Лабораторная работа № 1. «Функции»

Цель работы: ознакомиться с понятием функции в F#, на примере нескольких простых задач научиться работать с функциями.

Теоретические сведения

F# — это язык программирования, обеспечивающий поддержку функционального программирования, а также объектно-ориентированного и императивного (процедурного) программирования.

Язык программирования F# поддерживает следующие конструкции функционального программирования:

- Функции в качестве значений — позволяет гибко управлять функциями.
- Объединение и конвейеризация функций — позволяет объединять функции для создания новых функций и упрощения кодирования последующих операций с данными.
- Определение типа — устраняет необходимость явно вызывать типы без ущерба для безопасности типа.
- Автоматическое обобщение — позволяет повторно использовать код, упрощая написание кода, который работает с множеством разных типов без дополнительных усилий.
- Поддержка сопоставления шаблонов, упрощающая сложный код условия, и размеченные объединения, которые оптимизируются для использования с сопоставлением шаблонов.
- Типы коллекций для работы с неизменяемыми данными, включая типы list и sequence.
- Лямбда-выражения — важны для многих конструкций функционального программирования.
- Частичное применение аргументов функций — обеспечивает возможность неявного создания новых функций из существующих.
- Кавычки кода — функция, позволяющая программно манипулировать выражениями языка F#.

Visual F# предоставляет интерактивное окно, интегрированное в среду разработки Visual Studio. Данное окно позволяет вводить код F#, который сразу же компилируется и выполняется. Это позволяет легко создавать прототипы конструкций кода и проверять код при его написании. В интерактивном окне запускается средство интерактивного режима F# (fsi.exe), которое можно также запускать из командной строки. Такая функция дает возможность использовать язык F# в качестве скриптового языка.

Функции — это основной элемент выполнения программы в любом языке программирования. Как и в других языках, функция в языке F# имеет имя, может иметь параметры и принимать аргументы, а также функция имеет тело. Язык F# также поддерживает конструкции функционального программирования, например, обработку функций как значений, использование в выражениях неименованных функций, объединение функций для образования новых функций, каррированные функции и неявное определение функций посредством частичного применения аргументов функции.

Функции определяются с помощью ключевого слова *let* или, если функция рекурсивная, комбинации ключевых слов *let rec*.

Простое определение функции выглядит примерно следующим образом.

```
let f x = x + 1
```

В предыдущем примере имя функции *f*, аргумент *x* и он принадлежит к типу int, тело функции *x+1*, возвращаемое значение имеет тип *int*.

Область

На любом уровне области, отличной от области модуля, не будет ошибкой повторно использовать имя функции. Если имя используется повторно, то имя, объявленное позже, перекрывает имя, объявленное ранее.

```
let list1 = [ 1; 2; 3]
let sumPlus x =
    let list1 = [1; 5; 10]
    x + List.sum list1
```

Параметры

Имена параметров перечислены после имени функции. Можно задать тип для параметра, как показано в следующем примере.

```
let (x: int) = x + 1
```

Если тип задан, он следует за именем параметра, отделенный двоеточием. Если тип параметра не указан, он будет выведен компилятором. Например, в следующем определении функции тип аргумента *x* выведен как тип *int*, поскольку «1» принадлежит к типу *int*.

```
let f x = x + 1
```

Однако компилятор попытается сделать функцию насколько возможно универсальной. Например, рассмотрим следующий код:

```
let f x = (x, x)
```

Функция создает кортеж из одного аргумента типа *any*. Поскольку тип не задан, функция может использоваться с типом аргумента *any*.

Тело функции

Тело функции может содержать определения локальных переменных и функций. Область таких переменных и функций — тело текущей функции, но не вне его. Если включена возможность облегченного синтаксиса, необходимо использовать отступ для обозначения того, что определение находится внутри тела функции, как показано в следующем примере.

```
let cylinderVolume radius length =
    let pi = 3.14159
    length * pi * radius * radius
```

Возвращаемые значения

Компилятор использует результирующее значение в теле функции, чтобы определить возвращаемое значение и тип.

Компилятор может вывести тип результирующего выражения из предшествующих выражений. В функции *cylinderVolume*, показанной в предыдущем разделе, тип объекта *pi* определен по типу литерала 3.14159 как *float*. Компилятор использует тип *pi*, чтобы определить тип выражения *h * pi * r * r* как *float*. Поэтому общий возвращаемый тип функции — *float*.

Для того, чтобы явно задать возвращаемое значение, необходимо написать код следующим образом.

```
let cylinderVolume radius length : float =
    let pi = 3.14159
    length * pi * radius * radius
```

Вызов функции

Вызов функций осуществляется путем указания имени функции, пробела и следующих за ним аргументов через пробел. Например, чтобы вызвать функцию *cylinderVolume* и назначить результат значению *vol*, следует написать приведенный ниже код.

```
let vol = cylinderVolume 2.0 3.0
```

Частичное применение аргументов

Если предоставить меньшее число аргументов, чем задано, будет создана новая функция, ожидающая оставшиеся аргументы. Такой способ работы с аргументами называется каррированием и характерен для языков функционального программирования, таких как F#. Например, предположим, имеются две трубы с радиусом 2.0 и 3.0. Можно было бы создать функции, определяющие объем трубы следующим образом.

```
let smallPipeRadius = 2.0
let bigPipeRadius = 3.0

let smallPipeVolume = cylinderVolume smallPipeRadius
let bigPipeVolume = cylinderVolume bigPipeRadius
```

Затем можно было бы предоставить дополнительный аргумент длины трубы двух различных радиусов.

```
let length1 = 30.0
let length2 = 40.0
let smallPipeVol1 = smallPipeVolume length1
let smallPipeVol2 = smallPipeVolume length2
let bigPipeVol1 = bigPipeVolume length1
let bigPipeVol2 = bigPipeVolume length2
```

Рекурсивные функции

Рекурсивные функции — функции, вызывающие самих себя. Они требуют, чтобы вслед за ключевым словом *let* было указано ключевое слово *rec*. Рекурсивную функцию можно вызвать из тела функции как любую другую. Следующая рекурсивная функция вычисляет элемент последовательности Фибоначчи с индексом *n*. Последовательность чисел Фибоначчи является последовательностью, в которой каждый последующий элемент является суммой двух предыдущих.

```
let rec fib n = if n < 2 then 1 else fib (n - 1) + fib (n - 2)
```

Функциональные значения

В языке F# все функции считаются значениями, которые известны как функциональные значения. Так как функции являются значениями, их можно использовать как аргументы для других функций или в других контекстах, где используются значения. Далее приведен пример функции, которая принимает как аргумент функциональное значение.

```
let apply1 (transform : int -> int ) y = transform y
```

Тип функционального значения можно задать с помощью токена ->. В левой части токена находится тип аргумента, а в правой части — возвращаемое значение. В предыдущем примере *apply1* является функцией, принимающей функцию *transform* как аргумент, где *transform* является функцией, которая принимает целое число и возвращает другое целое число. Следующий код показывает, как использовать функцию *apply1*.

```
let increment x = x + 1
let result1 = apply1 increment 100
```

Значение *result* будет равно 101 после запуска показанного выше кода.

Несколько аргументов разделяются токенами ->, как показано в следующем примере.

```
let apply2 ( f: int -> int -> int) x y = f x y
let mul x y = x * y
let result2 = apply2 mul 10 20
```

Результат равен 200.

Лямбда-выражения

Лямбда-выражение — это неименованная функция. В предыдущих примерах вместо определения именованных функций *increment* и *mul* можно было бы использовать лямбда-выражения, как показано ниже.

```
let result3 = apply1 (fun x -> x + 1) 100
let result4 = apply2 (fun x y -> x * y ) 10 20
```

Лямбда-выражения определяются с помощью ключевого слова *fun*.

Композиция функций и конвейеризация

Функции в F# могут состоять из других функций. Композиция двух функций *function1* и *function2* является другой функцией, которая представляет собой применение функции *function1* и последующее применение *function2*:

```
let function1 x = x + 1
let function2 x = x * 2
let h = function1 >> function2
let result5 = h 100
```

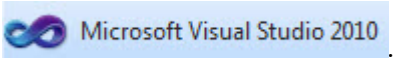
Результат равен 202.

Конвейеризация позволяет объединить вызовы функций в цепочку последовательных операций. Конвейеризация работает следующим образом.

```
let result = 100 |> function1 |> function2
```

Снова будет получен результат 202.

Практическая работа

Запустить 

.
Откроется начальная страница

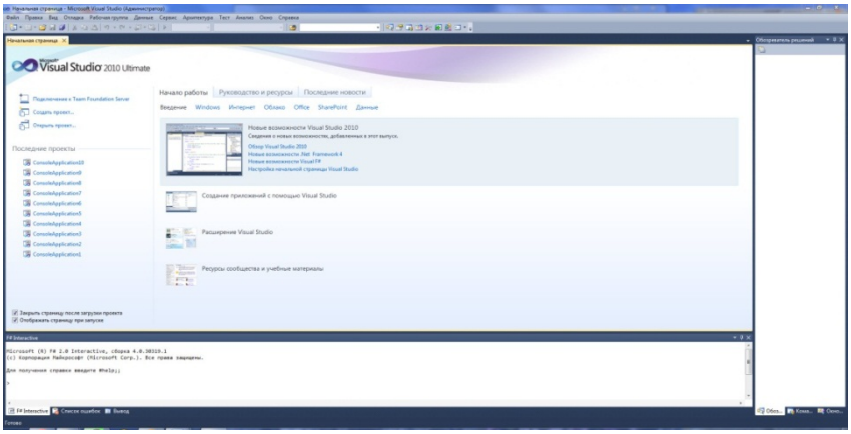


Рисунок 1.1

Выберите вкладку создания проекта

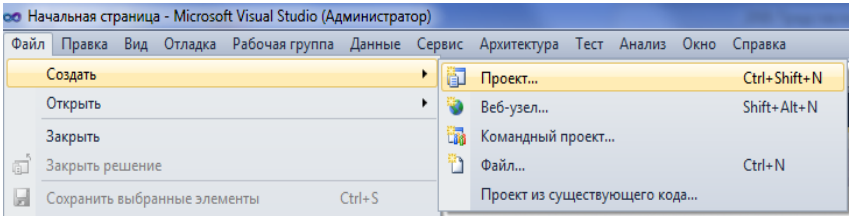


Рисунок 1.2

При создании проекта выберите пункт «Приложение F#»

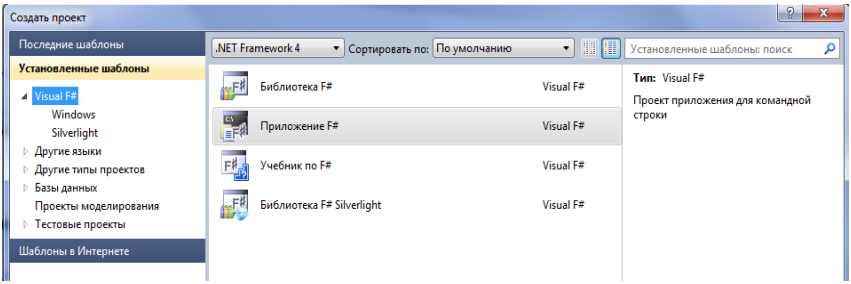


Рисунок 1.3

Откроется окно проекта для написания программы

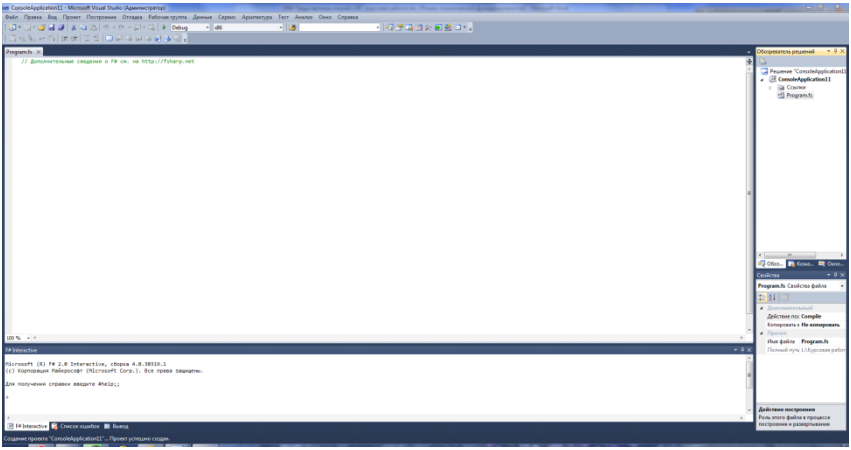


Рисунок 1.4

Вводим код программмы

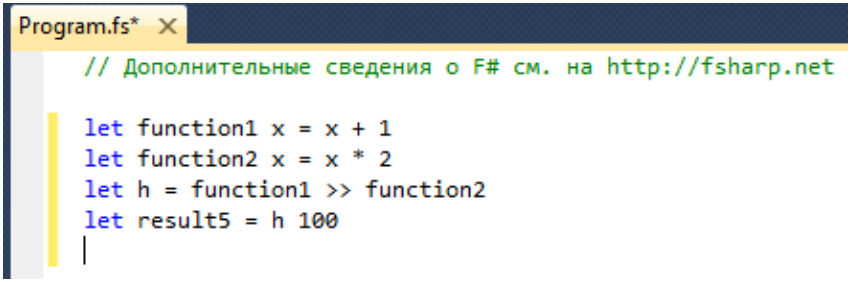


Рисунок 1.5

Для интерактивного выполнения программы выделяем её код и вызываем контекстное меню, выбираем пункт отправить в Interactive:

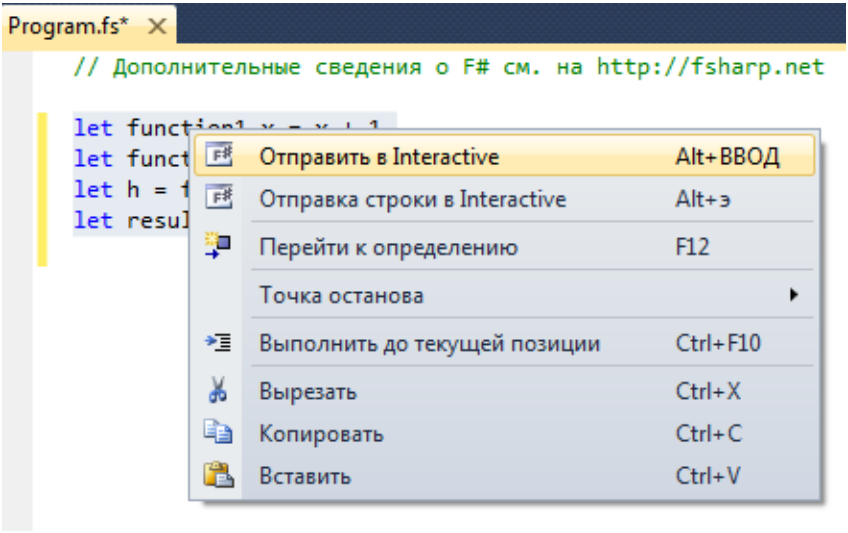


Рисунок 1.6

Смотрим результат выполнения программы в окне F# Interactive.

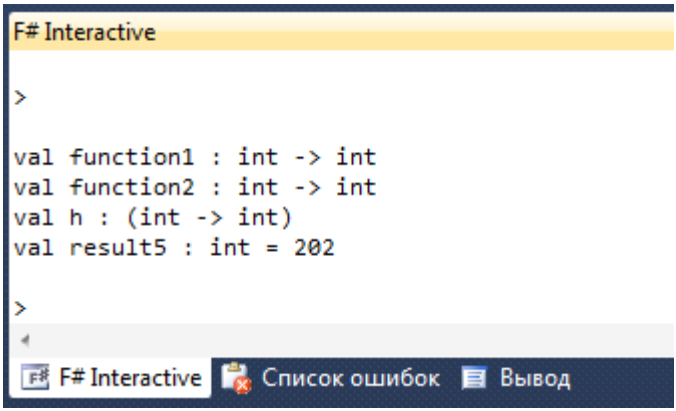


Рисунок 1.7

Задания

1. Написать функцию факториала числа.
2. Найти объем воздушного шарика, если его радиус равен 10 см.
3. С помощью конвейеризации возвести произвольное число в квадрат и умножить полученное значение на 3.
4. Написать функцию, определяющую среднее арифметическое трех целых чисел и возвращающую значение типа int.
5. Найти силу тока в полной цепи, если ЭДС источника равна 10 В, внутреннее сопротивление равно 2 Ом, а внешнее — 3 Ом.
6. Написать программу нахождения дискриминанта квадратного уравнения.
7. Найдите площадь равнобедренного треугольника, если его высота равна 10, а длина основания — 8.
8. В зоопарке 2 медведя, 3 страуса и 1 жираф. Сколько всего лап в зоопарке?
9. Файл занимает 521 Мб места на диске. Размер кластера составляет 0,064 Мб. Сколько кластеров занимает файл?
10. Сколько литров воды вмещается в чайник, если его высота 25 см, а радиус дна — 7 см.

Контрольные вопросы

1. Дайте определение функции.
2. Дайте определение понятию рекурсии.
3. Дайте определение лямбда — выражению.
4. Из каких элементов состоит функция.
5. Что такое токен?
6. Дайте определение каррированию.
7. Что такое конвейеризация?

Задания
для самостоятельной отработки

Комбинация №1	1000000011
Комбинация №2	0100110000
Комбинация №3	1100000001
Комбинация №4	0001000101
Комбинация №5	0010010001
Комбинация №6	0100000110
Комбинация №7	1010100000
Комбинация №8	1000001010
Комбинация №9	0100001100
Комбинация №10	1010000100
Комбинация №11	0100010001
Комбинация №12	1100001000
Комбинация №13	0100000101
Комбинация №14	1000010100
Комбинация №15	0100101000
Комбинация №16	0010000011
Комбинация №17	1000101000
Комбинация №18	0000110010
Комбинация №19	0010001100
Комбинация №20	1100000010
Комбинация №21	1001000010
Комбинация №22	0110000100
Комбинация №23	0000101001
Комбинация №24	0010010010
Комбинация №25	0101010000