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 №18.5**

Дисциплина: «Информатика»

Тема: “Объектно-ориентированное программирование. Виртуальные функции. Полиморфизм”

Вариант 12

Выполнила:

Студент группы ИВТ-20-2Б Ананина Арина Юрьевна

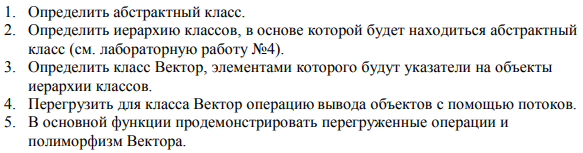
Проверила:

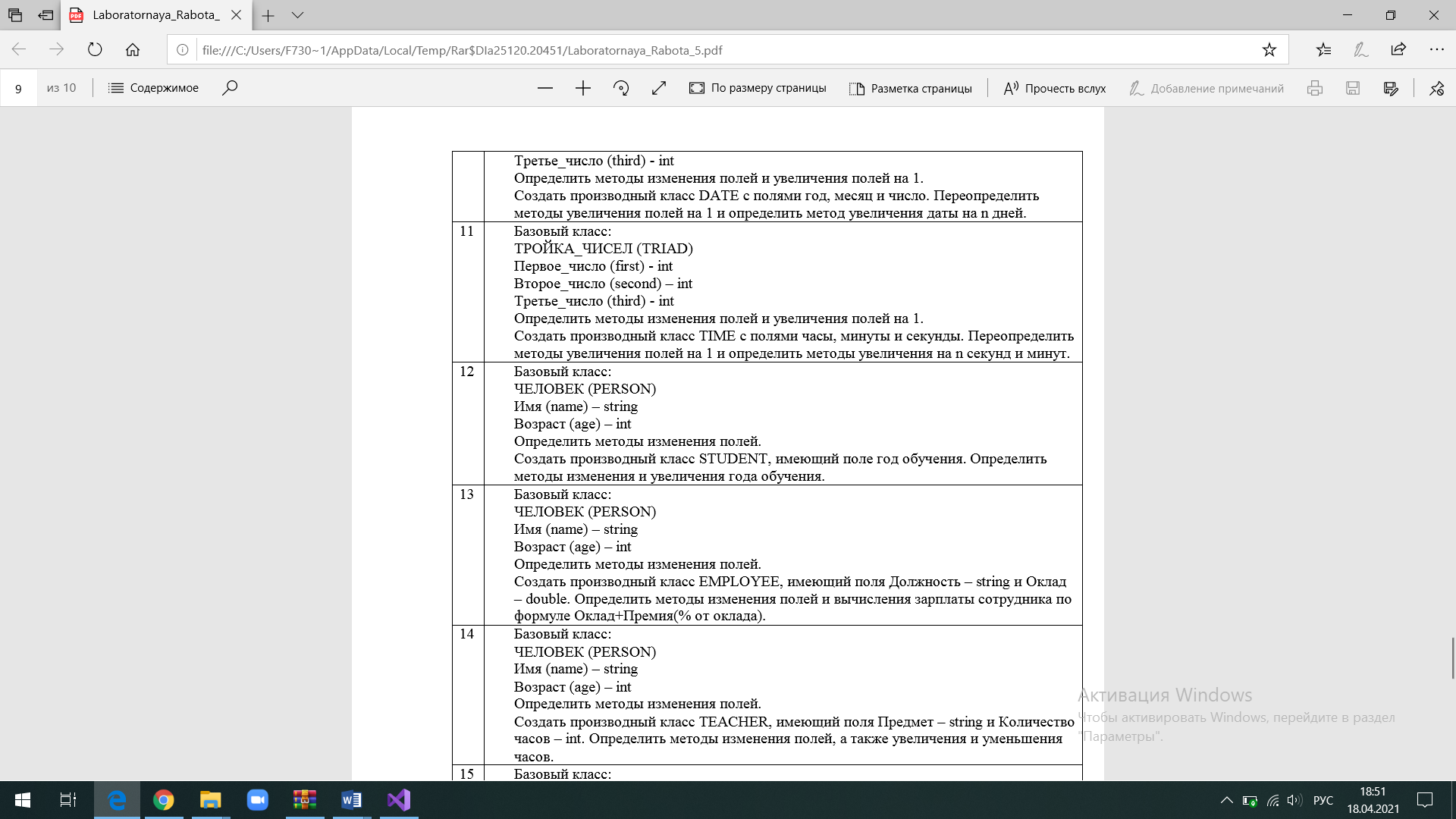
Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**





**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Abstract, который будет содержать чисто виртуальную функцию Show() для вывода объектов разных классов на консоль.

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

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

**1.4.** Организовать класс Vector с полями Abstract\*\* beg, int current, size.

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

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

**1.6.** Организовать перегрузку оператора << дружественной классу Vector.

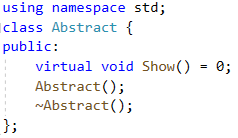
**1.7.** Организовать перегрузку оператора << дружественной классу Pair.

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

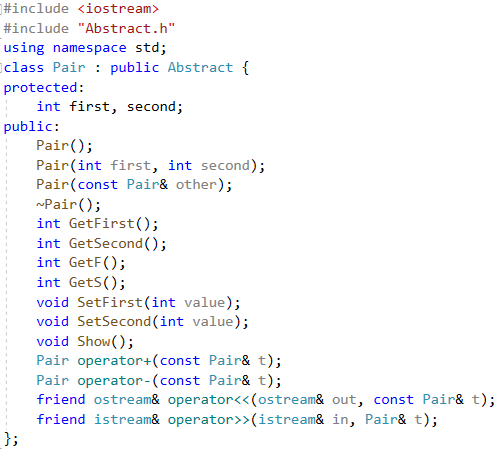
**1.9.** Организовать перегрузку оператора + дружественной классу Pair.

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

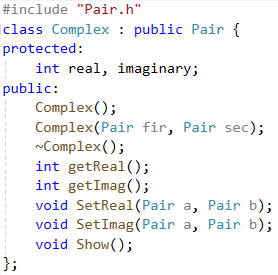
**2.1.** Класс Abstract, который будет содержать чисто виртуальную функцию Show() для вывода объектов разных классов на консоль, в заголовочном файле Abstract.h.



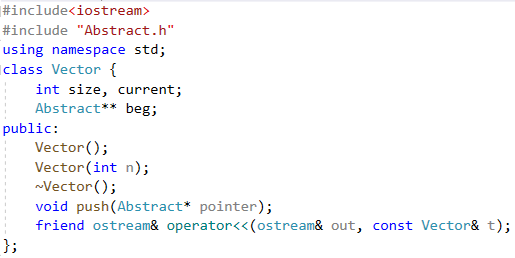
**2.2.** Класс Pair, который является производным классу Abstract, для хранения первого числа пары (first), второго числа пары (second) в заголовочном файле Pair.h.



**2.3.** Класс Complex с полями типа int real, imaginary, который является производным класса Pair в заголовочном файле Complex.h.



**2.4.** Класс Vector с полями Abstract\*\* beg, int current, size.



**2.5.** Переменную типа int: tmp, где tmp – введённое число.

int tmp;

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

Pair a, b;

Complex p(a, b);

**2.7.** Указатель ptr типа Abstract, который будет указывать ра обектры разных классов.

Abstract \*ptr = &a;

ptr = &b;

ptr = &p;

**2.8.** Объект vec типа Vector – массив объектов разных классов с помощью указателя ptr.

Vector vec(3);

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

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

Pair a, b;

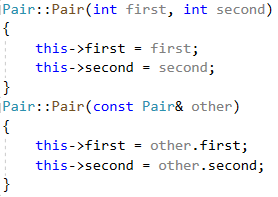
Complex p(a, b);

**3.2.** Указатель ptr типа Abstract, который будет указывать ра обектры разных классов.

Abstract \*ptr = &a;

ptr = &b;

ptr = &p;

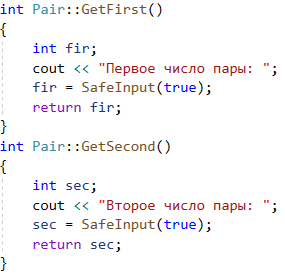


**3.3.** Объект vec типа Vector – массив объектов разных классов с помощью указателя ptr.

Vector vec(3);

**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.

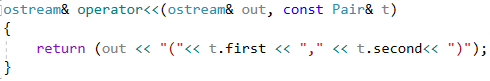


**4.2.** Для вывода на консоль суммы и разности чисел пар чисел используется функци cout.

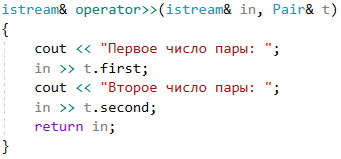
cout << endl<<"Сумма: " << a + b << endl;

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

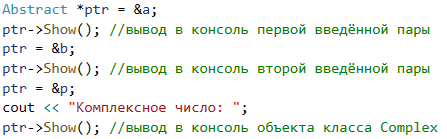
**4.4.** Так же вывод на консоль может быть осуществлён с помощью перегрузки оператора <<.



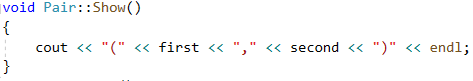
**4.5.** Так же ввод пар чисел может быть осуществлён с помощью перегрузки оператора >>.

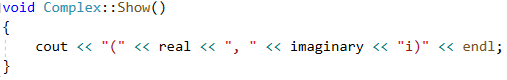


**4.5.** Кроме того, вывод введённых пар, вычисленных в классе Complex пары осуществляется через метод Show() класса Abstract по принципу полиморфизма.

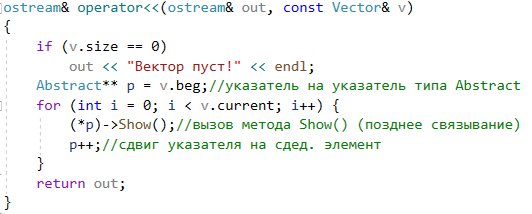






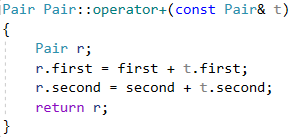


**4.6.** Элементы класса Vector выводятся через перегрузку оператора <<

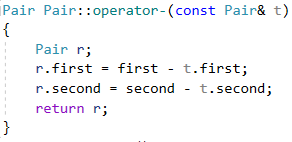


**5.** Поставленные задачи будут решены следующими действиями:

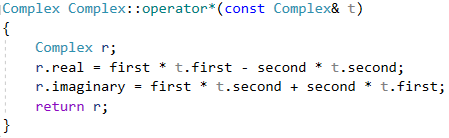
**5.1.** В перегрузке оператора + выполняется сложение полей first двух объектов класса Pair, затем сложение полей second этих объектов.



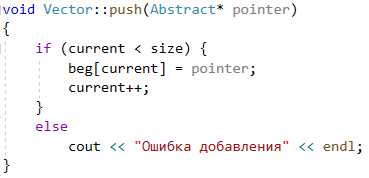
**5.2.** В перегрузке оператора - выполняется вычитание полей first двух объектов класса Pair, затем вычитание полей second этих объектов.



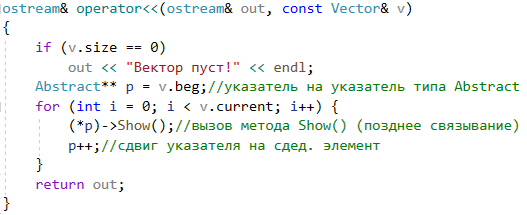
**5.3.** В перегрузке оператора \* выполняется преобразование полей real и imaginary по заданию.



**5.4.** В элемент класса вектор элементы добавляются через указатель pointer в методе push. Проверяется текущий индекс current на принадлежность множеству индексов vec, затем в вектор beg записывается элемент, на который указывает pointer.



**5.5.** В перегрузке оператора << происходит итерирование по циклу for от 0 до количества элементов в векторе. В цикле происходит вывод объектов через указатель и сдвиг к следующему элементу через инкремент.



**Код**

#pragma once

using namespace std;

class Object

{

public:

Object();

~Object();

virtual void Show()=0;

};

#include "Object.h"

Object::Object()

{

}

Object::~Object()

{

}

#include "Object.h"

using namespace std;

class Person :

public Object

{

public:

Person();

Person(string name, int age);

Person(const Person& p);

virtual ~Person();

void Show();

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;

};

#include <iostream>

#include <string>

#include "Person.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::Show()

{

cout << "Имя: " << this->name << endl;

cout << "Возраст: " << this->age << endl;

cout << endl;

}

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;

}

#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 Show();

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;

};

#include <iostream>

#include <string>

#include "Person.h"

#include "Student.h"

using namespace std;

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::Show()

{

cout << "Имя: " << this->name << endl;

cout << "Возраст: " << this->age << endl;

cout << "Год: " << this->year << endl;

cout << endl;

}

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;

}

#include <iostream>

#include "Student.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Student s;

cin >> s;

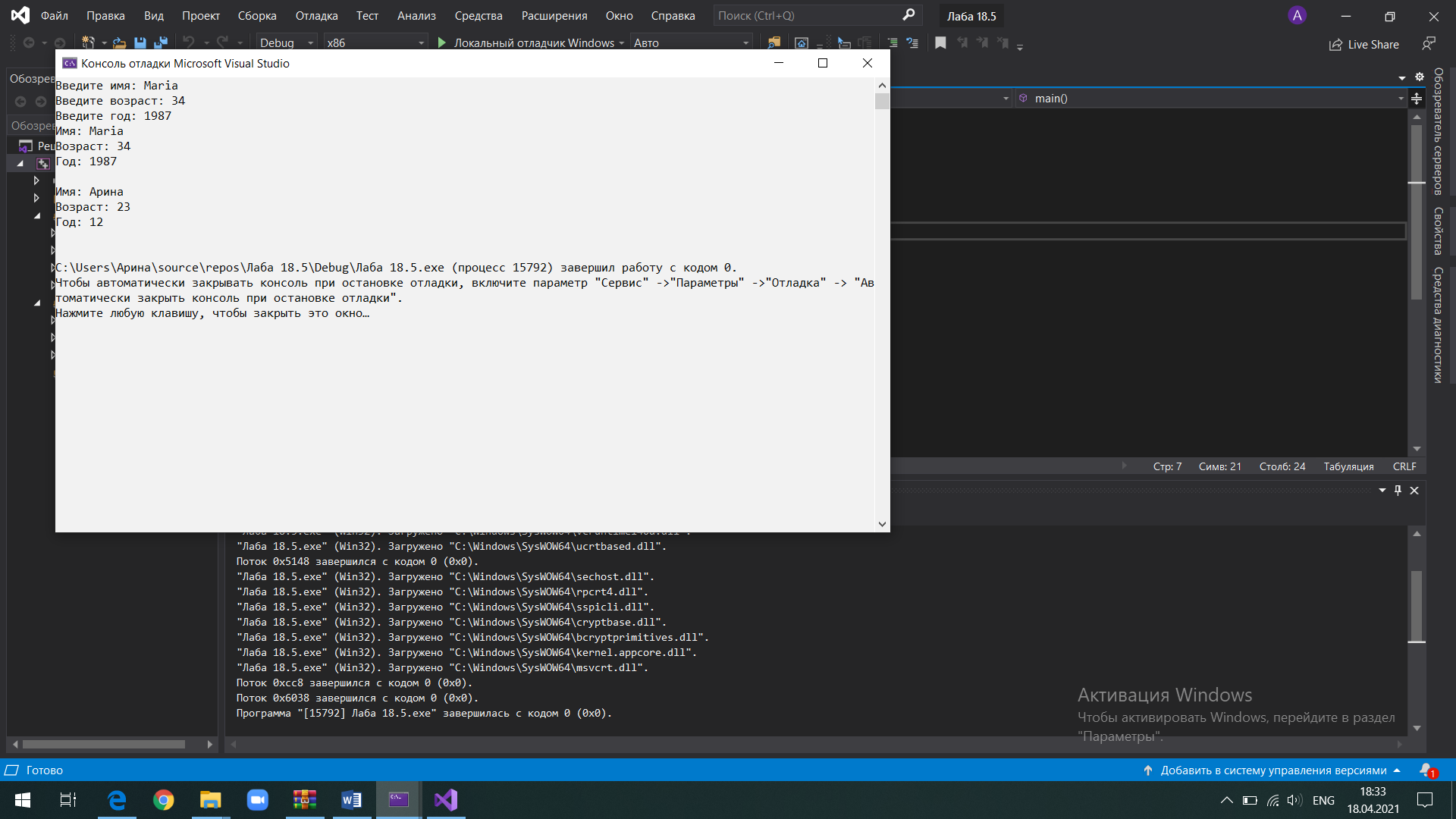
cout << s << endl;

Student s2("Арина", 23, 12);

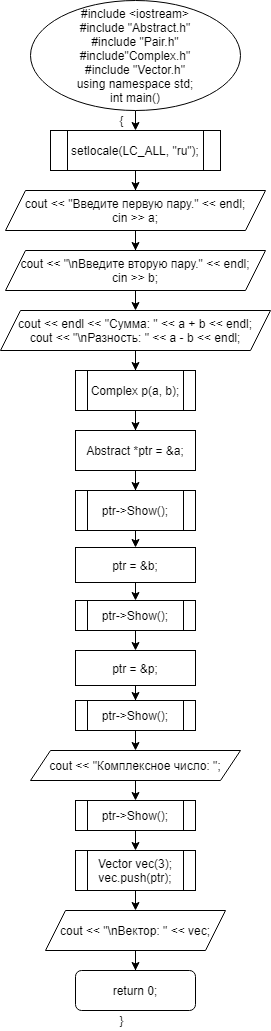
s2.Show();

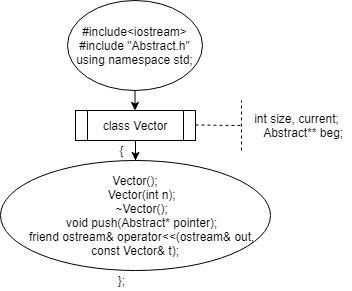
}

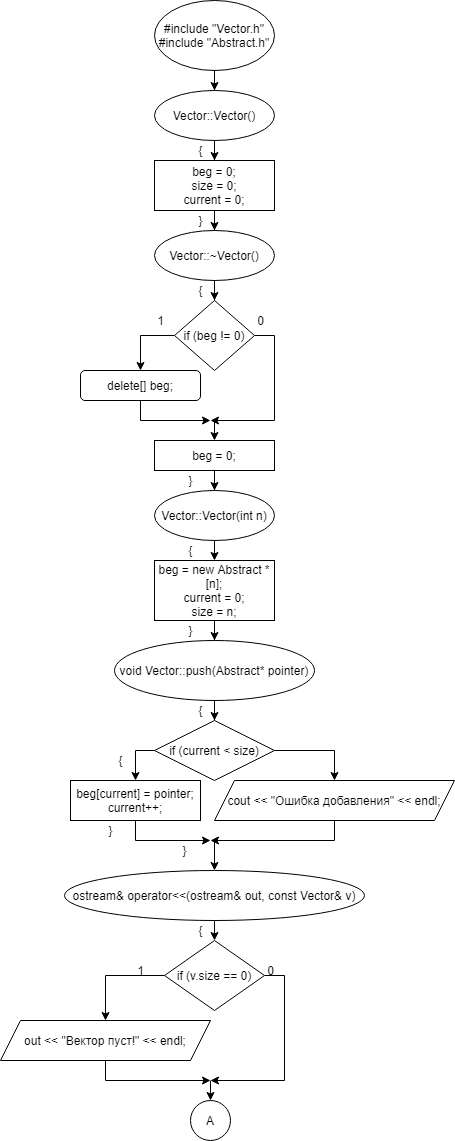
**Выполнение**

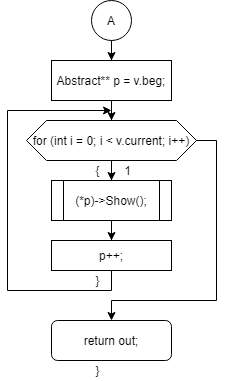


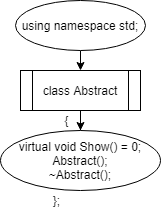
**Блок-схема**

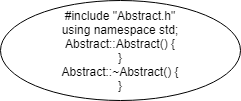


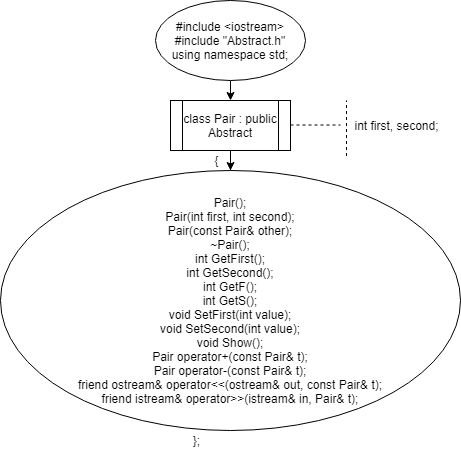


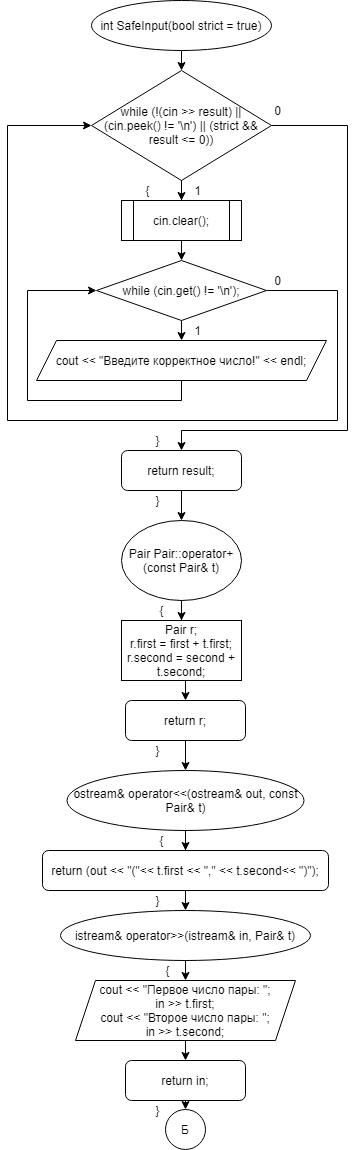


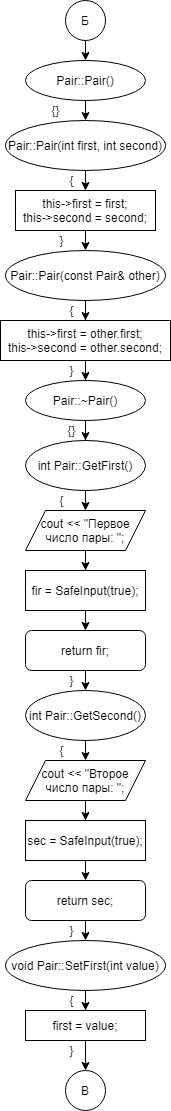


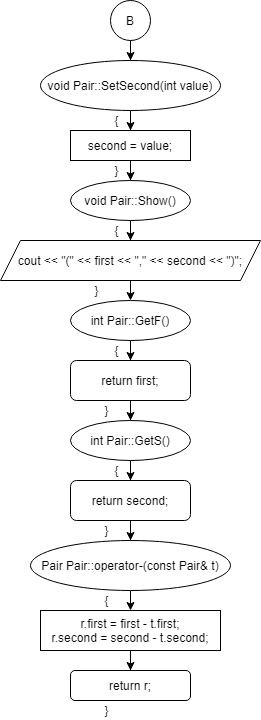


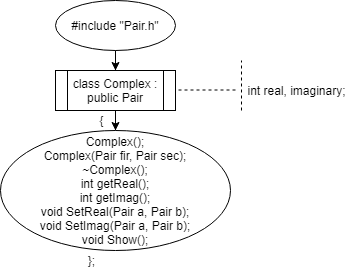


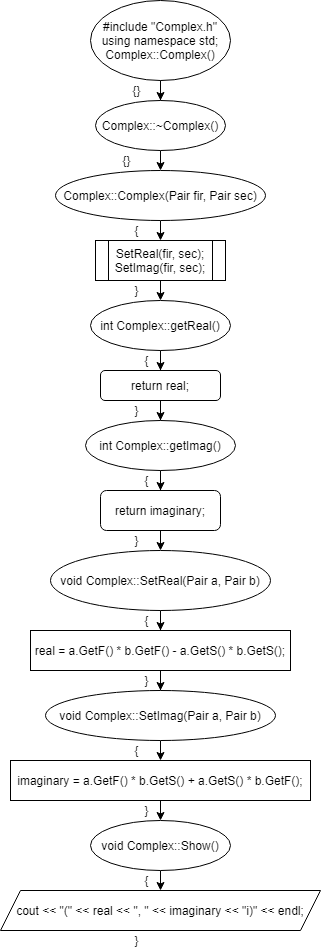


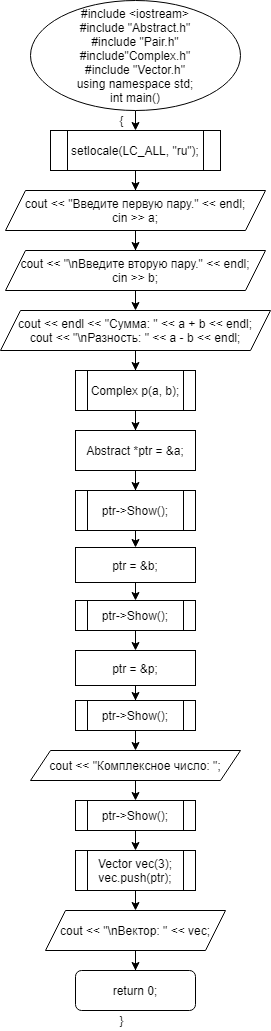






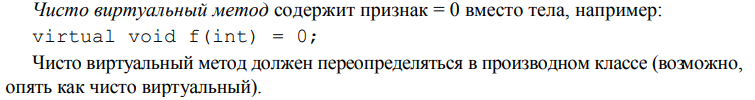






**Контрольные вопросы**

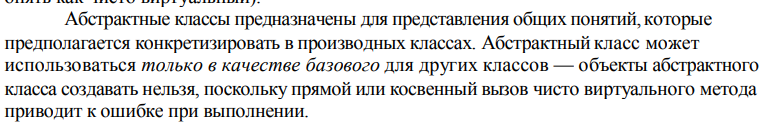


1. 



2. 



3. 



4. Полиморфные функции – это функции, которые работают с объектом любого типа в пределах одной иерархии.



5. При полиморфизме невозможно создать объект, так как базовым классом является абстрактный, когда в принципе подстановки используется наследование, что означает, что везде, где может быть использован объект базового класса, может быть и использован объект производного класса.



6. Пример: класс собака и класс кошка имеют одну из функций: говорить, но делают они это по-разному, так как собака гавкает, а кошка мяукает.



7. Метод Show, который наследуется от абстрактного класса, для вывода на консоль объектов разных классов.