Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №3

«Перегрузка операций»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Перегрузка операций

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование перегруженных операций в классах.

**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.

1. Перегрузить операции указанные в варианте.
2. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

**Задание**

Создать класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать:

* вычитание пар чисел
* добавление константы к паре (увеличивается первое число, если константа целая, второе, если константа вещественная).

**Анализ задачи**

1) Использование директивы #pragma once предотвращает повторную загрузку заголовочного файла, что может улучшить

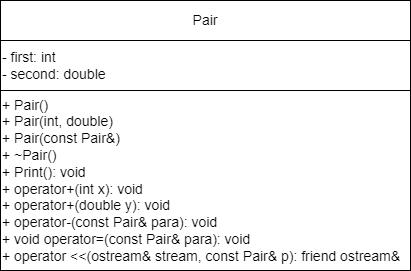
производительность программы, особенно при использовании множества включаемых файлов.

1. Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
2. void Pair::Print() функция для вывода информации об элементе

класса.

1. Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.
2. Функция-член operator+ для работы с парой и целым числом определена как дружественная функция, что позволяет использовать её для работы с объектами класса Pair без необходимости создания метода класса.
3. Дружественные функции operator+ для работы с парой и вещественным числом определены как глобальные функции, что также позволяет использовать их для работы с объектами класса Pair.
4. В главной функции показаны операции, которые можно выполнить с объектом класса, включая перегруженные операторы.

**Uml диграмма**



**Код**

Файл ООП3.cpp

#include <iostream>

using namespace std;

#include "Pair.h"

int main()

{

system("chcp 1251>null");

Pair pair1(3, 4.3);

pair1.Print();

Pair pair2(pair1);

pair2.Print();

pair2 + 3;

pair2.Print();

pair2 + 3.3;

pair2.Print();

pair2 - pair1;

pair2.Print();

Pair pair3;

cout << pair3 << endl;

pair3 = pair2;

cout << pair3 << endl;

Pair pair4;

cout << "Введите пару:\n";

cin >> pair4;

cout << pair4 << endl;

return 0;

}

Файл Pair.h

#pragma once

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int, double);

Pair(const Pair&);

~Pair();

void Print();

void operator - (const Pair&);

void operator + (const int&);

void operator + (const double&);

void operator = (const Pair&);

friend ostream& operator << (ostream&, const Pair&);

friend istream& operator >> (istream&, Pair&);

};

Файл Pair.cpp

#pragma once

#include <iostream>

using namespace std;

#include "Pair.h"

Pair::Pair()

{

cout << "\nВызов конструктора без параметров для объекта " << this << endl;

this->first = 0;

this->second = 0.0;

}

Pair::Pair(int first, double second)

{

cout << "Вызов конструктора с параметрами для объекта " << this << endl;

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

cout << "\nВызов конструктора копирования для объекта " << this << endl;

this->first = other.first;

this->second = other.second;

}

Pair::~Pair()

{

cout << "Вызов дуструктора для объекта " << this << endl;

}

void Pair::Print()

{

cout << "(" << this->first << ":" << this->second << ")\n";

}

void Pair::operator-(const Pair& other)

{

cout << "\nВызов оператора вычитания для объекта " << this << endl;

this->first -= other.first;

this->second -= other.second;

}

void Pair::operator+(const int&x)

{

cout << "\nВызов оператора суммирования int для объекта " << this << endl;

this->first += x;

}

void Pair::operator+(const double& x)

{

cout << "\nВызов оператора суммирования double для объекта " << this << endl;

this->second += x;

}

void Pair::operator=(const Pair& other)

{

cout << "\nВызов оператора присваивания для объекта" << this << endl;

this->first = other.first;

this->second = other.second;

}

ostream& operator<<(ostream& stream, const Pair& pair)

{

cout << "Вызов оператора вывода для объекта " << &pair << endl;

stream << "(" << pair.first << ":" << pair.second << ")";

return stream;

}

istream& operator>>(istream& stream, Pair& pair)

{

cout << "Вызов оператора ввода для объекта " << &pair << endl;

int first; double second;

stream >> first >> second;

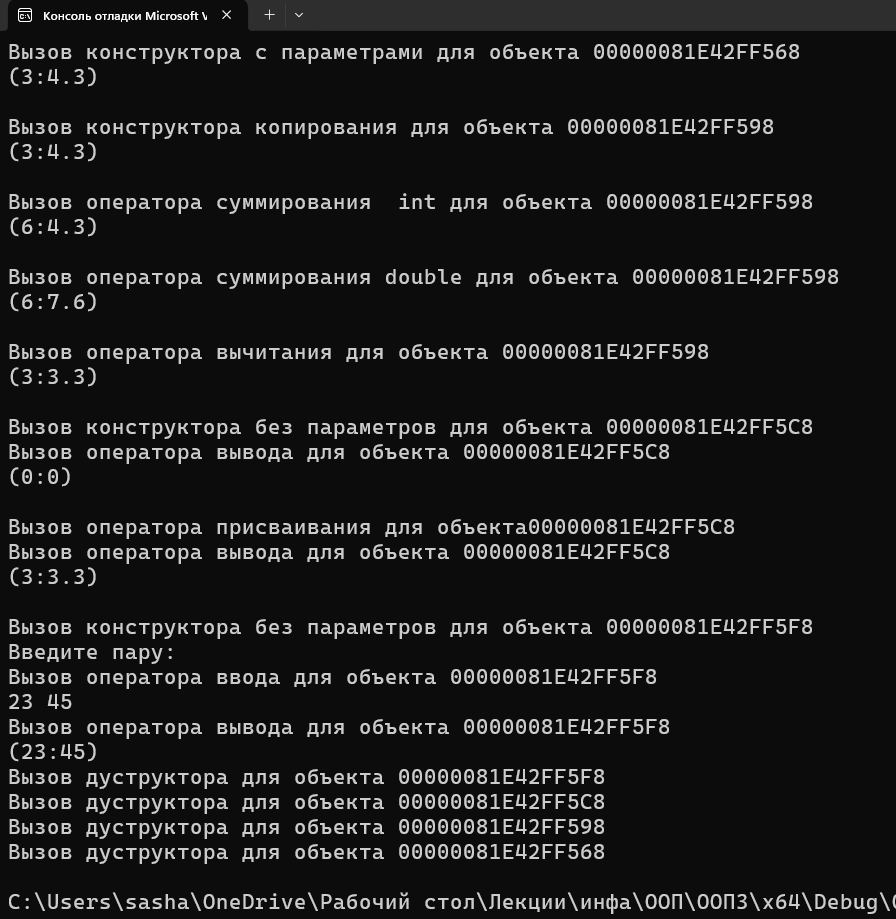
pair.first = first;

pair.second = second;

return stream;

}

**Результат работы**



**GitHub**

https://github.com/Korovay4ik/Laboratory-works

**Контрольные вопросы**

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

Дружественные функции и классы используются для предоставления доступа к закрытым членам класса извне этого класса. Это позволяет другим функциям или классам работать с частными данными и методами класса без необходимости делать их публичными.

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

**дружественных функций.**

* Ключевое слово friend используется для объявления дружественности.
* Дружественная функция может быть определена внутри или вне

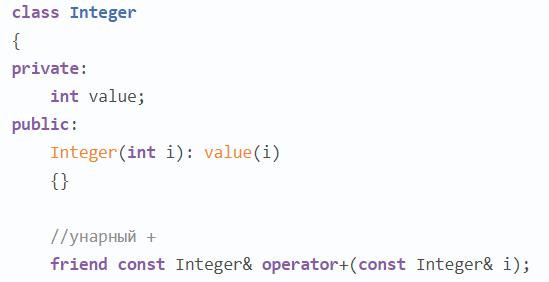
класса.

* Дружественная функция объявляется внутри класса.
* Класс может объявить функцию другом, но определение этой функции должно находиться вне класса.
* Дружественность должна быть явно объявлена для каждой функции или класса, которому предоставляется доступ.
* Нельзя сделать весь файл дружественным, но можно сделать все функции в файле дружественными.
* Одна функция может быть дружественной сразу нескольким классам.

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

Унарную операцию можно перегрузить:

* Как компонентную функцию класса



* Как внешнюю (глобальную) функцию

1. **Сколько операндов должна иметь унарная функция-операция,**

**определяемая внутри класса?**

0

1. **Сколько операндов должна иметь унарная функция-операция,**

**определяемая вне класса?**

1

1. **Сколько операндов должна иметь бинарная функция-**

**операция, определяемая внутри класса?**

1 – ее операнд

1. **Сколько операндов должна иметь бинарная функция-**

**операция, определяемая вне класса?**

Бинарная функция-операция, определяемая внутри класса, должна иметь два операнда: this и другой объект того же типа

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

Префиксные операции обычно изменяют значение объекта, а

постфиксные операции обычно возвращают текущее значение перед его

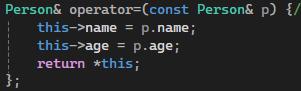
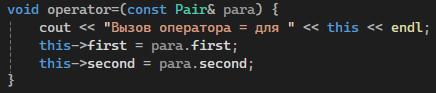
изменением.

Перегруженные префиксные операторы возвращают объект после того, как он был инкрементирован или декрементирован. Следовательно, перегрузить их довольно просто. Мы просто увеличиваем или уменьшаем наши переменные-члены, а затем возвращаем \*this.

Постфиксные операторы, напротив, должны возвращать состояние объекта до его инкремента или декремента.

* 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 ука-зывается 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 —

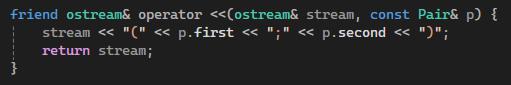
1. **Каким образом можно перегрузить операцию присваивания?**



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

Либо ничего, либо объект.

1. **Каким образом можно перегрузить операции ввода-вывода?**
2. **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 ++s; как обращение к префиксной операции инкремента, определенной внутри класса Student. Функция operator++() должна увеличивать значение объекта s на единицу.

1. **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 ++s; как обращение к префиксной операции инкремента, определенной как дружественная функция. Функция operator ++(Student&) должна увеличивать значение объекта s на единицу.

1. **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 cout << a < b;

как обращение к оператору сравнения <, определенному внутри класса

Student. Функция operator<(Student& P) должна сравнивать объекты a и b и возвращать результат сравнения. Если а меньше b вернется 1.

1. **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 вызов функции-операции cout << a > b; как обращение к оператору сравнения >, определенному как дружественная функция. Функция operator >(const Person&, Person&) должна сравнивать объекты a и b и возвращать результат сравнения. Если а больше b вернется 1.