Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Перегрузка операций»

Выполнил работу

Студент группы РИС-22-1Б

Шуракова А.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2024

**Постановка задачи:**

1. Определить пользовательский класс.

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

3. Определить в классе деструктор.

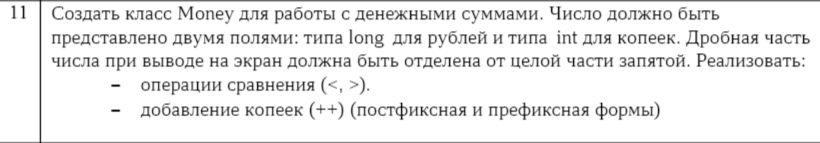
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

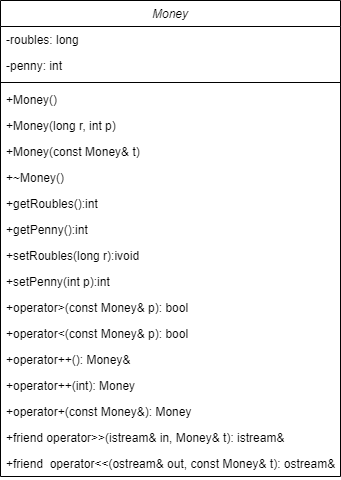
6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции указанные в варианте.

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



**UML диаграмма:**

****

**Код программы**

Money.h:

#pragma once

#include <iostream>

using namespace std;

class Money

{

long roubles;

int penny;

public:

//Конструкторы + деструктор

Money() { roubles = 0; penny = 0; };

Money(long r, int p) { roubles = r; penny = p; }

Money(const Money& t) { roubles = t.roubles; penny = t.penny; }

~Money() {};

//Селекторы и модификаторы

int getRoubles() { return roubles; }

int getPenny() { return penny; }

void setRoubles(long r) { roubles = r; }

void setPenny(int p) { penny = p; }

//перегруженные операции

bool operator>(const Money& p);

bool operator<(const Money& p);

//Money& operator=(const Money&);

Money& operator++(); //префиксная операция

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

Money operator+(const Money&);

//глобальные функции ввода-вывода

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

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

};

Money.cpp:

#include "Money.h"

#include <iostream>

using namespace std;

bool Money::operator>(const Money& p) {

if (roubles == p.roubles) {

return penny > p.penny;

}

return roubles > p.roubles ;

}

bool Money::operator<(const Money& p) {

if (roubles == p.roubles) {

return penny < p.penny;

}

return roubles < p.roubles;

}

//перегрузка префиксной операции инкремент

Money& Money::operator++()

{

int temp = roubles \* 100 + penny;

temp++;

roubles = temp / 100;

penny = temp % 100;

return \*this;

}

//перегрузка постфиксной операции инкремент

Money Money::operator ++(int)

{

int temp = roubles \* 100 + penny;

temp++;

Money t(roubles, penny);

roubles = temp / 100;

penny = temp % 100;

return t;

}

//перегрузка бинарной операции сложения

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

{

int temp1 = roubles \* 100 + penny;

int temp2 = t.roubles \* 100 + t.penny;

Money p;

p.roubles = (temp1 + temp2) / 100;

p.penny = (temp1 + temp2) % 100;

return p;

}

//перегрузка глобальной функции-операции ввода

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

{

cout << "Roubles: "; in >> t.roubles;

cout << "Penny: "; in >> t.penny;

return in;

}

//перегрузка глобальной функции-операции вывода

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

{

return (out << t.roubles << "," << t.penny);

}

Laba3\_main.cpp:

#include "Money.h"

#include <iostream>

using namespace std;

void main()

{

system("chcp 1251");

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<< " > " <<b<< endl;

}

else {

cout << a << " < " << b << endl;

}

++a;//префиксная операция инкремент

cout <<"Префиксная операция: " << a << endl;//вывод переменной

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

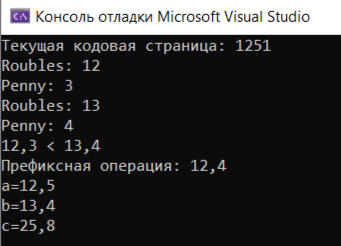
cout << "a=" << a << endl; //вывод переменной

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

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

}

**Результаты работы программы**



**Ответы на контрольные вопросы**

1. Для чего используются дружественные функции и классы?

Дружественная функция - это функция, которая может получить доступ к закрытым членам класса, как если бы она была членом этого класса.

2. Сформулировать правила описания и особенности дружественных функций.

* Дружественная функция объявляется внутри класса, к элементам которого ей нужен доступ, с ключевым словом friend. В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель this ей не передается.
* Дружественная функция может быть обычной функцией или методом другого ранее определенного класса. На нее не распространяется действие спецификаторов доступа, место размещения ее объявления в классе безразлично.
* Одна функция может быть дружественной сразу нескольким классам.

3. Каким образом можно перегрузить унарные операции?

В C++ можно перегрузить следующие унарные операторы:

* ! (логическое НЕ);
* & (адрес);
* ~ (дополнение);
* "\*" (разыменовка указателя);
* "+" (унарный плюс);
* "-" (унарное отрицание);
* ++ (добавочный префикс) или (приращение постфикса);
* -- (префикс декремент) или (postfix decrement).

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

0

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

1

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

1. Ее операнд – вызвавший ее же объект.

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

2 операнда типа класса.

8. Чем отличается перегрузка префиксных и постфиксных унарных операций?

* если перегружается префиксная форма оператора ++, то в классе нужно реализовать операторную функцию operator++() без параметров;
* если перегружается префиксная форма оператора —, то в классе нужно реализовать операторную функцию operator—() без параметров;
* если перегружается постфиксная форма оператора ++, то в классе нужно реализовать операторную функцию operator++(int d) с одним целочисленным параметром. В этом случае параметр d не используется в функции. Он указывается только для того, чтобы указать что это именно постфиксная реализация оператора ++. Имя d может быть заменено другим именем;
* если перегружается постфиксная форма оператора —, то в классе нужно реализовать операторную функцию operator—(int d) с одним параметром. Параметр d необходим для указания того, что перегружается именно постфиксная реализация оператора —

9. Каким образом можно перегрузить операцию присваивания?

* Перегрузка оператора присваивания — это процедура копирования величин из объекта в объект.
* Существует два основных способа перегрузки операторов:
* С помощью функций глобального масштаба.
* С помощью дружественных функций для класса или функций, которые представляют сам класс.
* Перегрузку возможно делать только для классов пользователя.

10. Что должна возвращать операция присваивания?

Объект

11. Каким образом можно перегрузить операции ввода-вывода?

Для того, чтобы перегрузить операции ввода и вывода для пользовательского класса, необходимо перегрузить операторы для объектов класса.

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?

При выполнении операции ++s;, компилятор будет интерпретировать это как префиксный инкремент объекта s, используя перегруженный оператор ++ в классе 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?

При выполнении операции ++s;, компилятор будет интерпретировать это как вызов дружественной функции operator++ вне класса, которая оперирует с объектом 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?

При выполнении операции cout << a < b;, компилятор будет интерпретировать это как вызов перегруженного оператора < для объектов a и b класса Student, который сравнивает их и возвращает булевое значение.

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?

При выполнении операции cout << a > b;, компилятор будет интерпретировать это как вызов дружественной функции operator> вне класса, которая оперирует с объектами a и b класса Student и сравнивает их.