Защищено:
Гапанюк Ю.Е.

"__"_____**2023** г.

Демонстрация:
Падалко К.Р.

"__"_____**2023** г.

Отчет по домашнему заданию по курсу Парадигмы и конструкции языков программирования

Тема работы: «Знакомство с языком F#»

7
(количество листов)
Вариант № 23

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-54Б _____

(подпись)

Падалко К.Р.

"__"_____**2023** г.

Москва, МГТУ - **2023**

СОДЕРЖАНИЕ

1. 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.

2. Введение

F# (F Sharp) - это функциональный статически типизированный язык программирования общего пользования, который создан и развивается компанией Microsoft. Отличительной чертой F# является то, что он работает поверх платформы .NET и тем самым позволяет в определенной степени использовать возможности, предоставляемые этой платформой, например, систему типов, различные библиотеки и т.д.

3. Объектно-ориентированное программирование в F#

Одной из ключевых особенностей F# является поддержка объектно-ориентированного программирования (ООП). В языке присутствуют все основные принципы ООП. В примере показан интерфейс и наследуемые классы.

Пример кода:

```
open System

//Интерфейс
type Empty = interface end

//Наследуемые классы с вариантами решения
type no_roots()=
 interface Empty

//Класс, содержит один параметр, который присваивается свойству
type one_root(x:double)=
 interface Empty

  member val root = x : double with get, set

//Класс, содержит два параметра, которые присваиваются свойству
type two_roots(x1:double,x2:double)=
 interface Empty

 member val root1 = x1 : double with get, set
 member val root2 = x2 : double with get, set


//Вычисление корней уравнения
let calculate_roots(a:double, b:double, c:double):Empty =
 let D = b*b - 4.0*a*c;
 if D < 0.0 then (new no_roots() :> Empty)
 else if D = 0.0 then
  let x = -b / (2.0 * a)
  //Требуется приведение к интерфейсному типу
  (one_root(x) :> Empty)
 else
  let sqrtD = Math.Sqrt(D)
  let x1 = (-b + sqrtD) / (2.0 * a);
  let x2 = (-b - sqrtD) / (2.0 * a);
  (two_roots(x1,x2) :> Empty)


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


[<EntryPoint>]
let main argv =
 //данные
 //2 корня
 let a1 = 5.0;
 let b1 = -8.0;
 let c1 = -3.0;
 //1 корень
 let a2 = 2.0;
 let b2 = -6.0;
 let c2 = 9.0;
```

```
//нет корней
let a3 = 1.0;
let b3 = 0.0;
let c3 = 4.0;
print_roots(a1,b1,c1)
print_roots(a2,b2,c2)
print_roots(a3,b3,c3)

 //|> ignore - перенаправление потока с игнорирование результата вычисления
Console.ReadLine() |> ignore
0 // выход
```

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

3. Работа с коллекциями

Для хранения набора данных в языке F# предназначены коллекции. Стоит отметить, что типы коллекций в F# являются неизменяемыми.

В F# есть следующие коллекции: List (список), Array (массив), seq (последовательность), Map (словарь), Set (набор данных, основанный на бинарных деревьях). Коллекции позволяют эффективно хранить и управлять группами объектов.

Ниже приведен пример создания списка людей (List) и его методы.

Пример кода:

```
open System

let obj = [] //пустой список

let people = ["Tom"; "Sam"; "Bob"] //Список включает 3 элемента

for person in people do
    printfn "%s" person

//IsEmply возвращает true, если в списке нет элементов, то false
printfn "Список пуст? %b" (people.IsEmpty)
printfn "Количество элементов: %d" (people.Length)
printfn "Первый элемент: %s" (people.Head)
printfn "Второй элемент %s" (people.Tail.Head)
printfn "Третий элемент %s" (people.Item(2))


let people1 = List.append people  ["Alice"; "Mike"; "Daniel"] //list.append() добавляет
элементы одного списка в другой
for p in people1 do printf "%s  " p
```

```
printfn ""


let people2 = List.updateAt 0 "Alex" people1   //List.update() изменяет элемент, по
индексу 0  устанавливаем значение  "Alex"
for p in people2 do printf "%s  " p
printfn ""


let people3 = List.removeAt 1 people2   //List.removeAt() удаляет элемент, удаляем элемент
по индексу 1
for p in people3 do printf "%s  " p
printfn ""


let people4 = List.insertAt 0 "Nick" people3    //List.insertAt() добавляет элемент по
определнному индексу, добавляем по индексу 0 строку "Nick"
for p in people4 do printf "%s  " p
printfn ""

Console.ReadLine() |> ignore
0
```

4. Pattern matching

Pattern matching (сопоставление шаблонов/паттернов) представляет механизм, который позволяет сопоставить некоторое выражение с определенным шаблоном.

Паттерн типов record позволяет разложить значения record на отдельные переменные.

В коде определена запись Person, которая состоит из двух свойств - Name (имя пользователя) и Language (язык пользователя).

Логические паттерны позволяют использовать операторы & и | для других паттернов. Операция | представляет паттерн OR и указывает, что выражение должно соответствовать хотя бы одному из двух паттернов этой операции.

```
open System

//использование паттерна типа record. Позволяет разложить значения record на отдельные
переменные.
type Person = { Name: string; Language: string }

let printHello person =
    match person with
    | { Name = username; Language = "russian"; } -> printfn "Привет, %s" username
    | { Name = username; Language = "english"; } -> printfn "Hello, %s" username
    | _ -> printfn "您好"

let Ivan = {Name="Иван"; Language="russian"}
let Nikita = {Name="Никита"; Language="english"}
```

```
let Vladislav = {Name="Владислав"; Language="china"}

printHello Ivan
printHello Nikita
printHello Vladislav

//логическое использование паттера OR
type UserStatus =
    | Admin = 0
    | Moderator = 1
    | User = 2

let checkStatus user =
    match user with
    | (_, UserStatus.Admin) | (_, UserStatus.Moderator) -> "Доступ разрешен"
    | _ -> "Доступ запрещен"

printfn "%s" (checkStatus ("Tom", UserStatus.Admin))
printfn "%s" (checkStatus ("Bob", UserStatus.Moderator))
printfn "%s" (checkStatus ("Sam", UserStatus.User))

Console.ReadLine() |> ignore
0
```

5. Заключение

F# предоставляет разработчикам мощные инструменты для реализации объектно-ориентированных концепций и эффективной работы с коллекциями данных. Эти возможности делают язык популярным среди разработчиков, создающих разнообразные приложения для платформы Windows. Ознакомление с основными конструкциями и парадигмой языка позволяет создавать надежные и эффективные программы.