МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**ОСНОВЫ РАБОТЫ С F#** — **СПИСКИ, КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ, БИБЛИОТЕКА WINFORMS.**

ЛАБОРАТОРНАЯ РАБОТА №5

(Вариант №5)

студента 4 курса 451 группы

направления 09.03.04 — Программная инженерия

факультета КНиИТ

Голикова Артема Олеговича

Проверил

к. пед. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_ В. А. Векслер

Саратов 2020

**Часть 1-я**

**«Списки»**

**Цель работы**: ознакомится 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 ] – список чисел в диапазоне от 1 до 10

> let list1 = [ 100 .. -1 .. 10 ];;

val list1 : int list =

[100; 99; 98; 97; 96; 95; 94; 93; 92; 91; 90; 89; 88; 87; 86; 85; 84; 83; 82;

81; 80; 79; 78; 77; 76; 75; 74; 73; 72; 71; 70; 69; 68; 67; 66; 65; 64; 63;

62; 61; 60; 59; 58; 57; 56; 55; 54; 53; 52; 51; 50; 49; 48; 47; 46; 45; 44;

43; 42; 41; 40; 39; 38; 37; 36; 35; 34; 33; 32; 31; 30; 29; 28; 27; 26; 25;

24; 23; 22; 21; 20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 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 создает 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](http://msdn.microsoft.com/ru-ru/library/ee340234.aspx) | 'T | Первый элемент. |
| [Empty](http://msdn.microsoft.com/ru-ru/library/ee353570.aspx) | bool | Значение true, если список не содержит элементов. |
| [IsEmpty](http://msdn.microsoft.com/ru-ru/library/ee353526.aspx) | bool | Значение true, если список не содержит элементов. |
| [Элемент](http://msdn.microsoft.com/ru-ru/library/ee353870.aspx)  Item | 'T | Элемент с указанным индексом (начинающимся с нуля). |
| [Length](http://msdn.microsoft.com/ru-ru/library/ee353415.aspx) | int | Количество элементов. |
| [Tail](http://msdn.microsoft.com/ru-ru/library/ee353436.aspx) | '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](http://msdn.microsoft.com/ru-ru/library/ee370323.aspx), [List.sortBy](http://msdn.microsoft.com/ru-ru/library/ee353676.aspx) и [List.sortWith](http://msdn.microsoft.com/ru-ru/library/ee370345.aspx) служат для сортировки списков. Функция сортировки определяет, какую их 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 = 0 }

]

let sortedWidgetList = List.sortWith compareWidgets listToCompare

printfn "%A" sortedWidgetList

Выходные данные выглядят следующим образом.

[{ID = 92;

Rev = 0;}; {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](http://msdn.microsoft.com/ru-ru/library/ee370243.aspx) 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.

Другой вариант: List.find (fun x -> x%5=0) [1..100]

Если элементы необходимо сначала преобразовать, вызовите функцию [List.pick](http://msdn.microsoft.com/ru-ru/library/ee370236.aspx), 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](http://msdn.microsoft.com/ru-ru/library/ee353506.aspx) 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](http://msdn.microsoft.com/ru-ru/library/ee340224.aspx) 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.

*Арифметически операции над списками*

В [модуль List](http://msdn.microsoft.com/ru-ru/library/ee353738.aspx) 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 1.000000.

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](http://msdn.microsoft.com/ru-ru/library/ee353467.aspx) принимает два списка отдельных элементов и создает один список из двухэлементных кортежей. Другая версия, [List.zip3](http://msdn.microsoft.com/ru-ru/library/ee370211.aspx), 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](http://msdn.microsoft.com/ru-ru/library/ee340249.aspx) и [List.unzip3](http://msdn.microsoft.com/ru-ru/library/ee353566.aspx), 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](http://msdn.microsoft.com/ru-ru/library/ee340249.aspx).

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](http://msdn.microsoft.com/ru-ru/library/ee353566.aspx).

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](http://msdn.microsoft.com/ru-ru/library/ee340469.aspx), которая позволяет вызвать некоторую функцию для каждого элемента списка. К другим ее вариантам относятся функция [List.iter2](http://msdn.microsoft.com/ru-ru/library/ee370562.aspx), которая позволяет выполнять операцию над элементами двух списков, функция [List.iteri](http://msdn.microsoft.com/ru-ru/library/ee340295.aspx), 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](http://msdn.microsoft.com/ru-ru/library/ee353682.aspx), которая объединяет функции List.iter2 и List.iteri. Функция List.iter возвращает значение unit.

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](http://msdn.microsoft.com/ru-ru/library/ee370378.aspx), которая позволяет применить функцию к каждому элементу списка и поместить результаты в новый список. Функции [List.map2](http://msdn.microsoft.com/ru-ru/library/ee340232.aspx) и [List.map3](http://msdn.microsoft.com/ru-ru/library/ee370502.aspx) — это варианты, принимающие несколько списков. Можно также использовать функции [List.mapi](http://msdn.microsoft.com/ru-ru/library/ee353425.aspx) и [List.mapi2](http://msdn.microsoft.com/ru-ru/library/ee340264.aspx), если, помимо элемента, функции также требуется передавать индекс каждого элемента. Единственное отличие между функциями List.mapi2 и List.mapi заключается в том, что функция List.mapi2 работает с двумя списками. В следующем примере демонстрируется использование функции [List.map](http://msdn.microsoft.com/ru-ru/library/ee370378.aspx).

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](http://msdn.microsoft.com/ru-ru/library/ee370406.aspx) 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](http://msdn.microsoft.com/ru-ru/library/ee370294.aspx), 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](http://msdn.microsoft.com/ru-ru/library/ee353428.aspx). Чтобы объединить более двух списков, следует использовать функцию [List.concat](http://msdn.microsoft.com/ru-ru/library/ee370370.aspx).

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

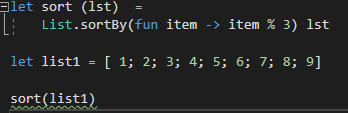
let listResult = List.concat [ [1; 2; 3]; [4; 5; 6]; [7; 8; 9] ]

Пример использования List.pick в рекурсивной функции, для отбора элементов списка и размещения квадратов данных элементов в новый список.   
let rec f lis lisa=    
   if lis<>[] then   
               let r = List.pick (fun x ->  Some(x) ) lis // берем элементы по порядку   
               let rr=r\*r //квадрат элемента   
               let lisa = rr::lisa // квадрат элемента в новый список   
               let lis = List.filter (fun x -> x<>r) lis //выбросим найденный элемент   
               f lis lisa //рекурсия с уменьшенным начальным списком и увеличенным итоговым   
               else  let lisa = List.sort lisa  // сортировка перед выводом  
                      printfn " Множество %A" lisa   
f [1;2;3] [] // два списка начальный и итоговый

**Ход работы:**

**Задание №1.**

Создать функцию, сортирующую элементы списка по остатку от деления на 3.



Результат выполнения:



**Задание №2.**

В списке от 1 до 20 найти четные числа и вывести на экран их в квадраты и кубы. Для решения задач используйте списки. Входные данные получите из файла. Результат запишите в файл.

Код программы:

open System

open System.IO

let stream = new StreamReader @"C:\Users\Makesar\source\repos\Lab5\_GolikovAO\Lab5\_GolikovAO\input.txt"

let mutable valid = true

while (valid) do

let line = stream.ReadLine()

if (line = null) then

valid <- false

else

// Создаем массив чисел из файлов

let mas = line.Split[|' '|]

// Функция выбора четных чисел

let list =

let tmp = List.ofArray mas

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

// Функция перевода элементов листа в строку

let list1 =

let mutable str = ""

for item in list do

str <- str + string(item) + " "

str <- str + "\n"

str

// Функция возведения чисел в квадрат

let list2 =

let tmp = List.map(fun x -> int(x) \* int(x)) list

let tmp = List.map(fun x -> string(x)) tmp

let mutable str = ""

for item in tmp do

str <- str + string(item) + " "

str <- str + "\n"

str

// Функция возведения числе в куб

let list3 =

let tmp = List.map(fun x -> int(x) \* int(x) \* int(x)) list

let tmp = List.map(fun x -> string(x)) tmp

let mutable str = ""

for item in tmp do

str <- str + string(item) + " "

str

File.WriteAllText(@"C:\Users\Makesar\source\repos\Lab5\_GolikovAO\Lab5\_GolikovAO\output.txt", list1 + list2 + list3)

Содержимое input.txt:



Содержимое output.txt после выполнения программы:



**Задание №3.**

Создайте список записей проведите фильтрацию и сортировку. Студенты (рост, вес, фамилия). Удалите студентов с ростом менее 100. Оставшихся отсортируйте по весу (по убыванию). Студенты с одинаковым весом сортируются по фамилии. Определите количество оставшихся студентов. Вычислите средний вес оставшихся студентов.

Код программы:

// Создадим тип Student с нужными параметрами

type Student = {height : int; weight: int; lastname: string}

// Создаем лист списков

let Students : list<Student> = [{height = 178; weight = 70; lastname = "Иванов"};

{height = 98; weight = 74; lastname = "Сидоров"};

{height = 187; weight = 70; lastname = "Высокова"};

{height = 80; weight = 55; lastname = "Одинцова"};

{height = 176; weight = 80; lastname = "Петров"}]

// Удаляем студентов с ростом менее 100

let Students1 = List.filter(fun x -> x.height > 100) Students

// Функции сортировки, которые сортируют по убыванию по весу, а если вес равный - по фамилии.

let compareStudents student1 student2 =

if student1.weight < student2.weight then 1 else

if student1.weight > student2.weight then -1 else

String.Compare(student1.lastname, student2.lastname)

let SortStudents =

List.sortWith compareStudents Students1

// Подсчитываем кол-во оставшихся студентов

let countStudents =

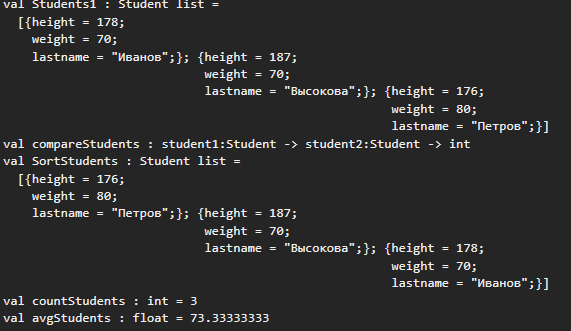
SortStudents.Length

// Подсчитываем средний вес оставшихся студентов

let avgStudents =

List.averageBy(fun x -> float(x.weight)) SortStudents

Результат выполнения:



**Контрольные вопросы**

1. Дайте определение списку и опишите его основные характеристики.

Список в языке F# — это упорядоченный, неизменный ряд элементов одного типа. 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).

1. Определите арифметическую операцию нахождения среднего арифметического элементов в списке через операцию нахождения суммы элементов и операцию нахождения длины списка.

let func (list) =

let count = List.length list

let summ = List.sum list

summ / count

1. Перечислите функции сортировки списков.

List.sort, List.sortDescending, List.sortBy, List.sortByDescending, List.sortWith

1. Какие функции используются при работе со списками, состоящими из кортежей?

List.zip, List.zip3, List.unzip, List.unzip3

1. Опишите функции, с помощью которых можно производить операции над каждым элементом списка.

List.iter, List.iter2, List.iteri, List.map, List.map2, List.map3, List.mapi, List.mapi2

1. Приведите способы объединения списков.

List.append, List.concat

1. Каким образом можно считать информацию из файла.

open System.IO

let stream = new StreamReader @"C:\1\1.txt"

let mutable valid = true

while (valid) do

let line = stream.ReadLine()

if (line = null) then

valid <- false

else

printfn "%A" line

или

open System

open System.IO

let lines = File.ReadAllLines(@"C:\1\2.txt")

**Часть 2. Консольное приложение.**

**Задание:**

Пользователь вводит дату. Программа сообщает время года, сколько дней осталось до конца года.

Код программы:

Console.Title <- "Голиков А.О. - Вывод времени года, сколько дней осталось до НГ"

Console.ForegroundColor <- ConsoleColor.Cyan

Console.BackgroundColor <- ConsoleColor.Black

printfn "Введите дату в формате DD.MM.YYYY"

// Считываем дату с клавиатуры

let mutable inputdate = System.Console.ReadLine().Split[|'.'|]

// Создадим массив из кол-ва дней в месяцах

let mutable days = [|31; 28; 31; 30; 31; 30; 31; 31; 30; 31; 30; 31|]

// Функция определяющая время года

match int(inputdate.[1]) with

| 1 | 2 | 12 -> printfn "Время года: Зима"

| 3 | 4 | 5 -> printfn "Время года: Весна"

| 6 | 7 | 8 -> printfn "Время года: Лето"

// Функция проверяющая год на високосность

let IsLeapYear =

if (int(inputdate.[2]) % 4 = 0) && (int(inputdate.[2]) % 100 <> 0) || (int(inputdate.[2]) % 400 = 0) then true

else false

// Функция высчитывающая кол-во дней до конца года

let DaysToNY =

let mutable tmp = 365

let mutable dayscount = 0

if IsLeapYear then

tmp <- 366

days.[1] <- 29

for i = 0 to (int(inputdate.[1]) - 2) do

dayscount <- dayscount + days.[i]

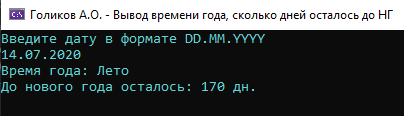
tmp - dayscount - int(inputdate.[0])

printfn "До нового года осталось: %A дн." DaysToNY

Console.ReadKey()

Console.Clear()

Результаты выполнения:



**Часть 3-я**

**«Библиотека WinForms – в F#»**

**Цель работы:** научиться создавать графический интерфейс, добавлять объекты и обрабатывать события с применением библиотеки WinForms.

Windows Forms - одна из наиболее интересных возможностей Microsoft .NET. Если вы знакомы с MFC (или Windows API), то Windows Forms хорошее начало для работы с .NET Framework class library, потому что она позволяет писать традиционные GUI приложения с окнами, формами и т.п. вещами. Однажды, начав работать с Windows Forms вы сможете быстро понять .NET Framework.

Для использования библиотеки WinForms при разработке приложений на F# требуются понимать следующие положения:

каждое окно на экране – это экземпляр класса Form, содержащий коллекцию элементов управления пользовательского интерфейса типа Control;

в процессе взаимодействия пользователей с элементами формы происходят события, которые могут обрабатываться программой;

Windows Forms – библиотека, представляющую объектно-ориентированную обертку вокруг Windows API.

Для качественного прототипирования формы можно воспользоваться дизайнером форм от C# (рис. 6).

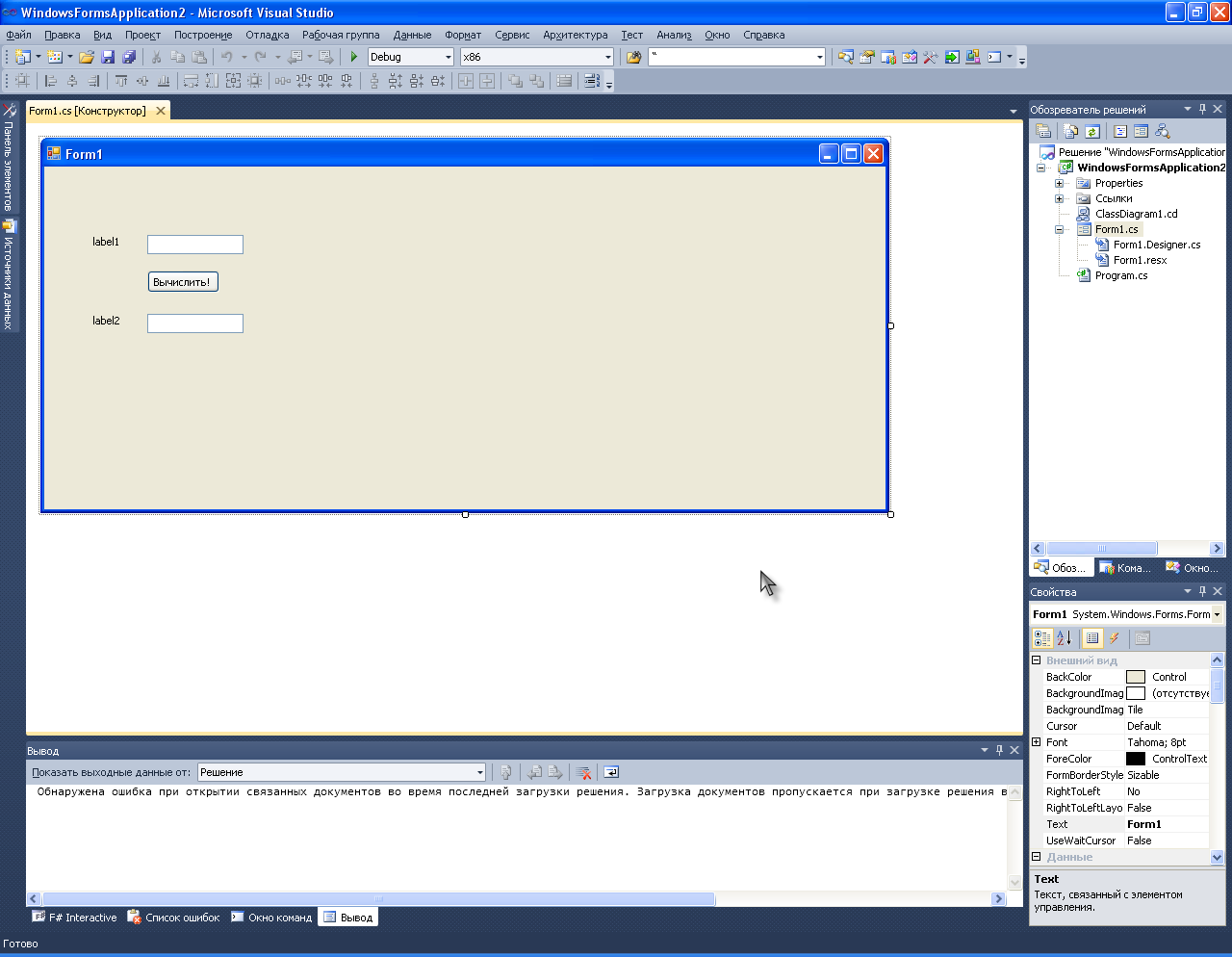


Рисунок 6 – Редактор форм от C#

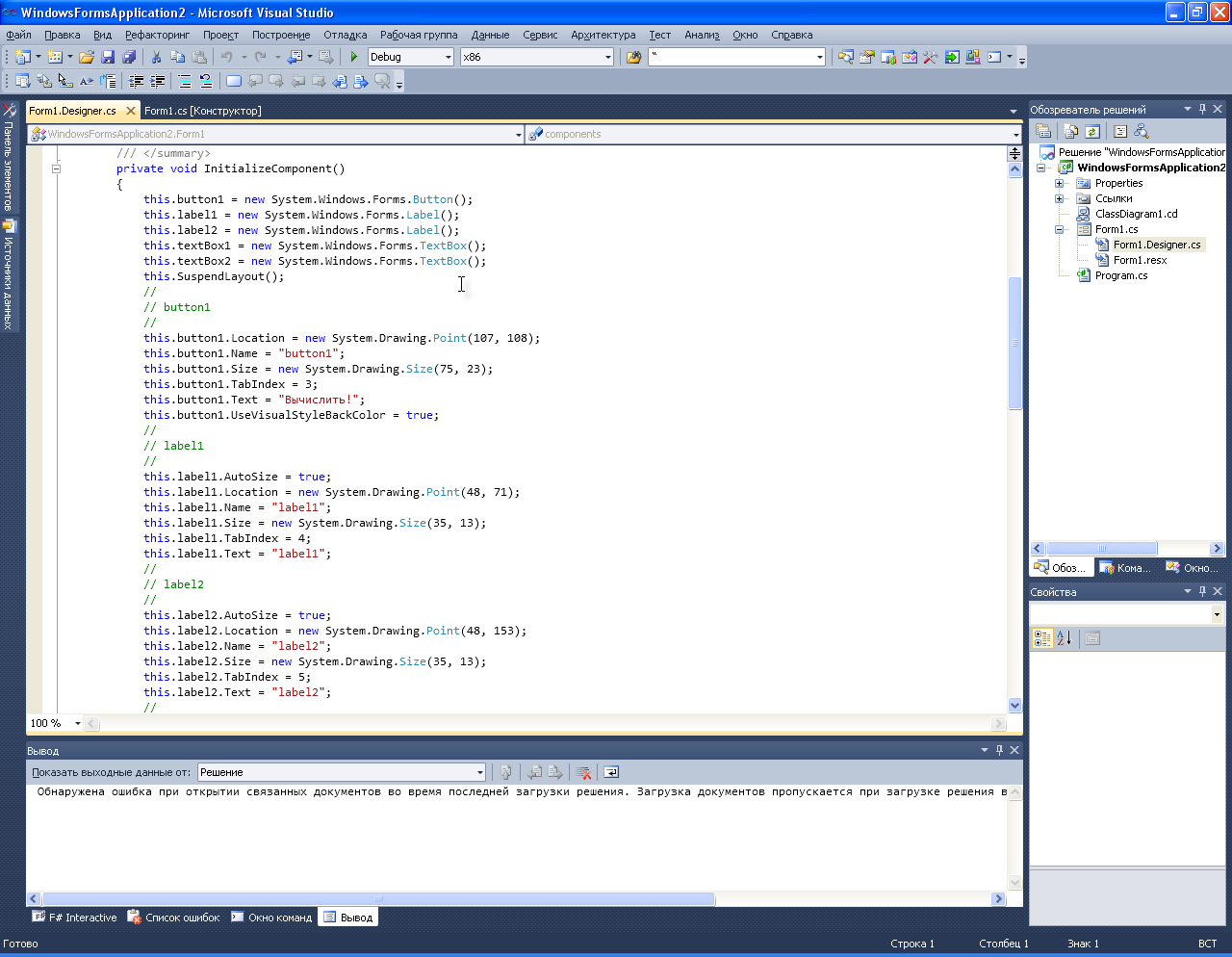


Рисунок 7 – Раздел с кодом C#

Для переноса дизайна формы, необходимо скопировать требуемый раздел в программе на C# и перенести его в программу на F#, внеся соответствующие изменения.

*Обзор основных компонентов и их свойств библиотеки WinForms*

*Классы:*

Form – Представляет окно или диалоговое окно, которое составляет пользовательский интерфейс приложения;

Button – Представляет элемент управления "кнопка" Windows;

MonthCalendar - Представляет элемент управления, который позволяет выбрать дату с помощью визуального календаря;

CheckBox – Представляет элемент управления — флажок, который пользователь может устанавливать и снимать;

ComboBox – Представляет элемент управления, предназначенный для выбора значения, с раскрывающимся списком, который можно отображать и скрывать нажатием принадлежащей этому элементу управления кнопки со стрелкой;

DataGrid – Представляет элемент управления, отображающий данные в настраиваемой сетке;

DatePicker – Представляет элемент управления, позволяющий выбрать дату;

GroupBox – Представляет элемент управления, создающий контейнер с границей и заголовком;

PictureBox – Представляет элемент управления, отображающий изображение;

Label – Предоставляет текстовую метку;

ListBox – Содержит список элементов для выбора;

Menu – Представляет Windows элементы управления меню, позволяющие иерархически организовать элементы, связанные с командами и обработчиками событий;

MenuItem – Представляет выбираемый элемент в составе Menu;

Panel – Предоставляет базовый класс для всех элементов Panel. Элементы Panel используются для размещения и организации дочерних объектов в приложениях;

ProgressBar – Показывает ход выполнения операции;

RadioButton – Представляет переключатель, который пользователь может устанавливать (выбирать), но не снимать (отменять выбор). Очистить свойство можно только программным способом;

RichTextBox – Представляет элемент управления с расширенным редактированием для работы с объектами FlowDocument;

Separator – Элемент управления, используемый для разделения элементов в элементах управления элементами;

TextBox – Представляет элемент управления, который можно использовать для отображения или редактирования неформатированного текста;

TreeView – Представляет элемент управления, отображающий иерархические данные в структуре дерева с элементами, которые могут разворачиваться и сворачиваться;

TrackBar – Представляет элемент управления, ползунок.

*Свойства:*

Height – Возвращает или задает предполагаемую высоту элемента;

Width – Возвращает или задает ширину элемента;

Top – Возвращает или задаёт расстояние (в точка) между верхней границей элемента управления и верхней границей клиентской области;

Left – Возвращает или задаёт расстояние (в точка) между левой границей элемента управления и левой границей клиентской области;

Text – Получает или задает текст, сопоставленный с этим элементом управления;

Enabled – Возвращает или задает значение, показывающее, сможет ли элемент управления отвечать на действия пользователя;

Visible – Получает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления;

Dock – Возвращает или задает границы элемента управления, прикрепленные к его родительскому элементу управления, и определяет способ изменения размеров элемента управления с его родительским элементом управления;

Checked – Возвращает или задает значение, указывающее, находится ли во включенном состоянии;

ReadOnly – Возвращает или задает значение, включающее режим использования только для чтения;

Minimum – Возвращает или задает минимальное возможное значение Value элемента диапазона;

Maximum – Возвращает или задает максимальное возможное значение Value элемента диапазона;

Name – Возвращает или задает идентификационное имя элемента;

Tag – Возвращает или задает произвольное значение объекта, которое может использоваться для хранения специальной информации о данном элементе

Value – Возвращает или задает текущую величину элемента управления диапазона;

FontSize – Получает или задает размер шрифта;

ImageLocation – Получает или задаёт путь или URL адрес изображения;

ShowDialog – Отображает форму в виде модального диалогового окна с текущим активным окном, заданным в качестве владельца;

MaxDate – Получает или задаёт максимально допустимую дату;

MinDate – Получает или задаёт минимально допустимую дату;

Clear – Удаляет весь текст из элемента управления;

SelectAll – Выбирает весть текст в текстовом поле;

Hide – Скрывает элемент управления от пользователя;

MenuItems – Получает значение, показывающее коллекцию объектов связанных с данным меню;

Add – Добавляет указанный элемент у правления в коллекцию элементов управления;

Visual Studio — это набор инструментов разработки, основанных на использовании компонентов, и других технологий для создания мощных, производительных приложений. Кроме того, среда Visual Studio оптимизирована для совместного проектирования, разработки и развертывания корпоративных решений.

Данный продукт позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework и Microsoft Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов цикла разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Так как формы являются основной частью приложения, важно уделять особое внимание их внешнему виду и функциям. В конечном счете, форма представляет собой пустую доску, которую разработчик оснащает элементами управления, формируя интерфейс пользователя, и кодом для управления данными. Для этого Visual Studio обеспечивает интегрированную среду разработки, способствующую написанию кодов, а также расширенный набор элементов управления .NET Framework. Дополняя функциональными возможностями этих элементов управления свои коды, пользователь может легко и быстро разработать необходимое приложение.

**Задание.**

Код программы:

open System

open System.Drawing

open System.Windows.Forms

open System.IO

let Form1 = new Form(Text="Пример формы в F#", Width=400, Height=300, Menu = new MainMenu())

let Button1 = new Button(Text="Change Picture", Width = 80, Height = 50, Top = 190, Left = 150)

let image1 = new PictureBox(SizeMode = PictureBoxSizeMode.AutoSize, Top = 30, Left = 120)

let image2 = new PictureBox(SizeMode = PictureBoxSizeMode.AutoSize, Top = 30, Left = 120)

// Меню бар

let mFile = Form1.Menu.MenuItems.Add("&Файл")

let mForms = Form1.Menu.MenuItems.Add("&Формы")

let mHelp = Form1.Menu.MenuItems.Add("&Помощь")

// Под меню

let miMessage = new MenuItem("&Пример сообщения")

let miSeparator = new MenuItem("-")

let miExit = new MenuItem("&Выход")

let miAbout = new MenuItem("&О программе...")

let miForm1 = new MenuItem("&Форма\_1")

let miForm2 = new MenuItem("&Форма\_2")

let miForm3 = new MenuItem("&Форма\_3")

// Добавление подменю в пункты меню

mFile.MenuItems.Add(miMessage)

mFile.MenuItems.Add(miSeparator)

mFile.MenuItems.Add(miExit)

mHelp.MenuItems.Add(miAbout)

mForms.MenuItems.Add(miForm1)

mForms.MenuItems.Add(miForm2)

mForms.MenuItems.Add(miForm3)

// Добавление картинок

image1.ImageLocation <- "C:/Users/Makesar/source/repos/Lab5\_3\_GolikovAO/sgu.png"

image2.ImageLocation <- "C:/Users/Makesar/source/repos/Lab5\_3\_GolikovAO/kniit.png"

// Добавляем элементы на форму

Form1.Controls.Add(Button1)

Form1.Controls.Add(image1)

Form1.Controls.Add(image2)

image2.Visible <- false

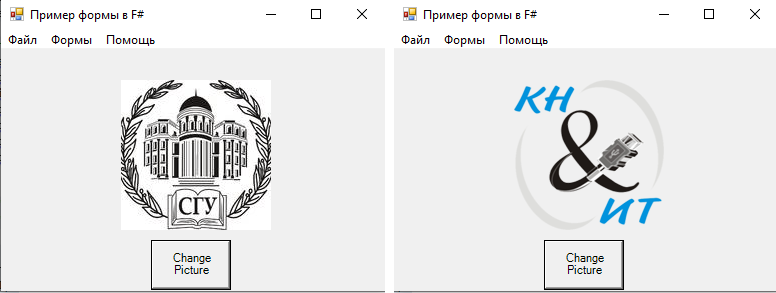
// Добавляем события при клике (чередует картинки при нажатии на кнопку)

Button1.Click.Add(fun i -> if (image1.Visible = false) then image1.Visible <- true else image1.Visible <- false)

Button1.Click.Add(fun i -> if (image2.Visible = false) then image2.Visible <- true else image2.Visible <- false)

do Application.Run(Form1)

Результат выполнения:



**Контрольные вопросы**

1. Основные компоненты библиотеки WinForms?

Классы и свойства. В конечном счете, форма представляет собой пустую доску, которую разработчик оснащает элементами управления, формируя интерфейс пользователя, и кодом для управления данными.

1. Как осуществляется добавление компонентов на форму?

Для добавления компонента на форму нужно указать форму на которую будет добавлен компонент. Form.Controls.Add(Component)

1. Какие ссылки необходимо добавить в проект для работы с библиотекой WinForms?

System.Windows.Forms.dll и System.Drawing

1. Какое свойство получает или задаёт путь или URL адрес изображения?

ImageLocation – получает или задаёт путь или URL адрес изображения

1. Существует ли визуальный дизайнер форм в F#?

В Visual Studio F# есть компоненты, как и в других языках программирования, существенная разница заключается в том, что в F# нет визуального конструктора форм. Для их размещения на форму нужно прописывать всё вручную.

1. Какие свойства отвечают за положения объекта на форме?

Top – Возвращает или задаёт расстояние (в точка) между верхней границей элемента управления и верхней границей клиентской области.

Left – Возвращает или задаёт расстояние (в точка) между левой границей элемента управления и левой границей клиентской области.

1. Как изменить консольное приложение на Windows приложение?

После создания проекта, изменяем его тип через Проект –> Свойства, меняя в типе выходных данных *Консольное приложение* на *Windows-приложение*, сохраняем изменения и закрываем вкладку.

1. Назовите основные свойства, которые использует кнопка?

Height – Возвращает или задает предполагаемую высоту элемента;

Width – Возвращает или задает ширину элемента;

Top – Возвращает или задаёт расстояние (в точка) между верхней границей элемента управления и верхней границей клиентской области;

Left – Возвращает или задаёт расстояние (в точка) между левой границей элемента управления и левой границей клиентской области;

Text – Получает или задает текст, сопоставленный с этим элементом управления;

Enabled – Возвращает или задает значение, показывающее, сможет ли элемент управления отвечать на действия пользователя;

Visible – Получает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления.

1. Как добавляется событие на нажатие кнопки?

Button.Click.Add(fun i -> image.Visible <- false)

**Вывод:** ознакомится с понятием списков в F#, консольного приложения в F# и графическим интерфейсом. Научился применять списки на примере нескольких тренировочных программ, а так же создавать консольное приложения на F# и графический интерфейс с добавлением объектов, обрабатывать события с применением библиотеки WinForms.