Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**Тема:** Лабораторная работа №3 по ООП

Семестр: 2

Выполнил студент ИВТ-23-2б:

Хомутов Сергей Алексеевич

(дата, подпись)

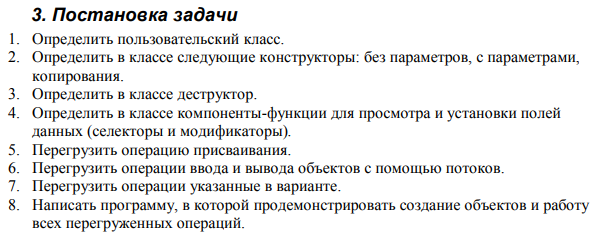
Проверила:

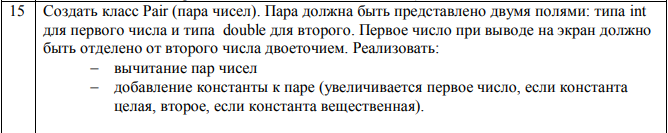
Полякова Ольга Андреевна

(дата, подпись)

Пермь 2024

**Задание**





**Анализ задачи**

1. Pair::Pair(int first, double second) - конструктор с параметрами

2. istream& operator>>(istream& in, Pair& m\_pair) - перегрузка оператора ввода

3. ostream& operator<<(ostream& out, const Pair& m\_pair) – перегрузка оператора вывода

4. Pair::Pair() - конструктор без параметров

5. Pair::~Pair() - деструктор

6. Pair::Pair(const Pair& m\_pair) - конструктор копирования

7. Pair& Pair::operator=(const Pair& m\_pair) - перегрузка оператора присваивания

8. Pair Pair::operator-(const Pair& m\_pair) - перегрузка оператора вычитания

9. Pair& Pair::operator+(int first) - перегрузка оператора сложения - целый аргумент

10. Pair& Pair::operator+(double second) - перегрузка оператора сложения - дробный аргумент

11. Pair& Pair::operator++() - перегрузка оператора ++ префикс

12. Pair Pair::operator ++(int) - перегрузка оператора ++ постфикс

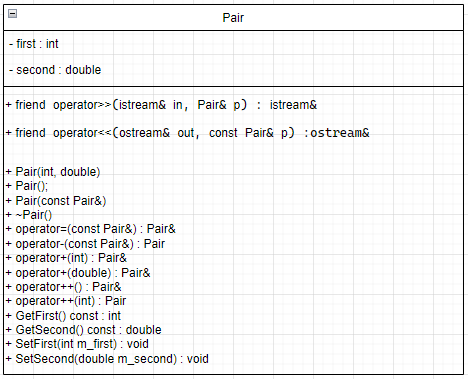
13. int Pair::GetFirst() const – геттер для первого числа

14. double Pair::GetSecond() const - геттер для второго числа

15. void Pair::SetFirst(int m\_first) – сеттер для первого числа

16. void Pair::SetSecond(double m\_second) – сеттер для второго числа

**UML-Диаграмма**



**Программное решение**

**Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair {

int first;

double second;

public:

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

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

Pair(int, double);

Pair();

Pair(const Pair&);

~Pair();

Pair& operator=(const Pair&);

Pair operator-(const Pair&);

Pair& operator+(int);

Pair& operator+(double);

Pair& operator++();

Pair operator++(int);

int GetFirst() const;

double GetSecond() const;

void SetFirst(int m\_first);

void SetSecond(double m\_second);

};

**Pair.cpp**

#include "Pair.h"

Pair::Pair(int first, double second) {//конструктор с параметрами

this->first = first;

this->second = second;

}

istream& operator>>(istream& in, Pair& m\_pair) {//перегрузка оператора ввода

in >> m\_pair.first;

in >> m\_pair.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& m\_pair) {//перегрузка оператора вывода

return (out << m\_pair.first << " : " << m\_pair.second);

}

Pair::Pair() {//конструктор без параметров

first = 0;

second = 0;

}

Pair::~Pair() {//деструктор

}

Pair::Pair(const Pair& m\_pair) {//конструктор копирования

first = m\_pair.first;

second = m\_pair.second;

}

Pair& Pair::operator=(const Pair& m\_pair) {//перегрузка оператора присваивания

if (&m\_pair != this) {

first = m\_pair.first;

second = m\_pair.second;

}

return \*this;

}

Pair Pair::operator-(const Pair& m\_pair) {//перегрузка оператора вычитания

Pair result(first - m\_pair.first, second - m\_pair.second);

return result;

}

Pair& Pair::operator+(int first) {//перегрузка оператора сложения - целый аргумент

this->first += first;

return \*this;

}

Pair& Pair::operator+(double second) {//перегрузка оператора сложения - дробный аргумент

this->second += second;

return \*this;

}

Pair& Pair::operator++() {//перегрузка оператора ++ префикс

++first;

++second;

return \*this;

}

Pair Pair::operator ++(int) {//перегрузка оператора ++ постфикс

Pair temp = \*this;

this->first++;

this->second++;

return temp;//сначала нужно вывести без изменений

}

int Pair::GetFirst() const{

return first;

}

double Pair::GetSecond() const{

return second;

}

void Pair::SetFirst(int m\_first) {

first = m\_first;

}

void Pair::SetSecond(double m\_second) {

second = m\_second;

}

**OOPLab3.cpp**

#include <iostream>

#include "Pair.h"

using namespace std;

Pair& operator+(int number, Pair& pair) { //заменяем местами для коммуникативности - сложение с целой константой

return pair + number;

}

Pair& operator+(double number, Pair& pair) { //заменяем местами для коммуникативности - сложение с вещественной константой

return pair + number;

}

int main() {

system("chcp 1251 > NULL");

Pair pair1;

Pair pair2(3, 4.5);

cout << "Вывод значений pair2:\n" << pair2 << '\n';

cout << "Ввод значений для pair1:\n";

cin >> pair1;

cout << "Вывод значений pair1:\n" << pair1 << '\n';

cout << "Вычитание из значений pair2 соответствующие значения pair1:\n" << pair2 - pair1 << '\n';

cout << "Присваивние: pair1 = pair2\n";

pair1 = pair2;

cout << "Вывод значений pair1:\n" << pair1 << '\n';

Pair tmp = pair1;//сохраняем значение

cout << "Сложение pair1 с целой константой 3:\n";

cout << pair1 + 3 << '\n';

pair1 = tmp;//возвращаем предыдущее значение

cout << "Проверка коммуникативности:\n";

cout << 3 + pair1 << '\n';

tmp = pair1;//сохраняем значение

cout << "Сложение pair1 с вещественной константой 4.5:\n";

cout << pair1 + 4.5 << '\n';

pair1 = tmp;//возвращаем предыдущее значение

cout << "Проверка коммуникативности:\n";

cout << 4.5 + pair1 << '\n';

cout << "++ префикс для pair1:\n";

cout << ++pair1 << '\n';

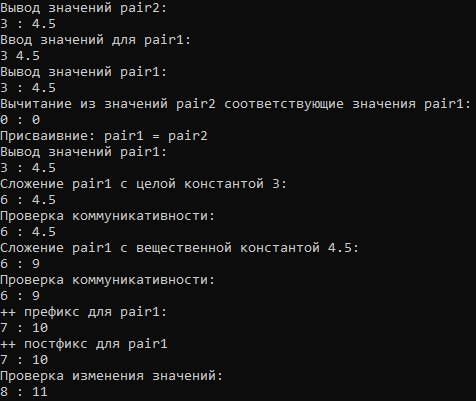
cout << "++ постфикс для pair1\n";

cout << pair1++ << '\n';

cout << "Проверка изменения значений:\n";

cout << pair1 << '\n';

}



**Контрольные вопросы**

1. Для чего используются дружественные функции и классы?

Для расширения возможности работы с классом.

1. Сформулировать правила описания и особенности дружественных функций.

* Дружественная функция объявляется внутри класса, к элементам которого ей нужен доступ
* Объявляется с ключевым словом friend.
* В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель this ей не передается.

Пример:



* 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 – если префиксная форма, 1 – int, если постфиксная.

Пример:



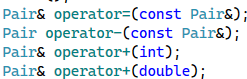
1. Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?

1 операнд – указатель на объект класса, т.к. this не передается. .

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

1 операнд.

Пример:

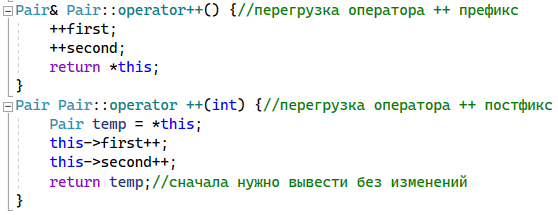


1. Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?

2 операнда.

1. Чем отличается перегрузка префиксных и постфиксных унарных операций?

* При перегрузке постфиксного оператора необходимо передать параметр int. – Чтобы компилятор правильно определил форму.
* Также эти операторы могут отличатся по типу возвращаемого значения.

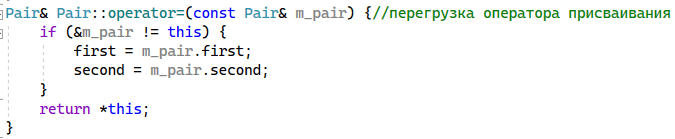


1. Каким образом можно перегрузить операцию присваивания?

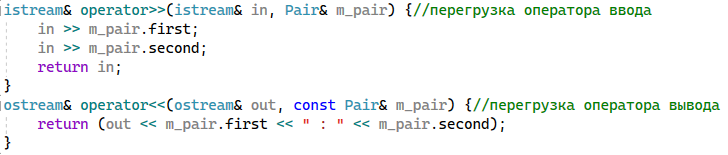
Как нестатическую компонентную функцию класса

1. Что должна возвращать операция присваивания?

Ссылку на объект класса, в который происходит копирование для реализации многочисленного присваивания: a = b = c;



1. Каким образом можно перегрузить операции ввода-вывода?

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?

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?

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 <. Поэтому программа не скомпилируется.

В ином случае, если взять выражение, идущее на вывод, в скобки, т. е. 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&)

….

};

Приоритет оператора << выше, чем у оператора >. Поэтому программа не скомпилируется.

В ином случае, если взять выражение, идущее на вывод, в скобки, т. е. cout << (a>b); то компилятор это будет воспринимать как вызов глобальной функции: cout << operator>(a, b);