

Университет им. Н.Э. Баумана

Факультет Радиотехнический

Кафедра РТ5

Курс «Парадигмы и конструкции языков программирования»

Домашнее задание

Тема «Язык программирования F#»

Выполнил: Кудрявцев Р. В.
Студент группы: РТ5-31Б

Проверяющий: Гапанюк Ю.Е.
Доцент кафедры ИУ5

Москва, 2023 г.

Описание задания

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Что такое F#?

F# — кроссплатформенный, мультипарадигмальный язык программирования из семейства языков .NET, поддерживающий функциональное программирование в дополнение к императивному (процедурному) и объектно-ориентированному программированию. Структура F# во многом схожа со структурой OCaml с той лишь разницей, что F# реализован поверх библиотек и среды исполнения .NET. Язык был разработан Доном Саймом в Microsoft Research в Кембридже, в настоящее время его разработку ведёт Microsoft Developer Division. F# достаточно тесно интегрируется со средой разработки Visual Studio.

F# позволяет писать краткий, надежный и эффективный код, в нем основное внимание уделяется проблемной области, а не деталям программирования.

Особенности языка

1. Функциональное программирование:

- F# поддерживает функциональное программирование, что означает, что функции рассматриваются как первоклассные объекты. Они могут передаваться как аргументы, возвращаться из функций, и сохраняться в структурах данных.

2. неизменяемость:

- Переменные в F# по умолчанию являются неизменяемыми. Это означает, что после присваивания значения переменной его нельзя изменить. Эта особенность способствует более безопасному и предсказуемому программированию.

3. Пайплайнинг:

- F# обеспечивает легкий синтаксис для создания функциональных конвейеров (pipelines). Это позволяет более лаконичным образом объединять функции в цепочки для обработки данных.

4. Pattern Matching:

- Язык поддерживает мощное средство сопоставления с образцом (pattern matching), которое упрощает обработку различных случаев в данных.

5. Асинхронное программирование:

- F# имеет встроенную поддержку асинхронного программирования с использованием асинхронных выражений, что облегчает написание эффективного и отзывчивого кода.

6. Типы данных:

- F# имеет богатую систему типов, включая типы данных кортежей, записей (records), активных паттернов (active patterns) и другие, что делает код более выразительным и безопасным.

7. Интеграция с .NET:

- F# полностью интегрирован с платформой .NET, что обеспечивает доступ к библиотекам .NET и возможность использовать F# вместе с другими языками, такими как C#.

8. Объектно-ориентированные возможности:

- В F# есть поддержка объектно-ориентированного программирования (ООП), включая определение классов и интерфейсов, но при этом язык остается функциональным в своей сути.

9. Active Patterns:

- Это механизм, который позволяет создавать собственные образцы сопоставления и определять собственные семантики для сопоставления с образцом.

10.Type Providers:

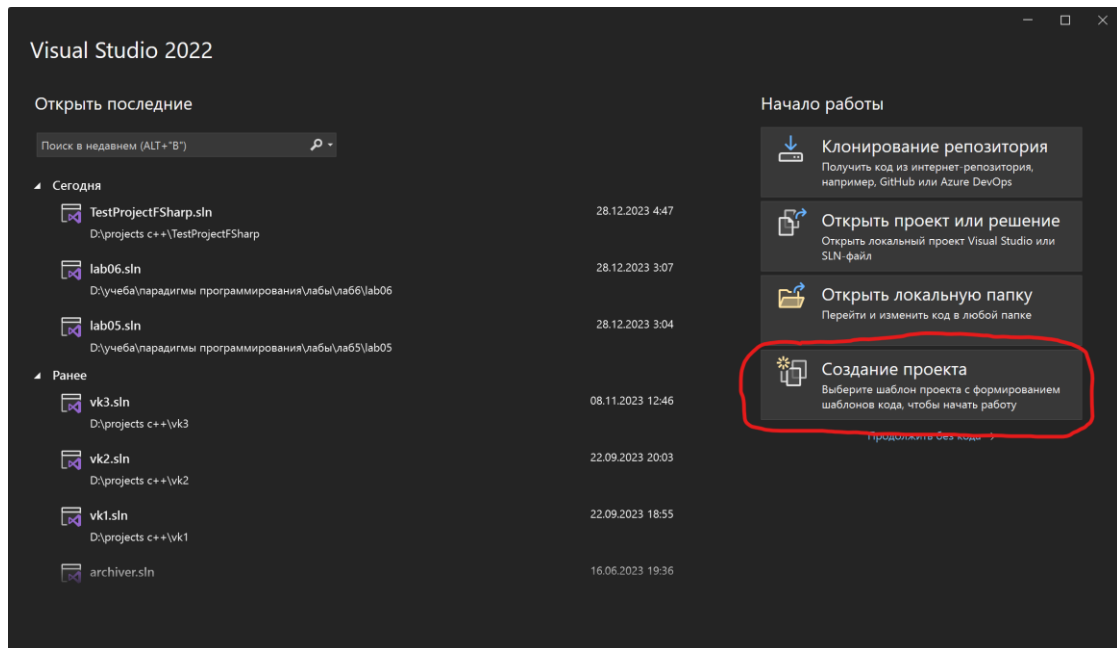
- F# включает в себя концепцию "type providers", которая автоматически предоставляет информацию о типах данных, основываясь на внешних источниках данных (например, базах данных, файловых форматах).

Среда разработки

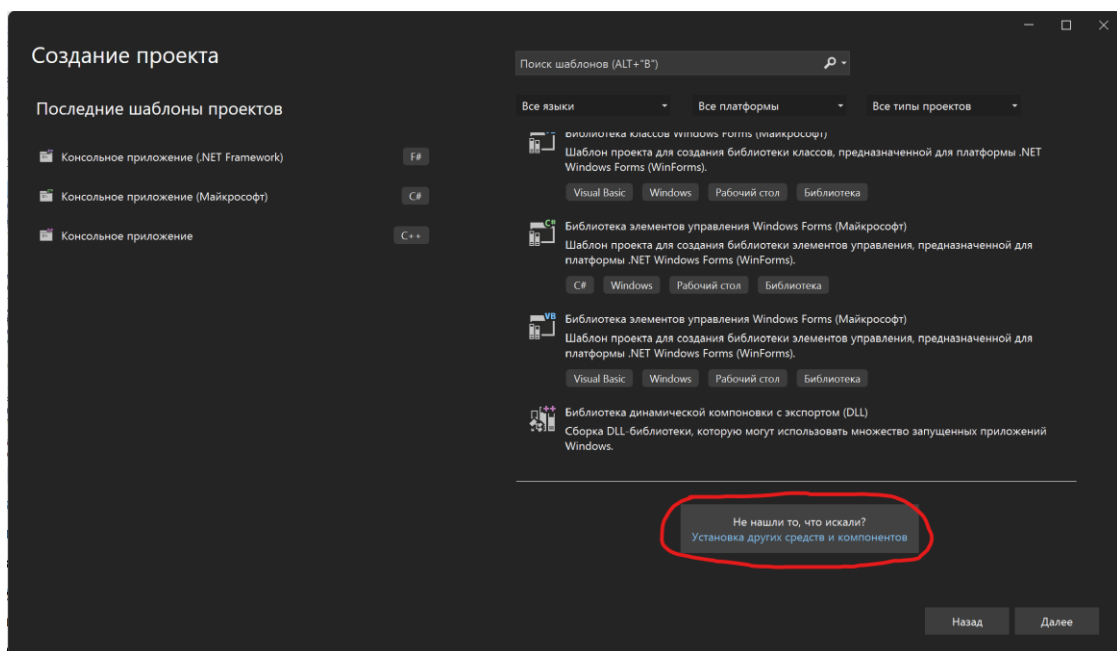
Проекты на F# разрабатываются в Microsoft Visual Studio/ Visual Studio Code. Но сообществом созданы множество других средств, например Fantomas для форматирования кода, FSharpLint для проверки кода, FAKE для автоматизации сборки. Я буду рассматривать только разработку в Visual Studio 2022 Community Edition.

Для начала работы с языком потребуется установить соответствующие компоненты в Visual Studio. Сделать это можно так:

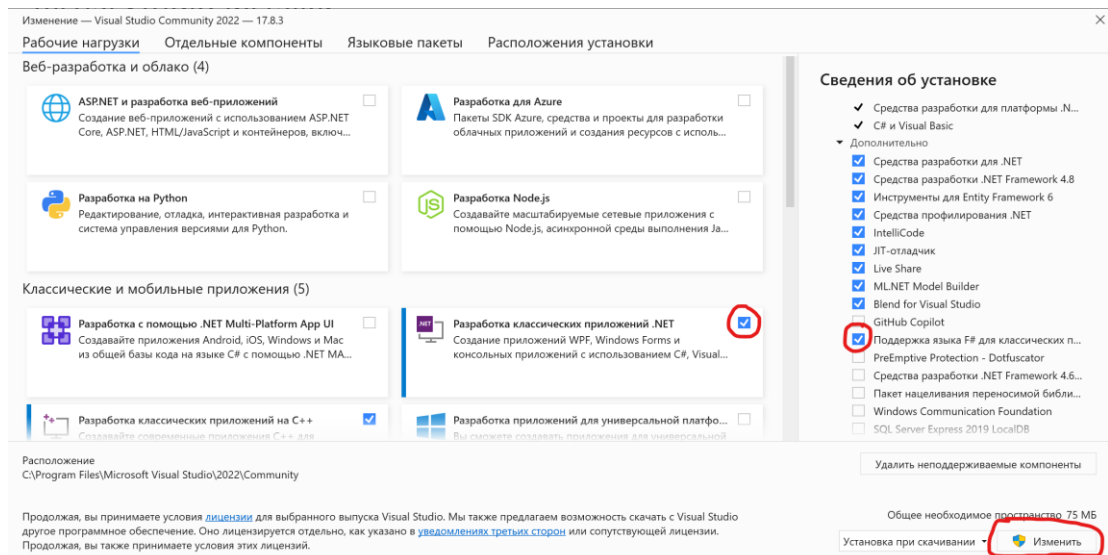
1. Запустить программу и нажать на кнопку “Создание проекта”.



2. В открывшемся окне “Создание проекта” в списке выбора шаблона пролистать до конца и найти кнопку “Установка других средств и компонентов”.



3. Во вкладке “Классические и мобильные приложения” выбрать “Разработка классических приложений .NET” и в дополнительных сведениях об установке поставить флажок перед “Поддержка языка F# для классических приложений”.



Программа

Я предлагаю к решению лабораторную работу №1, она же решение биквадратного уравнения.

open System;

// функция, которая принимает на вход коэффициенты уравнения, решает его и возвращает корни уравнения в виде списка

let solve_biquadratic_equation a b c =

// функция, которая решает обычное квадратное уравнение с заданными коэффициентами и возвращает список корней

let solve_q_eq d = [(-1.0*b + sqrt(d)) / (2.0 * a); (-1.0*b - sqrt(d)) / (2.0 * a)]

// функция, которая решает уже биквадратное уравнение путем взятия корня из корня решения квадратного уравнения

let solve_biq_eq r = [-sqrt(r); sqrt(r)]

// расчет дискриминанта

let d = b * b - 4.0 * a * c

// сравнение дискриминанта

match d with

| d when d > 0.0 ->

// расчет корней квадратного уравнения

Some (solve_biq_eq(solve_q_eq(d).[0]) @ solve_biq_eq(solve_q_eq(d).[1]))

| d when d = 0.0 ->

// расчет корней квадратного уравнения

Some (solve_biq_eq(solve_q_eq(d).[0]))

| ->

// если дискриминант отрицательный

None

// рекурсивная функция, которая помогает вводить коэффициенты с консоли

let rec coeff_setter coeff_name =

printfn "Введите коэффициент %s" coeff_name

// переменная a, в которую будет записываться число из консоли

let mutable a = 0.0

// определяем правильность ввода

match System.Console.ReadLine() with

```

| input when System.Double.TryParse(input, &a) && a <> 0.0 ->
    // возвращаем введеное число
    Some a
| _ ->
    // выводим ошибку
    printfn "Ошибка ввода. Попробуйте еще раз"

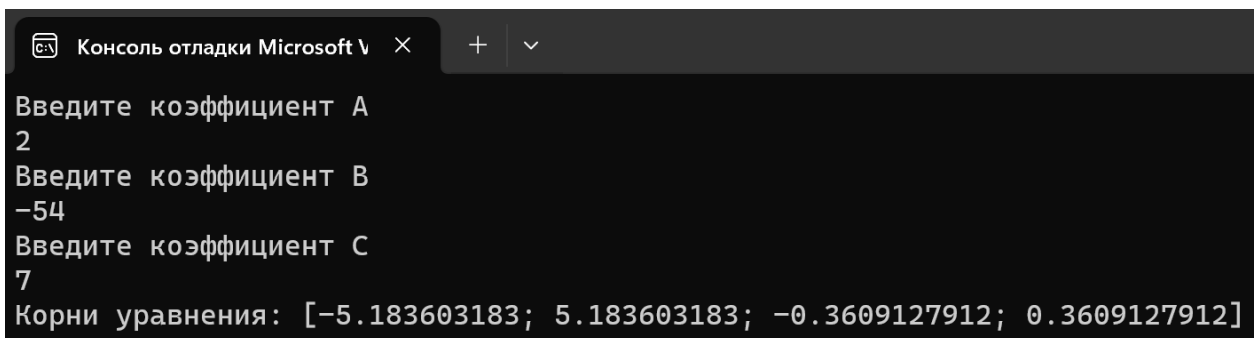
    // вызываем функцию еще раз для правильного ввода
    coeff_setter coeff_name

// функция, которая отвечает за ввод коэффициентов и возвращает кортеж из них
let rec getInput() =
    match coeff_setter "A" with
    | Some a ->
        match coeff_setter "B" with
        | Some b ->
            match coeff_setter "C" with
            | Some c ->
                (a, b, c)
            | None -> getInput()
        | None -> getInput()
    | None -> getInput()

[<EntryPoint>]
let main argv =
    let a, b, c = getInput()
    match solve_biquadratic_equation a b c with
    | Some roots ->
        printfn "Корни уравнения: %A" roots
    | _ ->
        printfn "Корней нет"
0

```

Экранные формы

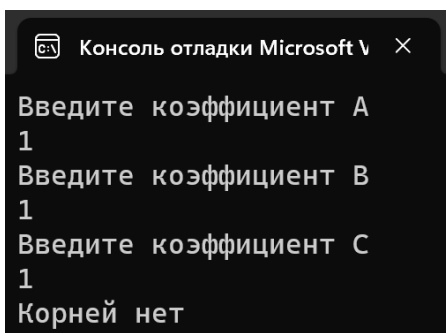


Консоль отладки Microsoft V

```

Введите коэффициент A
2
Введите коэффициент B
-54
Введите коэффициент C
7
Корни уравнения: [-5.183603183; 5.183603183; -0.3609127912; 0.3609127912]

```



Консоль отладки Microsoft V

```

Введите коэффициент A
1
Введите коэффициент B
1
Введите коэффициент C
1
Корней нет

```

Введите коэффициент A

ва

Ошибка ввода. Попробуйте еще раз

Введите коэффициент A

3

Введите коэффициент B

влатаы првьдлыажт

Ошибка ввода. Попробуйте еще раз

Введите коэффициент B

-8

Введите коэффициент C

4

Корни уравнения: $[-1.414213562; 1.414213562; -0.8164965809; 0.8164965809]$