Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.4**

Дисциплина: «Информатика»

Тема: Объектно-ориентированное программирование. Простое наследование

Вариант 4

Выполнил:

Студент группы РИС-20-1б

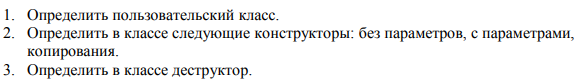
Еске Вячеслав Сергеевич

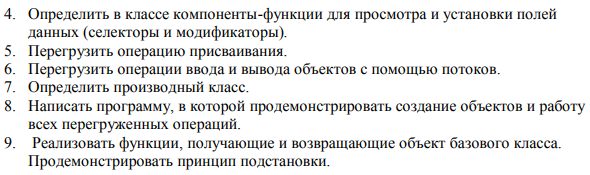
Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Постановка задачи**





Базовый класс: ПАРА\_ЧИСЕЛ (PAIR)   
Первое\_число (first) - int   
Второе\_число (second) – int   
Определить методы изменения полей и операцию сложения пар (a,b)+(c,d)=(a+b,c+d)   
Создать производный класс КОМПЛЕКСНОЕ\_ЧИСЛО(COMPLEX), с полями Действительная\_часть\_числа и Мнимая\_часть\_числа.   
Определить операции умножения (a,b)\*(c,d)= (a\*c-b\*d, a\*d+b\*c) и вычитания (a,b)-(c,d)= (a-b, с-d)

**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Person с полями name и age.

**1.2.** Организовать класс Employee с полями post, salary, который является производным класса Person.

**1.3.** Организовать необходимые методы для ввода данных в поля first и second: гетторы, сетторы, конструкторы, деструктор.

**1.4.** Организовать перегрузку оператора >> дружественной классу Person.

**1.5.** Организовать перегрузку оператора << дружественной классу Person.

**1.6.** Организовать перегрузку оператора - дружественной классу Person.

**1.7.** Организовать перегрузку оператора + дружественной классу Person.

**2.** В ходе работы были использованы следующие типы данных:

2.1 Тип данных int для хранения возраста;



2.2 Тип данных double для хранения зарплаты;



2.3 Тип данных string, для хранения имени.



**Код**

#include <iostream>

#include "Pair.h"

#include "Complex.h"

using namespace std;

void main()

{

setlocale(LC\_ALL, "Rus");

Pair a(2, 141);

Complex c,d;

cout << "Пары чисел"<<endl;

cout << "a: ";

cout << a;

Pair b(4, 115);

cout << "b: ";

cout << b;

cout << "a+b: ";

cout << a+b;

cout << "Комплексные числа" << endl;

cout << "c: ";

cin >> c;

cout << "c: ";

cout << c;

cout << "d: ";

cin >> d;

cout << "d: ";

cout << d;

cout << "c-d: ";

cout << c - d;

cout << "c\*d: ";

cout << c \* d;

}

#pragma once

#include <iostream>

using namespace std;

class Pair {

public:

//конструктор без параметров public:

Pair(void);

//деструктор virtual

~Pair(void);

//констрктор с параметрами

Pair(int,int);

//конструктор копирования

Pair(const Pair&);

//селекторы

int Get\_first(){return first;}

int Get\_second(){return second;}

//модификаторы

void Set\_first(int);

void Set\_second(int);

//перегрузка операции присваивания

Pair& operator=(const Pair&);

//глобальные операторы-функции ввода-вывода

friend Pair operator+(const Pair& p1, const Pair& p2);

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

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

//атрибуты

protected:

int first; int second;

};

#include "Pair.h"

//конструктор без параметров

Pair::Pair(void) {

first = 0;

second = 0;

}

//деструктор

Pair::~Pair(void) { }

//конструктор с параметрами

Pair::Pair(int F,int S) {

first= F;

second = S;

}

//конструктор копирования

Pair::Pair(const Pair& pair) {

first = pair.first;

second = pair.second;

}

//модификаторы

void Pair::Set\_first(int F) {

first = F;

}

void Pair::Set\_second(int S) {

second = S;

}

//перегрузка операции присваивания

Pair& Pair::operator=(const Pair&p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

return \*this;

}

//глобальная функция сложения

Pair operator+(const Pair& p1, const Pair& p2) {

Pair temp;

temp.first = p1.first + p2.first;

temp.second = p1.second + p2.second;

return temp;

}

//глобальная функция для ввода

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

{

cout << "\nПервое число:";

in >> p.first;

cout << "\nВторое число:";

in >> p.second;

return in;

}

//глобальная функция для вывода

ostream& operator<<(ostream&out,const Pair&p) {

out << "\nПервое число : " << p.first;

out << "\nВторое число : " << p.second;

out << "\n";

return out;

}

#pragma once

#include "Pair.h"

class Complex :

public Pair

{

public:

Complex(void); //конструктор без параметров

~Complex(void); //деструктор

Complex(int, int); //конструктор с параметрами

Complex(const Complex&); //конструктор копирования

Complex& operator=(const Complex&); //операция присваивания

friend istream& operator>>(istream& in, Complex& l); //операция ввода

friend ostream& operator<<(ostream&out,const Complex&l); //операция вывода

friend Complex operator\* (const Complex& c1, const Complex& c2);

friend Complex operator- (const Complex& c1, const Complex& c2);

};

#include "Complex.h"

Complex::Complex(void) :Pair() {};

Complex::~Complex(void) {};

Complex::Complex(int F, int S) :Pair(F, S) {};

Complex::Complex(const Complex &C) {

first = C.first;

second = C.second;

}

Complex& Complex::operator=(const Complex& c)

{

if (&c == this) return \*this;

first = c.first;

second = c.second;

return \*this;

}

Complex operator\* (const Complex& c1, const Complex& c2) {

Complex temp;

temp.first = (c1.first \* c2.first) - (c1.second \* c2.second);

temp.second = (c1.first \* c2.second) + (c1.second \* c2.first);

return temp;

}

Complex operator-(const Complex& c1, const Complex& c2)

{

Complex temp;

temp.first = c1.first-c2.first;

temp.second = c1.second-c2.second;

return temp;

}

istream& operator>>(istream& in, Complex& c)

{

cout << "\nПервое число:";

in >> c.first;

cout << "\nВторое число:";

in >> c.second;

return in;

}

ostream& operator<<(ostream& out, const Complex& c) {

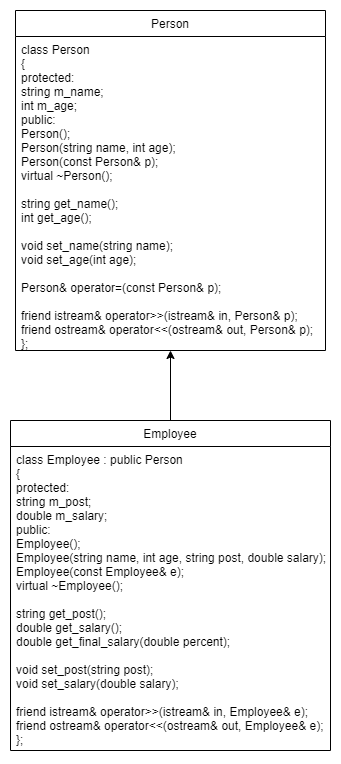
out << "\nКомплексное число : " << c.first << " + " << c.second << "i";

out << "\n";

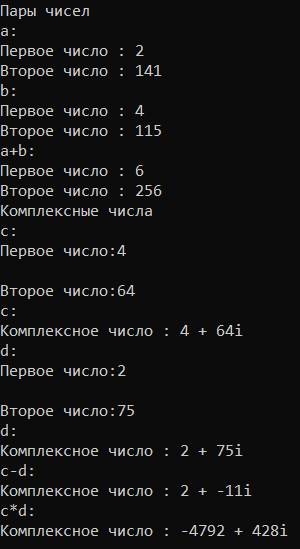
return out;

}

**UML диаграмма**



**Скриншоты тестов**



**Контрольные вопросы**



1. Наследование позволяет реализовать типизированное повторное использование объектов, создание иерархии «тип-подтип» с сохранением подтипом всех свойств своего предка. Можно рассматривать наследование для всех трех составляющих объекта.



2. Когда вы открыто наследуете родительский класс, то унаследованные public-члены остаются public.



3. Когда вы открыто наследуете родительский класс, то унаследованные private-члены остаются недоступными для дочернего класса.



4. К protected-членам родительского класса доступ открыт для членов дочернего класса.



5. 



6. Конструкторы не наследуются.



7. Деструкторы не наследуются.



8. 



9. 

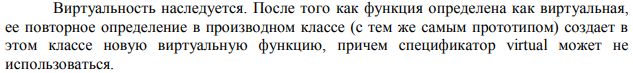


10. 

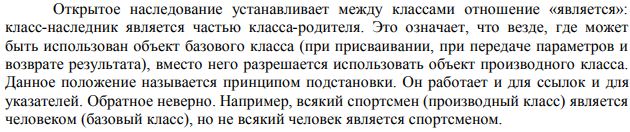


11. 



12. 



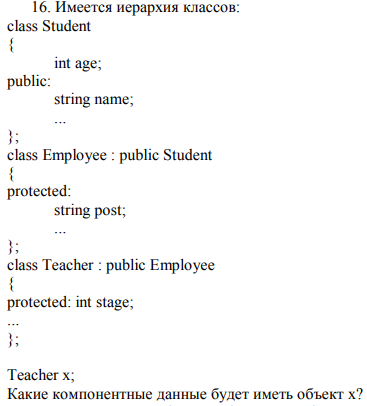
13. 



14. 



15. Пункт 13 и 14 – принцип подстановки.



16. age, name, post, stage.



17. Student(); Employee() : Student(); Teacher() : Employee();