

Московский государственный технический университет  
имени Н.Э. Баумана

---

Факультет «Информатика, искусственный интеллект и системы управления»  
Кафедра «Системы обработки информации и управления»

**Г.И. Ревунков, Ю.Е. Гапанюк, А.Н. Нардид**

**ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННУЮ И  
ФУНКЦИОНАЛЬНУЮ ПАРАДИГМУ  
ПРОГРАММИРОВАНИЯ НА ПРИМЕРЕ РЕШЕНИЯ  
ЗАДАЧИ КВАДРАТНОГО УРАВНЕНИЯ**

Электронное учебное издание – рабочая версия

Москва, 2023

## ОГЛАВЛЕНИЕ

<b>1</b>	<b>КРАТКИЙ ОБЗОР ИСТОЧНИКОВ ПО ПАРАДИГМАМ ПРОГРАММИРОВАНИЯ ...</b>	<b>3</b>
1.1	ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ .....	3
1.1.1	<i>Наиболее распространенные ФП-языки</i> .....	4
1.1.1.1	OCaml .....	4
1.1.1.2	F# .....	4
1.1.1.3	Scala .....	4
1.1.1.4	Erlang .....	4
1.1.1.5	Haskell .....	5
1.1.1.6	Rust .....	5
1.2	РЕФЛЕКСИВНОЕ ПРОГРАММИРОВАНИЕ .....	6
1.3	ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ .....	6
1.3.1	<i>Прямой логический вывод</i> .....	6
1.3.2	<i>Обратный логический вывод</i> .....	7
1.3.2.1	Язык Prolog .....	7
1.3.3	<i>Изоморфизм Карри-Ховарда</i> .....	7
<b>2</b>	<b>ПРИМЕНЕНИЕ РАЗЛИЧНЫХ ПАРАДИГМ ПРОГРАММИРОВАНИЯ НА ПРИМЕРЕ РЕШЕНИЯ КВАДРАТНОГО УРАВНЕНИЯ .....</b>	<b>8</b>
2.1	ПРИМЕР НА ЯЗЫКЕ C# С ИСПОЛЬЗОВАНИЕМ СПИСКА КОРНЕЙ .....	8
2.2	ПРИМЕР НА ЯЗЫКЕ C# С ИСПОЛЬЗОВАНИЕМ ПЕРЕЧИСЛЕНИЯ .....	9
2.3	ПРИМЕР НА ЯЗЫКЕ F# С ИСПОЛЬЗОВАНИЕМ АЛГЕБРАИЧЕСКОГО ТИПА .....	11
2.4	ПРИМЕР НА ЯЗЫКЕ C# С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА И НАСЛЕДУЕМЫХ КЛАССОВ .....	12
2.5	ПРИМЕР НА ЯЗЫКЕ F# С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА И НАСЛЕДУЕМЫХ КЛАССОВ .....	13
2.6	ПРИМЕР НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ (ПРОЦЕДУРНЫЙ ПОДХОД) .....	15
2.7	ПРИМЕР НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ КЛАССОВ (ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД) .....	16
2.8	ПРИМЕР НА ЯЗЫКЕ RUST .....	18
<b>3</b>	<b>ВЫВОДЫ .....</b>	<b>20</b>

# 1 Краткий обзор источников по парадигмам программирования

Описание [парадигм программирования](#).

## 1.1 Функциональное программирование

Описание

ФП

—

[https://ru.wikipedia.org/wiki/Функциональное\\_программирование](https://ru.wikipedia.org/wiki/Функциональное_программирование)

Языки ФП делятся на чистые (Haskell) и с побочными эффектами (большинство языков)

—

[https://ru.wikipedia.org/wiki/Чистота\\_языка\\_программирования](https://ru.wikipedia.org/wiki/Чистота_языка_программирования)

С точки зрения парадигмы программирования языки ФП можно разделить на:

1. Однопарадигмальные. Используют только приемы функционального программирования. Примеры – Haskell, Erlang.
2. Объектно-функциональные. Наряду с ФП позволяют использовать ООП. Примеры – F#, OCaml, Scala.
3. Функционально-логические. Сочетают концепции функционального и логического программирования. Пример – Mercury
4. Мультипарадигмальные с развитым метапрограммированием. Используя метапрограммирование (поддержку макросов) как ядро языка существует возможность реализовать разные парадигмы, в том числе функциональную. Пример – LISP. Как правило такие языки обладают свойством самоотображаемости (гомоиконичности) – <https://ru.wikipedia.org/wiki/Гомоиконичность>

### 1.1.1 Наиболее распространенные ФП-языки

#### 1.1.1.1 OCaml

<https://ru.wikipedia.org/wiki/OCaml>

##### Книга:

Ярон Мински, Анил Мадхавapedди и Джейсон Хикки  
«Программирование на языке OCaml».

На OCaml написана одна из самых известных систем автоматического доказательства теорем - <https://ru.wikipedia.org/wiki/Coq>

#### 1.1.1.2 F#

[https://ru.wikipedia.org/wiki/F\\_Sharp](https://ru.wikipedia.org/wiki/F_Sharp)

Является диалектом OCaml.

Документация - <http://fsharp.org/about/index.html#documentation>

##### Книги:

1. Сошников Д.В. «Функциональное программирование на F#»
2. Don Syme, Adam Granicz, Antonio Cisternino «Expert F# 4.0»
3. Tao Liu «F# for C# Developers»

#### 1.1.1.3 Scala

[https://ru.wikipedia.org/wiki/Scala\\_\(язык\\_программирования\)](https://ru.wikipedia.org/wiki/Scala_(язык_программирования))

##### Книга:

Кей Хостманн «Scala для нетерпеливых»

Дистанционный цикл из 5 курсов (специализация) на английском языке - <https://www.coursera.org/specializations/scala>

#### 1.1.1.4 Erlang

<https://ru.wikipedia.org/wiki/Erlang>

Синтаксис основан на логическом языке Пролог, хотя Erlang — функциональный язык.

Используется динамическая типизация, поэтому очень многие техники ФП, связанные с типами, не работают в Erlang.

Erlang широко применяется на практике для разработки высоконагруженных систем.

Хорошая вводная статья по Erlang -

<https://rdsn.org/article/erlang/GettingStartedWithErlang.xml>

Книга:

Фред Хеберт «Изучай Erlang во имя добра!»

#### **1.1.1.5 Haskell**

<https://ru.wikipedia.org/wiki/Haskell>

Книги:

**1. Миран Липовача «Изучай Haskell во имя добра!»**

2. Книги Р.В. Душкина

Сайт «Русский учебник по Haskell» - <http://anton-k.github.io/ru-haskell-book/book/toc.html>

Дистанционный курс на русском языке - <https://stepik.org/course/Функциональное-программирование-на-языке-Haskell-75>

#### **1.1.1.6 Rust**

[https://ru.wikipedia.org/wiki/Rust\\_\(язык\\_программирования\)](https://ru.wikipedia.org/wiki/Rust_(язык_программирования))

Современный мультипарадигмальный язык программирования, преобладает функциональная парадигма.

Элементы синтаксиса заимствованы из OCaml, Haskell, C/C++.

Использует базовую систему типов, похожую на язык C.

Реализует безопасный механизм управления памятью, позволяющий избегать «сборки мусора».

По производительности сравним с C/C++.

Используется как язык системного и прикладного программирования.

## 1.2 Рефлексивное программирование

Как правило такие языки обладают свойством самоотображаемости (гомоиконичности) – <https://ru.wikipedia.org/wiki/Гомоиконичность> (в статье приведены примеры языков).

Наиболее известный язык – LISP и его диалекты.

Наиболее известный диалект - Clojure

<https://ru.wikipedia.org/wiki/Clojure>

Документация - <https://clojure.org/reference/reader>

**Книга:**

Чаз Эмерик, Брайен Карпер, Кристоф Гранд «Программирование на Clojure»

## 1.3 Логическое программирование

### 1.3.1 Прямой логический вывод

От данным к цели (вывод, управляемый данными).

[https://en.wikipedia.org/wiki/Forward\\_chaining](https://en.wikipedia.org/wiki/Forward_chaining)

Программа записывается в виде продукционных правил «ЕСЛИ условие ТО действие». Порядок вызова правил определяется динамически машиной вывода. Результат выполнения предыдущих правил меняет состояние машины вывода (операционную память) и приводит к вызову следующих правил. Цель динамически выводится в результате выполнения правил.

Традиционно применялись в экспертных системах.

Пример – язык CLIPS - <https://ru.wikipedia.org/wiki/CLIPS>

В последнее время также применяются как средство программирования общего назначения.

Пример – система Drools - <https://en.wikipedia.org/wiki/Drools>

### 1.3.2 Обратный логический вывод

От цели к данным.

[https://en.wikipedia.org/wiki/Backward\\_chaining](https://en.wikipedia.org/wiki/Backward_chaining)

#### 1.3.2.1 Язык Prolog

[https://ru.wikipedia.org/wiki/Пролог\\_\(язык\\_программирования\)](https://ru.wikipedia.org/wiki/Пролог_(язык_программирования))

База знаний задается в виде предикатов. Правила вывода определяют связь между предикатами. Необходимо указать цель поиска и машина вывода пытается перебрать все комбинации предикатов, чтобы доказать цель. Задача машины вывода – найти комбинацию исходных данных при которых цель выполняется или доказать что их нет.

Фактически этот подход является аналогом языка запросов к базе знаний, выводимая цель соответствует конкретному запросу.

Существует диалект Пролога - Datalog (<https://en.wikipedia.org/wiki/Datalog>) который применяется как язык запросов к базам данных, в том числе к реляционным.

#### Книга:

Иван Братко «Алгоритмы искусственного интеллекта на языке Prolog»

#### Реализации Prolog:

1. Простая учебная реализация SWI Prolog - <http://www.swi-prolog.org/>
2. Одна из наиболее развитых реализаций, включающая логику высших порядков XSB Prolog - <http://xsb.sourceforge.net/>

### 1.3.3 Изоморфизм Карри-Ховарда

Существует важный принцип, который устанавливает связь между функциональным и логическим программированием – [https://ru.wikipedia.org/wiki/Соответствие\\_Карри\\_—\\_Ховарда](https://ru.wikipedia.org/wiki/Соответствие_Карри_—_Ховарда)

#### Книга:

Пирс Б. «Типы в языках программирования».

## 2 Применение различных парадигм программирования на примере решения квадратного уравнения

Рассмотрим пример вычисления корней квадратного уравнения.

### 2.1 Пример на языке C# с использованием списка корней

Проект «SquareRoot». Файл: SquareRoot\_Simple.cs

```
using System;
using System.Collections.Generic;

namespace SquareRoot
{
    /// <summary>
    /// Простое вычисление корней
    /// </summary>
    class SquareRoot_Simple
    {
        /// <summary>
        /// Вычисление корней
        /// </summary>
        public List<double> CalculateRoots(double a, double b, double c)
        {
            List<double> roots = new List<double>();
            double D = b * b - 4 * a * c;
            //Один корень
            if (D == 0)
            {
                double root = -b / (2 * a);
                roots.Add(root);
            }
            //Два корня
            else if (D > 0)
            {
                double sqrtD = Math.Sqrt(D);
                double root1 = (-b + sqrtD) / (2 * a);
                double root2 = (-b - sqrtD) / (2 * a);
                roots.Add(root1);
                roots.Add(root2);
            }
            return roots;
        }

        /// <summary>
        /// Вывод корней
        /// </summary>
        public void PrintRoots(double a, double b, double c)
        {
            List<double> roots = this.CalculateRoots(a, b, c);
            Console.WriteLine("Коэффициенты: a={0}, b={1}, c={2}. ", a, b, c);
            if(roots.Count == 0)
```



```

        {
            Console.WriteLine("Корней нет.");
        }
        else if (roots.Count == 1)
        {
            Console.WriteLine("Один корень {0}", roots[0]);
        }
        else if (roots.Count == 2)
        {
            Console.WriteLine("Два корня {0} и {1}", roots[0], roots[1]);
        }
    }
}
}

```

Корни возвращаются в виде списка.

Не очень надежно. Вдруг произошел сбой и вернулась пустая коллекция?

Для увеличения надежности будем использовать перечисление (enum).

## 2.2 Пример на языке C# с использованием перечисления

Проект «SquareRoot». Файл: SquareRoot\_Enum.cs

```

using System;
using System.Collections.Generic;

namespace SquareRoot
{
    /// <summary>
    /// Перечисление для обозначения количества корней
    /// </summary>
    enum RootsEnum { NoRoots, OneRoot, TwoRoots }

    /// <summary>
    /// Вычисление корней с использованием перечисления
    /// </summary>
    class SquareRoot_Enum
    {
        /// <summary>
        /// Вычисление корней
        /// </summary>
        public void CalculateRoots(double a, double b, double c, out List<double>
roots, out RootsEnum rootFlag)
        {
            rootFlag = RootsEnum.NoRoots;
            roots = new List<double>();
            double D = b * b - 4 * a * c;
            //Один корень
            if (D == 0)
            {
                rootFlag = RootsEnum.OneRoot;
                double root = -b / (2 * a);
                roots.Add(root);
            }
        }
    }
}

```

```

//Два корня
else if (D > 0)
{
    rootFlag = RootsEnum.TwoRoots;
    double sqrtD = Math.Sqrt(D);
    double root1 = (-b + sqrtD) / (2 * a);
    double root2 = (-b - sqrtD) / (2 * a);
    roots.Add(root1);
    roots.Add(root2);
}
}

/// <summary>
/// Вывод корней
/// </summary>
public void PrintRoots(double a, double b, double c)
{
    List<double> roots;
    RootsEnum rootFlag;
    this.CalculateRoots(a, b, c, out roots, out rootFlag);
    Console.WriteLine("Коэффициенты: a={0}, b={1}, c={2}. ", a, b, c);
    if (rootFlag == RootsEnum.NoRoots)
    {
        Console.WriteLine("Корней нет.");
    }
    else if (rootFlag == RootsEnum.OneRoot)
    {
        Console.WriteLine("Один корень {0}", roots[0]);
    }
    else if (rootFlag == RootsEnum.TwoRoots)
    {
        Console.WriteLine("Два корня {0} и {1}", roots[0], roots[1]);
    }
}
}
}

```

Корни возвращаются в виде списка, а также возвращается значение перечисления, которое задает количество корней.

Не очень удобно. Список корней и значение перечисления возвращаются по отдельности. При возникновении ошибки они могут не соответствовать друг другу.

Можно ли их объединить? Теоретически их можно объединить в отдельный класс, но все равно список корней и значение перечисления можно изменить отдельно друг от друга.

В F# они объединены в единую структуру, которая называется алгебраическим типом.

## 2.3 Пример на языке F# с использованием алгебраического типа

Проект «SquareRootFSharp». Файл: Program.fs

```
//Для использования классов Math и Console
open System

// Алгебраический тип или "Discriminated Unions"
// Алгебраический тип - тип сумма из типов произведений
// | - означает "или" и задает тип-сумму
// * - означает "и" и задает произведение (кортеж, который соединяет все элементы)
// В абстрактных алгебрах наиболее близкой алгеброй является полукольцо

///Тип решения квадратного уравнения
type SquareRootResult =
    | NoRoots
    | OneRoot of double
    | TwoRoots of double * double //кортеж из двух double

///Функция вычисления корней уравнения
let CalculateRoots(a:double, b:double, c:double):SquareRootResult =
    let D = b*b - 4.0*a*c;
    if D < 0.0 then NoRoots
    else if D = 0.0 then
        let rt = -b / (2.0 * a)
        OneRoot rt
    else
        let sqrtD = Math.Sqrt(D)
        let rt1 = (-b + sqrtD) / (2.0 * a);
        let rt2 = (-b - sqrtD) / (2.0 * a);
        TwoRoots (rt1,rt2)

///Вывод корней (тип unit - аналог void)
let PrintRoots(a:double, b:double, c:double):unit =
    printf "Коэффициенты: a=%A, b=%A, c=%A. " a b c
    let root = CalculateRoots(a,b,c)
    //Оператор сопоставления с образцом
    let textResult =
        match root with
        | NoRoots -> "Корней нет"
        | OneRoot(rt) -> "Один корень " + rt.ToString()
        | TwoRoots(rt1,rt2) -> "Два корня " + rt1.ToString() + " и " + rt2.ToString()
    printfn "%s" textResult

[<EntryPoint>]
let main argv =
    //Тестовые данные
    //2 корня
    let a1 = 1.0;
    let b1 = 0.0;
    let c1 = -4.0;
    //1 корень
    let a2 = 1.0;
    let b2 = 0.0;
    let c2 = 0.0;
    //нет корней
    let a3 = 1.0;
    let b3 = 0.0;
    let c3 = 4.0;
```

```
PrintRoots(a1,b1,c1)
PrintRoots(a2,b2,c2)
PrintRoots(a3,b3,c3)
```

```
//|> ignore - перенаправление потока с игнорирование результата вычисления
Console.ReadLine() |> ignore
0 // возвращение целочисленного кода выхода
```

У алгебраического типа есть недостаток. Его нельзя расширять в процессе реализации. А в C# можно воспользоваться решением на основе интерфейсов.

## 2.4 Пример на языке C# с использованием интерфейса и наследуемых классов

Проект «SquareRoot». Файл: SquareRoot\_WithInterface.cs

```
using System;
using System.Collections.Generic;

namespace SquareRoot
{
    interface RootsResult { }
    class NoRoots : RootsResult { }
    class OneRoot : RootsResult
    {
        public double root { get; set; }
    }
    class TwoRoots : RootsResult
    {
        public double root1 { get; set; }
        public double root2 { get; set; }
    }

    /// <summary>
    /// Возможные варианты решения расширяются за счет использования интерфейса
    /// </summary>
    class SquareRoot_WithInterface
    {
        /// <summary>
        /// Вычисление корней
        /// </summary>
        public RootsResult CalculateRoots(double a, double b, double c)
        {
            List<double> roots = new List<double>();
            double D = b * b - 4 * a * c;
            //Один корень
            if (D == 0)
            {
                double rt = -b / (2 * a);
                return new OneRoot()
                {
                    root = rt
                };
            }
            //Два корня
            else if (D > 0)
```

```

    {
        double sqrtD = Math.Sqrt(D);
        double rt1 = (-b + sqrtD) / (2 * a);
        double rt2 = (-b - sqrtD) / (2 * a);
        return new TwoRoots()
        {
            root1 = rt1,
            root2 = rt2
        };
    }
    //Нет корней
    else
    {
        return new NoRoots();
    }
}

/// <summary>
/// Вывод корней
/// </summary>
public void PrintRoots(double a, double b, double c)
{
    RootsResult result = this.CalculateRoots(a, b, c);
    Console.Write("Коэффициенты: a={0}, b={1}, c={2}. ", a, b, c);
    string resultType = result.GetType().Name;
    if (resultType == "NoRoots")
    {
        Console.WriteLine("Корней нет.");
    }
    else if (resultType == "OneRoot")
    {
        OneRoot rt1 = (OneRoot)result;
        Console.WriteLine("Один корень {0}", rt1.root);
    }
    else if (resultType == "TwoRoots")
    {
        TwoRoots rt2 = (TwoRoots)result;
        Console.WriteLine("Два корня {0} и {1}", rt2.root1, rt2.root2);
    }
}
}
}

```

Этот вариант лучше, потому что он позволяет расширять варианты решения, добавляя новые классы, наследуемые от интерфейса. Но в F# тоже так можно.

## 2.5 Пример на языке F# с использованием интерфейса и наследуемых классов

Проект «SquareRootFSharpClass». Файл: Program.fs

```

open System

///Интерфейс
type SquareRootEmpty = interface end
//Наследуемые классы с вариантами решения

```

```

type NoRoots()=
    interface SquareRootEmpty
//Клсс содержит параметры, которые присваиваются свойству
type OneRoot(p:double)=
    interface SquareRootEmpty
    // Объявление свойства
    member val root = p : double with get, set

type TwoRoots(p1:double,p2:double)=
    interface SquareRootEmpty
    // Объявление свойства
    member val root1 = p1 : double with get, set
    member val root2 = p2 : double with get, set

///Функция вычисления корней уравнения
let CalculateRoots(a:double, b:double, c:double):SquareRootEmpty =
    let D = b*b - 4.0*a*c;
    if D < 0.0 then (new NoRoots() :> SquareRootEmpty)
    else if D = 0.0 then
        let rt = -b / (2.0 * a)
        //Требуется явное приведение к интерфейсному типу
        (OneRoot(rt) :> SquareRootEmpty)
    else
        let sqrtD = Math.Sqrt(D)
        let rt1 = (-b + sqrtD) / (2.0 * a);
        let rt2 = (-b - sqrtD) / (2.0 * a);
        (TwoRoots(rt1,rt2) :> SquareRootEmpty)

///Вывод корней (тип unit - аналог void)
let PrintRoots(a:double, b:double, c:double):unit =
    printf "Коэффициенты: a=%A, b=%A, c=%A. " a b c
    let root = CalculateRoots(a,b,c)
    //Оператор сопоставления с образцом по типу - :?
    let textResult =
        match root with
        | :? NoRoots -> "Корней нет"
        | :? OneRoot as r -> "Один корень " + r.root.ToString()
        | :? TwoRoots as r -> "Два корня " + r.root1.ToString() + " и " + r.root2.ToString()
        | _ -> "" // Если не выполняется ни один из предыдущих шаблонов
    printfn "%s" textResult

[<EntryPoint>]
let main argv =
    //Тестовые данные
    //2 корня
    let a1 = 1.0;
    let b1 = 0.0;
    let c1 = -4.0;
    //1 корень
    let a2 = 1.0;
    let b2 = 0.0;
    let c2 = 0.0;
    //нет корней
    let a3 = 1.0;
    let b3 = 0.0;
    let c3 = 4.0;

    PrintRoots(a1,b1,c1)
    PrintRoots(a2,b2,c2)
    PrintRoots(a3,b3,c3)

    //|> ignore - перенаправление потока с игнорирование результата вычисления

```

```
Console.ReadLine() |> ignore
0 // возвращение целочисленного кода выхода
```

Таким образом в F# можно использовать как «закрытые» алгебраические типы так и «открытую» к расширению реализацию на основе интерфейса и наследуемых классов.

## **2.6 Пример на языке Python с использованием функций (процедурный подход)**

Проект «SquareRootPython». Файл: roots\_proc.py

Реализован подход из SquareRoot\_Simple.cs

Язык Python поддерживает процедурную, объектно-ориентированную, и, отчасти, функциональную парадигмы.

```
import sys
import math

def get_coef(index, prompt):
    """
    Читаем коэффициент из командной строки или вводим с клавиатуры

    Args:
        index (int): Номер параметра в командной строке
        prompt (str): Приглашение для ввода коэффициента

    Returns:
        float: Коэффициент квадратного уравнения
    """
    try:
        # Пробуем прочитать коэффициент из командной строки
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt)
        coef_str = input()
    # Переводим строку в действительное число
    coef = float(coef_str)
    return coef

def get_roots(a, b, c):
    """
    Вычисление корней квадратного уравнения

    Args:
        a (float): коэффициент A
        b (float): коэффициент B
        c (float): коэффициент C

    Returns:
        list[float]: Список корней
    """
```

```

...
result = []
D = b*b - 4*a*c
if D == 0.0:
    root = -b / (2.0*a)
    result.append(root)
elif D > 0.0:
    sqD = math.sqrt(D)
    root1 = (-b + sqD) / (2.0*a)
    root2 = (-b - sqD) / (2.0*a)
    result.append(root1)
    result.append(root2)
return result

def main():
    ...
    Основная функция
    ...
    a = get_coef(1, 'Введите коэффициент A:')
    b = get_coef(2, 'Введите коэффициент B:')
    c = get_coef(3, 'Введите коэффициент C:')
    # Вычисление корней
    roots = get_roots(a,b,c)
    # Вывод корней
    len_roots = len(roots)
    if len_roots == 0:
        print('Нет корней')
    elif len_roots == 1:
        print('Один корень: {}'.format(roots[0]))
    elif len_roots == 2:
        print('Два корня: {} и {}'.format(roots[0], roots[1]))

# Если сценарий запущен из командной строки
if __name__ == "__main__":
    main()

# Пример запуска
# qr.py 1 0 -4

```

## 2.7 Пример на языке Python с использованием классов (объектно-ориентированный подход)

Проект «SquareRootPython». Файл: roots\_oop.py

Реализован подход из SquareRoot\_Simple.cs, но с использованием класса.

```

import sys
import math

class SquareRoots:

```



```

def __init__(self):
    """
    Конструктор класса
    """
    # Объявление коэффициентов
    self.coef_A = 0.0
    self.coef_B = 0.0
    self.coef_C = 0.0
    # Количество корней
    self.num_roots = 0
    # Список корней
    self.roots_list = []

def get_coef(self, index, prompt):
    """
    Читаем коэффициент из командной строки или вводим с клавиатуры
    Args:
        index (int): Номер параметра в командной строке
        prompt (str): Приглашение для ввода коэффициента
    Returns:
        float: Коэффициент квадратного уравнения
    """
    try:
        # Пробуем прочитать коэффициент из командной строки
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt)
        coef_str = input()
    # Переводим строку в действительное число
    coef = float(coef_str)
    return coef

def get_coefs(self):
    """
    Чтение трех коэффициентов
    """
    self.coef_A = self.get_coef(1, 'Введите коэффициент A:')
    self.coef_B = self.get_coef(2, 'Введите коэффициент B:')
    self.coef_C = self.get_coef(3, 'Введите коэффициент C:')

def calculate_roots(self):
    """
    Вычисление корней квадратного уравнения
    """
    a = self.coef_A
    b = self.coef_B
    c = self.coef_C
    # Вычисление дискриминанта и корней
    D = b*b - 4*a*c
    if D == 0.0:
        root = -b / (2.0*a)
        self.num_roots = 1
        self.roots_list.append(root)
    elif D > 0.0:
        sqD = math.sqrt(D)
        root1 = (-b + sqD) / (2.0*a)
        root2 = (-b - sqD) / (2.0*a)
        self.num_roots = 2
        self.roots_list.append(root1)

```

```

        self.roots_list.append(root2)

    def print_roots(self):
        # Проверка отсутствия ошибок при вычислении корней
        if self.num_roots != len(self.roots_list):
            print(('Ошибка. Уравнение содержит {} действительных корней, ' + \
                  'но было вычислено {} корней.').format(self.num_roots,
                  len(self.roots_list)))
        else:
            if self.num_roots == 0:
                print('Нет корней')
            elif self.num_roots == 1:
                print('Один корень: {}'.format(self.roots_list[0]))
            elif self.num_roots == 2:
                print('Два корня: {} и {}'.format(self.roots_list[0], \
                  self.roots_list[1]))

def main():
    """
    Основная функция
    """
    # Создание объекта класса
    r = SquareRoots()
    # Последовательный вызов необходимых методов
    r.get_coefs()
    r.calculate_roots()
    r.print_roots()

# Если сценарий запущен из командной строки
if __name__ == "__main__":
    main()

# Пример запуска
# roots_oop.py 1 0 -4

```

## 2.8 Пример на языке Rust

Проект «SquareRootRust». Файл: src/main.rs

Идея реализации напоминает «SquareRootFSharp».

```

use std::io;

#[derive(Debug, Copy, Clone)]
///Тип решения квадратного уравнения
enum SquareRootResult {
    /// Unit-тип
    NoRoots,
    /// Один корень - кортежная структура
    OneRoot(f64),
    /// C-подобная структура
    TwoRoots { root1: f64, root2: f64 },
}

```

```

#[derive(Debug, Copy, Clone)]
/// Структура, соответствующая уравнению
struct Equation {
    /// Коэффициент A
    c_a: f64,
    /// Коэффициент B
    c_b: f64,
    /// Коэффициент C
    c_c: f64,
    /// Дискриминант
    disk: f64,
    /// Корни
    res: SquareRootResult,
}

impl Equation {
    /// Функция вычисления корней
    fn calculate_roots(&mut self) {
        self.disk = self.c_b.powi(2) - 4.0 * self.c_a * self.c_c;
        self.res = {
            if self.disk < 0.0 {
                SquareRootResult::NoRoots
            } else if self.disk == 0.0 {
                let rt = -self.c_b / (2.0 * self.c_a);
                SquareRootResult::OneRoot(rt)
            } else {
                let rt1 = (-self.c_b - self.disk.sqrt()) / (2.0 * self.c_a);
                let rt2 = (-self.c_b + self.disk.sqrt()) / (2.0 * self.c_a);
                SquareRootResult::TwoRoots {
                    root1: rt1,
                    root2: rt2,
                }
            }
        };
    }

    /// Ввод одного коэффициента
    fn get_coef(message: &str) -> f64 {
        return loop {
            let mut input = String::new();
            println!("{}", message);
            io::stdin()
                .read_line(&mut input)
                .expect("Неверно введена строка");
            match input.trim().parse() {
                Ok(val) => {
                    break val;
                }
                Err(_) => {
                    continue;
                }
            }
        }
    }
}

```

```

    };
}

fn get_coefs(&mut self) -> () {
    self.c_a = Equation::get_coef("Введите коэффициент A: ");
    self.c_b = Equation::get_coef("Введите коэффициент B: ");
    self.c_c = Equation::get_coef("Введите коэффициент C: ");
}

}

fn main() {
    use SquareRootResult::*;
    let mut eq = Equation {
        c_a: 0.0,
        c_b: 0.0,
        c_c: 0.0,
        disk: 0.0,
        res: SquareRootResult::NoRoots,
    };
    eq.get_coefs();
    eq.calculate_roots();
    let text_res = match eq.res {
        NoRoots => format!("Корней нет"),
        OneRoot(rt) => format!("Один корень => {}", rt),
        TwoRoots { root1, root2 } => format!("Два корня => {} и {}", root1,
root2),
    };
    println!("{}", text_res);
}

```

### 3 Выводы

В настоящее время функциональные языки программирования широко используются для реализации различных задач.

Использование таких механизмов функционального программирования, как алгебраические типы и сопоставление с образцом, позволяют наглядно реализовать классические алгоритмы.