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: основы алгоритмизации и программирования

Тема: “Классы и объекты. Полиморфизм.”

Вариант 4

Выполнил:

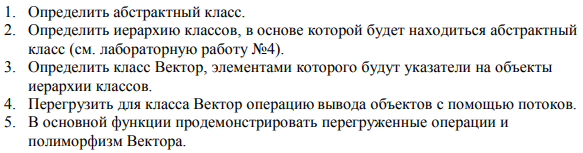
студент группы ИВТ-20-2Б Тедеев А.З.

Проверила: доцент кафедры ИТАС

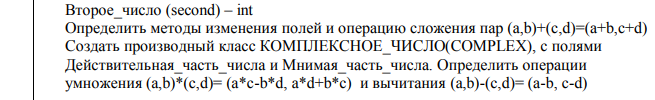
Полякова О.А.

Пермь, 2021

**Постановка задачи**







2

**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.

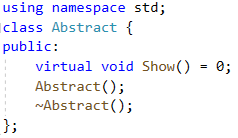
**1.7.** Организовать перегрузку оператора << дружественной классу Pair.

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

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

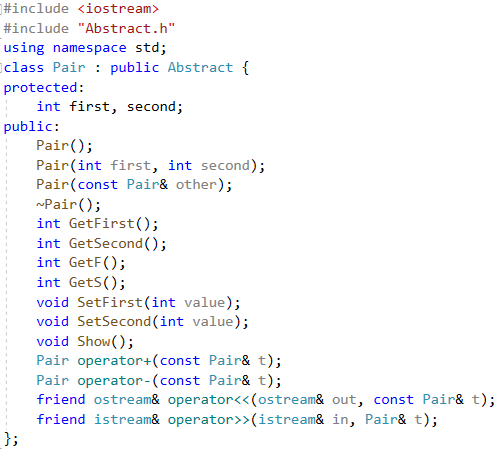
**2.** В ходе работы были использованы следующие типы данных:

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

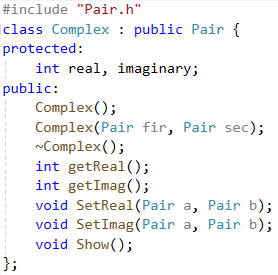


**2.2.** Класс Pair, который является производным классу Abstract, для хранения первого числа пары (first), второго числа пары (second) в заголовочном файле Pair.h.

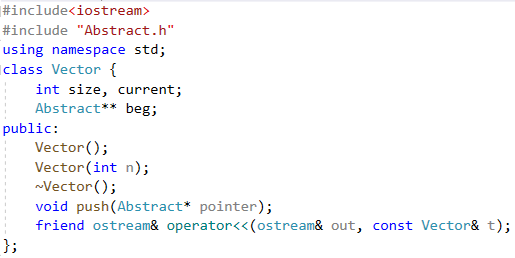
3



**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.



4

**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;

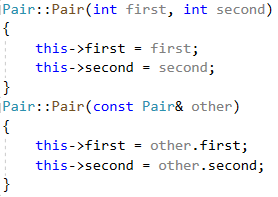
Complex p(a, b);

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

Abstract \*ptr = &a;

ptr = &b;

ptr = &p;



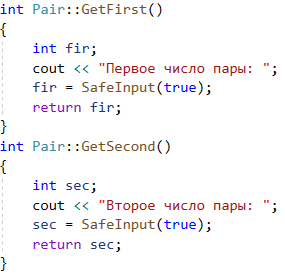
5

**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.

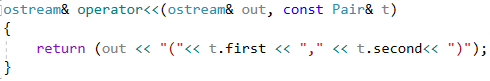


**4.2.** Для вывода на консоль суммы и разности чисел пар чисел используется функци cout.

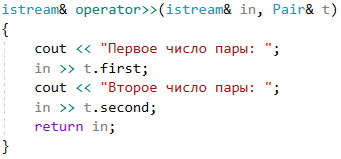
cout << endl<<"Сумма: " << a + b << endl;

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

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

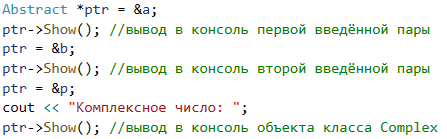


**4.5.** Так же ввод пар чисел может быть осуществлён с помощью перегрузки оператора >>.

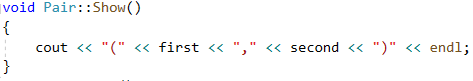


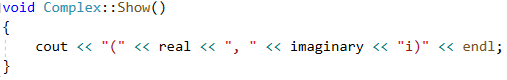
6

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

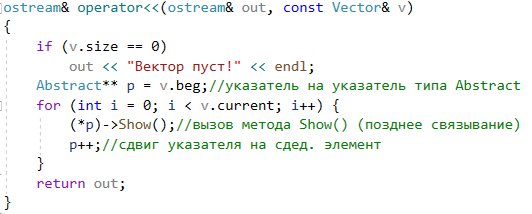






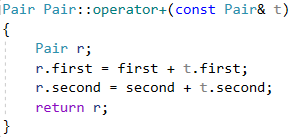


**4.6.** Элементы класса Vector выводятся через перегрузку оператора <<



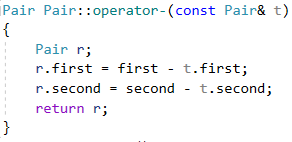
**5.** Поставленные задачи будут решены следующими действиями:

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

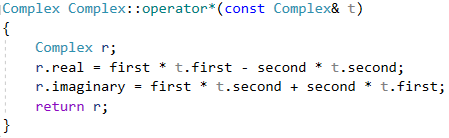


7

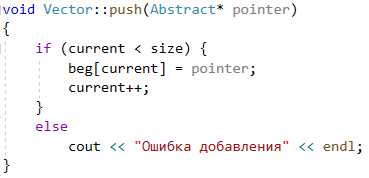
**5.2.** В перегрузке оператора - выполняется вычитание полей first двух объектов класса Pair, затем вычитание полей second этих объектов.



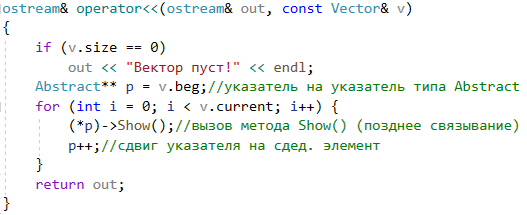
**5.3.** В перегрузке оператора \* выполняется преобразование полей real и imaginary по заданию.



**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.

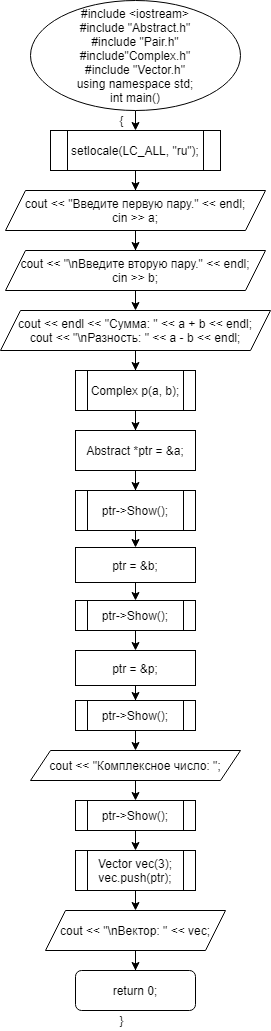


**5.5.** В перегрузке оператора << происходит итерирование по циклу for от 0 до количества элементов в векторе. В цикле происходит вывод объектов через указатель и сдвиг к следующему элементу через инкремент.

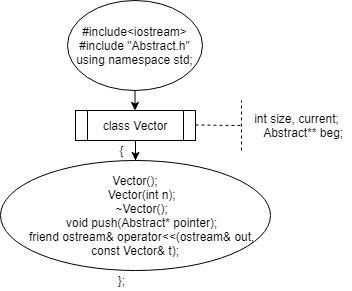


8

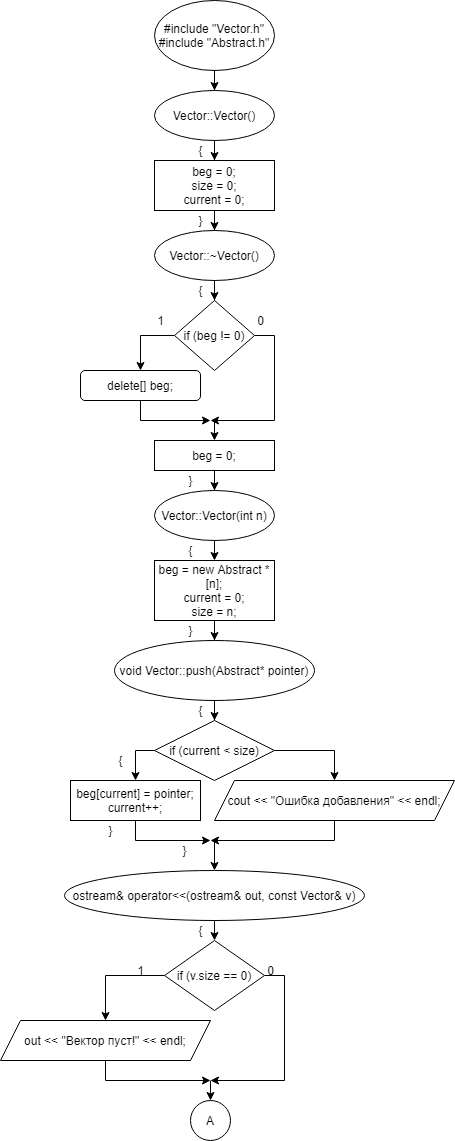
**Блок-схема**



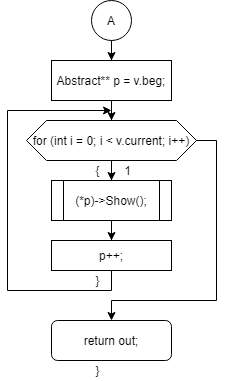
9



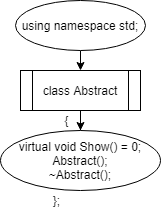
10



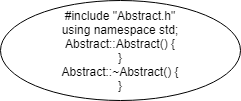
11



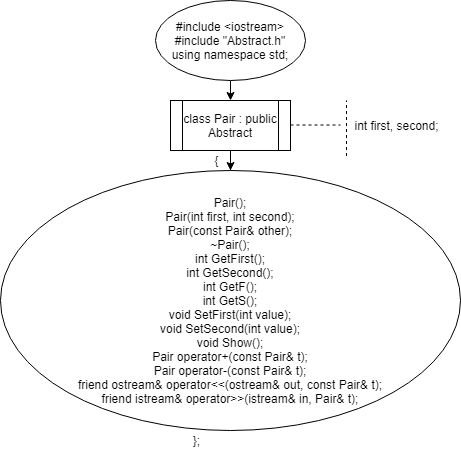
12



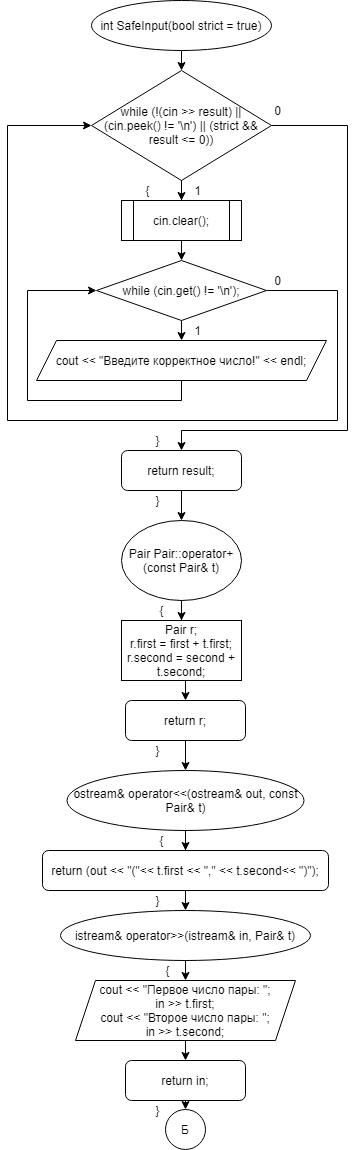
13



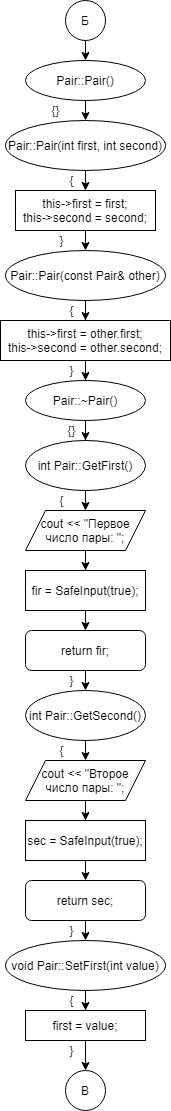
14



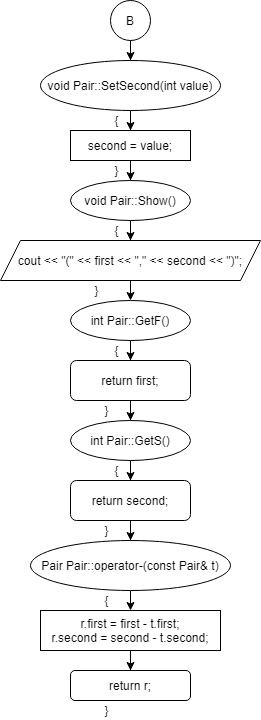
15



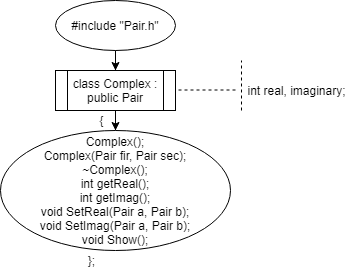
16



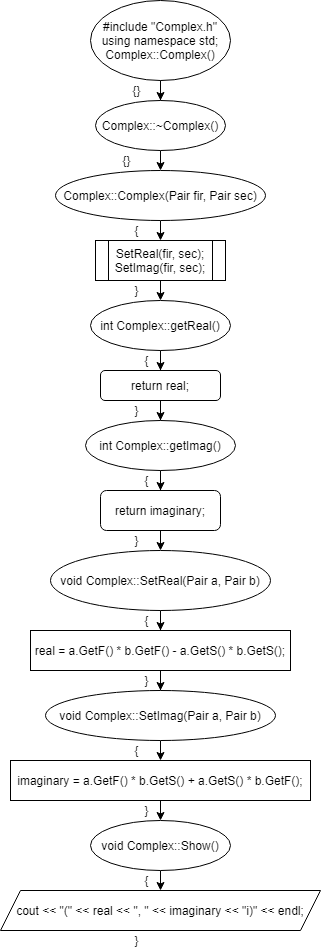
17



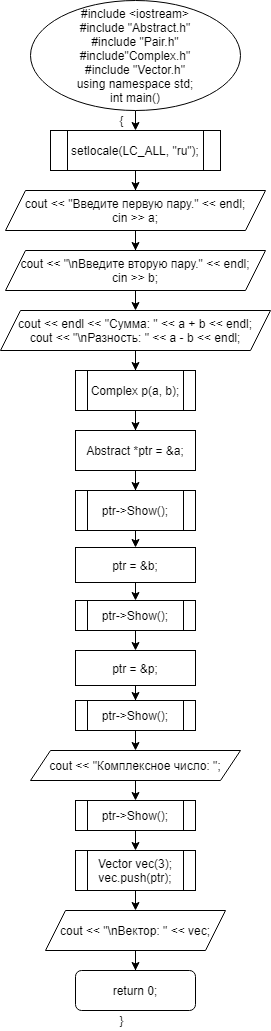
18



19



20



21

**Код**

#include <iostream>

#include "Abstract.h"

#include "Pair.h"

#include"Complex.h"

#include "Vector.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

Pair a, b;

int tmp;

cout << "Введите первую пару." << endl;

//cin >> a;

tmp = a.GetFirst();

a.SetFirst(tmp);

tmp = a.GetSecond();

a.SetSecond(tmp);

cout << "\nВведите вторую пару." << endl;

//cin >> b;

tmp = b.GetFirst();

b.SetFirst(tmp);

tmp = b.GetSecond();

b.SetSecond(tmp);

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

Complex p(a, b); //создание объекта класса Complex

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

Abstract \*ptr = &a;

ptr->Show(); //вывод в консоль первой введённой пары

ptr = &b;

ptr->Show(); //вывод в консоль второй введённой пары

ptr = &p;

cout << "Комплексное число: ";

ptr->Show(); //вывод в консоль объекта класса Complex

Vector vec(3);

vec.push(ptr); //добавление обекта класса Complex в вектор

cout << "\nВектор: " << vec;

return 0;

}

22

#pragma once

#include<iostream>

#include "Abstract.h"

using namespace std;

class Vector {

int size, current;

Abstract\*\* beg;

public:

Vector();

Vector(int n);

~Vector();

void push(Abstract\* pointer);

friend ostream& operator<<(ostream& out, const Vector& t);

};

23

#include "Vector.h"

#include "Abstract.h"

Vector::Vector()

{

beg = 0;

size = 0;

current = 0;

}

Vector::~Vector()

{

if (beg != 0)

delete[] beg;

beg = 0;

}

Vector::Vector(int n)

{

beg = new Abstract \* [n];

current = 0;

size = n;

}

void Vector::push(Abstract\* pointer)

{

if (current < size) {

beg[current] = pointer;

current++;

}

else

cout << "Ошибка добавления" << endl;

}

ostream& operator<<(ostream& out, const Vector& v)

{

if (v.size == 0)

out << "Вектор пуст!" << endl;

Abstract\*\* p = v.beg;//указатель на указатель типа Abstract

for (int i = 0; i < v.current; i++) {

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//сдвиг указателя на сдед. элемент

}

return out;

}

24

#pragma once

using namespace std;

class Abstract {

public:

virtual void Show() = 0;

Abstract();

~Abstract();

};

25

#include "Abstract.h"

using namespace std;

Abstract::Abstract() {

}

Abstract::~Abstract() {

}

26

#pragma once

#include <iostream>

#include "Abstract.h"

using namespace std;

class Pair : public Abstract {

protected:

int first, second;

public:

Pair();

Pair(int first, int second);

Pair(const Pair& other);

~Pair();

int GetFirst();

int GetSecond();

int GetF();

int GetS();

void SetFirst(int value);

void SetSecond(int value);

void Show();

Pair operator+(const Pair& t);

Pair operator-(const Pair& t);

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

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

};

27

#include "Pair.h"

int SafeInput(bool strict = true) {

int result;

while (!(cin >> result) || (cin.peek() != '\n') || (strict && result <= 0)) {

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число!" << endl;

}

return result;

}

Pair Pair::operator+(const Pair& t)

{

Pair r;

r.first = first + t.first;

r.second = second + t.second;

return r;

}

ostream& operator<<(ostream& out, const Pair& t)

{

return (out << "(" << t.first << "," << t.second << ")");

}

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

{

cout << "Первое число пары: ";

in >> t.first;

cout << "Второе число пары: ";

in >> t.second;

return in;

}

Pair::Pair() {

}

Pair::Pair(int first, int second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

Pair::~Pair() {

}

int Pair::GetFirst()

{

int fir;

cout << "Первое число пары: ";

fir = SafeInput(true);

return fir;

}

int Pair::GetSecond()

{

int sec;

cout << "Второе число пары: ";

sec = SafeInput(true);

return sec;

}

void Pair::SetFirst(int value)

{

first = value;

}

void Pair::SetSecond(int value)

28

{

second = value;

}

void Pair::Show()

{

cout << "(" << first << "," << second << ")" << endl;

}

int Pair::GetF()

{

return first;

}

int Pair::GetS()

{

return second;

}

Pair Pair::operator-(const Pair& t)

{

Pair r;

r.first = first - t.first;

r.second = second - t.second;

return r;

}

29

#pragma once

#include "Pair.h"

class Complex : public Pair {

protected:

int real, imaginary;

public:

Complex();

Complex(Pair fir, Pair sec);

~Complex();

int getReal();

int getImag();

void SetReal(Pair a, Pair b);

void SetImag(Pair a, Pair b);

void Show();

};

30

#include "Complex.h"

using namespace std;

Complex::Complex()

{

}

Complex::~Complex()

{

}

Complex::Complex(Pair fir, Pair sec)

{

SetReal(fir, sec);

SetImag(fir, sec);

}

int Complex::getReal()

{

return real;

}

int Complex::getImag()

{

return imaginary;

}

void Complex::SetReal(Pair a, Pair b)

{

real = a.GetF() \* b.GetF() - a.GetS() \* b.GetS();

}

void Complex::SetImag(Pair a, Pair b)

{

imaginary = a.GetF() \* b.GetS() + a.GetS() \* b.GetF();

}

void Complex::Show()

{

cout << "(" << real << ", " << imaginary << "i)" << endl;

}

31

#pragma once

#include<iostream>

#include "Abstract.h"

using namespace std;

class Vector {

int size, current;

Abstract\*\* beg;

public:

Vector();

Vector(int n);

~Vector();

void push(Abstract\* pointer);

friend ostream& operator<<(ostream& out, const Vector& t);

};

32

#include "Vector.h"

#include "Abstract.h"

Vector::Vector()

{

beg = 0;

size = 0;

current = 0;

}

Vector::~Vector()

{

if (beg != 0)

delete[] beg;

beg = 0;

}

Vector::Vector(int n)

{

beg = new Abstract \* [n];

current = 0;

size = n;

}

void Vector::push(Abstract\* pointer)

{

if (current < size) {

beg[current] = pointer;

current++;

}

else

cout << "Ошибка добавления" << endl;

}

ostream& operator<<(ostream& out, const Vector& v)

{

if (v.size == 0)

out << "Вектор пуст!" << endl;

Abstract\*\* p = v.beg;//указатель на указатель типа Abstract

for (int i = 0; i < v.current; i++) {

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//сдвиг указателя на сдед. элемент

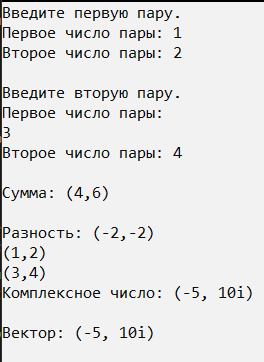
}

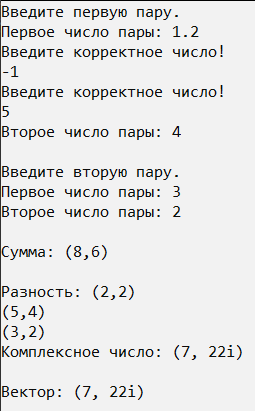
return out;

}

33

**Скриншоты**

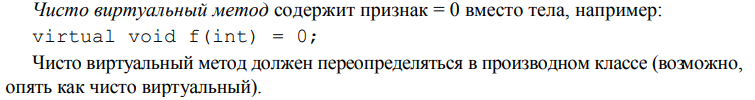




34

**Контрольные вопросы**

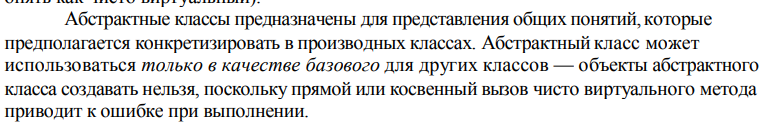


1. 



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.