Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№3»**

Выполнил:

студент группы РИС-23-1б

Кривошеин Александр Антонович

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

2024 г.

**Разработка алгоритма**

**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).

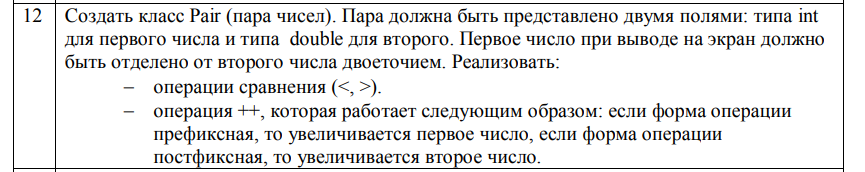
5. Перегрузить операцию присваивания.

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:



**Анализ задачи:**

1. В классе Pair определяем пару значений с типами данных integer и double.
2. Создаем конструкторы, деструктор, операторы сравнения (< и >), префиксные и постфиксные операторы приращения, операторы ввода/вывода (<<) и методы получения/установки полей класса get() set().
3. Операторы сравнения позволяют нам сравнивать два объекта Pair на основе суммы их значений.
4. Перегрузка оператора инкремента позволяют увеличивать либо первое, либо второе поле, в зависимости от того, постфиксный или префиксный оператор был применен к обьекту.
5. Перегрузка оператора вывода (<<) позволяет выводить объекты Pair в стандартный поток вывода.

Код на C++:

Pair.h:

#pragma once

#include <iostream>

class Pair

{

private:

    int first;

    double second;

public:

    Pair();

    Pair(int, double);

    Pair(Pair &);

    ~Pair();

    bool operator<(Pair &);

    bool operator>(Pair &);

    Pair &operator++();

    Pair &operator++(int);

    friend std::ostream &operator<<(std::ostream &out, Pair i);

    void get();

    void set(int, double);

};

Pair::Pair()

{

    this->first = 0;

    this->second = 0;

}

Pair::Pair(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

Pair::Pair(Pair &copying)

{

    this->first = copying.first;

    this->second = copying.second;

}

Pair::~Pair()

{

    std::cout << "\nDestructor was called";

}

bool Pair::operator<(Pair &temp)

{

    if ((this->first + this->second) < (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator>(Pair &temp)

{

    if ((this->first + this->second) > (temp.first + temp.second))

        return true;

    return false;

}

Pair &Pair::operator++()

{

    ++this->first;

    return \*this;

}

Pair &Pair::operator++(int)

{

    ++this->second;

    return \*this;

}

void Pair::get()

{

    std::cout << "first = " << this->first << ", second = " << this->second;

}

void Pair::set(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

std::ostream &operator<<(std::ostream &pairout, Pair temp)

{

    return (pairout << temp.first << ':' << temp.second);

}

main.cpp:

#include <iostream>

using namespace std;

#include "Pair.h"

int main()

{

    Pair a(1, 5.5);

    Pair b;

    a.get();

    cout << endl;

    b.get();

    cout << endl;

    b.set(6, 2.2);

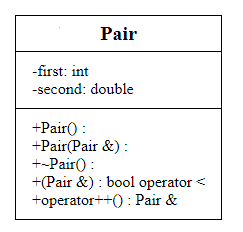
    cout << endl << a << endl << b<< endl;

    cout << endl << (a > b);

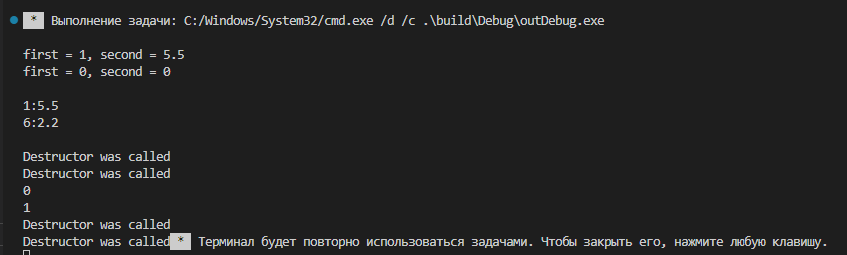
    cout << endl << (a < b);

}

Диаграмма класса:



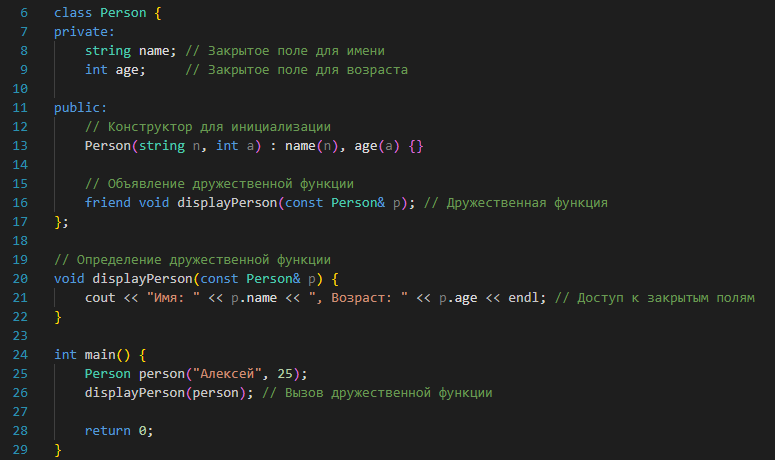
Пример работы программы:



**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.

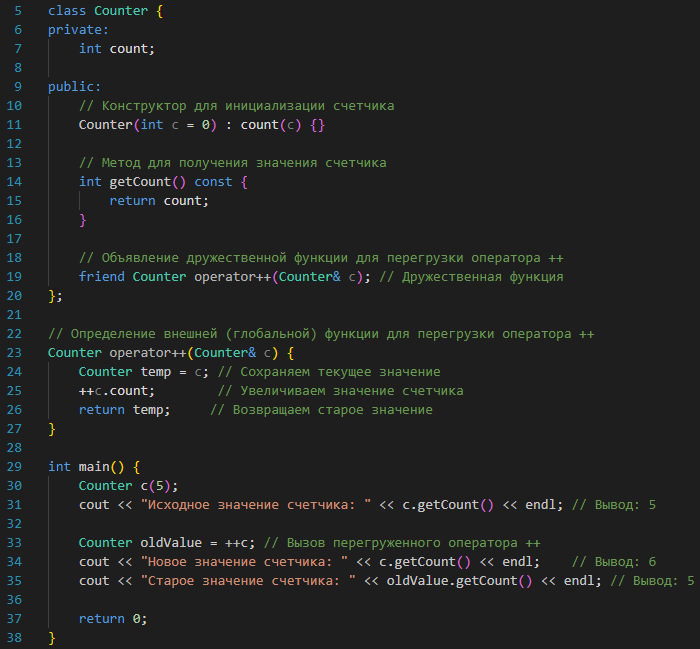
1. Каким образом можно перегрузить унарные операции?

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

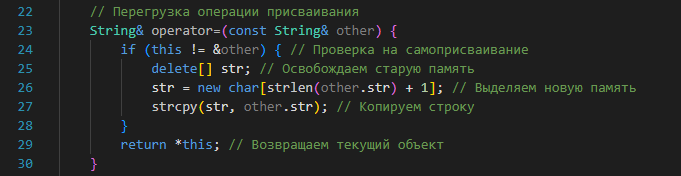
изменением.

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 —

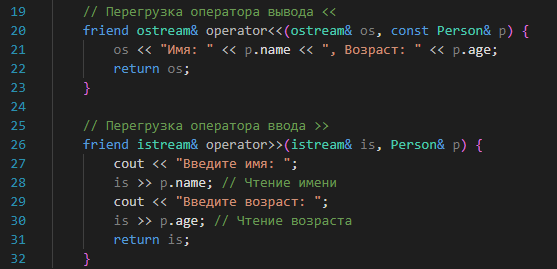
1. Каким образом можно перегрузить операцию присваивания?



1. Что должна возвращать операция присваивания?

Либо ничего, либо объект.

1. Каким образом можно перегрузить операции ввода-вывода?



1. 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

{

…

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&)

….

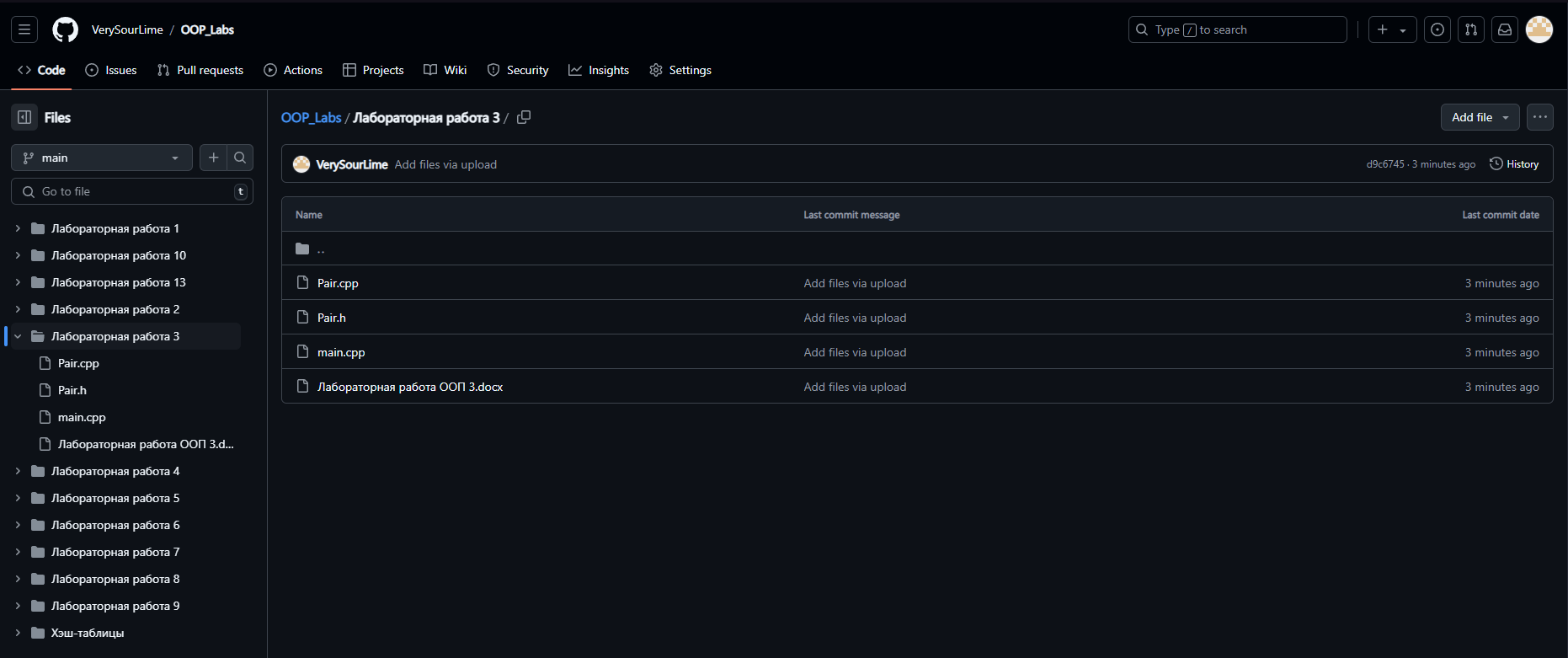
};

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.

Скриншот из GitHub:

  
GitHub: https://github.com/VerySourLime/OOP\_Labs