Пермский национальный исследовательский политехнический университет

**Лабораторная работа №3**

**Дисциплина: Информатика**

**Вариант № 9**

**"Перегрузка операций"**

Выполнил: Зайченко Никита Геннадьевич

Проверила: Доцент кафедры ИТАС Полякова О. А.

Пермь 2022

**Содержание отчета**

1) Постановка задачи (общая и конкретного варианта).

2) Описание класса.

3) Определение компонентных функций.

4) Определение глобальных функций.

5) Функция main().

6) Объяснение результатов работы программы.

7) Ответы на контрольные вопросы

**Постановка задачи (общая и конкретного варианта)**

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Использование перегруженных операций в классах.3) Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать:  вычитание дробного числа из суммы  операции сравнения (==, !=)

**Описание класса.**

class Money

{

long rub; int cop;

public:

Money() { rub = 0; cop = 0; }; //конструктор без параметров

Money(long r, int c) { rub = r; cop = c; } //конструктор с параметрами

Money(const Money& m) { rub = m.rub; cop = m.cop; } //конструктор копирования

~Money() {}; //деструктор

int get\_rub() { return rub; } //селектор

int get\_cop() { return cop; } //селектор

void set\_rub(long r) { rub = r; } //модификатор

void set\_cop(int c) { cop = c; } //модификатор

//перегруженные операции

Money& operator=(const Money&); //перегрузка операции присваивания

Money& operator++(); //перегрузка префиксной операции инкремент

Money operator++(int); //постфиксная операция

Money operator+(const Money&); //перегрузка бинарной операции сложения

Money operator-(const Money&); //перегрузка бинарной операции вычитания

bool operator==(const Money&); //перегрузка операции сравнения

bool operator!=(const Money&); //перегрузка операции сравнения

//глобальные функции ввода-вывода

friend istream& operator>>(istream& in, Money& m);

friend ostream& operator<<(ostream& out, const Money& m);

};



**Определение компонентных функций.**

//перегрузка операции присваивания

Money& Money::operator=(const Money& m)

{

//проверка на самоприсваивание

if (&m == this) return \*this;

rub = m.rub;

cop = m.cop;

return \*this;

}

//перегрузка префиксной операции инкремент

Money& Money::operator++()

{

long e = rub + (cop / 100);

rub = e;

cop = cop % 100;

return \*this;

}

//перегрузка постфиксной операции инкремент

Money Money::operator ++(int)

{

long e = rub + (cop / 100);

Money t(rub, cop);

rub = e;

cop = cop % 100;

return t;

}

//перегрузка бинарной операции сложения

Money Money::operator+(const Money& m)

{

long rubs = rub + m.rub;

int cops = cop + m.cop;

if (cops >= 100) { // обработка случая, когда cop >= 100

rubs += 1;

cops -= 100;

}

Money p(rubs,cops);

return p;

}

//перегрузка бинарной операции вычитания

Money Money::operator-(const Money& m)

{

double total = rub + cop / 100.0 - (m.rub + m.cop / 100.0);

long rub = static\_cast<long>(total); // static\_cast: оператор преобразования типов данных

int cop = static\_cast<int>((total - rub) \* 100);

if (cop < 0) { // обработка случая, когда cop < 0

rub -= 1;

cop += 100;

}

if (rub < 0) { // обработка случая, когда rub < 0

rub = 0;

cop = 0;

}

Money p(rub, cop);

return p;

}

//перегрузка операции сравнения ==

bool Money::operator==(const Money& m)

{

if (rub == m.rub && cop == m.cop){ return true; }

else { return false; }

}

//перегрузка операции сравнения !=

bool Money::operator!=(const Money& m)

{

if (rub != m.rub || cop != m.cop) { return true; }

else { return false; }

}

**Определение глобальных функций.**

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Money& m)

{

cout << "Введите рубли = "; in >> m.rub;

cout << "Введите копейки = "; in >> m.cop;

cout << endl;

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Money& m)

{

return (out << m.rub << "," << m.cop);

}

**Функция main().**

void main()

{

setlocale(LC\_ALL, "Russian");

Money a; //конструктор без параметров

Money b; //конструктор c параметрами

b.set\_rub(12902346);

b.set\_cop(534);

Money c = b; //конструктор копирования

c.set\_rub(12902346);

c.set\_cop(534);

cin >> a; //ввод переменной

++a; //префиксная операция инкремент

++b; //префиксная операция инкремент

++c; //префиксная операция инкремент

Money d;

d = (a++) + b + c; //сложение и постфиксная операция инкремент

cout << "a = " << a << endl; //вывод переменной

cout << "b = " << b << endl; //вывод переменной

cout << "c = " << c << endl << endl; //вывод переменной

cout << "Сумма всех значений d = " << d << endl; //вывод переменной

d = d - b - a; //операция вычитания

cout << "Сумма после вычитания d = " << d << endl << endl; //вывод переменной

if (b == c) {cout << "Числа b и с равны" << endl;}

else { cout << "Числа b и с не равны" << endl;}

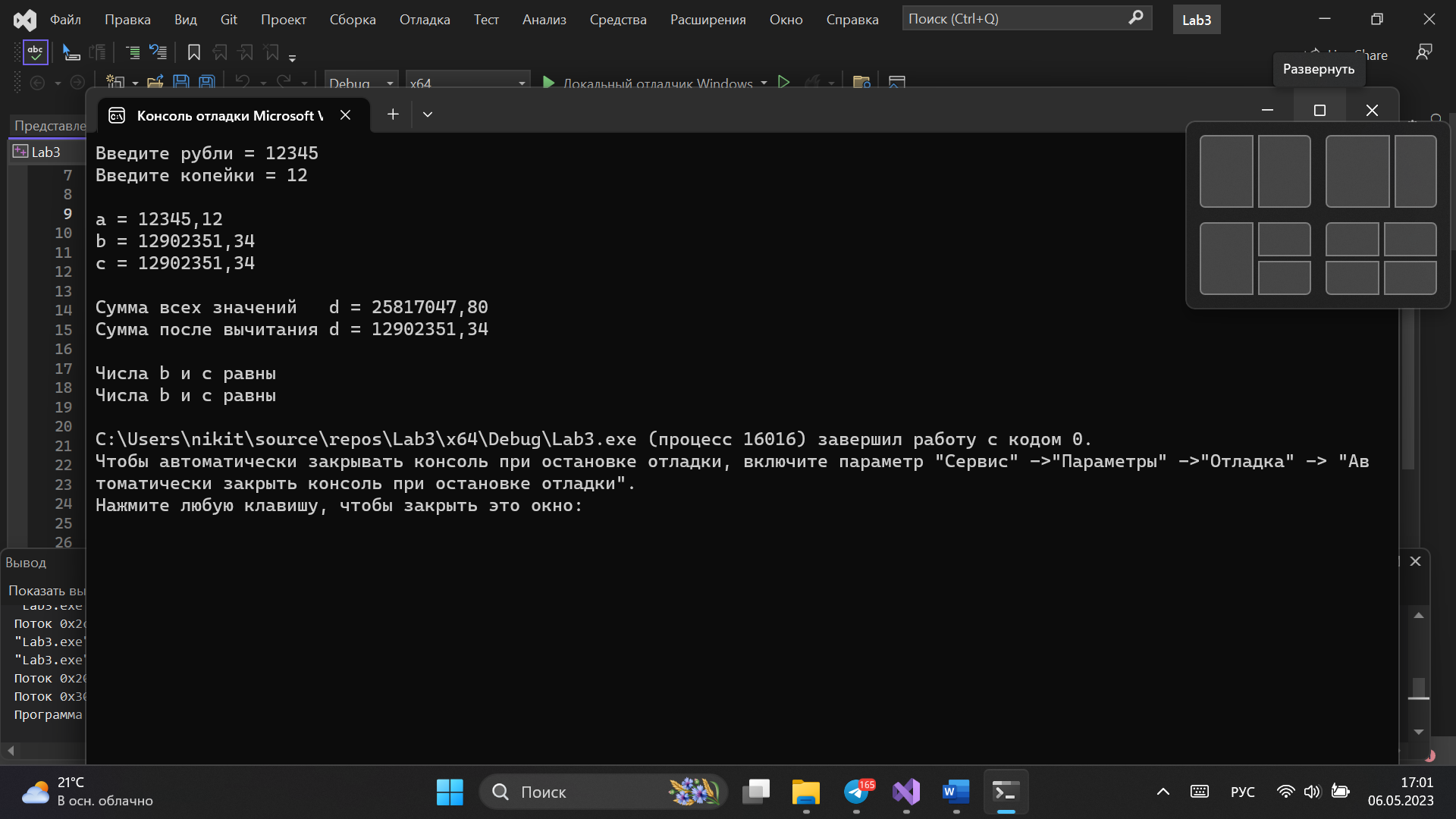
if (b != c) { cout << "Числа b и с не равны" << endl; }

else { cout << "Числа b и с равны" << endl; }

}

**Объяснение результатов работы программы.**

Данный код представляет собой определение класса Money для работы с денежными суммами, используя перегрузку операторов, реализовано сравнение, сложение, префиксные операции и постфиксные операции. Таким образом, класс Money позволяет создавать объекты для хранения денежных сумм и выполнять над ними различные операции с помощью перегруженных операторов и методов класса.



**Ответы на контрольные вопросы**

**1. Для чего используются дружественные функции и классы?**

Дружественные функции и классы используются для предоставления доступа к приватным и защищенным атрибутам и методам класса или для реализации операций, которые логически связаны с данным классом, но не являются его методами.

Дружественные функции или классы объявляются с помощью ключевого слова friend. Класс или функция, объявленные дружественными, имеют доступ к приватным и защищенным членам класса, как если бы они были определены внутри класса.

Примеры использования дружественных функций или классов могут включать операции ввода-вывода объектов класса, перегрузку операторов, реализацию функций, связанных с данным классом (например, класс Array может иметь дружественную функцию сортировки массива).

**2. Сформулировать правила описания и особенности дружественных функций.**

Правила описания дружественных функций:

- Название функции должно ясно отражать ее назначение.

- Входные параметры функции должны быть определены и описаны как можно более точно.

- Возвращаемое значение должно быть описано и должно быть понятно, что данная функция выполняет.

**3. Каким образом можно перегрузить унарные операции?**

Можно перегрузить унарные операции путем определения специальных методов с именами, соответствующими операторам. Для каждого оператора существует специальный метод, который вызывается во время выполнения программы, когда этот оператор применяется к объекту.

class Person

{

string name;

int age;

public:

Person(string, int);//конструктор

.....

//компонентная функция

Person& operator ++() //префиксная операция

{

++age;

return \*this; //указатель на объект, вызвавший метод

}

};

**4. Сколько операндов должна иметь унарная функция-операция, определяемая внутри класса?**

Унарная функция-операция определяется внутри класса для перегрузки унарного оператора. Унарный оператор - это оператор, который действует только на один операнд. Соответственно, унарная функция-операция должна принимать только один параметр

**5. Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?**

Унарная функция-операция, определяемая вне класса, должна иметь один операнд.

**6. Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?**

Бинарная функция-операция, определяемая внутри класса, должна иметь один параметр - второй операнд будет передаваться в качестве аргумента в функцию.

**7. Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?**

Бинарная функция-операция, определенная вне класса, должна иметь два операнда.

**8. Чем отличается перегрузка префиксных и постфиксных унарных операций?**

Префиксная унарная операция выполняется перед операцией, постфиксная - после. Также, префиксная операция возвращает изменённое значение, в то время как постфиксная - старое значение до изменения. ++a / a++.

**9. Каким образом можно перегрузить операцию присваивания?**

Операция присваивания может быть перегружена с помощью оператора "=" и функции-операции.

**10. Что должна возвращать операция присваивания?**

Операция присваивания должна возвращать ссылку на объект, который присваивается.

**11. Каким образом можно перегрузить операции ввода-вывода?**

Операции ввода-вывода могут быть перегружены с помощью операторов ">>" и "<<". Также нужно определить соответствующие функции-операции для каждой из этих операций.

**12. В программе описан класс**

**class Student**

**{…**

**Student& operator++(); …. };**

**и определен объект этого класса Student s;**

**Выполняется операция ++s;**

**Каким образом, компилятор будет воспринимать вызов функции-операции?**

Компилятор будет воспринимать операцию как вызов префиксной функции-операции инкремента для объекта s.

**13. В программе описан класс**

**class Student**

**{…**

**friend Student& operator ++( Student&); …. };**

**и определен объект этого класса Student s;**

**Выполняется операция ++s;**

**Каким образом, компилятор будет воспринимать вызов функции-операции?**

Компилятор будет воспринимать операцию как вызов префиксной дружественной функции-операции инкремента для объекта s.

**14. В программе описан класс**

**class Student**

**{…**

**bool operator<(Student &P); …. };**

**и определены объекты этого класса**

**Student a,b;**

**Выполняется операция cout<<a<b;**

**Каким образом, компилятор будет воспринимать вызов функции-операции?**

Компилятор будет воспринимать операцию как вызов функции-операции сравнения "меньше" для объектов a и b, и результат выполнения данной операции будет передан в функцию cout.

**15. В программе описан класс**

**class Student**

**{…**

**friend bool operator >(const Person&, Person&) …. };**

**и определены объекты этого класса Student a,b;**

**Выполняется операция cout<<a>b;**

**Каким образом, компилятор будет воспринимать вызов функции-операции?**

Компилятор будет воспринимать операцию как вызов дружественной функции-операции сравнения "больше" для объектов a и b, и результат выполнения данной операции будет передан в функцию cout.