**МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ   
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт прикладной математики и компьютерных наук

Кафедра информационной безопасности

**ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

Отчет по выполнению лабораторной работы №15  
Вариант №27

Выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ст. гр.230711 Якунин Роман Витальевич

Проверила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доц. каф. ИБ Басалова Галина Валерьевна

Тула 2022

# ЛАБОРАТОРНАЯ РАБОТА №15. НАСЛЕДОВАНИЕ В С++

## ЦЕЛЬ РАБОТЫ

Изучить основные принципы создания и использования иерархии классов; разработать приложения по своим вариантам заданий.

## ЗАДАНИЕ НА РАБОТУ

**Задание 1.** Ознакомиться с теоретическим материалом, приведенным в пункте «Краткие теоретические положения» данных методических указаний, а также с конспектом лекций и рекомендуемой литературой по данной теме.

**Задание 2.** Разработать иерархию классов «Алгебраическая функция, линейная функция, квадратичная функция, функция квадратного корня». Кроме указанных в варианте задания свойств и методов, можно добавить свои, необходимые по смыслу предметной области, свойства и методы классов. Минимальные требования:

- не менее двух виртуальных функций,

- не менее трех свойств у классов-потомков;

- не менее трех методов;

- наличие конструкторов у всех классов.

Составить диаграмму классов, реализовать составленную иерархию классов на языке С++.

**Задание 3.** Разработать основную программу (cpp-файл с функцией main), в которой используются созданные классы. В программе должны демонстрироваться возможности созданных классов.

## ХОД РАБОТЫ

Согласно заданию варианта №27, необходимо разработать программу с иерархией классов «Алгебраическая функция, линейная функция, квадратичная функция, функция квадратного корня». Диаграмма разработанных классов представлена на рисунке 1.



Рисунок 1 – Диаграмма классов

Описание разработанных классов и структур представлено в таблицах 1-4.

Таблица 1 – Описание разработанного базового класса Function

|  |  |  |
| --- | --- | --- |
| class Function | | |
| **Поля/свойства (элементы данных) класса** | | |
| Название и тип | Описание | |
| string name | Название функции | |
| string formula | Общая формула | |
| string domainOfDef | Область определения | |
| string setOfValues | Множество значений | |
| **Методы (функции-элементы) класса** | | |
| Название и тип возвращаемого значения | Аргументы | Описание |
| Function() | string newName, string newFormula, string domain, string setVal | Конструктор класса |
| virtual void Print() | void | Вывод свойств функции |
| virtual void PrintNulls() | void | Вывод нулей функции |
| virtual void Calculate() | double argument | Вычисление значения y по x |
| string GetName() | void | Функция для доступа к полю название |
| string GetFormula() | void | Функция для доступа к полю формула |
| string GetDef() | void | Функция для доступа к полю обл. определения функции |
| string GetVal() | void | Функция для доступа к полю множество значений функции |
| string GetName() | void | Функция для изменения поля название |
| string GetFormula() | void | Функция для изменения поля формула |
| string GetDef() | void | Функция для изменения поля обл. определения функции |
| string GetVal() | void | Функция для изменения поля множество значений функции |
| ~Function() | void | Деструктор класса |

Таблица 2 – Описание разработанного наследуемого класса LinearFunction

|  |  |  |
| --- | --- | --- |
| class LinearFunction : public Function | | |
| **Поля/свойства (элементы данных) класса** | | |
| Название и тип | Описание | |
| vector<double> coefficients | Массив коэффициентов | |
| Increase increasing | Монотонность | |
| Parity parity | Чётность | |
| **Методы (функции-элементы) класса** | | |
| Название и тип возвращаемого значения | Аргументы | Описание |
| LinearFunction() | double k, double b | Конструктор класса |
| void SetCoeffs | double k, double b | Задать коэффициенты |
| void Print() | void | Функция вывода свойств функции |
| void PrintNulls() | void | Вывод нулей функции |
| void Calculate() | double argument | Функция вычисления значения по аргументу |
| double GetNull() | void | Вернёт x, при котором y = 0 |
| double CalculateSolve | double argument | Вернёт значение функции при данном аргументе |
| void PrintIntervalsOfSign() | void | Вывод промежутков знакопостоянства |
| ~LinearFunction() | void | Деструктор класса |
| void UpdateInfo() | void | Обновить свойства для новых коэффициентов |

Таблица 3 – Описание разработанного наследуемого класса QuadraticFunction

|  |  |  |
| --- | --- | --- |
| class QuadraticFunction : public Function | | |
| **Поля/свойства (элементы данных) класса** | | |
| Название и тип | Описание | |
| vector<double> coefficients | Массив коэффициентов | |
| double minValue | Максимальное значение функции | |
| double maxValue | Максимальное значение функции | |
| bool isUp | Флаг true, если ветви вверх, иначе false | |
| const Parity parity | Чётность: чётная | |
| **Методы (функции-элементы) класса** | | |
| Название и тип возвращаемого значения | Аргументы | Описание |
| QuadraticFunction() | double a, double b, double c | Конструктор класса |
| bool isNullsExist() | void | Есть ли корни (y = 0 – пересечения с OX) |
| void Print() | void | Функция вывода свойств функции |
| void PrintNulls() | void | Вывод нулей функции |
| void Calculate() | double argument | Функция вычисления значения по аргументу |
| double GetNull() | void | Вернёт x, при котором y = 0 |
| double Discriminant() | void | Вернёт дискриминант уравнения |
| pair <double, double> GetRoots() | void | Вернёт пару корней |
| void PrintIntervalsOfSign() | void | Вывод промежутков знакопостоянства |
| double GetVertexAbsciss() | void | Вернёт абсциссу вершины параболы |
| ~QuadraticFunction() | void | Деструктор класса |

Таблица 4 – Описание разработанного наследуемого класса SquareRootFunction

|  |  |  |
| --- | --- | --- |
| class SquareRootFunction : public Function | | |
| **Поля/свойства (элементы данных) класса** | | |
| Название и тип | Описание | |
| const double minValue | Минимальное значение функции | |
| const Parity parity | Чётность: функция общего вида, симметрии нет | |
| double aCoeff | Коэффициент | |
| Bulge bulge | Выпуклость/вогнутость | |
| bool isIncrease | Флаг true, если ветвь возрастает, иначе false | |
| **Методы (функции-элементы) класса** | | |
| Название и тип возвращаемого значения | Аргументы | Описание |
| SquareRootFunction() | double a | Конструктор класса |
| bool isNullsExist() | void | Есть ли корни (y = 0 – пересечения с OX) |
| void Print() | void | Функция вывода свойств функции |
| void PrintNulls() | void | Вывод нулей функции |
| void Calculate() | double argument | Функция вычисления значения по аргументу |
| void PrintIntervalsOfSign() | void | Вывод промежутков знакопостоянства |
| ~ SquareRootFunction() | void | Деструктор класса |

## КОД ПРОГРАММЫ

## Содержимое заголовочного файла Function.h

#pragma once

#include <iostream>

using std::string;

enum Parity //чётность функции

{

even = 1, //чётная (симметрия относительно оси ординат OY)

odd = 2, //нечётная (симметрична относительно точки O)

general = 3 //функция общего вида (не явл. ни чётной, ни нечётной)

};

class Function

{

public:

/\*конструкторы\*/

Function();

Function(string newName, string newFormula, string domain, string setVal);

/\*функции-геттеры для доступа к закрытым полям\*/

string GetName(); //для доступа к полю name

string GetFormula(); //для доступа к полю formula

string GetDef(); //для доступа к полю domainOfDefinition

string GetVal(); //для доступа к полю setOfValues

/\*функции-сеттеры для изменения закрытых полей\*/

void SetName(string newName);

void SetFormula(string newFormula);

void SetDef(string newDef);

void SetVal(string newVal);

/\*виртуальные функции\*/

virtual void PrintNulls() = 0; //вывести нули функции

virtual void Calculate(double argument) = 0; //вычисление значения

virtual void Print() = 0; //вывести свойства функции

/\*деструктор\*/

~Function(){};

private:

string name; //название

string formula; //формула

string domainOfDef; //область определения функции (возможные x)

string setOfValues; //множество значений функции (возможные y)

};

## Содержимое файла реализации Function.cpp

#include "Function.h"

Function::Function()

{

name = "";

formula = "";

domainOfDef = "";

setOfValues = "";

}

Function::Function(string newName, string newFormula, string domain, string setVal)

{

name = newName;

formula = newFormula;

domainOfDef = domain;

setOfValues = setVal;

}

string Function::GetName()

{

return (\*this).name;

}

string Function::GetFormula()

{

return (\*this).formula;

}

string Function::GetDef()

{

return (\*this).domainOfDef;

}

string Function::GetVal()

{

return (\*this).setOfValues;

}

void Function::SetName(string newName)

{

name = newName;

}

void Function::SetFormula(string newFormula)

{

formula = newFormula;

}

void Function::SetDef(string newDef)

{

domainOfDef = newDef;

}

void Function::SetVal(string newVal)

{

setOfValues = newVal;

}

## Содержимое заголовочного файла LinearFunction.h

#pragma once

#include "Function.h"

#include <vector>

using std::vector;

enum Increase //возрастание/убывание функции

{

increase = 1,

decrease = 2,

constant = 3

};

class LinearFunction : public Function

{

public:

vector<double> coefficients; //массив коэффициентов

Increase increasing; //монотонность

Parity parity; //чётность

/\*конструкторы\*/

LinearFunction();

LinearFunction(double k, double b);

void SetCoeffs(double k, double b);

double GetNull(); //вернуть такое x, при котором y = 0

void PrintNulls(); //вывести нули функции

void Calculate(double argument); //вычисление значения

double CalculateSolve(double argument); //вернуть корень уравнения

void Print(); //вывести свойства функции

void PrintIntervalsOfSign(); //вывести промежутки знакопостоянства

~LinearFunction() {};

private:

void UpdateInfo(); //обновить свойства для новых коэффициентов

};

## Содержимое файла реализации LinearFunction.cpp

#include "LinearFunction.h"

#include <stdio.h>

LinearFunction::LinearFunction() : Function (

"Linear function",

"k\*x + b",

"x in R",

"y in R"

){

coefficients.resize(2);

coefficients[0] = 0;

coefficients[1] = 0;

increasing = constant;

parity = general;

}

LinearFunction::LinearFunction(double k, double b) : Function (

"Linear function",

"k\*x + b",

"x in R",

"y in R"

){

coefficients = { k, b };

if (coefficients[0] > 0)

increasing = increase;

if (coefficients[0] < 0)

increasing = decrease;

if (coefficients[0] == 0)

increasing = constant;

if (coefficients[0] == 0 && coefficients[1] != 0)

parity = even;

if (coefficients[0] != 0 && coefficients[1] == 0)

parity = odd;

if (coefficients[0] == 0 && coefficients[1] == 0)

parity = general;

if (coefficients[0] != 0 && coefficients[1] != 0)

parity = general;

}

void LinearFunction::SetCoeffs(double k, double b)

{

coefficients[0] = k;

coefficients[1] = b;

UpdateInfo();

}

void LinearFunction::UpdateInfo()

{

if (coefficients[0] > 0)

increasing = increase;

if (coefficients[0] < 0)

increasing = decrease;

if (coefficients[0] == 0)

increasing = constant;

if (coefficients[0] == 0 && coefficients[1] != 0)

parity = even;

if (coefficients[0] != 0 && coefficients[1] == 0)

parity = odd;

if (coefficients[0] == 0 && coefficients[1] == 0)

parity = general;

if (coefficients[0] != 0 && coefficients[1] != 0)

parity = general;

}

double LinearFunction::GetNull()

{

return ((-coefficients[1]) / coefficients[0]); //нуль функции достигается при x = (-b)/k

}

void LinearFunction::PrintNulls()

{

printf("\n\t\tНуль функции достигается при x = %3.3f\n", (-coefficients[1]) / coefficients[0]);

}

void LinearFunction::Calculate(double argument)

{

printf("\n\t\tФормула: y = (%3.3f) \* x + (%3.3f)\n", coefficients[0], coefficients[1]);

printf("\t\tПри x = %3.3f функция принимает значение y = %3.3f\n", argument,

argument \* (coefficients[0]) + coefficients[1]);

}

double LinearFunction::CalculateSolve(double argument)

{

return argument \* (coefficients[0]) + coefficients[1];

}

void LinearFunction::Print()

{

printf("\n\t\tНазвание функции: %s\n", (\*this).GetName().c\_str());

printf("\t\tФормула: y = (%3.3f) \* x + (%3.3f)\n", coefficients[0], coefficients[1]);

printf("\t\tОбласть определения функции: %s\n", (\*this).GetDef().c\_str());

printf("\t\tМножество значений функции: %s\n", (\*this).GetVal().c\_str());

if (parity == even)

printf("\t\tСвойство чётности: чётная\n");

if (parity == odd)

printf("\t\tСвойство чётности: нечётная\n");

if (parity == general)

printf("\t\tСвойство чётности: функция общего вида\n");

if (increasing == increase)

printf("\t\tМонотонность: возрастающая\n");

if (increasing == decrease)

printf("\t\tМонотонность: убывающая\n");

if (increasing == constant)

printf("\t\tМонотонность: константная\n");

printf("\t\tСвойством выпуклости не обладает\n");

}

void LinearFunction::PrintIntervalsOfSign()

{

printf("\n\t\tПромежутки знакопостоянства:\n");

if (coefficients[0] > 0)

{

printf("\t\ty > 0 при x > %3.3f\n", (-coefficients[1]) / coefficients[0]);

printf("\t\ty < 0 при x < %3.3f\n\n", (-coefficients[1]) / coefficients[0]);

}

if (coefficients[0] < 0)

{

printf("\t\ty > 0 при x < %3.3f\n", (-coefficients[1]) / coefficients[0]);

printf("\t\ty < 0 при x > %3.3f\n\n", (-coefficients[1]) / coefficients[0]);

}

if (coefficients[0] == 0 && coefficients[1] == 0)

printf("\t\ty = 0 при любом х\n");

if (coefficients[0] == 0 && coefficients[1] > 0)

printf("\t\ty > 0 при любом х\n");

if (coefficients[0] == 0 && coefficients[1] < 0)

printf("\t\ty < 0 при любом х\n");

}

## Содержимое заголовочного файла QuadraticFunction.h

#pragma once

#include "Function.h"

#include <vector>

using namespace std;

#define inf DBL\_MAX //бесконечноть

class QuadraticFunction : public Function

{

public:

vector<double> coefficients; //массив коэффициентов

double minValue;

double maxValue;

bool isUp; //true = ветви вверх, false = ветви вниз

/\*конструкторы\*/

QuadraticFunction();

QuadraticFunction(double a, double b, double c);

bool isNullsExist(); //есть ли корни (y = 0)

void PrintNulls(); //вывести корни уравнения

pair <double, double> GetRoots(); //вернуть корни уравнения

void Calculate(double argument); //вычисление значения функции

void Print(); //вывести свойства функции

void GetIntervalsOfSign(); //вывести промежутки знакопостоянства

double GetVertexAbsciss(); //вовзращает абсциссу вершины параболы

~QuadraticFunction() {};

private:

double Discriminant();

const Parity parity = even; //всегда чётна и симметрична относительно OY

};

## Содержимое файла реализации QuadraticFunction.cpp

#include "QuadraticFunction.h"

#include <string>

#include <iostream>

using namespace std;

QuadraticFunction::QuadraticFunction() : Function

(

"Quadratic function",

"a\*(x^2) + b\*x + c, a!=0, b!=0",

"x in R",

"a > 0 => y > -b/2a\n a < 0 => y < -b/2a"

)

{

coefficients.resize(2);

coefficients[0] = 1;

coefficients[1] = 2;

coefficients[2] = 1;

maxValue = inf;

minValue = GetVertexAbsciss();

isUp = true;

}

QuadraticFunction::QuadraticFunction(double a, double b, double c) : Function

(

"Quadratic function", "", "x in R", ""

)

{

coefficients.resize(3);

coefficients[0] = a;

coefficients[1] = b;

coefficients[2] = c;

string formula = to\_string(a) + "(x ^ 2) + " + to\_string(b) + " \* x + " + to\_string(c);

string val;

SetFormula(formula);

if (a > 0)

{

isUp = true;

maxValue = inf;

minValue = -b / (2 \* a);

val = "y > " + to\_string(minValue);

SetVal(val);

}

else

{

isUp = false;

maxValue = -b / (2 \* a);

minValue = -inf;

val = "y < " + to\_string(maxValue);

SetVal(val);

}

}

bool QuadraticFunction::isNullsExist()

{

return Discriminant() >= 0;

}

void QuadraticFunction::PrintNulls()

{

printf("\n\n\t\tФункция: y = (%3.3f) \* (x^2) + (%3.3f) \* x + (%3.3f)",

coefficients[0],

coefficients[1],

coefficients[2]);

if (isNullsExist())

{

printf("\n\t\tНули функции x1 = %3.3f, x2 = %3.3f", GetRoots().first, GetRoots().second);

return;

}

printf("\n\n\t\tФункция не принимает нулевых значений (нет пересечения с OX)");

}

pair<double,double> QuadraticFunction::GetRoots()

{

if (isNullsExist())

{

double x1 = (sqrt(Discriminant()) + GetVertexAbsciss());

double x2 = (-sqrt(Discriminant()) + GetVertexAbsciss());

return {x1,x2};

}

return {0,0};

}

void QuadraticFunction::Calculate(double argument)

{

printf("\n\n\t\tФункция: y = (%3.3f) \* (x^2) + (%3.3f) \* x + (%3.3f)",

coefficients[0],

coefficients[1],

coefficients[2]);

if (!isNullsExist())

{

printf("\n\t\tКорней нет\n");

return;

}

printf("\n\t\tПервый корень x1 = %3.3f\n\t\tВторой корень x2 = %3.3f\n", GetRoots().first, GetRoots().second);

}

void QuadraticFunction::Print()

{

printf("\n\n\t\tНазвание функции: %s\n", GetName().c\_str());

printf("\t\tФормула: y = (%3.3f) \* (x^2) + (%3.3f) \* x + (%3.3f)\n",

coefficients[0],

coefficients[1],

coefficients[2]);

printf("\t\tОбласть определения функции: %s\n", GetDef().c\_str());

printf("\t\tМножество значений функции: ");

if (isUp)

{

printf("y > %3.3f\n", minValue);

printf("\t\tКонфигурация параболы: ветви вверх\n");

printf("\t\tСвойство чётности: чётная\n");

printf("\t\tМонотонность: \n\t\t\tубывает на ( - inf ; %3.3f )\n\t\t\tвозрастает на ( %3.3f ; + inf )\n",

GetVertexAbsciss(), GetVertexAbsciss());

printf("\t\tВыпуклость: вогнута вниз\n");

}

else

{

printf("y < %3.3f\n", maxValue);

printf("\t\tКонфигурация параболы: ветви вниз\n");

printf("\t\tСвойство чётности: чётная\n");

printf("\t\tМонотонность: \n\t\t\tвозрастает на ( - inf ; %3.3f )\n\t\t\tубывает на ( %3.3f ; + inf )\n",

GetVertexAbsciss(), GetVertexAbsciss());

printf("\t\tВыпуклость: выпукла вверх\n");

}

}

void QuadraticFunction::GetIntervalsOfSign()

{

printf("\n\t\tПромежутки знакопостоянства:\n");

if (isUp && isNullsExist())

{

printf("\t\ty > 0 на ( -inf ; %3.3f )\n", GetRoots().first);

printf("\t\ty < 0 на ( %3.3f ; %3.3f )\n", GetRoots().first, GetRoots().second);

printf("\t\ty > 0 на ( %3.3f ; +inf )\n", GetRoots().second);

return;

}

if (!isUp && isNullsExist())

{

printf("\t\ty < 0 на ( -inf ; %3.3f )\n", GetRoots().first);

printf("\t\ty > 0 на ( %3.3f ; %3.3f )\n", GetRoots().first, GetRoots().second);

printf("\t\ty < 0 на ( %3.3f ; +inf )\n", GetRoots().second);

return;

}

if (!isNullsExist() && isUp)

{

printf("\t\ty > 0 на ( -inf ; inf )");

return;

}

printf("\t\ty < 0 на ( -inf ; inf )");

return;

}

double QuadraticFunction::GetVertexAbsciss()

{

return (-coefficients[1]) / (2 \* coefficients[0]);

}

double QuadraticFunction::Discriminant()

{

return coefficients[1] \* coefficients[1] - 4 \* coefficients[0] \* coefficients[2];

}

## Содержимое заголовочного файла SquareRootFunction.h

#pragma once

#include "Function.h"

#include <vector>

using namespace std;

#define inf DBL\_MAX //бесконечность

enum Bulge //выпуклость

{

up = 1, //выпукла вверх

down = 2, //выпукла вниз (вогнута)

notConvex = -1 //не обладает свойством выпуклости

};

class SquareRootFunction : public Function

{

private:

const double minValue = 0; //мин. значение функции всегда 0

const Parity parity = general; //не обладает симметрией

public:

double aCoeff; //коэффициент при sqrt(x)

Bulge bulge; //выпуклость/вогнутость

bool isIncrease; //true = возрастает (aCoeff > 0), иначе false

/\*конструкторы\*/

SquareRootFunction();

SquareRootFunction(double a);

void PrintNulls(); //вывести корни уравнения

void Calculate(double argument); //вычисление значения функции

void Print(); //вывести свойства функции

void GetIntervalsOfSign(); //вывести промежутки знакопостоянства

~SquareRootFunction() {};

};

## Содержимое файла реализации SquareRootFunction.cpp

#include "SquareRootFunction.h"

SquareRootFunction::SquareRootFunction() : Function

(

"Square root function",

"y = a\*sqrt(x)",

"x > 0",

"y > 0"

)

{

aCoeff = 0;

bulge = notConvex;

isIncrease = false;

}

SquareRootFunction::SquareRootFunction(double a) : Function

(

"Square root function",

"y = a\*sqrt(x)",

"x > 0",

"y > 0"

)

{

aCoeff = a;

if (a > 0)

{

bulge = up;

isIncrease = true;

}

if (a < 0)

{

bulge = down;

isIncrease = false;

}

if (a == 0)

{

bulge = notConvex;

isIncrease = false;

}

}

void SquareRootFunction::PrintNulls()

{

printf("\n\t\tНуль функции достигается при x = 0\n");

}

void SquareRootFunction::Calculate(double argument)

{

printf("\n\t\tФормула: y = (%3.3f) \* sqrt(x)\n", aCoeff);

printf("\t\tПри x = %3.3f функция принимает значение y = %3.3f\n", argument,

aCoeff \* sqrt(argument));

}

void SquareRootFunction::Print()

{

printf("\n\t\tНазвание функции: %s", (\*this).GetName().c\_str());

printf("\n\t\tФормула: y = (%3.3f) \* sqrt(x)\n", aCoeff);

printf("\t\tОбласть определения функции: %s\n", (\*this).GetDef().c\_str());

printf("\t\tМножество значений функции: %s\n", (\*this).GetVal().c\_str());

if (parity == general)

printf("\t\tСвойство чётности: функция общего вида\n");

if (isIncrease)

printf("\t\tМонотонность: непрерывно возрастает\n");

else

printf("\t\tМонотонность: непрерывно убывает\n");

if (bulge == up)

printf("\t\tВыпуклость: выпукла вверх\n");

if (bulge == down)

printf("\t\tВыпуклость: вогнута вниз\n");

if (bulge == notConvex)

printf("\t\tСвойством выпуклости не обладает\n");

}

void SquareRootFunction::GetIntervalsOfSign()

{

if (aCoeff > 0)

printf("\t\ty > 0 при любом х из D(x)\n");

if (aCoeff < 0)

printf("\t\ty < 0 при любом х из D(x)\n");

if (aCoeff == 0)

printf("\t\ty = 0 при любом х из D(x)\n");

}

## Содержимое файла Main.cpp

#include <Windows.h>

#include <iostream>

#include <vector>

#include "Function.h"

#include "LinearFunction.h"

#include "QuadraticFunction.h"

#include "SquareRootFunction.h"

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

/\*LINEAR FUNCTION\*/

LinearFunction linFunc(2,2);

linFunc.Print();

linFunc.PrintIntervalsOfSign();

linFunc.Calculate(1);

linFunc.PrintNulls();

/\*QUADRATIC FUNCTION\*/

QuadraticFunction quadraticFunc(1, 4, 4);

quadraticFunc.Print();

quadraticFunc.Calculate(2);

quadraticFunc.PrintNulls();

quadraticFunc.GetIntervalsOfSign();

/\*QUADRATIC ROOT FUNCTION\*/

SquareRootFunction rootFunc(1);

rootFunc.Print();

rootFunc.Calculate(4);

rootFunc.GetIntervalsOfSign();

rootFunc.PrintNulls();

return 0;}

## РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Пример работы с разработанными классами приведён на рисунках 2-4.

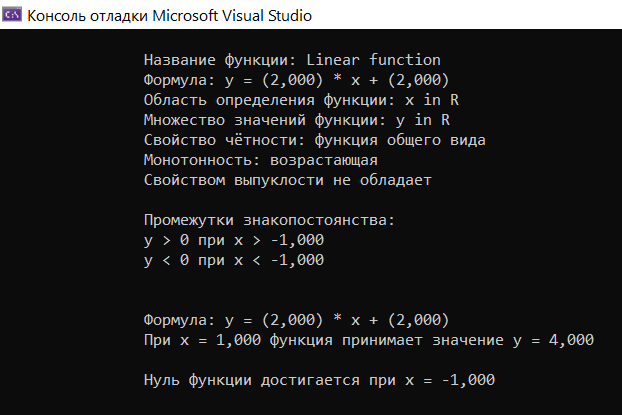


Рис. 2 – Пример работы с линейной функцией

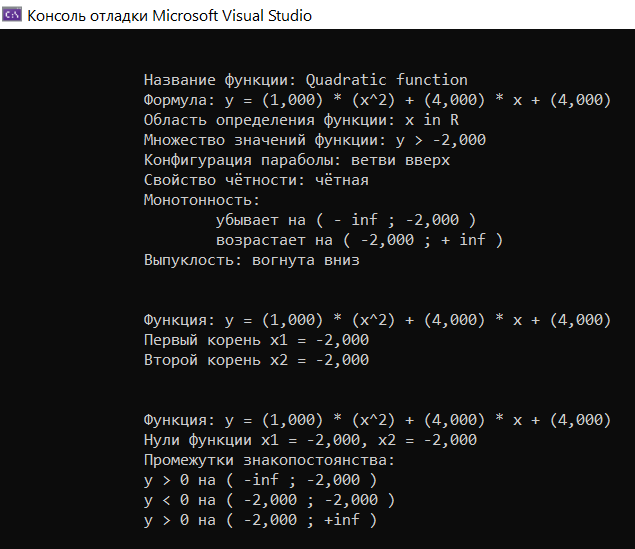


Рис. 3 – Пример работы с квадратичной функцией

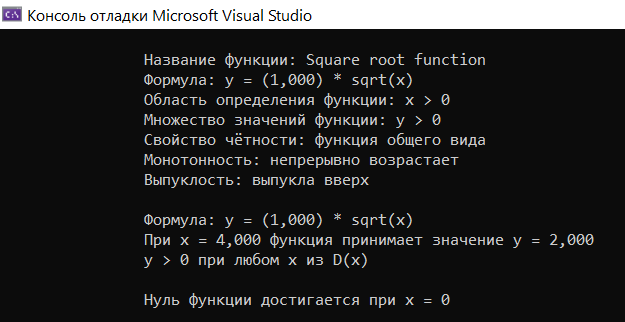


Рис. 4 – Пример работы с функцией квадратного корня

# ВЫВОД

В ходе выполнения лабораторной работы были изучены основные принципы наследования классов в языке С++. Для практического применения изученных понятия было разработано приложение по варианту.