|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю.Е.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2024** г. |  | Демонстрация:  Алябушев А. Д.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2024** г. |

**Отчет по домашнему заданию по курсу**

**Парадигмы и конструкции языков программирования**

#### Тема работы: " Основные конструкции языка F#. "

6

(количество листов)

Вариант № **1**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5-52Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Алябушев А. Д. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_**2024**  г. |

Москва, МГТУ - **2024**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

[1. Описание задания 3](#_Toc152712824)

[2. Ход работы 3](#_Toc152712825)

[3. Заключение 6](#_Toc152712826)

# 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.

# Ход работы

В качестве домашнего задания я выбрал изучения языка программирования F#. Целью моей работы является изучение основных функций данного инструмента и применение его в дальнейшей карьере.

F# (произносится эф-шарп) — мультипарадигмальный язык программирования из семейства языков .NET, поддерживающий функциональное программирование. Функциональное программирование позволяет создавать более надежные и устойчивые программы, которые проще тестировать и поддерживать. Благодаря своим особенностям функциональное программирование распространено при работе с важными данными или при решении задач, где нужны сложные вычисления. В F# важно соблюдать табуляцию, ибо это влияет на компиляцию кода.

Первым шагом была установка .NET SDK, необходимого для работы с F#. Я посетил официальный сайт .NET и загрузил последнюю версию SDK. Затем установил Visual Studio Code и расширение Ionide для удобной разработки на F#. Эти инструменты обеспечили мне комфортное окружение для написания кода.

Следующим шагом было создание простого консольного приложения на F#. С использованием команды **dotnet new console -lang f#**, я создал новый проект, который автоматически включал в себя структуру F#-приложения. В файле Program.fs я получил простую программу "Hello, World".

printfn "Hello from F#"

Далее я запустил программу через терминал командой dotnet run .\Program.fs на что получил ответ:

Hello from F#

Далее я попробовал усложнить программу путем добавления простейшей функции:

let sayHello () =

    printfn "Hello, World!"

sayHello()

let sayHello () = определяет функцию sayHello внутри этого модуля.

sayHello() является вызовом функции sayHello.

Примечательно, что в F# используется ключевое слово let для создания неизменяемых значений. Если вам нужна изменяемая переменная, вы можете использовать ключевое слово mutable.

let mutable a = 5

printfn "Начальное значение  'a': %d" a

// Присвоение нового значения

a <- 10

printfn "Новое значение 'a': %d" a

Тут мы плавно переходим к выводу переменных в консоль.

printfn в F# является функцией, предназначенной для вывода форматированной строки в консоль. Форматирование выполняется с использованием специальных символов, таких как %d, которые представляют значения, которые будут вставлены в строку.

В нашем случае, **printfn "Начальное значение 'a': %d" a** это строка формата, где %d является спецификатором формата для целочисленных значений. Он указывает, что в этом месте должно быть вставлено целое число.

Также, помимо %d для целочисленных значений, есть и другие спецификаторы формата для различных типов данных. %s – строка, %.2f – числа с плавающей точкой (2 цифры после запятой). В общем, форматирование как в языке Си. Особенным является %A тип, который может выводить значения любых типов. Пример использования будет показан позже.

На следующем этапе я углубился в изучение основных конструкций F#. Изучил сопоставление с образцом, функции высшего порядка, кортежи, записи и активные образцы. Проиллюстрирую пример использования функций высшего порядка:

let add x y = x + y

let multiply x y = x \* y

let applyOperation operation x y =

    operation x y

let result1 = applyOperation add 3 4

let result2 = applyOperation multiply 2 5

printfn "Результат 1: %d" result1

printfn "Результат 2: %d" result2

Вывод в консоль при этом:  
Результат 1: 7

Результат 2: 10

Как мы видим, тут используется особенность синтаксиса функциональных языков программирования - передача функций в функции. В конкретно данном примере, мы создаем функции сложения и умножения, далее создаем функцию applyOperation, которая принимает операцию и переменные, а затем создаем переменные result1, result2 и присваиваем им значения функции applyOperation, куда передаем нашу функцию add & multiply со значениями переменных.

На следующем этапе освоения F# я изучил работу с коллекциями данных. Приведенный ниже пример демонстрирует создание списка, его фильтрацию и преобразование:

let numbers = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

let evenNumbers = numbers |> List.filter (fun x -> x % 2 = 0)

let squaredNumbers = numbers |> List.map (fun x -> x \* x)

printfn "Четные: %A" evenNumbers

printfn "Нечетные: %A" squaredNumbers

В данном коде numbers представляет собой коллекцию (в данном случае, список) целых чисел от 1 до 10

List.filter - Фильтрация элементов списка:

fun x -> x % 2 = 0 - это анонимная функция (лямбда-функция), которая возвращает true для четных чисел (деление на 2 без остатка).

numbers |> List.filter (fun x -> x % 2 = 0) использует оператор |>, который передает список numbers в функцию List.filter с условием фильтрации. Результатом является новый список, содержащий только четные числа из исходного списка.

List.map - Преобразование элементов списка:

fun x -> x \* x - это анонимная функция, которая возвращает квадрат числа.

numbers |> List.map (fun x -> x \* x) использует оператор |>, чтобы передать список numbers в функцию List.map с преобразованием. Результатом является новый список, содержащий квадраты элементов из исходного списка.

Так же следует упомянуть, что функции можно применять поочередно. Это называется декларативное программирование ( Еще одна особенность функциональных языков), например:

let numbers = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

let evenSquaredNumbers =

    numbers

    |> List.filter (fun x -> x % 2 = 0)

    |> List.map (fun x -> x \* x)

printfn "%A" evenSquaredNumbers

Тут функции фильтрации и возведения в квадрат идут последовательно и переменная evenSquaredNumbers возвращает четные квадрируемые числа.

# Заключение

Изучение F# оказалось увлекательным и познавательным процессом. Я освоил основные концепции функционального программирования, овладел работой с коллекциями данных и научился писать базовый код. F# оказался мощным и гибким инструментом, который можно успешно использовать для разработки различных типов приложений.