Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

**Лабораторная работа №3.**

**«Перегрузка операций.»**

Выполнил студент гр. РИС-24-1б

Чижов Денис Николаевич

Проверил:

Доц. Каф. ИТАС 

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2025

**Задача:**

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции указанные в варианте.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

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:

 сложение суммы и дробного числа

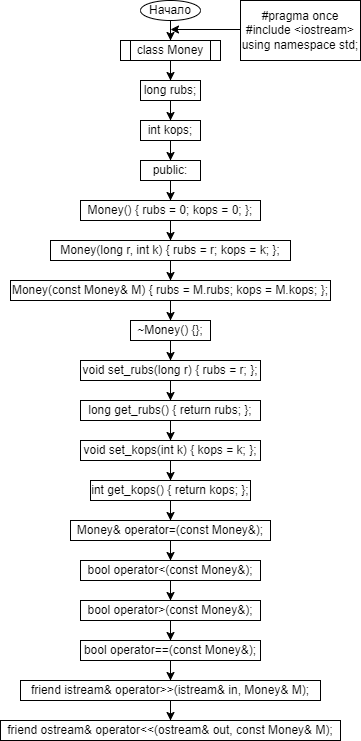
 операции сравнения (>, <, ==).

**Анализ:**

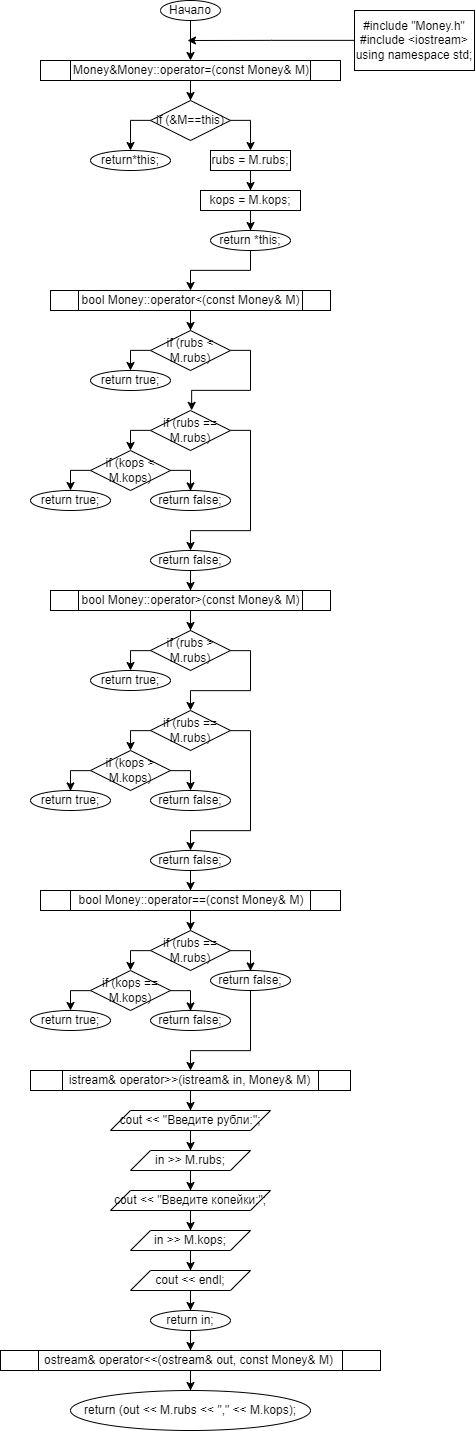
1. **Класс Money**
   * Представляет денежную сумму в рублях и копейках.
   * Поля:
     + long rubs – рубли (целая часть).
     + int kops – копейки (дробная часть).
2. **Конструкторы**
   * Без параметров (Money()) – инициализирует нулями.
   * С параметрами (Money(long, int)) – задаёт конкретные значения.
   * Копирования (Money(const Money&)) – копирует данные из другого объекта.
3. **Деструктор**
   * Пустой (~Money()), так как нет динамической памяти.
4. **Селекторы и модификаторы**
   * Позволяют получать и изменять значения полей (get\_rubs(), set\_kops() и т.д.).
5. **Перегрузка операторов**
   * Присваивания (=), сравнения (<, >, ==), ввода/вывода (>>, <<).
   * Операторы сравнения корректно учитывают и рубли, и копейки.
6. **Демонстрация в main()**
   * Создание объектов (a, b, c).
   * Ввод значений через перегруженный >>.
   * Сравнение объектов (>, <, ==).
   * Копирование (c = a) и вывод (<<).
7. **Особенности**
   * Вывод формата "рубли,копейки" (например, 100,50).
   * Проверка на самоприсваивание в операторе =.

**Блок-схема:**

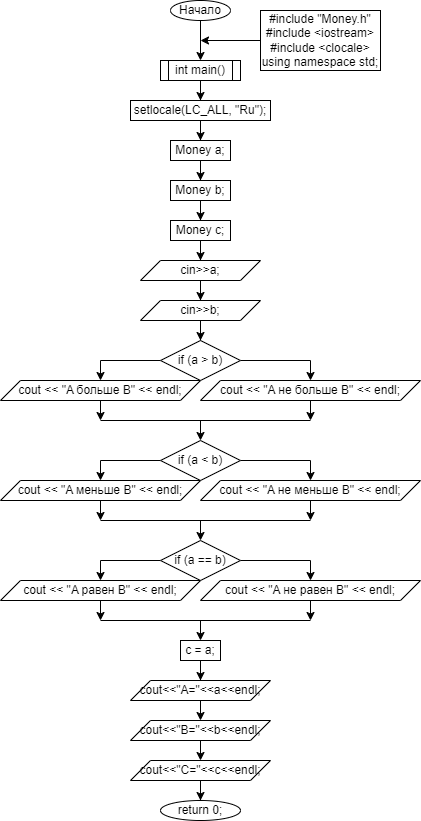
Money.h:



Money.cpp:



Main.cpp:



**Код:**

Money.h:

#pragma once

#include <iostream>

using namespace std;

class Money {

long rubs;

int kops;

public:

//Конструкторы

Money() { rubs = 0; kops = 0; }; //Конструктор без параметров

Money(long r, int k) { rubs = r; kops = k; };//Конструктор с параметрами

Money(const Money& M) { rubs = M.rubs; kops = M.kops; }; //Конструктор копирования

//Деструктор

~Money() {};

//Селекторы и модификаторы

void set\_rubs(long r) { rubs = r; }; //модификатор rubs

long get\_rubs() { return rubs; }; //селектор rubs

void set\_kops(int k) { kops = k; }; //модификатор kops

int get\_kops() { return kops; }; //селектор kops

//Перегрузки операторов

Money& operator=(const Money&);

bool operator<(const Money&);

bool operator>(const Money&);

bool operator==(const Money&);

//Глобальные функции ввода-вывода

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

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

};

Money.cpp:

#include "Money.h"

#include <iostream>

using namespace std;

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

{

//проверка на самоприсваивание

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

rubs = M.rubs;

kops = M.kops;

return \*this;

}

bool Money::operator<(const Money& M)

{

if (rubs < M.rubs)return true;

if (rubs == M.rubs) {

if (kops < M.kops) {

return true;

}

else return false;

}

return false;

}

bool Money::operator>(const Money& M)

{

if (rubs > M.rubs)return true;

if (rubs == M.rubs) {

if (kops > M.kops) {

return true;

}

else return false;

}

return false;

}

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

{

if (rubs == M.rubs) {

if (kops == M.kops) {

return true;

}

else return false;

}

else return false;

}

istream& operator>>(istream& in, Money& M) {

cout << "Введите рубли:";

in >> M.rubs;

cout << "Введите копейки:";

in >> M.kops;

cout << endl;

return in;

}

ostream& operator<<(ostream& out, const Money& M) {

return (out << M.rubs << "," << M.kops);

}

Main.cpp:

#include "Money.h"

#include <iostream>

#include <clocale>

using namespace std;

int main() {

setlocale(LC\_ALL, "Ru");

Money a;//конструктор без параметров

Money b; //конструктор без параметров

Money c; //конструктор без параметров

cin>>a;//ввод переменной

cin>>b;//ввод переменной

if (a > b) {

cout << "A больше B" << endl;

}

else {

cout << "A не больше B" << endl;

}

if (a < b) {

cout << "A меньше B" << endl;

}

else {

cout << "A не меньше B" << endl;

}

if (a == b) {

cout << "A равен B" << endl;

}

else {

cout << "A не равен B" << endl;

}

c = a;

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

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

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

return 0;

}

**Вывод:**

Программа работает. Задача решена.

**Вопросы:**

1. **Для чего используются дружественные функции и классы?**  
   Дружественные функции и классы используются для предоставления доступа к закрытым (private) и защищенным (protected) членам класса извне. Они объявляются внутри класса с ключевым словом friend и не являются членами класса, но могут обращаться к его приватным данным.
2. **Сформулировать правила описания и особенности дружественных функций.**
   * Дружественная функция объявляется внутри класса с ключевым словом friend.
   * Она не является членом класса и не имеет указателя this.
   * Может быть определена как внутри, так и вне класса.
   * Не наследуется в производных классах.
   * Может быть обычной функцией или функцией-оператором.
3. **Каким образом можно перегрузить унарные операции?**  
   Унарные операции можно перегрузить:
   * Как метод класса (без аргументов, работает с текущим объектом через this).
   * Как дружественную функцию (с одним аргументом — объектом, к которому применяется операция).
4. **Сколько операндов должна иметь унарная функция-операция, определяемая внутри класса?**  
   Унарная функция-операция, определяемая внутри класса, не должна иметь явных операндов (работает с текущим объектом через this).
5. **Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?**  
   Унарная функция-операция, определяемая вне класса, должна иметь один операнд (объект, к которому применяется операция).
6. **Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?**  
   Бинарная функция-операция, определяемая внутри класса, должна иметь один явный операнд (второй — текущий объект через this).
7. **Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?**  
   Бинарная функция-операция, определяемая вне класса, должна иметь два операнда.
8. **Чем отличается перегрузка префиксных и постфиксных унарных операций?**
   * Префиксная перегрузка (например, ++obj) объявляется без параметров (operator++()).
   * Постфиксная перегрузка (например, obj++) принимает фиктивный параметр int (operator++(int)).
9. **Каким образом можно перегрузить операцию присваивания?**  
   Операция присваивания (=) перегружается только как метод класса:

ClassName& operator=(const ClassName& other);

1. **Что должна возвращать операция присваивания?**  
   Операция присваивания должна возвращать ссылку на текущий объект (\*this), чтобы поддерживать цепочку присваиваний (a = b = c).
2. **Каким образом можно перегрузить операции ввода-вывода?**  
   Операции ввода (>>) и вывода (<<) перегружаются как дружественные функции:  
   friend ostream& operator<<(ostream& os, const ClassName& obj);  
   friend istream& operator>>(istream& is, ClassName& obj);
3. **Пример с Student& operator++(); и вызовом ++s:**  
   Компилятор воспримет вызов как s.operator++(); (вызов метода класса).
4. **Пример с friend Student& operator++(Student&); и вызовом ++s:**  
   Компилятор воспримет вызов как operator++(s); (вызов дружественной функции).
5. **Пример с bool operator<(Student &P); и вызовом cout << a < b:**  
   Компилятор воспримет вызов как a.operator<(b), но синтаксис cout << a < b ошибочен (нужны скобки: cout << (a < b)).
6. **Пример с friend bool operator>(const Person&, Person&) и вызовом cout << a > b:**  
   Компилятор воспримет вызов как operator>(a, b), но синтаксис cout << a > b ошибочен (нужны скобки: cout << (a > b)).