**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИТЕТ**

**им. И.Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра «**Программное обеспечение компьютерных систем**»

Направление: 710400 «**Программная инженерия**»

Дисциплина: ««**Объектно-ориентированное программирование**»»

ОТЧЕТ

Лабораторная работа №5

Выполнил: Тойчубекова Айчурок

Группа: ПИ-2-21

Проверил: Мусабаев Э. Б.

Бишкек – 2024

**Задание №1**

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**.

**Код программы:**

#include <iostream>

using namespace std;

class Int {

private:

int value;

public:

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

Int operator+(const Int& other) const {

return checkOverflow(value, other.value, '+');

}

Int operator-(const Int& other) const {

return checkOverflow(value, other.value, '-');

}

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

return checkOverflow(value, other.value, '\*');

}

Int operator/(const Int& other) const {

if (other.value == 0) {

cerr << "Ошибка деление на 0!!!\n";

exit(EXIT\_FAILURE);

}

return Int(value / other.value);

}

Int& operator++() {

return \*this = checkOverflow(value, 1, '+');

}

Int operator++(int) {

Int temp = \*this;

\*this = checkOverflow(value, 1, '+');

return temp;

}

Int checkOverflow(int a, int b, char op) const {

long double result;

switch (op) {

case '+':

result = static\_cast<long double>(a) + static\_cast<long double>(b);

break;

case '-':

result = static\_cast<long double>(a) - static\_cast<long double>(b);

break;

case '\*':

result = static\_cast<long double>(a) \* static\_cast<long double>(b);

break;

default:

cerr << "Ошибка не верный оператор!!!\n";

exit(EXIT\_FAILURE);

}

const int INT\_MAX\_VALUE = 2147483647;

const int INT\_MIN\_VALUE = -2147483648;

if ((result > INT\_MAX\_VALUE && b > 0) || (result < INT\_MIN\_VALUE && b < 0)) {

cerr << "Ошибка арифметическое переполнение!!!\n";

exit(EXIT\_FAILURE);

}

return Int(static\_cast<int>(result));

}

friend ostream& operator<<(ostream& os, const Int& integer);

};

ostream& operator<<(ostream& os, const Int& integer) {

os << integer.value;

return os;

}

int main() {

setlocale(LC\_ALL, "rus");

Int a = 5;

Int b = 0;

Int c = a + b;

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

/\*c = a - b;

cout << "a - b = " << c << endl;

c = a / b;

cout << "a / b = " << c << endl;

c = b / a;

cout << "b / a = " << c << endl;\*/

return 0;

}

**Результат:**







**Задание №2**

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.

**Код программы:**

#include <iostream>

using namespace std;

class Fraction {

private:

double chislo;

public:

Fraction(double value = 0.0) : chislo(value) {}

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) {

cerr << "Ошибка: деление на ноль!\n";

exit(EXIT\_FAILURE);

}

return Fraction(chislo / other.chislo);

}

Fraction& operator++() {

chislo++;

return \*this;

}

Fraction operator++(int) {

Fraction temp = \*this;

chislo++;

return temp;

}

void print() const {

cout << chislo;

}

};

int main() {

Fraction a(3.5);

Fraction b(2.5);

Fraction c = a + b/a;

cout << "a + b = ";

c.print();

cout << endl;

Fraction d = a \* 2;

cout << "a \* 2 = ";

d.print();

cout << endl;

Fraction e = ++a;

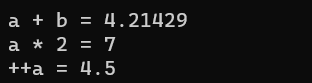
cout << "++a = ";

e.print();

return 0;

}

**Результат:**



**Задание №3**

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().**

**Код программы:**

#include <iostream>

using namespace std;

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l = 0.0, double w = 0.0) : length(l), width(w) {}

double rect\_area(double l, double w) const {

return l \* w;

}

double rect\_area(double side) const {

return side \* side;

}

};

int main() {

setlocale(LC\_ALL, "rus");

Rectangle rectangle;

cout << "Площадь прямоугольника: " << rectangle.rect\_area(4.0, 5.0) << endl;

cout << "Площадь квадрата: " << rectangle.rect\_area(3.0) << endl;

return 0;

}

**Результат:**

****

**Ответы на вопросы**

**Вопросы:**  
1)Полиморфизм: Полиморфизм в объектно-ориентированном программировании - это способность объекта использовать методы базового класса таким образом, чтобы они были адаптированы под конкретный производный класс. Это может быть достигнуто через виртуальные функции и интерфейсы.

2)Перегрузка операторов: Перегрузка операторов - это возможность определения собственного поведения для стандартных операторов языка программирования в контексте пользовательских типов данных. Это делает код более читаемым и удобным, поддерживая привычный синтаксис.

3)Переопределение операторов: Переопределение операторов - это предоставление новой реализации для существующего оператора. В отличие от перегрузки, где добавляется поддержка для пользовательских типов, переопределение меняет поведение оператора для существующих типов.

4) Синтаксис операторной функции: Синтаксис операторной функции в C++ выглядит следующим образом:

cppCopy code

тип\_возвращаемого\_значения operator оператор(параметры) { // реализация оператора }

5) Операторы, которые можно перегружать, и те, которые нельзя: Можно перегружать многие бинарные и унарные операторы, такие как +, -, \*, /, ==, != и др. Нельзя перегружать операторы, такие как ::, .\*, ., ?:.

6) Зависимость между количеством аргументов в операторной функции и количеством операндов: Количество аргументов в операторной функции зависит от количества операндов, с которыми оператор работает. Например, бинарные операторы требуют два аргумента (левый и правый операнд), унарные - один аргумент.

7) Как перегруженные операции делают код более читаемым: Перегруженные операции позволяют использовать привычный синтаксис для пользовательских типов данных, делая код более читаемым и естественным. Например, перегрузка оператора + для класса может позволить использовать выражение obj1 + obj2 вместо вызова функции add(obj1, obj2).