangle area: " << rectangle.rect\_area() << std::endl;

// For square

std::cout << "Square area: " << rectangle.rect\_area(5.0) << std::endl;

return 0;

}

Изображение выглядит как текст, Шрифт, снимок экрана, Цвет электрик

Автоматически созданное описание