

## Лабораторная работа «Списки»

**Цель работы:** ознакомиться с понятием списков в F# и научиться применять списки на примере нескольких тренировочных программ.

### Теоретические сведения

Список в языке F# — это упорядоченный, неизменный ряд элементов одного типа. Список можно определить, явно перечислив его элементы между квадратными скобками с точкой с запятой в качестве разделителя:

```
let list123 = [1; 2; 3]
```

Можно также вставить между элементами разрывы строк; в этом случае точки с запятой являются необязательными. Последний вариант синтаксиса может повысить удобочитаемость кода, особенно если выражения инициализации элементов имеют значительную длину, или если для каждого элемента требуется ввести комментарий.

```
let list123 = [  
1  
2  
3]
```

Обычно все элементы списка должны быть одного типа.

Элементы списка можно также определить, используя диапазон, заданный целыми числами, разделенными оператором диапазона (..):

```
let list1 = [1 ..10]
```

Также можно задать список с помощью циклической конструкции, как в следующем коде.

```
let listOfSquares = [for i in 1 ..10 -> i*i]
```

Пустой список задается парой квадратных скобок, между которыми ничего нет.

### *Операторы для работы со списками*

Элементы можно добавить в список с помощью оператора *::* (*cons*). Если список *list1* содержит [2; 3; 4], следующий код создаст список *list2*, содержащий [100; 2; 3; 4].

```
let list2 = 100:: list1
```

Списки, имеющие совместимые типы, можно сцепить с помощью оператора *@*, как в следующем коде. Если список *list1* содержит [2; 3; 4], а список *list2* содержит [100; 2; 3; 4], этот код создает список *list3*, содержащий [2; 3; 4; 100; 2; 3; 4].

```
let list3 = list1 @ list2
```

Поскольку списки в языке F# неизменяемы, в результате всех операций изменения создаются новые списки, а не изменяются старые.

Списки в языке F# реализованы в виде однократно связанных списков, что означает, что операции, обращающиеся только к началу списка, имеют порядок  $O(1)$ , а операции, обращающиеся к элементу, —  $O(n)$ .

Таблица 4.1

### Свойства списков

Свойство	Тип	Описание
Head	'T	Первый элемент.
Empty	bool	Значение true, если список не содержит элементов.
IsEmpty	bool	Значение true, если список не содержит элементов.
Элемент	'T	Элемент с указанным индексом (начинающимся с нуля).
Length	int	Количество элементов.
Tail	'T list	Список без первого элемента.

Примеры использования этих свойств.

```

let list1 = [1; 2; 3]
printfn «list1.Length is %d» (list1.Length)
printfn «list1.Head is %d» (list1.Head)
printfn «list1.Tail.Head is %d» (list1.Tail.Head)
printfn «list1.Tail.Tail.Head is %d» (list1.Tail.Tail.Head)
printfn «list1.Item(1) is %d» (list1.Item(1))

```

Программирование с использованием списков позволяет выполнять сложные операции с помощью небольших фрагментов кода.

### *Операции сортировки в списках*

Функции *List.sort*, *List.sortBy* и *List.sortWith* служат для сортировки списков. Функция сортировки определяет, какую из этих трех функций следует использовать. В функции *List.sort* используется универсальное сравнение по умолчанию. Функция *List.sortBy* принимает функцию, возвращающую значение, используемое в качестве критерия сортировки, а функция *List.sortWith* принимает функцию сравнения в качестве аргумента.

В следующем примере показано использование функции *List.sort*.

```

let sortedList1 = List.sort [1; 4; 8; -2; 5]
printfn «%A» sortedList1

```

Выходные данные выглядят следующим образом:

```
[-2; 1; 4; 5; 8]
```

В следующем примере показано использование функции *List.sortBy*.

```

let sortedList2 = List.sortBy (fun elem -> abs elem) [1; 4; 8; -2; 5]
printfn «%A» sortedList2

```

Выходные данные выглядят следующим образом:

```
[1; -2; 4; 5; 8]
```

В следующем примере показано использование функции *List.sortWith*. В этом примере пользовательская функция сравнения

*compareWidgets* используется сначала для сравнения значений одного поля пользовательского типа, а затем другого, если значения первого поля равны.

```
type Widget = { ID: int; Rev: int }
```

```
let compareWidgets widget1 widget2 =  
  if widget1.ID < widget2.ID then -1 else  
  if widget1.ID > widget2.ID then 1 else  
  if widget1.Rev < widget2.Rev then -1 else  
  if widget1.Rev > widget2.Rev then 1 else  
  0
```

```
let listToCompare = [  
  { ID = 92; Rev = 1 }  
  { ID = 110; Rev = 1 }  
  { ID = 100; Rev = 5 }  
  { ID = 100; Rev = 2 }  
  { ID = 92; Rev = 1 }  
]
```

```
let sortedWidgetList = List.sortWith compareWidgets listToCompare  
printfn «%A» sortedWidgetList
```

Выходные данные выглядят следующим образом:

```
[{ID = 92;  
  Rev = 1;}; {ID = 92;  
  Rev = 1;}; {ID = 100;  
  Rev = 2;}; {ID = 100;  
  Rev = 5;}; {ID = 110;  
  Rev = 1;}]
```

### *Операции поиска в списках*

Списки поддерживают большое число операций поиска. Самая простая функция *List.find* позволяет найти первый элемент, который удовлетворяет заданному условию.

В следующем примере кода демонстрируется использование функции *List.find* для поиска в списке первого числа, которое делится на 5.

```
let isDivisibleBy number elem = elem % number = 0  
let result = List.find (isDivisibleBy 5) [1 ..100]  
printfn «%d» result
```

Результат применения функции — 5.

Если элементы необходимо сначала преобразовать, вызовите функцию *List.pick*, которая принимает функцию, возвращающую значение типа *option*, а затем ищет первое значение типа *option*, равное *Some(x)*. Вместо возврата элемента функция *List.pick* возвращает результат *x*. Если соответствующий элемент не найден, функция *List.pick* вызывает исключение. В следующем коде показано использование функции *List.pick*.

```
let valuesList = [(«a», 1); («b», 2); («c», 3)]
let result = List.pick (fun elem -> if (snd elem = 2) then Some(fst elem) else
None) valuesList
printfn «%A» result
```

Выходные данные выглядят следующим образом:

(«b», 2)

Другая группа операций поиска, *List.tryFind* и связанные с ней функции, возвращает значение типа *option*. Функция *List.tryFind* возвращает первый элемент списка, удовлетворяющий условию, если такой элемент существует, и значение *None*, если элемент не существует. Версия *List.tryFindIndex* возвращает индекс элемента, если элемент найден, а не сам элемент. Эти функции демонстрируются в следующем коде.

```
let list1 = [1; 3; 7; 9; 11; 13; 15; 19; 22; 29; 36]
let isEven x = x % 2 = 0
match List.tryFind isEven list1 with
| Some value -> printfn «The first even value is %d.» value
| None -> printfn «There is no even value in the list.»
match List.tryFindIndex isEven list1 with
| Some value -> printfn «The first even value is at position %d.» value
| None -> printfn «There is no even value in the list.»
```

Выходные данные выглядят следующим образом:

The first even value is 22. The first even value is at position 8.

### *Арифметические операции над списками*

В модуль *List* встроены стандартные арифметические операции, такие как вычисление суммы или среднего.

В следующем коде показано использование функций *List.sum*, *List.sumBy* и *List.average*.

```
let sum1 = List.sum [1 ..10]
let avg1 = List.average [0.0; 1.0; 1.0; 2.0]
printfn «%f» avg1
```

В результате получается 1.000.000.

В следующем коде показано использование функции *List.averageBy*.

```
let avg2 = List.averageBy (fun elem -> float elem) [1 ..10]
printfn «%f» avg2
```

В результате получается 5.5.

### *Списки и кортежи*

Со списками, содержащими кортежи, можно использовать функции *zip* и *unzip*. Эти функции объединяют два списка отдельных значений в один список кортежей или разделяют один список кортежей на два списка отдельных значений. Самая простая функция *List.zip* принимает два списка отдельных элементов и создает один список из двухэлементных кортежей. Другая версия, *List.zip3*, принимает три списка отдельных элементов и создает один список из кортежей, содержащих по три элемента. В следующем примере кода показано использование функции *List.zip*.

```
let list1 = [1; 2; 3]
let list2 = [-1; -2; -3]
let listZip = List.zip list1 list2
printfn «%A» listZip
```

Выходные данные выглядят следующим образом:

(1, -1); (2, -2); (3, -3)]

В следующем примере кода показано использование функции *List.zip3*

```
let list3 = [0; 0; 0]
let listZip3 = List.zip3 list1 list2 list3
printfn «%A» listZip3
```

Выходные данные выглядят следующим образом:

[(1, -1, 0); (2, -2, 0); (3, -3, 0)]

Соответствующие версии функций распаковки, *List.unzip* и *List.unzip3*, принимают списки кортежей и возвращают кортеж списков, где первый список содержит все первые элементы кортежей, второй список содержит все вторые элементы кортежей и т. д.

В следующем примере кода показано использование функции *List.Unzip*.

```
let lists = List.unzip [(1, 2); (3, 4)]
printfn «%A» lists
printfn «%A %A» (fst lists) (snd lists)
```

Выходные данные выглядят следующим образом:

[(1; 3], [2; 4])

[1; 3] [2; 4]

В следующем примере кода показано использование функции *List.unzip3*.

```
let listsUnzip3 = List.unzip3 [(1,2,3); (4,5,6)]
printfn «%A» listsUnzip3
```

Выходные данные выглядят следующим образом:

[(1; 4], [2; 5], [3; 6])

### *Операции над элементами списков*

Язык F# поддерживает разнообразные операции над элементами списков. Самая простая функция — *List.iter*, которая позволяет вызвать некоторую функцию для каждого элемента списка. К другим ее вариантам относятся функция *List.iter2*, которая позволяет выполнять операцию над элементами двух списков, функция *List.iteri*, которая подобна функции *List.iter* за тем исключением, что в качестве аргумента функции, которая вызывается для каждого элемента, передается индекс этого элемента, и функция *List.iteri2*, которая объединяет функции *List.iter2* и *List.iteri*.

```

let list1 = [1; 2; 3]
let list2 = [4; 5; 6]
List.iter (fun x -> printfn «List.iter: element is %d» x) list1
List.iteri(fun i x -> printfn «List.iteri: element %d is %d» i x) list1
List.iter2 (fun x y -> printfn «List.iter2: elements are %d %d» x y) list1 list2
List.iteri2 (fun i x y ->
    printfn «List.iteri2: element %d of list1 is %d element %d of list2 is %d»
        i x i y)
    list1 list2

```

ВЫХОДНЫЕ ДАННЫЕ ВЫГЛЯДЯТ СЛЕДУЮЩИМ ОБРАЗОМ:

```

List.iter: element is 1
List.iter: element is 2
List.iter: element is 3
List.iteri: element 0 is 1
List.iteri: element 1 is 2
List.iteri: element 2 is 3
List.iter2: elements are 1 4
List.iter2: elements are 2 5
List.iter2: elements are 3 6
List.iteri2: element 0 of list1 is 1; element 0 of list2 is 4
List.iteri2: element 1 of list1 is 2; element 1 of list2 is 5
List.iteri2: element 2 of list1 is 3; element 2 of list2 is 6

```

Еще одна часто используемая функция, преобразующая элементы списка, это — функция *List.map*, которая позволяет применить функцию к каждому элементу списка и поместить результаты в новый список. Функции *List.map2* и *List.map3* — это варианты, принимающие несколько списков. Можно также использовать функции *List.map1* и *List.map12*, если, помимо элемента, функции также требуется передавать индекс каждого элемента. Единственное отличие между функциями *List.map12* и *List.map1* заключается в том, что функция *List.map12* работает с двумя списками. В следующем примере демонстрируется использование функции *List.map*.

```

let list1 = [1; 2; 3]
let newList = List.map (fun x -> x + 1) list1
printfn «%A» newList

```

ВЫХОДНЫЕ ДАННЫЕ ВЫГЛЯДЯТ СЛЕДУЮЩИМ ОБРАЗОМ:

```

[2; 3; 4]

```



В следующем примере демонстрируется использование функции *List.map2*.

```
let list1 = [1; 2; 3]
let list2 = [4; 5; 6]
let sumList = List.map2 (fun x y -> x + y) list1 list2
printfn «%A» sumList
```

Выходные данные выглядят следующим образом:

```
[5; 7; 9]
```

В следующем примере демонстрируется использование функции *List.map3*.

```
let newList2 = List.map3 (fun x y z -> x + y + z) list1 list2 [2; 3; 4]
printfn «%A» newList2
```

Выходные данные выглядят следующим образом:

```
[7; 10; 13]
```

В следующем примере демонстрируется использование функции *List.mapI*.

```
let newListAddIndex = List.mapI (fun i x -> x + i) list1
printfn «%A» newListAddIndex
```

Выходные данные выглядят следующим образом:

```
[1; 3; 5]
```

В следующем примере демонстрируется использование функции *List.mapI2*.

```
let listAddTimesIndex = List.mapI2 (fun i x y -> (x + y) * i) list1 list2
printfn «%A» listAddTimesIndex
```

Выходные данные выглядят следующим образом:

```
[0; 7; 18]
```

Функция *List.collect* аналогична функции *List.map*, за исключением того, что для каждого элемента создается список, и все эти списки сцепляются в конечный список. В следующем коде

для каждого элемента списка создаются три числа. Все эти числа собираются в один список.

```
let collectList = List.collect (fun x -> [for i in 1..3 -> x * i]) list1
printfn «%A» collectList
```

Выходные данные выглядят следующим образом:  
[1; 2; 3; 2; 4; 6; 3; 6; 9]

Можно также использовать функцию `List.Filter`, которая принимает логическое условие и создает новый список, содержащий только элементы, которые удовлетворяют заданному условию.

```
let evenOnlyList = List.filter (fun x -> x % 2 = 0) [1; 2; 3; 4; 5; 6]
```

В результате будет получен список [2; 4; 6].

#### *Работа с несколькими списками*

Списки можно объединять. Чтобы объединить два списка в один, следует использовать функцию `List.append`. Чтобы объединить более двух списков, следует использовать функцию `List.concat`.

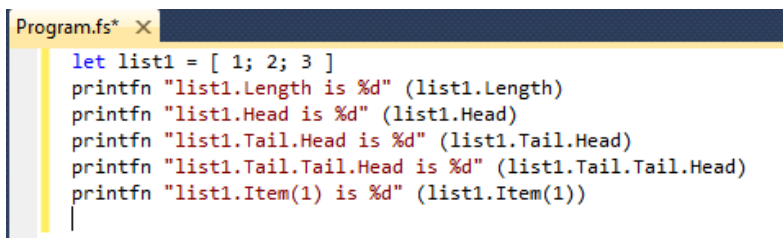
```
let list1to10 = List.append [1; 2; 3] [4; 5; 6; 7; 8; 9; 10]
let listResult = List.concat [ [1; 2; 3]; [4; 5; 6]; [7; 8; 9] ]
```

## Практическая работа

Запустите Visual Studio.

Создайте приложение на F#.

Введите программу, выводящую характеристики списков.



```
Program.fs* X
let list1 = [ 1; 2; 3 ]
printfn "list1.Length is %d" (list1.Length)
printfn "list1.Head is %d" (list1.Head)
printfn "list1.Tail.Head is %d" (list1.Tail.Head)
printfn "list1.Tail.Tail.Head is %d" (list1.Tail.Tail.Head)
printfn "list1.Item(1) is %d" (list1.Item(1))
```

Рисунок 4.1.

Запустите введенную программу на исполнение в интерактивном режиме.

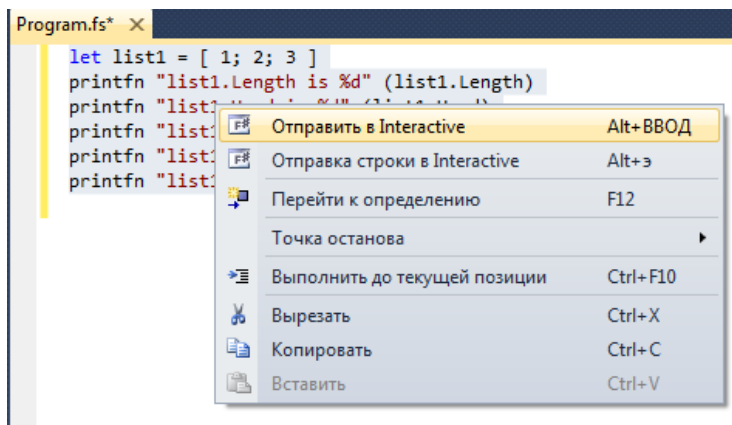


Рисунок 4.2.

Проанализируйте результат выполнения программы.

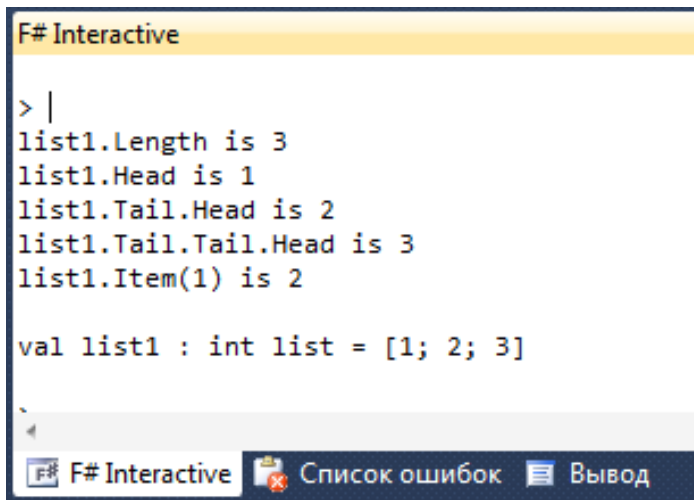


Рисунок 4.3.

## Задания начального уровня

1. Найти сумму квадратов чисел от списка из 9 элементов. Произвольный список создать программно. Результат вывести на экран.
2. Написать функцию, определяющую, есть ли данный элемент в списке из 8 целочисленных элементов. Произвольный целочисленный список с шагом списка 1 и элемент поиска создать программно. Результат вывести на экран в виде False или True.
3. Слить 2 целочисленных списка в один. Списки задать программно. Величина 1-го списка 5 элементов, величина 2-го списка 4 элемента. Сформированный список вывести на экран.
4. Найти сумму кубов чисел от списка из 7 элементов. Произвольный список создать программно. Шаг списка 3. Результат вывести на экран.
5. Написать функцию, определяющую, есть ли данный элемент в списке из 8 символьных элементов. Произвольный символьный список и элемент поиска создать программно. Результат вывести на экран в виде False или True.
6. Даны два целочисленных списка. Списки задать программно. Величина первого списка 3 элемента, величина второго списка 4 элемента. Из первого списка взять первый элемент, из второго – последний. Вывести их на экран.
7. Найти произведение квадратов чисел от списка из 6 элементов. Произвольный список создать программно. Результат вывести на экран.
8. Написать функцию, определяющую, есть ли данный элемент в списке из 7 целочисленных элементов. Произвольный целочисленный список и элемент поиска создать программно. Шаг списка 2. Результат вывести на экран в виде False или True.
9. Слить два целочисленных списка в один. Величина первого списка 3 элемента, величина второго списка 4 элемента. Сформированный список без первого элемента вывести на экран.
10. Найти произведение кубов чисел от списка из 7 элементов. Произвольный список создать программно. Шаг списка 1. Результат вывести на экран.

## Задания

1. Дан список [1; 2; 3; 4]. Получить список квадратов этих чисел и список их кубов. Затем получить список сумм квадратов и кубов. Вывести результат.

2. В коллективе 6 человек. 4 мужчины и 2 женщины. Возраст мужчин 25, 37, 48, 60 лет. Возраст женщин 28, 43 года. Найти средний возраст женщин и средний возраст мужчин.

3. Оклады работников фирмы составляют 10, 11, 12,5, 15, 13 и 7,5 тысяч рублей. Произведено повышение окладов на 15%. Сколько рублей теперь составляют оклады работников?

4. В группе детского сада раздали конфеты. Оле досталось 10 конфет, Ире — 3, Славе — 7, Каролине — 0. У кого сколько конфет надо забрать и кому сколько надо отдать, чтобы у всех было поровну конфет?

5. В списке от 1 до 20 найти четные числа и возвести их в квадрат.

6. Поезд состоит из 20 вагонов. Из этих вагонов — каждый четвертый — вагон-ресторан. Вывести номера вагонов-ресторанов.

7. В поезде 20 вагонов. Каждый третий вагон — ресторан. В пассажирском вагоне 20 мест. Сколько всего пассажирских мест в поезде?

8. Шел караван верблюдов a, b, c, d. Позже к ним присоединились верблюды e, f, g, а позже — еще и верблюды h, i. Вывести список верблюдов в караване.

9. В списке чисел от 1 до 20 все числа, кратные 3, возвести в квадрат, а кратные 5 — в куб. Объединить полученные списки кубов и квадратов.

10. Ввести произвольный список и вывести его характеристики.

## Контрольные вопросы

1. Дайте определение списку и опишите его основные характеристики.
2. Перечислите функции сортировки списков.
3. Опишите функции, с помощью которых можно производить операции над каждым элементом списка.
4. Приведите способы объединения списков.
5. В чем суть операций свертки?
6. Перечислите способы задания списков.
7. Основные возможности функции List.map.
8. Основные возможности функции List.fold.
9. Основные возможности функции List.exists.
10. Основные возможности функции List.rev.
11. Основные возможности функции List.zip.
12. Основные возможности функции List.filter.
13. Основные возможности функции List.tail.
14. Основные возможности функции List.length.
15. Основные возможности функции List.head.
16. Основные возможности функции List.partition.

**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