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
Акбашева Софья Руслановна

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.

7. Перегрузить операции указанные в варианте.

8. 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.

2) Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.

3) void Pair::Print() функция для вывода информации об элементе класса.

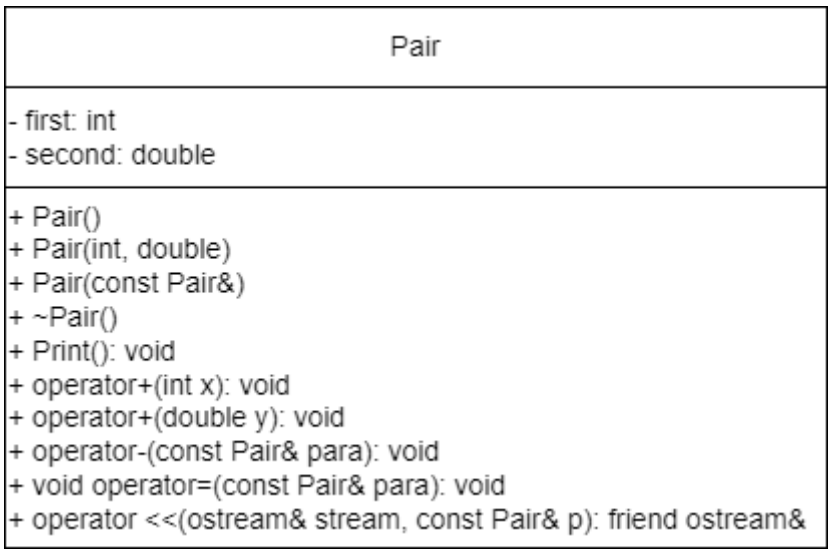
4) Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.

5) 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.

6) 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.

7) В главной функции показаны операции, которые можно выполнить с объектом класса, включая перегруженные операторы.

Блок схема



Код

Файл Class_3.cpp

```
#include <iostream>
#include "pair.h"

using namespace std;

int main() {
	system("chcp 1251 > Null");

	int number_1 = 5;
	double number_2 = 5.1;

	Pair a(3, 5);
	cout << "a = " << a << endl;
	Pair b(2, 8);
	cout << "b = " << b << endl;
	Pair c;
	cout << "c = " << c << endl;
	Pair d(a);
	cout << "d = " << d << endl << endl;

	c = b;
	cout << "c = " << c << endl << endl;
	cout << "Вычитание a - b: " << a << " - " << b << " = " << a - b << endl;
	cout << "a + " << number_1 << " = " << a + number_1 << endl;
	cout << "b + " << number_2 << " = " << b+ number_2 << endl;
}
```

Файл pair.cpp

```
#include "pair.h"
#include <iostream>
#include <string>
using namespace std;

Pair::Pair() {
	this->first = 0;
	this->second = 0;
	cout << "Конструктор по умолчанию для объекта " << this << endl;
}

Pair::Pair(int first, double second) {
	this->first = first;
	this->second = second;
	cout << "Конструктор с параметрами для объекта " << this << endl;
}

Pair::Pair(const Pair& other) {//конструктор копирования
	this->first = other.first;
	this->second = other.second;
	cout << "Конструктор копирования для объекта " << this << endl;
}

Pair::~Pair() {//деструктор
	cout << "Деструктор для объекта " << this << endl;
}

void Pair::Print() {//вывод параметров класса person
	cout << "(" << this->first <<  " ; " << this->second << ")" << endl;
```

```
}
```

Файл pair.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

class Pair {
	int first;
	double second;

public:
	Pair();//конструктор класса по умолчанию
	Pair(int, double);//конструктор класса с параметрами
	Pair(const Pair&);//конструктор копирования класса
	~Pair();//деструктор класса
	void Print();

	Pair operator+(const int& x){
		cout << endl << "Вызов оператора + для " << this << endl;
		return (Pair(this->first + x, this->second));
	}

	Pair operator+(const double& y){
		cout << endl << "Вызов оператора + для " << this << endl;
		return (Pair(this->first, this->second + y));
	}

	Pair operator-(const Pair& para) {
		cout << "Вызов оператора - для " << this << endl;
		return (Pair(this->first - para.first, this->second - para.second));
	}

	void operator=(const Pair& para) {
		cout << "Вызов оператора = для " << this << endl;
		this->first = para.first;
		this->second = para.second;
	}

	friend ostream& operator <<(ostream& stream, const Pair& p) {
		stream << "(" << p.first << ":" << p.second << ")";
		return stream;
	}
};
```

Результат работы

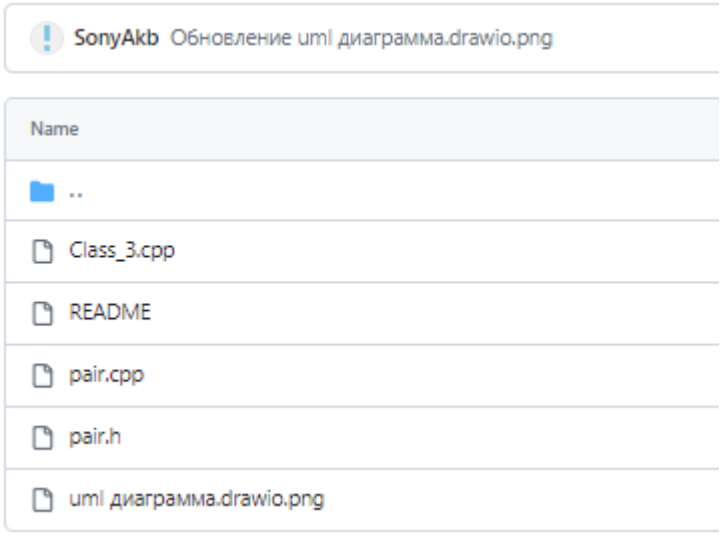
```
Конструктор с параметрами для объекта 00000013325BF4A8
a = (3;5)
Конструктор с параметрами для объекта 00000013325BF4D8
b = (2;8)
Конструктор по умолчанию для объекта 00000013325BF508
c = (0;0)
Конструктор копирования для объекта 00000013325BF538
d = (3;5)

Вызов оператора = для 00000013325BF508
c = (2;8)

Вызов оператора - для 00000013325BF4A8
Вычитание a - b: (1;-3) - (2;8) = (1;-3)

Вызов оператора + для 00000013325BF4A8
a + 5 = (6;-3)

Вызов оператора + для 00000013325BF4D8
b + 5.1 = (2;13.1)
Деструктор для объекта 00000013325BF538
Деструктор для объекта 00000013325BF508
Деструктор для объекта 00000013325BF4D8
Деструктор для объекта 00000013325BF4A8
```

Вывод

В ходе работы я применила знания о работе с классами. В частности, о перегрузке операций при работе с классами. По итогу работы была реализованная поставленная задача.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_3



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.

3. Каким образом можно перегрузить унарные операции?

Унарную операцию можно перегрузить:

- Как компонентную функцию класса

```
class Integer
{
private:
    int value;
public:
    Integer(int i): value(i)
    {}

    //унарный +
    friend const Integer& operator+(const Integer& i);
```

- Как внешнюю (глобальную) функцию

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

7. 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

8. 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 —

9. Каким образом можно перегрузить операцию присваивания?

```
void operator=(const Pair& para) {
    cout << "Вызов оператора = для " << this << endl;
    this->first = para.first;
    this->second = para.second;
}
```

```
Person& operator=(const Person& p) {/
    this->name = p.name;
    this->age = p.age;
    return *this;
};
```

10. Что должна возвращать операция присваивания?

Либо ничего, либо объект.

11. Каким образом можно перегрузить операции ввода-вывода?

```
friend ostream& operator <<(ostream& stream, const Pair& p) {
    stream << "(" << p.first << ";" << p.second << ")";
    return stream;
}
```

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 ++s; как обращение к префиксной операции инкремента, определенной внутри класса Student. Функция operator++() должна увеличивать значение объекта s на единицу.

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

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 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 a меньше b вернется 1.

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?

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 a больше b вернется 1.