Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №3

Тема: «Перегрузка операций.»

Выполнил

Студент группы РИС-22-2б

Радостев И.А.

Проверил доц. Кафедры ИТАС

Полякова Ольга Андреевна

Пермь 2023

# Постановка задачи

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).

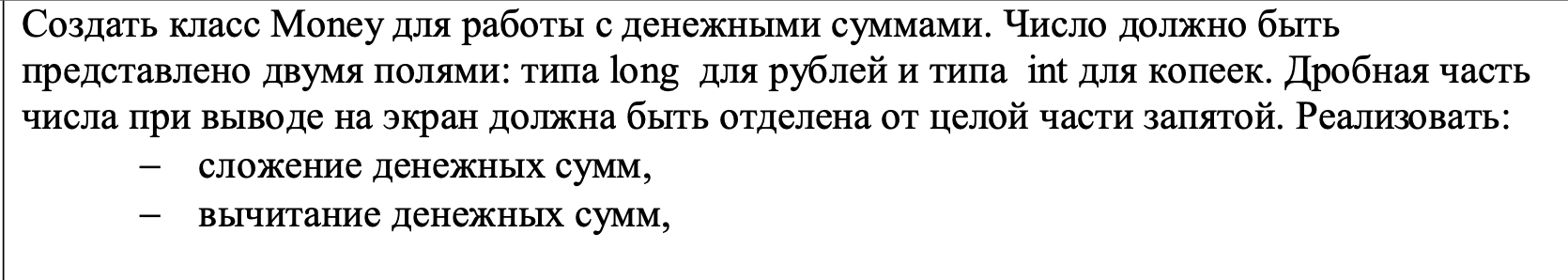
5. Перегрузить операцию присваивания.

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.

ВАРИАНТ 6:



**Контрольные вопросы**

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

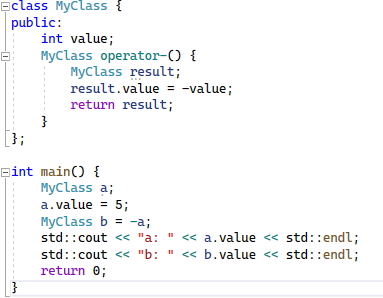
Дружественные функции и классы используются для предоставления доступа к закрытым (private) членам класса другим функциям и классам, что позволяет избежать использования глобальных переменных и упрощает реализацию сложных алгоритмов.

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

Правила и особенности описания дружественных функций включают объявление функции с ключевым словом `friend` внутри класса, с определением самой функции вне класса. Дружественная функция не является методом класса и не может обращаться к ключевому слову `this`. С помощью дружественных функций можно улучшить инкапсуляцию, увеличить удобство использования класса и повысить его гибкость.

*3. Каким образом можно перегрузить унарные операции?*

Унарные операции можно перегрузить с помощью перегрузки операторов в классе.



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

Унарная функция-операция должна иметь один операнд.

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

Унарная функция-операция должна иметь один операнд.

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

Бинарная функция-операция должна иметь два операнда.

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

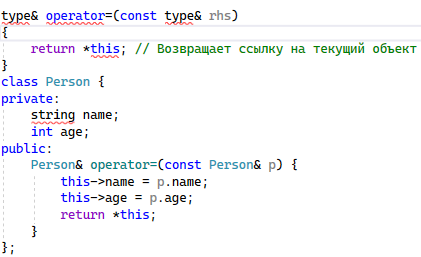
Бинарная функция-операция должна иметь два операнда.

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

Перегрузка префиксных унарных операций выполняется непосредственно перед операцией, а постфиксных - непосредственно после операции.

*9. Каким образом можно перегрузить операцию присваивания?*

Операция присваивания может быть перегружена в классе через перегрузку оператора "=" для определения собственной логики копирования одного объекта в другой.

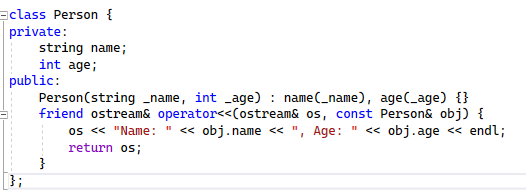


*10. Что должна возвращать операция присваивания?*

Операция присваивания должна возвращать значение, присвоенное переменной.

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

Операции ввода-вывода можно перегрузить путем определения функций операторов << и >> для пользовательских типов данных.



*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 вызов функции-операции как инкремент объекта s. Это будет эквивалентно вызову функции-операции вида s.operator++(), где s - объект класса Student.

*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 вызов функции-операции как унарный оператор инкремента (++). Так как в определении класса указан friend для функции-операции, она может получить доступ к закрытым членам класса и изменить их значение после инкремента. В результате вызова ++s, значение s будет увеличено на 1.

*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 вызов функции-операции как сравнение объектов класса Student по определенным критериям, заданным в функции-операции `<`. Результатом операции будет значение типа bool, которое будет выведено с помощью оператора `<<` в поток вывода данных.

*15. В программе описан класс*

*class Money*

*{*

*…*

*friend bool operator >(const Person&, Person&)*

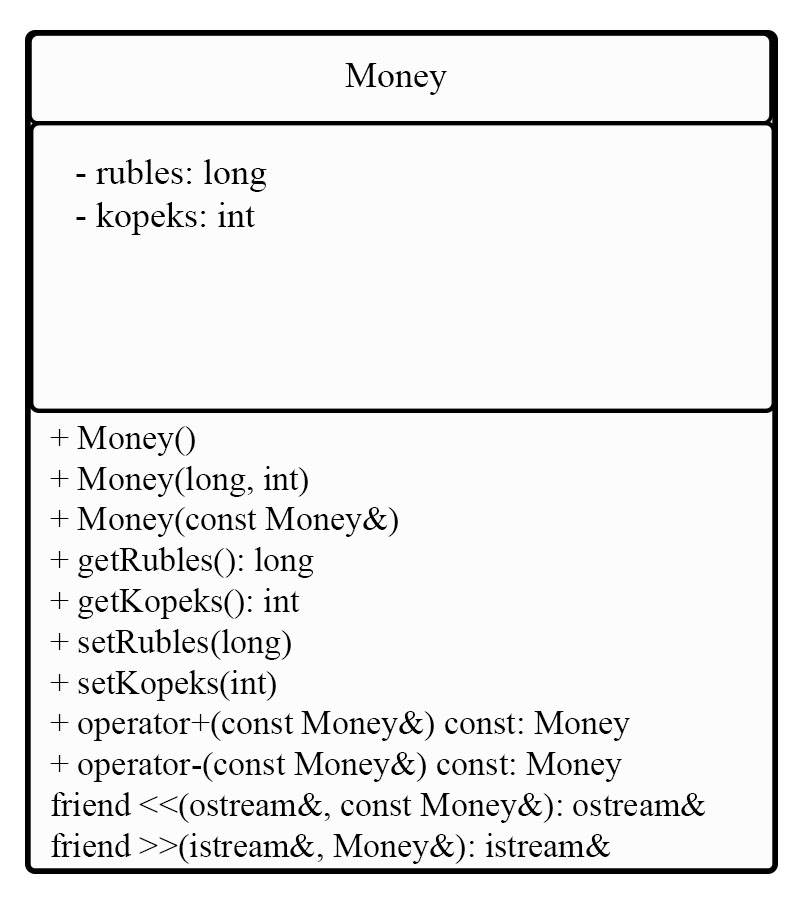
*….*

*};*

*и определены объекты этого класса Student a,b; Выполняется операция cout<<a>b; Каким образом, компилятор будет воспринимать вызов функции-операции?*

Компилятор будет воспринимать вызов функции-операции ">" как бинарную операцию с двумя аргументами: объектом "a" класса Student и объектом "b" класса Student. Поскольку эта функция-операция определена как friend в классе Student, то она имеет доступ к приватным членам класса и может сравнивать объекты по необходимым свойствам. Результатом выполнения операции будет логическое значение (true или false), которое потом будет передано в объект cout для печати на экране.

UML таблица



Код программы

Money.h:

#pragma once // предотвращает множественные включения заголовочного файла

#include <iostream> // подключение стандартной библиотеки ввода-вывода

class Money { // определение класса Money

private: // модификатор доступа private

long rubles; // поле для хранения рублей

int kopeks; // поле для хранения копеек

public: // модификатор доступа public

Money(); // конструктор по умолчанию

Money(long r, int k); // конструктор с параметрами

Money(const Money& other); // конструктор копирования

long getRubles() const; // метод получения рублей

int getKopeks() const; // метод получения копеек

void setRubles(long r); // метод установки рублей

void setKopeks(int k); // метод установки копеек

Money operator+ (const Money& other) const; // перегрузка оператора сложения

Money operator- (const Money& other) const; // перегрузка оператора вычитания

friend std::ostream& operator<< (std::ostream& out, const Money& m); // перегрузка оператора вывода в поток

friend std::istream& operator>> (std::istream& is, Money& m); // перегрузка оператора ввода из потока

};

Money.cpp:

#include "Money.h" // подключение заголовочного файла

Money::Money() { // реализация конструктора по умолчанию

rubles = 0;

kopeks = 0;

}

Money::Money(long r, int k) { // реализация конструктора с параметрами

rubles = r;

kopeks = k;

}

Money::Money(const Money& other) { // реализация конструктора копирования

rubles = other.rubles;

kopeks = other.kopeks;

}

long Money::getRubles() const { // реализация метода получения рублей

return rubles;

}

int Money::getKopeks() const { // реализация метода получения копеек

return kopeks;

}

void Money::setRubles(long r) { // реализация метода установки рублей

rubles = r;

}

void Money::setKopeks(int k) { // реализация метода установки копеек

kopeks = k;

}

Money Money::operator+ (const Money& other) const { // реализация перегрузки оператора сложения

long r = rubles + other.rubles;

int k = kopeks + other.kopeks;

if (k >= 100) {

r += k / 100;

k %= 100;

}

return Money(r, k);

}

Money Money::operator- (const Money& other) const { // реализация перегрузки оператора вычитания

long r = rubles - other.rubles;

int k = kopeks - other.kopeks;

if (k < 0) {

r -= 1;

k += 100;

}

return Money(r, k);

}

std::ostream& operator<< (std::ostream& out, const Money& m) { // реализация перегрузки оператора вывода в поток

out << m.rubles << "," << m.kopeks;

return out;

}

std::istream& operator>> (std::istream& is, Money& m) { // реализация перегрузки оператора ввода из потока

long r;

int k;

char comma;

is >> r >> comma >> k;

m.setRubles(r);

m.setKopeks(k);

return is;

}

main.cpp:

#include <iostream> // подключение стандартной библиотеки ввода-вывода

#include "Money.h" // подключение заголовочного файла

using namespace std; // использование пространства имен std

int main() { // точка входа в программу

Money m1, m2; // создание двух объектов класса Money

cout << "Введите первую сумму: "; // вывод на экран приглашения к вводу

cin >> m1; // ввод первой суммы с клавиатуры

cout << "Введите вторую сумму: "; // вывод на экран приглашения к вводу

cin >> m2; // ввод второй суммы с клавиатуры

Money sum = m1 + m2; // сложение двух сумм

cout << "Сумма: " << sum << endl; // вывод результата на экран

Money diff = m1 - m2; // вычитание двух сумм

cout << "Разность: " << diff << endl; // вывод результата на экран

return 0; // завершение программы

}