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**

Тема: «Перегрузка операций»

Выполнил работу

Студент группы РИС-22-2Б

Барыбин К.Ю.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

Постановка задачи.

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 (==, !=).

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.

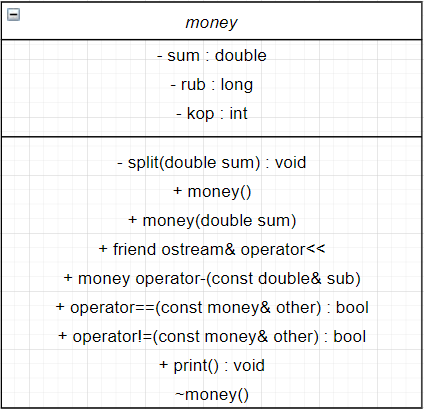
6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции, указанные в варианте.

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

всех перегруженных операций.

UML диаграмма.



Код программы.

#include<iostream>

using namespace std;

class money

{

private:

double sum;

long rub;

int kop;

void split(double sum)

{

rub = (int)sum;

kop = abs(((sum - (int)sum) \* 100));

}

public:

money()

{

cout << "Вызвался крнструктор без параметров " << this << endl;

this->sum = 0.0;

this->rub = 0;

this->kop = 0;

};

money(double sum)

{

cout << "\nВызвался крнструктор c параметрами " << this << endl;

this->sum = sum;

split(sum);

}

friend ostream& operator<<(ostream& stream, const money& m);

money operator-(const double& sub)

{

return (this->sum - sub);

}

bool operator==(const money& other)

{

if (this->sum == other.sum)

return true;

else

return false;

}

bool operator!=(const money& other)

{

return !(this->sum == other.sum);

}

void print()

{

cout << "\nСумма: " << rub << "," << kop << endl;

}

~money()

{

cout << "\nВызвался деструктор " << this;

cout << endl;

}

};

ostream& operator<<(ostream& stream, const money& m)

{

stream <<"\nРубли: " << m.rub << "\nКопейки: " << m.kop;

return stream;

};

int main()

{

system("chcp 1251 >> null");

double sub,sum;

money one;

one.print();

cout << "\n\nВведите сумму(double): ";

cin >> sum;

money two(sum);

two.print();

cout << "\nВведите дробное число, которое нужно вычесть из суммы: ";

cin >> sub;

money tre = sum - sub;

tre.print();

bool res = two == tre;

cout << "\nРезультат сравнения сумм ==(1 - равны; 0 - неравны): " << res << endl;

bool res2 = two != tre;

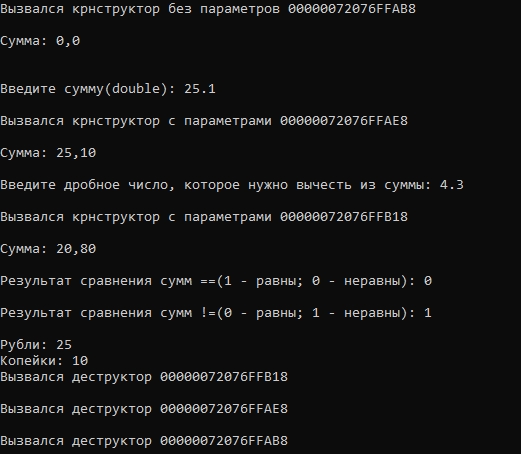
cout << "\nРезультат сравнения сумм !=(0 - равны; 1 - неравны): " << res2 << endl;

cout << two;

return 0;

}

Вывод.



Контрольные вопросы.

1. Для чего используются дружественные функции и классы?

Чтобы предоставить доступ к private – полям класса, методам другого класса, который является дружественным.

Чтобы предоставить доступ к private-полям глобальным не компонентным функциям.

2. Сформулируйте правила описания и особенности дружественных функций.

- Дружественная функция объявляется внутри класса с ключевым словом friend.

- Поскольку дружественная функция не является компонентной (ей не передается указатель this), то необходимо, чтобы она принимала в качестве параметра объект класса по ссылке, по значению или по адресу.

- Дружественная функция может быть дружественной сразу к нескольким классам.

- На неё не распространяются спецификаторы доступа, поэтому то место, где она описана в классе, неважно.

3. Каким образом можно перегрузить унарные операции.

-Как компонентные нестатические функции класса.

тип operator “знак оператора”(void);

-Как обычная глобальная не компонентная функция, которая также может быть дружественная классу.

тип operator “знак оператора”(class A), где class A – передача объекта класса.

- Унарные операторы перегружаются чаще всего как методы класса.

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

- Унарная операция по определению работает с одним операндом. Раз она перегружается как компонентная нестатическая функция, то она не должна принимать параметров. (неявно принимает указатель this).

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

- Т.к. оператор перегружается как глобальная функция, то параметр this ему не передается, следовательно необходимо явно передавать объект класса.

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

- Если оператор перегружается как компонентная функция, то левым операндом по умолчанию является объект класса – this. Правым операндом является тот объект, что передается в качестве параметра в перегружаемый оператор.

Например: A operator+(const int& value); - тут для класса A перегружается оператор сложения как метод класса. –>

a.operator+(5) ≡ a + 5. где a – объект класса.

Ответ: один.

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

- Если оператор перегружается как не компонентная функция, чаще всего дружественная классу. То указатель this не передается, поэтому, необходимо явно передавать объект класса в качестве параметра. Также необходимо передавать объект другого класса, с которым должен взаимодействовать исходный класс посредством оператора.

Например:

Из лабораторной работы №3: -

friend Pair operator+(const double& y, const Pair& p).

Существуют бинарные операторы, перегрузка которых вне класса невозможна: это “->”, “[]”, “()”, “=”

Ответ: два

8. Чем отличается перегрузка префиксных и постфиксных унарных операций.

-Префиксные и постфиксные операции по сути являются версией одного оператора в разных формах. Если при перегрузке префиксного оператора не нужно передавать никаких параметров, то при перегрузке постфиксного оператора необходимо передать незначащий параметр int. – Чтобы объяснить компилятору разницу.

Также эти операторы могут отличатся по типу возвращаемого значения. Допустим, если префиксный оператор (инкремента или декремента) модифицирует какое-либо информационное поле, а затем возвращает ссылку на объект этого класса, то постфиксный оператор должен сохранить состояние объекта класса во временную переменную, затем модифицировать поле класса, затем вернуть копию предыдущего состояния. – Это накладывает некоторые ограничения на использование постфиксных операторов, т.к. они не позволяют взаимодействовать напрямую с объектом класса.

9. Каким образом можно перегрузить операцию присваивания.

- Оператор присваивания можно перегрузить только как нестатическую компонентную функцию класса.

10. Что должна возвращать операция присваивания?

- Ссылку на объект класса, в который происходит копирование (левый операнд). Это нужно для реализации многочисленного присваивания.

Например: a = b = c; Где a, b и c – Объекты одного класса.

11. Каким образом можно перегрузить операции ввода-вывода?

- Для того, чтобы обеспечить взаимодействие пользовательского класса и потокового класса (левым операндом является объект потокового класса, правым операндом является объект пользовательского класса), необходимо перегрузить оператор<< или оператор>> как дружественную функцию. С двумя параметрами – первый: объект класса std::ostream или std::istream, второй: объект пользовательского класса.

Например: friend std::ostream& operator<<(std::ostream& stream, const Pair& p)

Тут operator << объявлен в классе как дружественная функция.

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?

-Как вызов метода класса: s.operator++();

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?

-Как вызов глобальной функции: operator++(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?

Приоритет оператора << выше, чем у оператора<. Программа просто не скомпилируется.

Если выполнялась бы операция cout<<(a<b); То компилятор это воспринял бы как вызов метода класса a.operator<(b);

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?

Приоритет у оператора << выше, программа не будет работать.

Если выполняется операция cout << (a>b); то компилятор это будет воспринимать как вызов глобальной функции: cout << operator>(a, b);