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 РИС-20-1Б

Еске Вячеслав Сергеевич

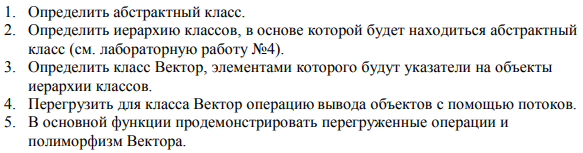
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**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 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.

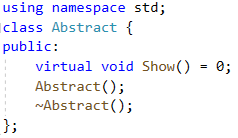
**1.7.** Организовать перегрузку оператора << дружественной классу Pair.

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

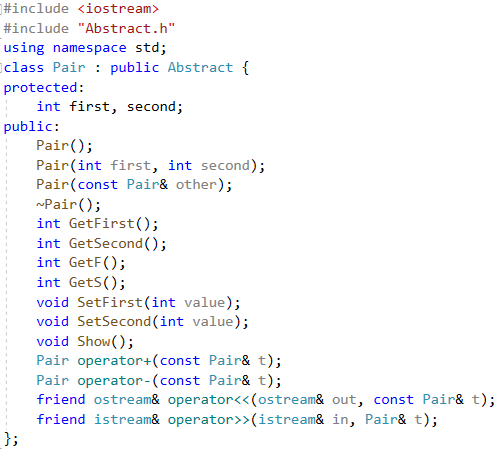
**1.9.** Организовать перегрузку оператора + дружественной классу Pair.

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

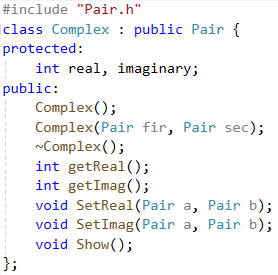
**2.1.** Класс Abstract, который будет содержать чисто виртуальную функцию Show() для вывода объектов разных классов на консоль, в заголовочном файле Abstract.h.



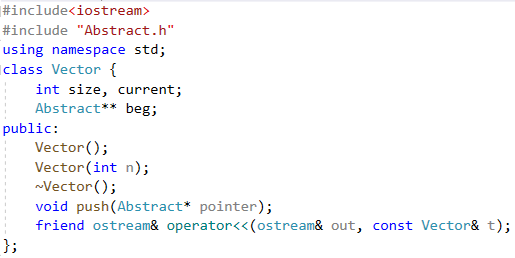
**2.2.** Класс Pair, который является производным классу Abstract, для хранения первого числа пары (first), второго числа пары (second) в заголовочном файле Pair.h.



**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;

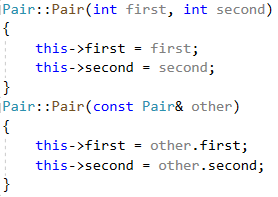
Complex p(a, b);

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

Abstract \*ptr = &a;

ptr = &b;

ptr = &p;

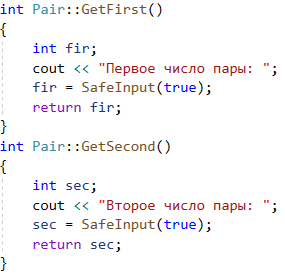


**3.3.** Объект vec типа Vector – массив объектов разных классов с помощью указателя ptr.

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.

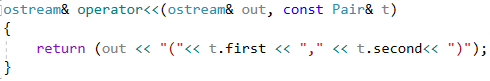


**4.2.** Для вывода на консоль суммы и разности чисел пар чисел используется функци cout.

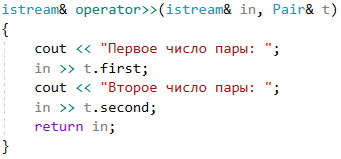
cout << endl<<"Сумма: " << a + b << endl;

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

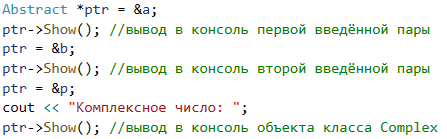
**4.4.** Так же вывод на консоль может быть осуществлён с помощью перегрузки оператора <<.



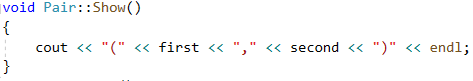
**4.5.** Так же ввод пар чисел может быть осуществлён с помощью перегрузки оператора >>.

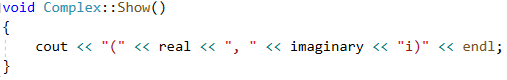


**4.5.** Кроме того, вывод введённых пар, вычисленных в классе Complex пары осуществляется через метод Show() класса Abstract по принципу полиморфизма.

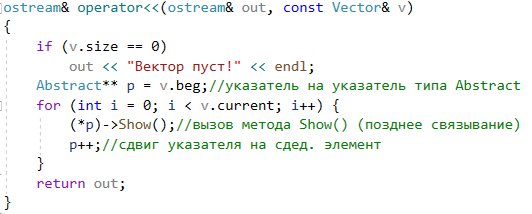






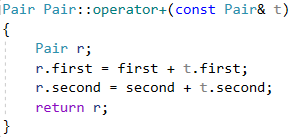


**4.6.** Элементы класса Vector выводятся через перегрузку оператора <<

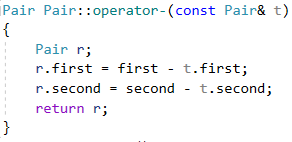


**5.** Поставленные задачи будут решены следующими действиями:

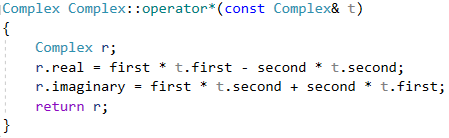
**5.1.** В перегрузке оператора + выполняется сложение полей first двух объектов класса Pair, затем сложение полей second этих объектов.



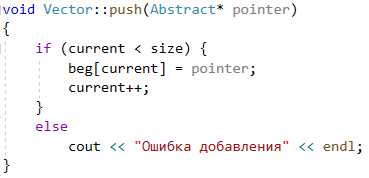
**5.2.** В перегрузке оператора - выполняется вычитание полей first двух объектов класса Pair, затем вычитание полей second этих объектов.



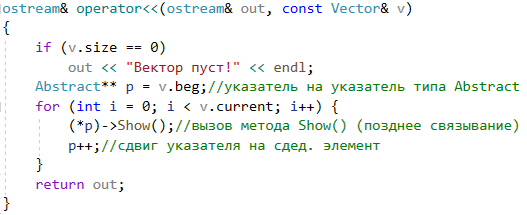
**5.3.** В перегрузке оператора \* выполняется преобразование полей real и imaginary по заданию.



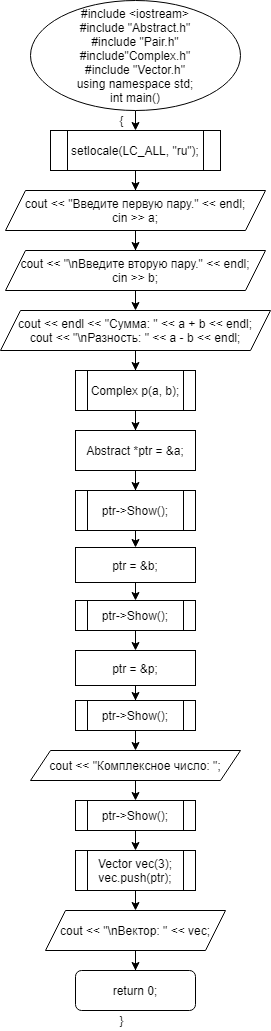
**5.4.** В элемент класса вектор элементы добавляются через указатель pointer в методе push. Проверяется текущий индекс current на принадлежность множеству индексов vec, затем в вектор beg записывается элемент, на который указывает pointer.



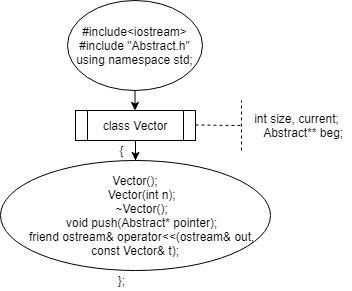
**5.5.** В перегрузке оператора << происходит итерирование по циклу for от 0 до количества элементов в векторе. В цикле происходит вывод объектов через указатель и сдвиг к следующему элементу через инкремент.

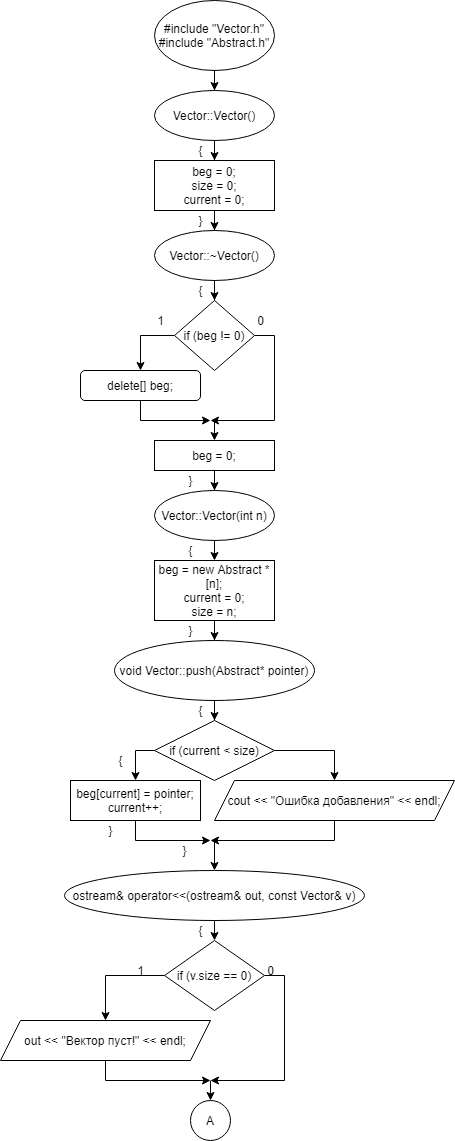


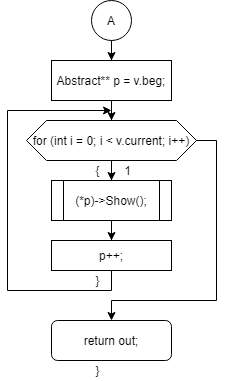
**Блок-схема**

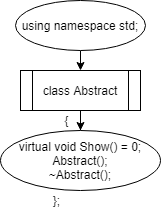


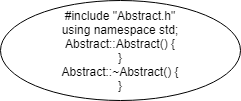
9

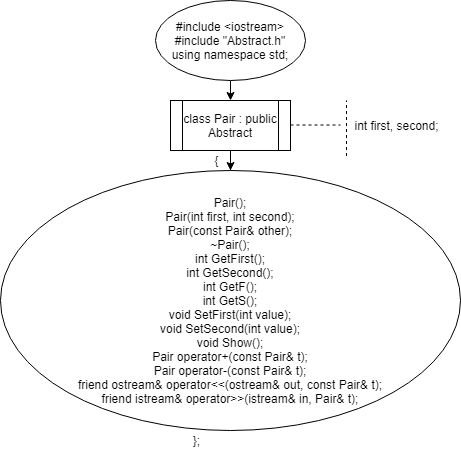


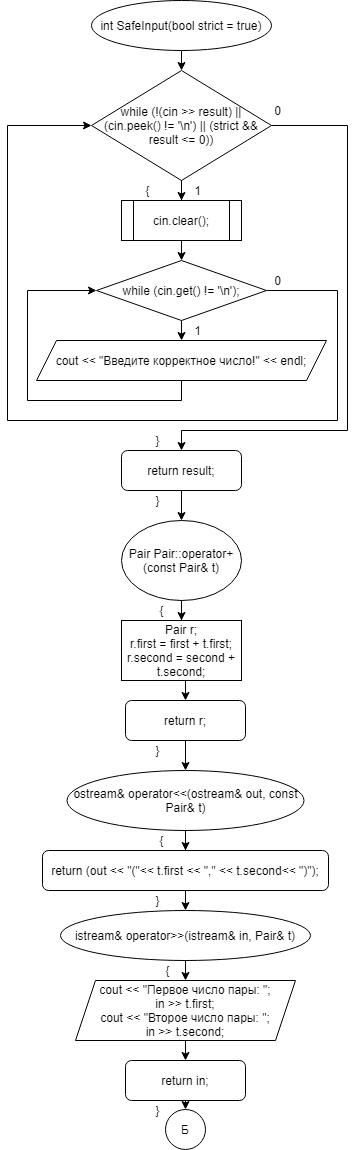


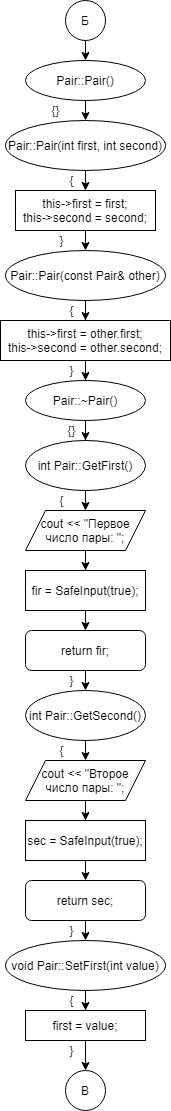


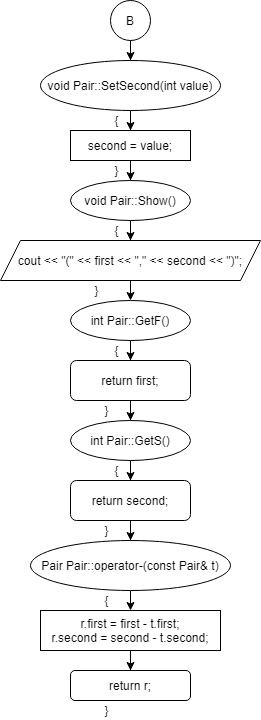


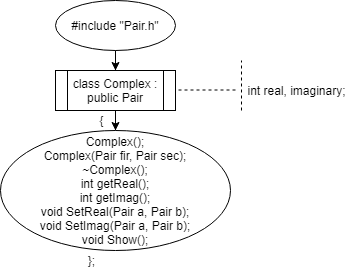


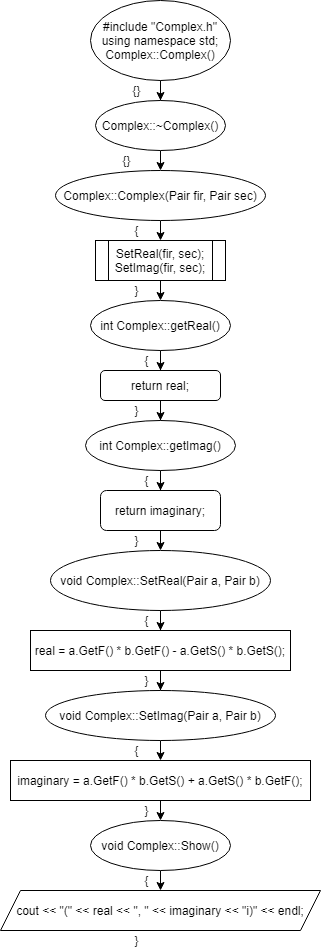


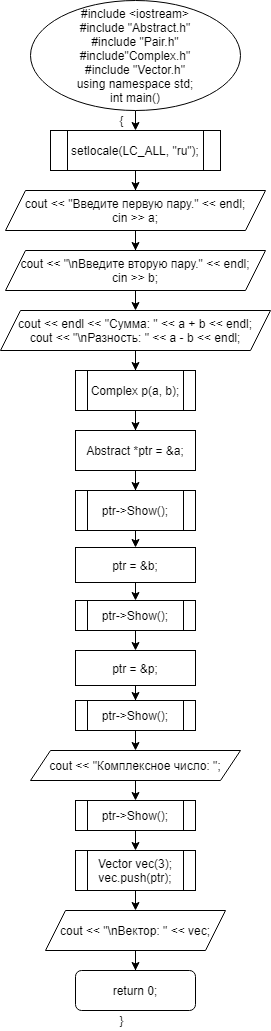












**Код**

#include <iostream>

#include "Object.h"

#include "Pair.h"

#include "Complex.h"

#include "Vector.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

/\*Pair a;

cin >> a;

cout << a << endl;//вывод с помощью перегруженной операции

Object \*p=&a;

p->Show();//вывод с помощью метода Show() и указателя

Complex b;

cin >> b;

cout << b << endl;//вывод с помощью перегруженной операции

p=&b;

p->Show();//вывод с помощью метода Show() и указателя \*/

Vector v(10);//вектор из 5 элементов

Pair a;//объект класса Pair

cin>>a;

Complex b;// объект класса Complex

cin>>b;

Complex c;// объект класса Complex

cin >> c;

Complex d;// объект класса Complex

Complex e;// объект класса Complex

Object \*p=&a;//ставим указатель на объект класса Pair

v.Add(p);//добавляем объект в вектор

p=&b;//ставим указатель на объект класса Complex

v.Add(p); //добавляем объект в вектор

p = &c;//ставим указатель на объект класса Complex

v.Add(p); //добавляем объект в вектор

d = b - c;

p = &d;

v.Add(p);

e = b \* c;

p = &e;

v.Add(p);

cout << v;//вывод вектора

}

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Pair:

public Object

{

public:

//конструктор без параметров public:

Pair();

//деструктор virtual

~Pair(void);

//функция для просмотра атрибутов класса с помощью указателя

void Show();

//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;

}

void Pair::Show() {

cout << "\nПервое число : " << first;

cout << "\nВторое число : " << second;

cout << "\n";

}

#pragma once

#include "Pair.h"

class Complex :

public Pair

{

public:

Complex(void); //конструктор без параметров

~Complex(void); //деструктор

void Show();

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;

}

void Complex::Show() {

cout << "\nКомплексное число : " << first << " + " << second << "i";

cout << "\n";

}

#pragma once

using namespace std;

class Object {

public:

Object(void) {};

~Object(void) {};

virtual void Show() = 0;//чисто виртуальная функция

};

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Vector {

public:

Vector(void);//конструктор без параметров

Vector(int);//конструктор копирования

public: ~Vector(void);//деструктор

void Add(Object \*);//добавление элемента в вектор

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

private:

Object\*\* beg;//указатель на первый элемент вектора

int size;//размер

int cur;//текущая позиция

};

#include "Vector.h"

//конструктор без параметров

Vector::Vector(void) {

beg = 0;

size = 0;

cur = 0;

}

//деструктор

Vector::~Vector(void) {

if (beg != 0)delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n) {

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add(Object \*p) {

if (cur < size) {

beg[cur] = p;

cur++;

}

}

//операция вывода

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

{

if (v.size == 0) out << "Empty" << endl; Object\*\* p = v.beg;

//указатель на указатель типа Object

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

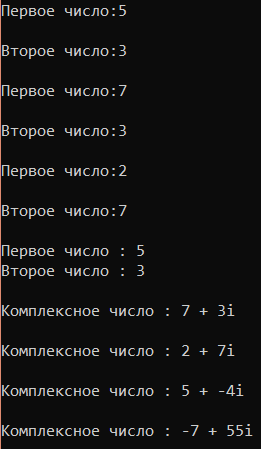
(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//передвигаем указатель на следующий объект

} return out;

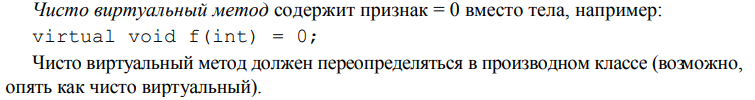
}

**Скриншоты**



**Контрольные вопросы**

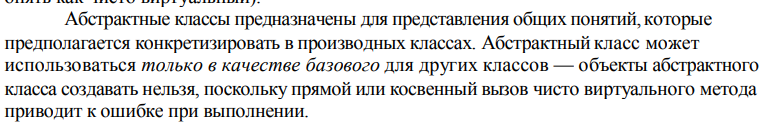


1. 



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.