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 12

Выполнила:

студент группы ИВТ-20-2Б Ананина Арина Юрьевна

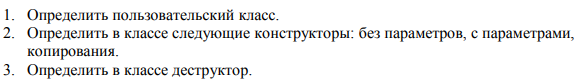
Проверила:

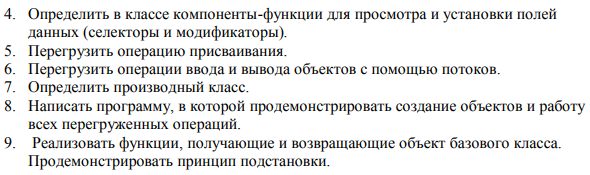
Доцент кафедры ИТАС

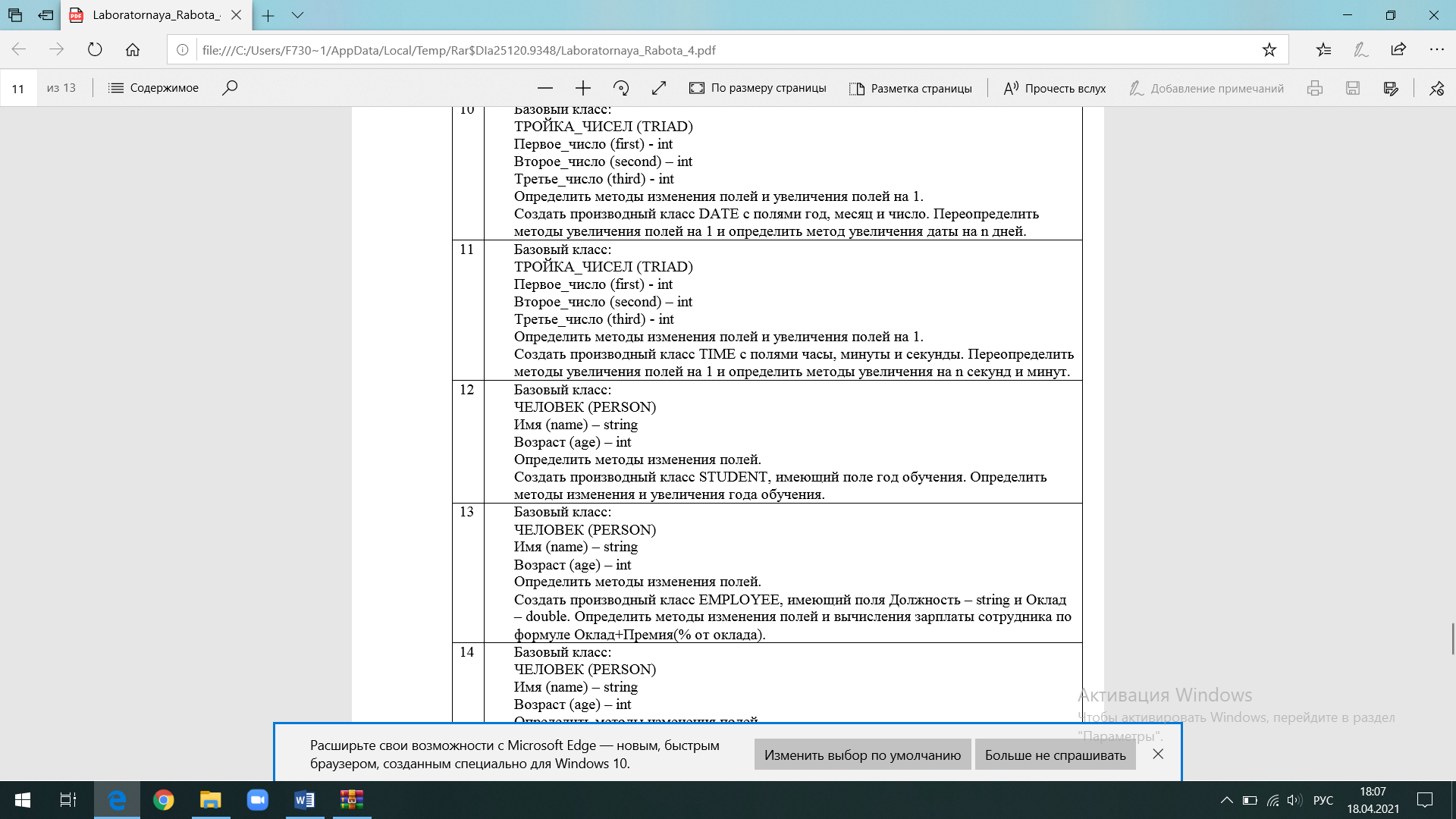
Полякова О.А.

Пермь, 2021

**Постановка задачи**







**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Pair для хранения первого числа пары (first), второго числа пары (second).

**1.2.** Организовать класс Complex с полями типа int real, imaginary, который является производным класса Pair.

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

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

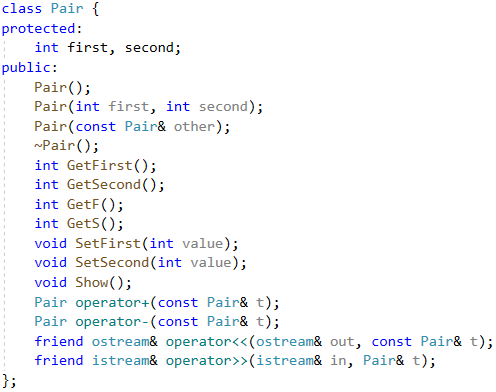
**1.5.** Организовать перегрузку оператора << дружественной классу Pair.

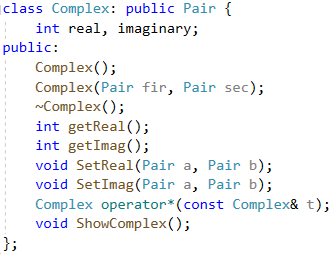
**1.6.** Организовать перегрузку оператора - дружественной классу Pair.

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

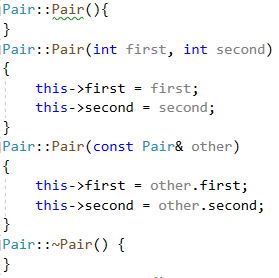
**2.** В ходе работы были использованы следующие типы данных:

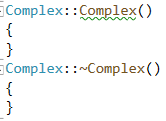
**2.1.** Организовать класс Pair для хранения первого числа пары (first), второго числа пары (second). в заголовочном файле Pair.h.

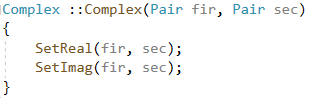




**2.2.** Организовать соответствующий файл Pair.cpp, в котором будут описаны методы, конструкторы и дружественные функции класса Pair.







**2.3.** Переменные типа int: tmp, res1, res2, где tmp – введённое число, res1– значение поля real класса Complex (производный класса Pair), res2 – значение поля imaginary класса Complex.

int res1=p.getReal(), res2=p.getImag();

int tmp;

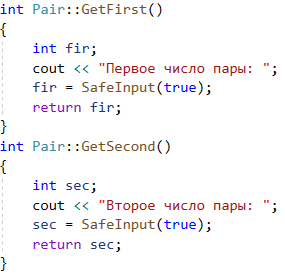
**2.4.** Объекты a, b, tmp класса Pair. Объект p принадлежит классу Complex.

Pair a, b;

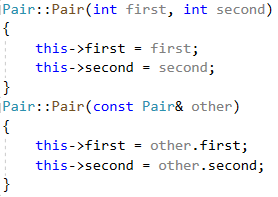
Complex p(a, b);

**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Данные вводятся через объекты класса Pair через гетторы.

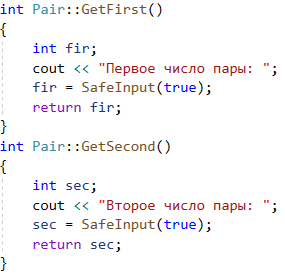


**3.2.** Данные также вводятся через объекты класса Pair через конструктор с параметрами.



**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для ввода пар чисел используются гетторы (методы, описанные в классе для ввода), в которых используется функция cin.



**4.2.** Для вывода на консоль суммы и разности чисел пар чисел используется функци cout.

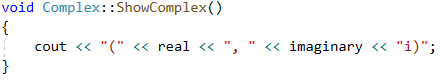
cout << endl<<"Сумма: " << a + b << endl;

cout << "\nРазность: " << a-b << endl;

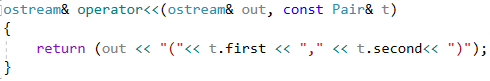
**4.3.** Для вывода на консоль произведения и разности чисел пар чисел используется метод ShowComplex() класса Complex, в котором используется функция cout.

cout << "Комплексное число: ";

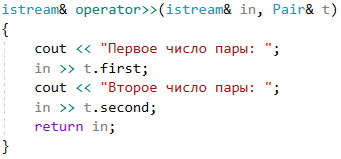
p.ShowComplex();



**4.4.** Так же вывод на консоль может быть осуществлён с помощью перегрузки оператора <<.

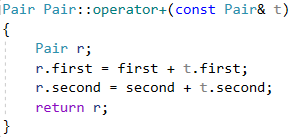


**4.5.** Так же ввод пар чисел может быть осуществлён с помощью перегрузки оператора >>.

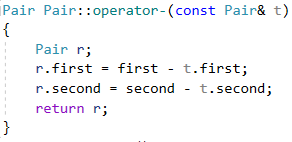


**5.** Поставленные задачи будут решены следующими действиями:

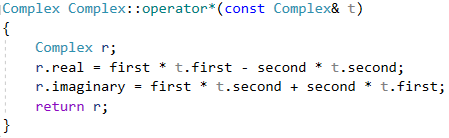
**5.1.** В перегрузке оператора + выполняется сложение полей first двух объектов класса Pair, затем сложение полей second этих объектов.



**5.2.** В перегрузке оператора - выполняется вычитание полей first двух объектов класса Pair, затем вычитание полей second этих объектов.



**5.3.** В перегрузке оператора \* выполняется преобразование полей real и imaginary по заданию.



**Код**

Person.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Person

{

public:

Person();

Person(string name, int age);

Person(const Person& p);

~Person();

void SetName(string name);

void SetAge(int age);

string GetName();

int GetAge();

//overloads

Person& operator= (const Person& p);

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

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

protected:

string name;

int age;

};

Student.h

#pragma once

#include <iostream>

#include <string>

#include "Person.h"

using namespace std;

class Student : public Person

{

public:

Student();

Student(string name, int age, int year);

Student(int year);

Student(const Student& s);

~Student();

void SetYear(int year);

int GetYear();

void AddYear(int year);

Student& operator= (const Student& s);

friend istream& operator>> (istream& in, Student& s);

friend ostream& operator<< (ostream& out, const Student& s);

protected:

int year;

};

StudentPerson.cpp

#include <iostream>

#include <string>

#include "Person.h"

#include "Student.h"

using namespace std;

Person::Person()

{

name = "noname";

age = 0;

}

Person::Person(string name, int age)

{

this->name = name;

this->age = age;

}

Person::Person(const Person& p)

{

this->name = p.name;

this->age = p.age;

}

Person::~Person()

{

}

void Person::SetName(string name)

{

this->name = name;

}

void Person::SetAge(int age)

{

this->age = age;

}

string Person::GetName()

{

return this->name;

}

int Person::GetAge()

{

return this->age;

}

Person& Person::operator= (const Person& p)

{

if (this == &p)

{

return \*this;

}

name = p.name;

age = p.age;

return \*this;

}

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

{

cout << "Введите имя: ";

in >> p.name;

cout << "Введите возраст: ";

in >> p.age;

return in;

}

ostream& operator<< (ostream& out, const Person& p)

{

out << "Имя: " << p.name << endl;

out << "Возраст: " << p.age << endl;

return out;

}

Student::Student(): Person()

{

this->year = 0;

}

Student::Student(string name, int age, int year): Person(name, age)

{

this->year = year;

}

Student::Student(int year): Person()

{

this->year = year;

}

Student::Student(const Student& s): Person(s)

{

this->year = s.year;

}

Student::~Student()

{

}

void Student::SetYear(int year)

{

this->year = year;

}

int Student::GetYear()

{

return this->year;

}

void Student::AddYear(int year)

{

this->year += year;

}

Student& Student::operator= (const Student& s)

{

if (this == &s)

{

return \*this;

}

name = s.name;

age = s.age;

year = s.year;

return \*this;

}

istream& operator>> (istream& in, Student& s)

{

cout << "Введите имя: ";

in >> s.name;

cout << "Введите возраст: ";

in >> s.age;

cout << "Введите год: ";

in >> s.year;

return in;

}

ostream& operator<< (ostream& out, const Student& s)

{

out << "Имя: " << s.name << endl;

out << "Возраст: " << s.age << endl;

out << "Год: " << s.year << endl;

return out;

}

Main.cpp

#include <iostream>

#include "Student.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Student s;

cin >> s;

cout << s << endl;

Student s2("Арина", 23, 12);

cout << s2 << endl;

Student s3(10);

cout << s3 << endl;

s3.SetAge(20);

s3.SetName("Имя");

s3.SetYear(1990);

cout << s3 << endl;

Student s4;

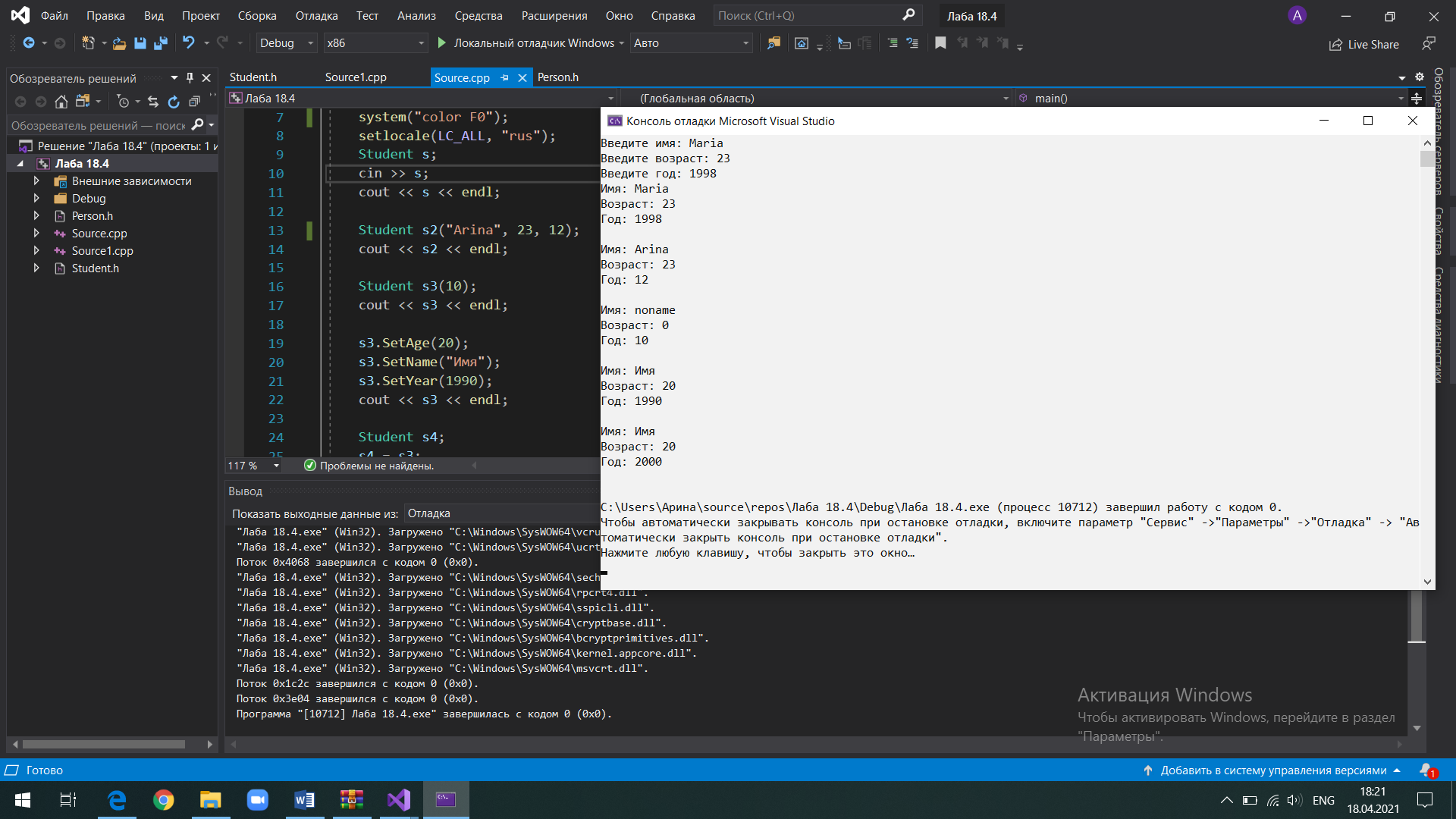
s4 = s3;

s4.AddYear(10);

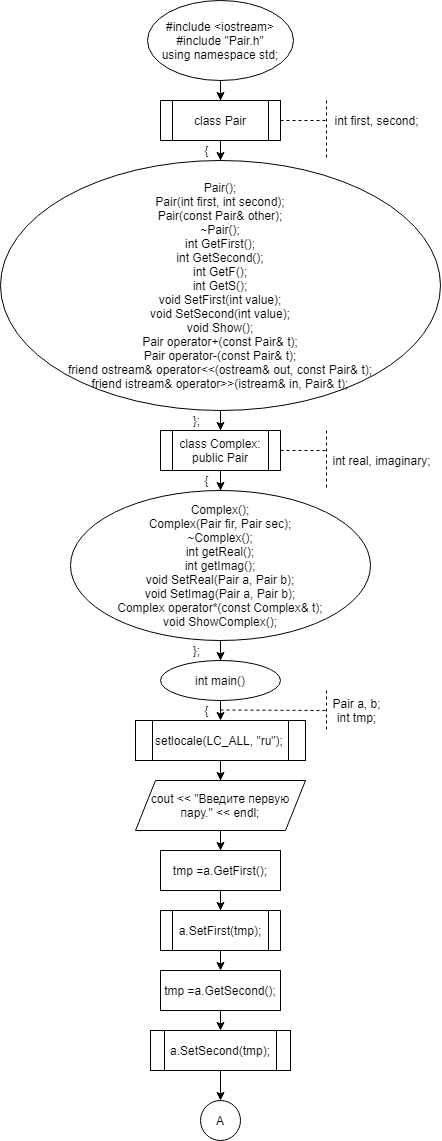
cout << s4 << endl;

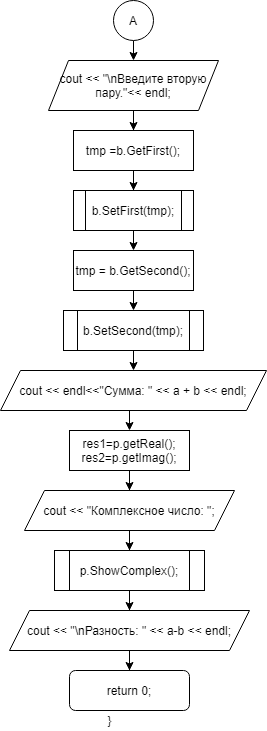
}

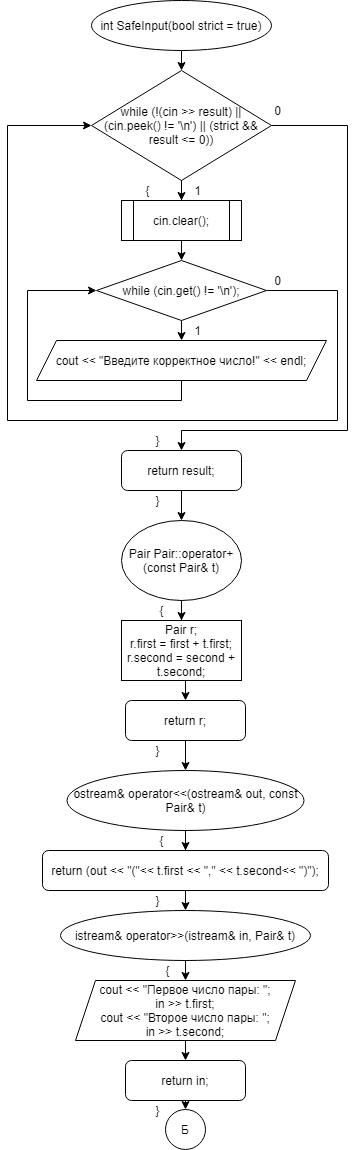
**Выполнение**

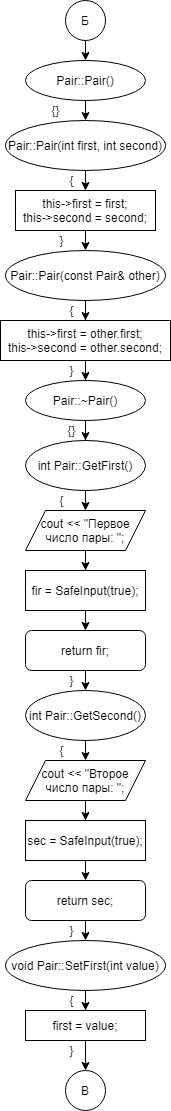


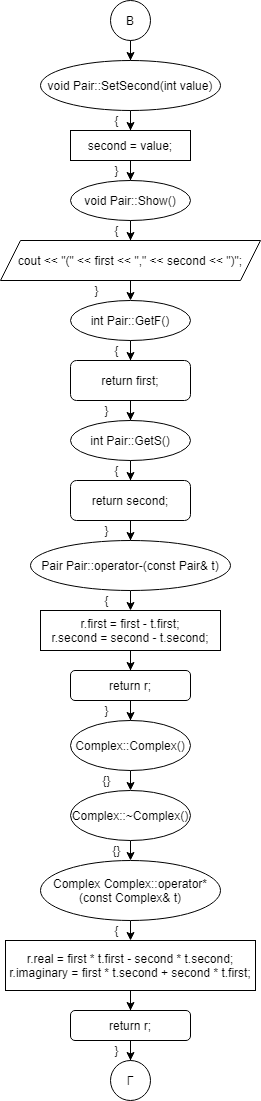
**Блок-схема**

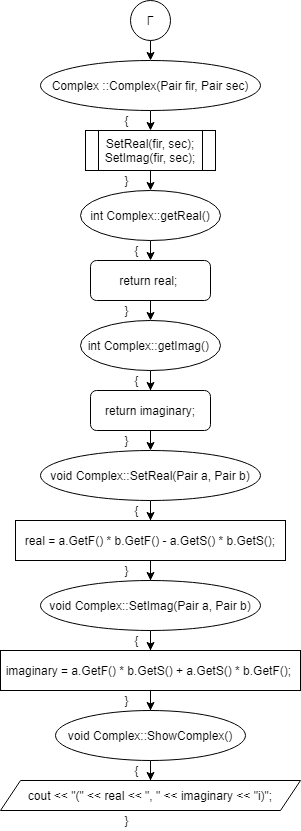












**Контрольные вопросы**



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. 

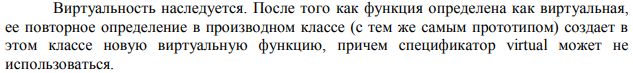


10. 

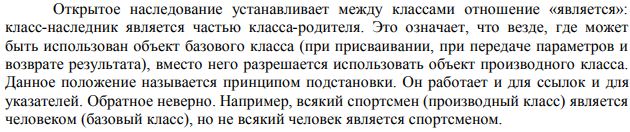


11. 



12. 



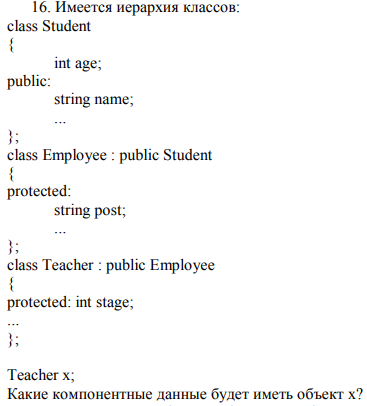
13. 



14. 



15. Пункт 13 и 14 – принцип подстановки.



16. age, name, post, stage.



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