**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**федеральное государственное образовательное бюджетное учреждение высшего образования**

**«Вологодский государственный университет»**

**(ВоГУ)**

* 1. **ОБЪЕКТНО-ОРЕНТИРОВАННОЕ программирование**
  2. **лабораторнАЯ РАБОТА №3**

**«Наследование в С++. Виртуальные функции»**

**Направление подготовки: 09.03.01 Информатика и вычислительная техника**

**Направленность (профиль): Программное обеспечение средств вычислительной техники и автоматизированных систем**

**Форма обучения: очная**

**Институт: Математики, естественных и компьютерных наук**

**Кафедра: Автоматики и вычислительной техники**

**Группа: 4Б09 ПО-41**

**Студент: Махонин М.Н.**

**Руководитель: Сорокин А.Н.**

Вологда

2021 г.

**Цель работы**: научиться применять виртуальные функции при разработке классов на языке С++.

**Вариант 2**

1. Создать базовый класс с виртуальными функциями: норма, вывод элемента на экран.

* Создать производные классы: комплексные числа, вектор из 10 элементов, матрица (3х3).
* В производных классах определить: скрытые поля, конструктор по умолчанию, конструктор с параметрами, методы для установки и получения значение полей, метод для вывода элемента на экран, метод для расчета нормы (для комплексных чисел - модуль в квадрате, для вектора - корень квадратный из суммы элементов по модулю, для матрицы - максимальное значение по модулю).

2. Разработать программу, которая демонстрировала бы работоспособность всех методов классов. Для проверки определить массив ссылок на базовый класс, которым присваиваются адреса различных объектов.

#include <iostream>

using namespace std;

class Base

{

public:

Base();

virtual double Normal();

virtual void Print();

~Base();

private:

};

Base::Base()

{

}

double Base::Normal()

{

return -1;

}

void Base::Print()

{

cout << "Base class" << endl;

}

Base::~Base()

{

}

class ComplexNumbers:public Base

{

public:

ComplexNumbers();

ComplexNumbers(int \_real, int \_imagine);

void SetReal(int value);

void SetImagine(int value);

int GetReal();

int GetImagine();

virtual double Normal();

virtual void Print();

~ComplexNumbers();

private:

int real;

int imagine;

};

ComplexNumbers::ComplexNumbers()

{

real = 0;

imagine = 0;

}

ComplexNumbers::ComplexNumbers(int \_real, int \_imagine)

{

real = \_real;

imagine = \_imagine;

}

void ComplexNumbers::SetReal(int value)

{

real = value;

}

void ComplexNumbers::SetImagine(int value)

{

imagine = value;

}

int ComplexNumbers::GetReal()

{

return real;

}

int ComplexNumbers::GetImagine()

{

return imagine;

}

double ComplexNumbers::Normal()

{

return pow(real, 2) + pow(imagine, 2);

}

void ComplexNumbers::Print()

{

cout << real << " + " << imagine << "i" << endl;

}

ComplexNumbers::~ComplexNumbers()

{

}

class Vector:public Base

{

public:

Vector();

Vector(int n);

void SetElement(int index, int elem);

int GetElement(int index);

virtual double Normal();

virtual void Print();

~Vector();

private:

int\* mas;

int count;

};

Vector::Vector()

{

mas = new int[3];

count = 3;

for (int i = 0; i < 3; i++) {

mas[i] = 0;

}

}

Vector::Vector(int n)

{

mas = new int[n];

count = n;

for (int i = 0; i < n; i++) {

mas[i] = 0;

}

}

void Vector::SetElement(int index, int elem)

{

if (index > count - 1) {

cout << "Индекс вне границ массива" << endl;

}

else {

mas[index] = elem;

}

}

int Vector::GetElement(int index)

{

if (index > count - 1) {

cout << "Индекс вне границ массива" << endl;

return LONG\_MIN;

}

else {

return mas[index];

}

}

double Vector::Normal()

{

int result = 0;

for (int i = 0; i < count; i++) {

result += abs(mas[i]);

}

return sqrt(result);

}

void Vector::Print()

{

for (int i = 0; i < count; i++) {

cout << mas[i] << " ";

}

cout << endl;

}

Vector::~Vector()

{

}

class Matrix:public Base

{

public:

Matrix();

Matrix(int \_row, int \_column);

void SetElement(int indexRow, int indexColumn, int value);

int GetElement(int indexRow, int indexColumn);

virtual double Normal();

virtual void Print();

~Matrix();

private:

int\*\* mas;

int row;

int column;

};

Matrix::Matrix()

{

row = column = 2;

mas = new int\* [row];

for (int i = 0; i < row; i++) {

mas[i] = new int[column];

for (int j = 0; j < column; j++) {

mas[i][j] = 0;

}

}

}

Matrix::Matrix(int \_row, int \_column)

{

row = \_row;

column = \_column;

mas = new int\* [row];

for (int i = 0; i < row; i++) {

mas[i] = new int[column];

for (int j = 0; j < column; j++) {

mas[i][j] = 0;

}

}

}

void Matrix::SetElement(int indexRow, int indexColumn, int value)

{

if (indexRow >= row || indexColumn >= column) {

cout << "Индекс вне границ массива" << endl;

return;

}

mas[indexRow][indexColumn] = value;

}

int Matrix::GetElement(int indexRow, int indexColumn)

{

if (indexRow >= row || indexColumn >= column) {

cout << "Индекс вне границ массива" << endl;

return LONG\_MIN;

}

return mas[indexRow][indexColumn];

}

double Matrix::Normal()

{

int\* maxNormals = new int[column];

for (int i = 0; i < column; i++) {

maxNormals[i] = 0;

for (int j = 0; j < row; j++) {

maxNormals[i] += abs(mas[j][i]);

}

}

int maxx = 0;

maxx = maxNormals[0];

for (int i = 1; i < column; i++) {

maxx = max(maxx, maxNormals[i]);

}

return maxx;

}

void Matrix::Print()

{

for (int i = 0; i < row; i++) {

for (int j = 0; j < column; j++) {

cout << mas[i][j] << " ";

}

cout << endl;

}

}

Matrix::~Matrix()

{

}

void main() {

setlocale(LC\_ALL, "RU");

ComplexNumbers number(33, 42);

cout << "Комплексное число" << endl;

number.Print();

cout << "Меняем мнимую часть на 10" << endl;

number.SetImagine(10);

number.Print();

cout << "Выводим норму" << endl;

cout << number.Normal() << endl;

cout << "------------------------------------------------------" << endl;

cout << "Вектор" << endl;

Vector vector(3);

vector.Print();

cout << "Меняем 2ое число на 3, 3 на 8" << endl;

vector.SetElement(1, 3);

vector.SetElement(2, 8);

vector.Print();

cout << "Выводим норму" << endl;

cout << vector.Normal() << endl;

cout << "------------------------------------------------------" << endl;

cout << "Матрица" << endl;

Matrix matrix;

matrix.Print();

cout << "Меняем элементы на 2 4 7 0" << endl;

matrix.SetElement(0, 0, 2);

matrix.SetElement(1, 0, 4);

matrix.SetElement(1, 1, 7);

matrix.Print();

cout << "Выводим норму" << endl;

cout << matrix.Normal() << endl;

cout << "------------------------------------------------------" << endl;

cout << "Проверка виртуальных функций" << endl;

Base\*\* base = new Base\*[3];

base[0] = &number;

base[1] = &vector;

base[2] = &matrix;

for (int i = 0; i < 3; i++) {

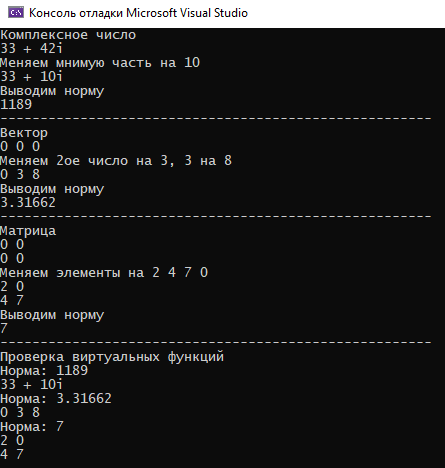
cout << "Норма: " << base[i]->Normal() << endl;

base[i]->Print();

}

}

Вывод:



**Контрольные вопросы**

1. **Что понимается под термином «наследование»? Перечислите преимущества и недостатки наследования.**

Наследование - На основе одних классов можно создавать новые классы, при этом новый класс называется производным или наследником, а класс основа — базовым или предком.

Класс наследник может использовать все элементы класса предка без повторного определения, то есть в наследнике определяются только дополнительные элементы, а остальные достаются по наследству.

Часто от одного предка можно произвести несколько потомков, более того в языке С++ разрешено и множественное наследование, то есть несколько предков у одного потомка.

*Достоинства:* возможность написания универсального кода в предках, который будет использоваться всеми потомками, то есть сокращение работы программиста, повышение надежности программы.

*Недостатки:* увеличение требуемого объема памяти, так как в памяти при выполнении программы должны находится все одноименные методы, реализованные в потомках. Обычно они оформляются в виде таблицы виртуальных методов (там хранится адрес начала каждого метода). Время выполнения программы замедляется, поскольку требуется дополнительное время на поиск в таблице виртуальных методов.

1. **Что понимается под термином «виртуальный метод»? Для каких целей применяются виртуальные методы?**

Виртуальный метод (виртуальная функция) — в объектно-ориентированном программировании метод (функция) класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения.

Виртуальный метод становится полезным только тогда, когда нужно программно один и тот же метод менять по некоторому алгоритму в программном коде.