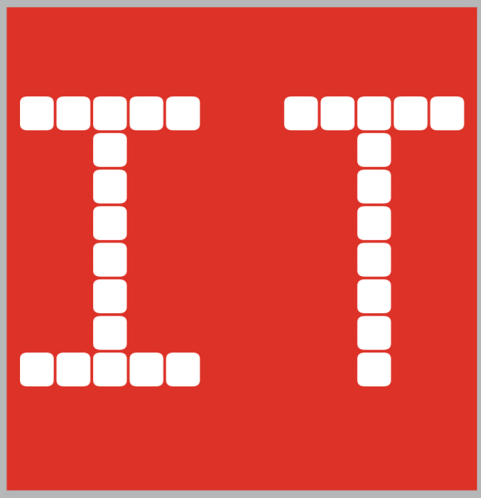
Р.И.БАЖЕНОВ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ФУНКЦИОНАЛЬНОМУ ПРОГРАММИРОВАНИЮ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ



DirectMEDIA

Р. И. Баженов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ФУНКЦИОНАЛЬНОМУ ПРОГРАММИРОВАНИЮ

Учебно-методическое пособие

Утверждено Редакционно-издательским советом университета в качестве учебно-методического пособия для студентов, обучающихся по направлениям подготовки 01.03.02 Прикладная математика и информатика, 09.03.02 Информационные системы и технологии, 09.04.02 Информационные системы и технологии, 09.03.03 Прикладная информатика, 44.03.05 Педагогическое образование (направленность Информатика и математика)

DirectMEDIA

Москва
Берлин
2017

УДК 004(075)
ББК 32.97я7
Б16

Рецензенты:
Векслер В. А., кандидат педагогических наук, доцент, доцент кафедры информационных систем и технологий обучения, Саратовский государственный университет им. Н. Г. Чернышевского;
Штепа Ю. П., кандидат педагогических наук, доцент, доцент кафедры информационных систем, математики и методики обучения, Приамурский государственный университет им. Шолом-Алейхема

Баженов, Р. И.

Б16 Лабораторный практикум по функциональному программированию : учебно-методическое пособие / Р. И. Баженов. — М. ; Берлин : Директ-Медиа, 2017. — 90 с.

ISBN 978-5-4475-9458-9

В учебно-методическом пособии рассматриваются различные приемы разработки простых приложений на языке функционального программирования F#. Представлены темы функций, циклов, кортежей, списков. Выделены работы с библиотекой WinForms и WPF. Приведены примеры программирования с использованием библиотеки WinForms кортежей, массивов, списков.

Пособие адресовано студентам направлений 01.03.02 — Прикладная математика и информатика, 09.03.02 — Информационные системы и технологии, 09.04.02 — Информационные системы и технологии, 09.03.03 — Прикладная информатика, 44.03.05 — Педагогическое образование (направленность Информатика и математика), может быть использовано при изучении одноименного курса (курса по выбору) или в курсах «Функциональное и логическое программирование», «Интеллектуальные системы и технологии».

Текст печатается в авторской редакции.

УДК 004(075)
ББК 32.97я7

ISBN 978-5-4475-9458-9

Введение

В настоящее время произошло возрождение функциональных языков программирования. Компания Microsoft обратила внимание на функциональную парадигму и выпустила на рынок язык F#, который в данный момент интегрирован в платформу Visual Studio 201X. Программы на языке F# получаются компактными, простыми для понимания и потому актуально изучать такой язык функционального программирования, стоящий в ряду LISP, CLIPS, Haskell.

Предлагаемое пособие представляет основные простые приемы функционального программирования. К каждому подобранному теоретическому материалу прилагается его проработка на лабораторных занятиях.

Пособие адресовано студентам направлений 01.03.02 — Прикладная математика и информатика, 09.03.02 — Информационные системы и технологии, 09.04.02 — Информационные системы и технологии, 09.03.03 — Прикладная информатика, 44.03.05 — Педагогическое образование (направленность Информатика и математика), может быть использовано при изучении одноименного курса (курса по выбору) или в курсах «Функциональное и логическое программирование», «Интеллектуальные системы и технологии».

В изложении материала автор опирался на собственный опыт преподавания в Приамурском государственном университете им. Шолом-Алейхема.

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
```

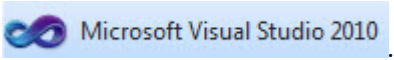
Результат равен 202.

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

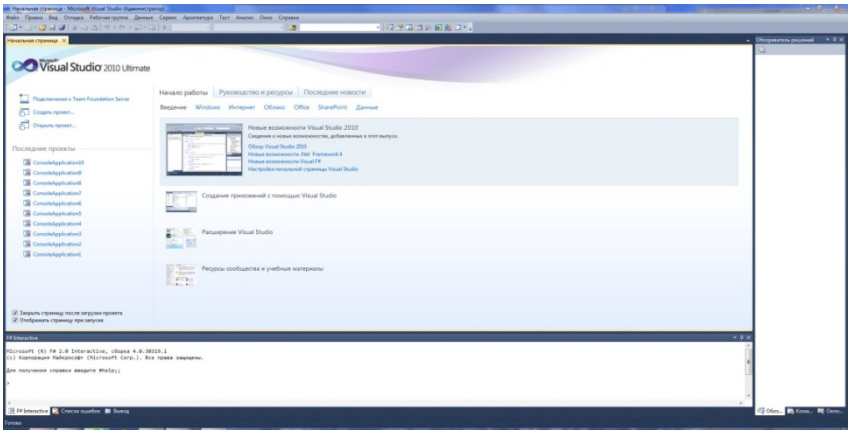


Рисунок 1.1

Выберите вкладку создания проекта

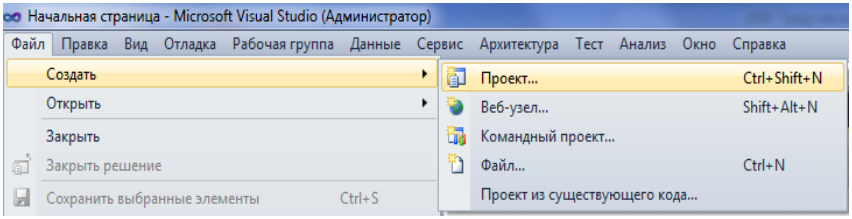


Рисунок 1.2

При создании проекта выберите пункт «Приложение F#»

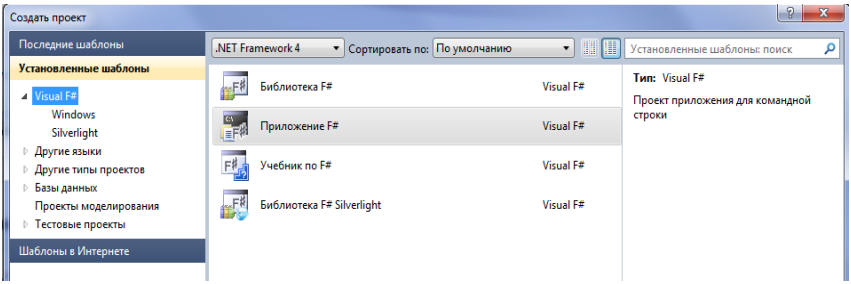


Рисунок 1.3

Откроется окно проекта для написания программы

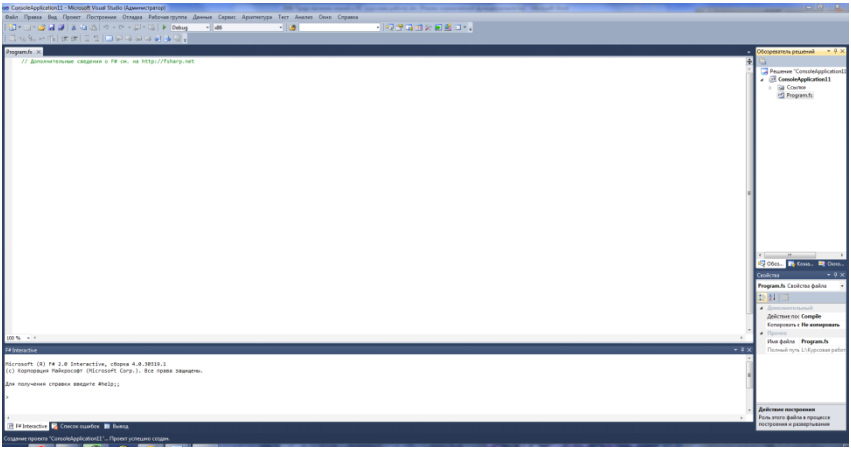


Рисунок 1.4

Вводим код программмы

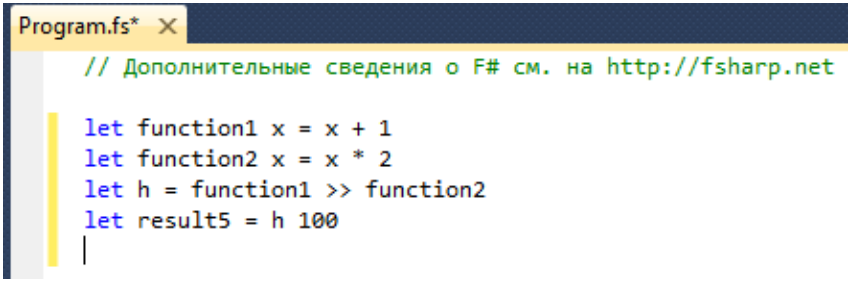


Рисунок 1.5

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

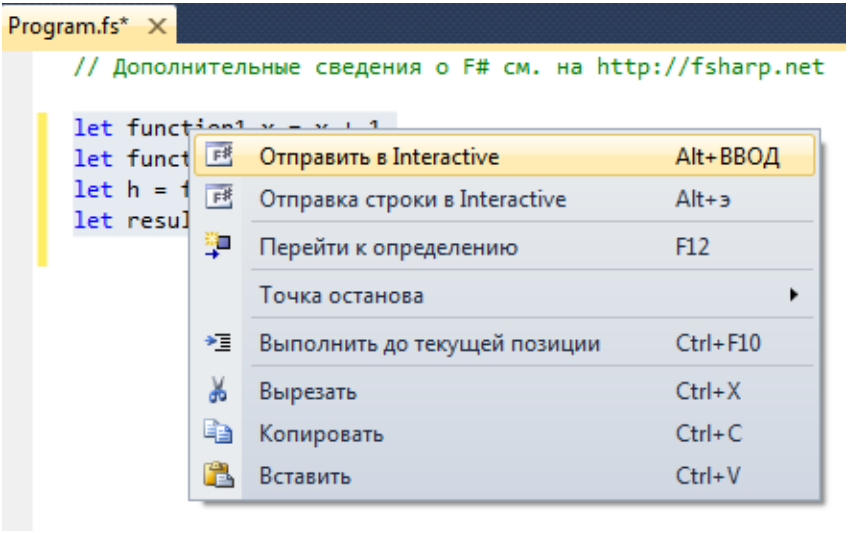


Рисунок 1.6

Смотрим результат выполнения программы в окне F# Interactive.

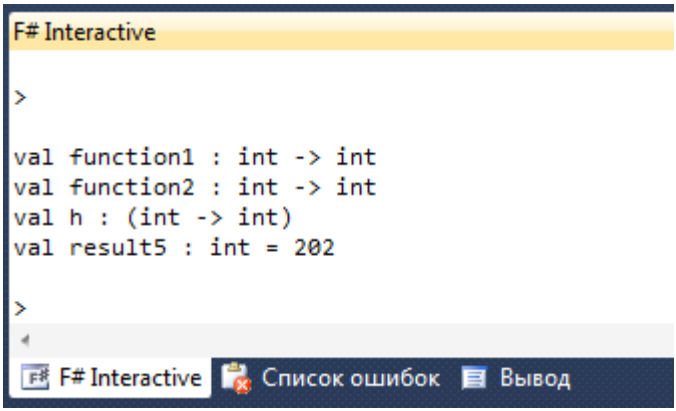


Рисунок 1.7

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?

Лабораторная работа № 2. «Циклы»

Цель работы: ознакомиться с понятием циклов в F#, на примере нескольких тренировочных задач закрепить знания.

Теоретические сведения

Выражение for...to

Выражение *for...to* используется для прохождения в цикле диапазона значений переменной цикла.

```
for identifier = start [ to | downto ] finish do
    body-expression
```

Несмотря на то, что технически *for...to* является выражением, оно больше похоже на традиционный оператор в императивном языке программирования. В следующих примерах показаны различные варианты использования выражения *for...to*.

```
let function1() =
  for i = 1 to 10 do
    printf «%d» i
  printfn «»

let function2() =
  for i = 10 downto 1 do
    printf «%d» i
  printfn «»

function1()
function2()

let beginning x y = x – 2*y
let ending x y = x + 2*y

let function3 x y =
  for i = (beginning x y) to (ending x y) do
    printf «%d» i
  printfn «»

function3 10 4
```

Результат выполнения приведенного кода будет следующим:

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

Выражение for...in

Эта конструкция для организации циклов используется для перебора в перечислимой коллекции при соответствии шаблона, например в выражении диапазона, в последовательности, в списке, в массиве или в другой конструкции, поддерживающей перечисление.

```
for pattern in enumerable-expression do
  body-expression
```

Перечислимое выражение можно задать в виде перечислимой коллекции или с помощью оператора .. в виде диапазона целочисленных значений. К перечислимым коллекциям относятся списки, последовательности, массивы, наборы, сопоставления и т. д.

При определении диапазона с помощью оператора .. можно использовать следующий синтаксис.

```
start .. finish
```

Можно также использовать версию, в которой инкремент имеет значение пропуска, как показано в следующем примере.

```
start .. skip .. finish
```

Если в качестве шаблона в целочисленном диапазоне используется простая переменная-счетчик, то обычно счетчик при каждой итерации увеличивается на 1, но если диапазон содержит значение пропуска, то счетчик будет увеличиваться на это значение.

Значения, соответствующие шаблону, также можно использовать в теле цикла.

В следующих примерах кода показано использование выражения *for...in*.

```
let list1 = [ 1; 5; 100; 450; 788 ]
for i in list1 do
  printfn «%d» i
```

Выходные данные выглядят следующим образом.

```
1
5
100
450
788
```

В следующем примере показано применение цикла к последовательности и использование шаблона в виде кортежа вместо обычной переменной.

```
let seq1 = seq { for i in 1 .. 10 -> (i, i*i) }
for (a, asqr) in seq1 do
  printfn «%d squared is %d» a asqr
```

Выходные данные выглядят следующим образом.

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
10 squared is 100
```

В следующем примере показано применение цикла к простому целочисленному диапазону.

```
let function1() =
  for i in 1 .. 10 do
    printf «%d» i
 printfn «»
function1()
```

Выходные данные функции *function1* выглядят следующим образом.

```
1 2 3 4 5 6 7 8 9 10
```

В следующем примере показано применение цикла со значением пропуска 2, в результате чего выбирается каждый второй элемент диапазона.

```
let function2() =
 for i in 1 .. 2 .. 10 do
   printf «%d» i
 printfn «»
function2()
```

Выходные данные функции *function2* выглядят следующим образом:

```
1 3 5 7 9
```

В следующем примере показано использование диапазона знаков.

```
let function3() =
  for c in 'a' .. 'z' do
   printf «%c» c
 printfn «»
function3()
```

Выходные данные функции *function3* выглядят следующим образом:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

В следующем примере показано использование отрицательного значения пропуска для перебора в обратном порядке.

```
let function4() =
 for i in 10 .. –1 .. 1 do
  printf «%d» i
 printfn « ... Lift off!»
function4()
let function4() =
 for i in 10 ..–1 ..1 do
  printf «%d» i
```

```
 printfn «»
function4()
```

Выходные данные функции *function4* выглядят следующим образом.

```
10 9 8 7 6 5 4 3 2 1 ...Lift off!
```

Начало и конец диапазона также могут представлять собой выражения, например функции, как показано в следующем примере кода.

```
let beginning x y = x – 2*y
let ending x y = x + 2*y

let function5 x y =
 for i in (beginning x y) .. (ending x y) do
  printf «%d» i
printfn «»

function5 10 4
```

Выходные данные функции *function5* со входными значениями выглядят следующим образом:

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

В следующем примере показано использование подстановочного знака (_), если элемент не требуется использовать в цикле.

```
let mutable count = 0
 for _ in list1 do
  count <- count + 1
 printfn «Number of elements in list1: %d» count
```

Выходные данные выглядят следующим образом.

```
Number of elements in list1: 5
```

Циклы: выражение while...do (F#)

Выражение *while...do* используется для выполнения итерации (в цикле), пока заданное проверяемое условие истинно.

```
while test-expression do
  body-expression
```

Проверяется истинность условия *test-expression*. Если результат проверки — значение *true*, выполняется выражение *body-expression* и снова проверяется истинность условия. Если результат проверки истинности условия — значение *false*, итерация заканчивается.

В следующем примере демонстрируется использование выражения while...do.

```
open System

let lookForValue value maxValue =
 let mutable continueLooping = true
 let randomNumberGenerator = new Random()
  while continueLooping do
  // Generate a random number between 1 and maxValue.
  let rand = randomNumberGenerator.Next(maxValue)
  printf «%d» rand
  if rand = value then
   printfn «\nFound a %d!» value
   continueLooping <- false

lookForValue 10 20
```

Выходные данные приведенного выше кода — серия случайных чисел от 1 до 20, последнее из которых — 10.

```
13 19 8 18 16 2 10
Found a 10!
```

Практическая работа

Запустите Visual Studio.

Создайте приложение на F#.

Введите код программы, которая будет считать квадраты чисел от 1 до 15.

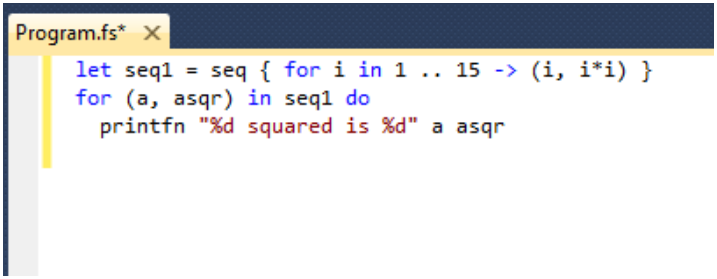


Рисунок 2.1

Запустите программму на выполнение в интерактивном режиме.

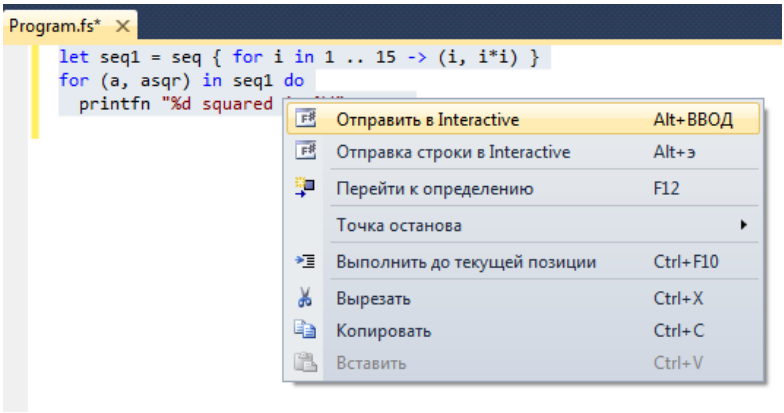


Рисунок 2.2

Получаем результат:

1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81

```
10 squared is 100
11 squared is 121
12 squared is 144
13 squared is 169
14 squared is 196
15 squared is 225
```

Проанализируйте результат и выполните задачи.

Задания

1. Все числа от 5 до 17 возведите в квадрат и от полученных значений отнимите единицу.
2. Выведите все четные числа от 10 до 20, предварительно возведя их в квадрат.
3. Создайте программу, которая выводила бы произвольные числа от 1 до 27 до тех пор, пока она не выдаст число 17.
4. Какие примут значения *x* и *y*, если *x=2*(i+1)*, $y=3i^2$, где *i* изменяется от 1 до 3?
5. Напишите программу, которая выводила бы квадраты и кубы
6. Выведите все заглавные буквы русского алфавита.
7. Выведите все числа в промежутке между *(x + y)*2* и *(x * * y)*3*, если *x=3, y=2*.
8. Числа от 100 до 80 увеличить на 5 и найти квадрат.
9. Все числа от 1 до 50 через каждые 4 , умножить на 8.
10. Выведите все целые значения функции $y = x^2 + 2x - 3$ при *x*, принимающем значения от 1 до 10.

Контрольные вопросы

1. Для чего используется выражение for...to?
2. Для чего используется выражение for...in?
3. Для чего используется выражение while...do?
4. Опишите принцип работы выражения while...do.
5. Каким образом обозначается шаг в выражении for...in?
6. Как можно выразить начало и конец диапазона в выражении for...in?
7. Перечислите варианты использования выражения for... to.

Лабораторная работа № 3. «Кортежи»

Цель работы: ознакомиться с понятием кортежа и на основе нескольких тренировочных программ научиться использовать кортежи в программировании на F#.

Теоретические сведения

Кортеж — это группировка неименованных, но упорядоченных значений, возможно, различных типов.

Примерами кортежей являются пары, тройки и т. п. одного и того же или разных типов.

Некоторые примеры показаны в следующем коде.

```
// Tuple of two integers.
(1, 2)
// Triple of strings.
(«one», «two», «three» )
// Tuple of unknown types.
(a, b)
// Tuple that has mixed types.
(«one», «1», «2.0»)
// Tuple of integer expressions.
(a + 1, b + 1)
```

Для доступа к элементам кортежа и присвоения им имен можно использовать подбор шаблона, как показано в следующем коде.

```
let print tuple1 =
  match tuple1 with
    | (a, b) -> printfn «Pair %A %A» a b
```

Шаблоны кортежей можно использовать в привязках *let* следующим образом.

```
let (a, b) = (1, 2)
```

Здесь привязываются одновременно значения *a* и *b*. Если требуется только один элемент кортежа, можно использовать

подстановочный знак (символ подчеркивания) во избежание создания нового имени для ненужной переменной.

```
let (a, _) = (1, 2)
```

Функции *fst* и *snd* возвращают первый и второй элементы кортежа соответственно.

```
let c = fst (1, 2)
let d = snd (1, 2)
```

Встроенной функции, возвращающей третий элемент тройки, нет, однако ее легко написать следующим образом.

```
let third (_, _, c) = c
```

В общем случае для доступа к отдельным элементам кортежа лучше использовать подбор шаблона.

Использование кортежей

Кортежи — это удобный способ возвращения из функции сразу нескольких значений, как показано в следующем примере. В этом примере выполняется целочисленное деление и возвращается округленный результат операции в качестве первого члена кортежной пары и остаток в качестве второго члена пары.

```
let divRem a b =
    let x = a / b
    let y = a % b
    (x, y)
```

Кортежи также можно использовать в качестве аргументов функций, когда требуется избежать неявного каррирования аргументов функции, что подразумевается обычным синтаксисом функций.

```
let sumNoCurry (a, b) = a + b
```

Обычный синтаксис определения функции *let sum a b = a + b* позволяет определить функцию, являющуюся частичным

применением первого аргумента функции, как показано в следующем коде.

```
let addTen = sum 10
let result = addTen 95
// Result is 105.
```

Использование кортежа в качестве параметра предотвращает каррирование.

Имена кортежных типов

При записи имени типа, который является кортежем, для разделения элементов используется символ *. В случае кортежа, состоящего из int, float и string, такого как (10, 10.0, «ten»), тип будет записан следующим образом.

```
int * float * string
```

Практическая работа

Запустите Visual Studio.

Создайте приложение F#.

Введите программу, в ходе которой выполняется целочисленное деление и возвращается округленный результат операции в качестве первого члена кортежной пары и остаток в качестве второго члена пары.

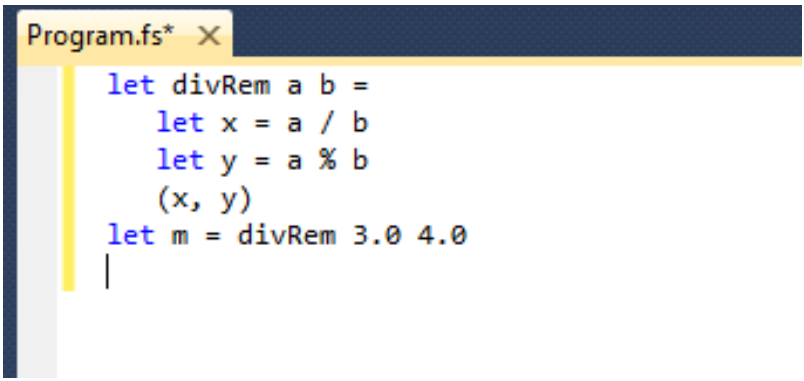


Рисунок 3.1.

Запустите программу в интерактивном режиме выполнения.

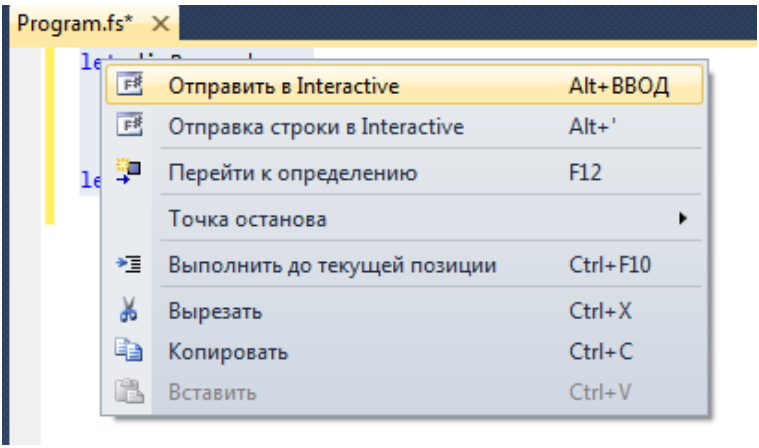


Рисунок 3.2.

Проанализируйте данные рис. 3.3.

```
>

val divRem : float -> float -> float * float
val m : float * float = (0.75, 3.0)

>
```

Рисунок 3.3

Задания

1. В магазине продают одну пару босоножек, 2 пары туфель и 3 пары сапог. Сколько всего продают единиц обуви и сколько из них можно носить летом? Вывести ответ в виде кортежа.
2. Первая лягушка поймала 10 мух, вторая — 15 мух, а третья — 30. Определить коэффициент везения каждой из лягушек, приняв коэффициент везения первой лягушки за единицу.
3. На новогодней елке у детей было 100 конфет, 4 торта и 50 мандаринов. Дед мороз принес 50 конфет, 2 торта и 30 мандаринов. Сколько теперь у детей конфет, тортов и мандаринов?

4. Найдите длину отрезка, если координаты его точек (1, 2) и (3, 4).

5. Найдите длину радиус-вектора с координатами конца (3, 4).

6. Первый купец занял для торговли бойкое место, и его дневной доход составляет 5 золотых. Второй купец занял менее бойкое место, и его доход за день составляет 3,5 золотых. Пошлина за бойкое торговое место равна 20% дохода, а за место менее людное — 5%. Сколько составляет выручка каждого из купцов за день?

7. У тигра 20 темных полос, у зебры — 25 темных полос, а у кота Тимоши — 5. Кот Тимоша в длину вместе с хвостом составляет 40 см, тигр — 200 см, а зебра — 170 см. Определить полосатость зверей.

8. Морская свинка съела 3 капустных листа, полторы морковки и полмандарина. Сколько овощей и сколько фруктов съела свинка?

9. Ночью за 6 часов Вася должен подготовиться к экзамену. После чашки кофе Вася не спит 1 час, а после банки энергетика — 3 часа. Сколько Васе нужно чашек кофе или банок энергетика, чтобы не спать?

10. Съел царь-батюшка молодильное яблочко и помолодел на 10 лет. Съел еще 2 яблока и помолодел на 20 лет. А потом еще 3 яблока — и помолодел еще на 30 лет. На сколько лет помолодел царь-батюшка?

Контрольные вопросы

1. Дайте определение кортежу.
2. Чем удобно использование кортежей?
3. С помощью каких функций возвращаются элементы кортежей?
4. Каким образом задаётся кортеж?
5. Каким образом определяются имена кортежных типов?
6. Как можно осуществить доступ к элементам кортежа?
7. Каких типов являются элементы кортежей?

Лабораторная работа № 4. «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]
```

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	'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 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 = 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.mapi* и *List.mapi2*, 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*.

```
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*

```
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.

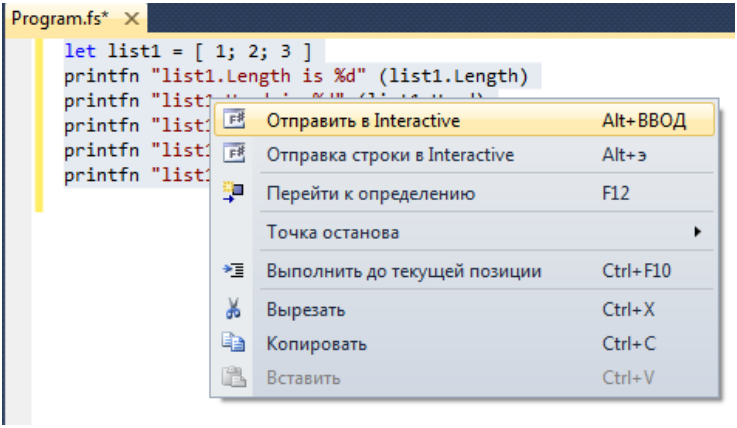


Рисунок 4.2.

Проанализируйте результат выполнения программы.

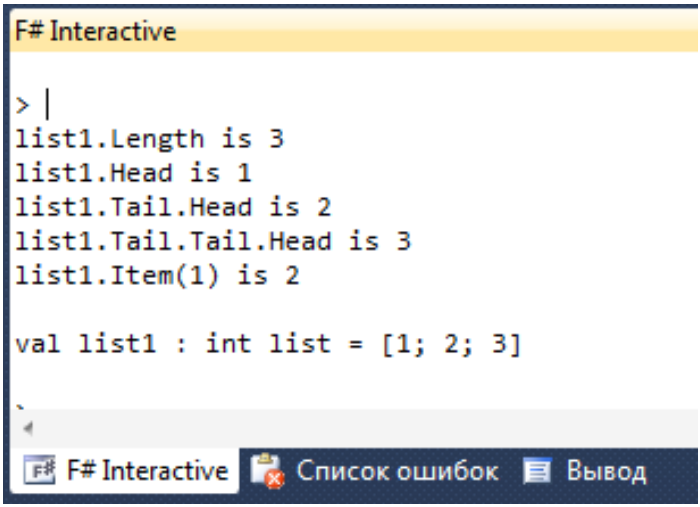


Рисунок 4.3.

Задания

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

3. Перечислите функции сортировки списков.

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

5. Дайте определение кортежам.

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

7. Приведите способы объединения списков.

Лабораторная работа № 5.
«Библиотека WinForms»

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

Практическая работа

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;
- в Visual Studio 2010 для F# отсутствует визуальный дизайнер форм.

Разработаем простую программу, состоящую из главной формы и трех дочерних форм, с помощью библиотеки WinForms.

Запускаем Visual Studio 2010 и создаем проект — Приложение F#, проект приложения для командной строки (рис. 5.1).

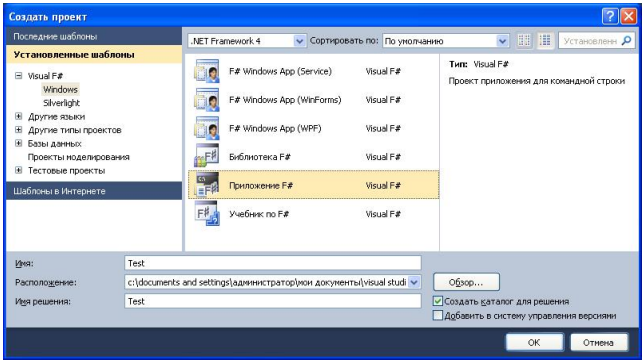


Рисунок 5.1. Создание проекта

После создания проекта, изменяем его тип через Проект -> Свойства, меняя в типе выходных данных Консольное приложение на Windows-приложение (рис. 5.2), сохраняем изменения и закрываем вкладку.

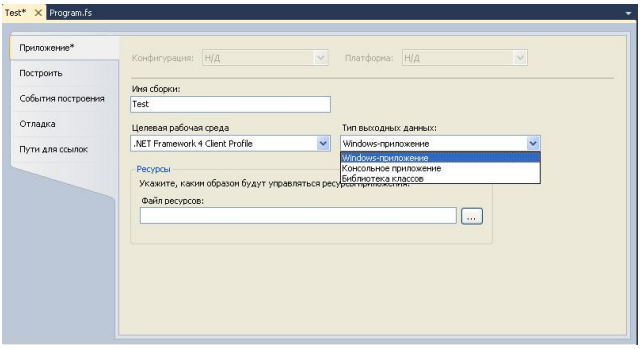


Рисунок 5.2. Изменение типа приложения

Теперь необходимо добавить в проект ссылки на требуемые библиотеки: System.Windows.Forms, System.Drawing (рис. 5.3). Это осуществляется через контекстное меню на пункте Ссылки в Обозревателе решений.

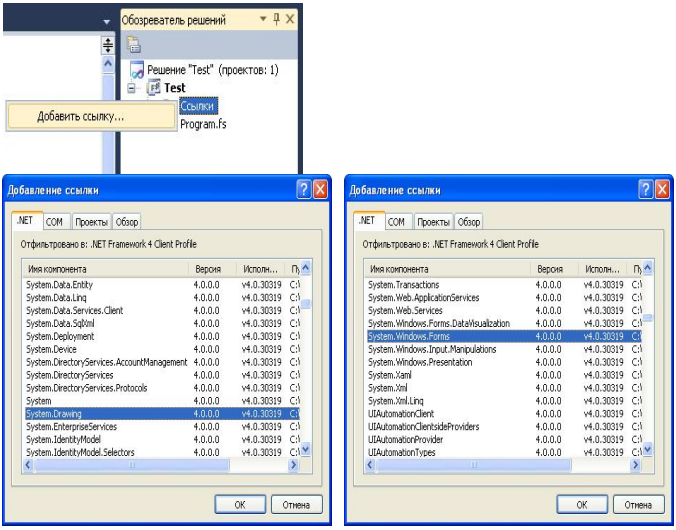


Рисунок 5.3. Добавление ссылок в проект

Для качественного прототипирования формы можно воспользоваться дизайнером форм от C# (рис. 5.4).

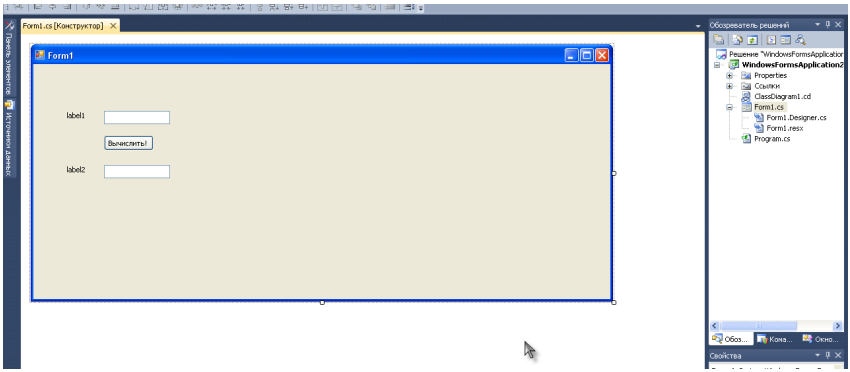


Рисунок 5.4. Редактор форм от C#

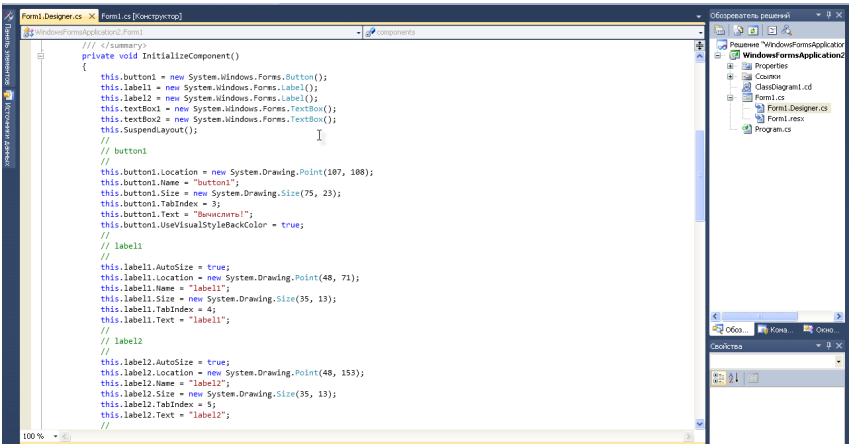


Рисунок 5.5. Раздел с кодом C#

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

Теперь внесем текст программы. Создадим главную форму, на которой поместим меню, три рисунка и надпись.

```
open System
open System.Drawing
open System.IO
open System.Windows.Forms

let form = new Form(Width= 400, Height = 300, Text = «F# Главная форма», Menu = new MainMenu())

// Меню бар
let mFile = form.Menu.MenuItems.Add(«&Файл»)
let mForms = form.Menu.MenuItems.Add(«&Формы»)
let mHelp = form.Menu.MenuItems.Add(«&Помощь»)

// Подменю
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)

// Создаем картинки и задаем им свойства
let image1 = new PictureBox(SizeMode = PictureBoxSizeMode.AutoSize, Top = 5)
let image2 = new PictureBox(SizeMode = PictureBoxSizeMode.AutoSize, Top = 5, Left = 133)
let image3 = new PictureBox(SizeMode = PictureBoxSizeMode.AutoSize, Top = 5, Left = 266)
```

Скопируем картинки в папку с программой здесь это: 1.png, 2.phg, 3.png.

```
// Указываем местоположение картинок
image1.ImageLocation <- «1.PNG»
```

```
image2.ImageLocation <- «2.PNG»
image3.ImageLocation <- «3.PNG»

// Добавляем картинки на форму
form.Controls.Add(image1)
form.Controls.Add(image2)
form.Controls.Add(image3)
// Создаём и добавляем надпись на форму
let Label1 = new Label(Text=«Пример вывода изображений на экран», Top=150)
Label1.Width <- form.Width
Label1.Left <- 80
form.Controls.Add(Label1)

// Запускаем форму
do Application.Run(form)
```

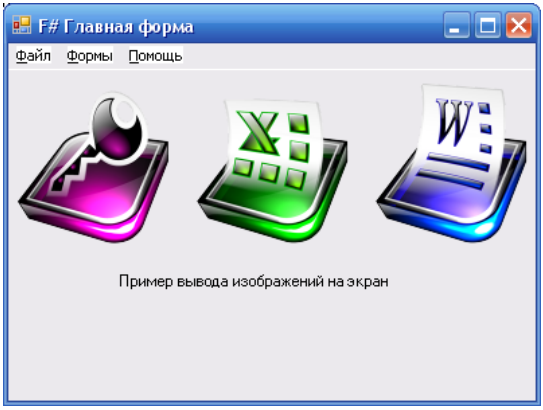


Рисунок 5.6. Вид главной формы

Внесем дополнения в программу. При нажатии на «Файл -> Пример сообщения» будет отображаться сообщение (см. рис. 13), а при нажатии на «Файл -> Выход» приложение будет закрываться.

```
// Создание сообщения
let Msg _ = MessageBox.Show(«Пример сообщения в F#!», «Сообщение») |> ignore
// Событие на нажатие пункта меню
```

```
let _ = miMessage.Click.Add(Msg)
// Закрытие приложения
let Exit _ = form.Close ()
let _ = miExit.Click.Add(Exit)
```

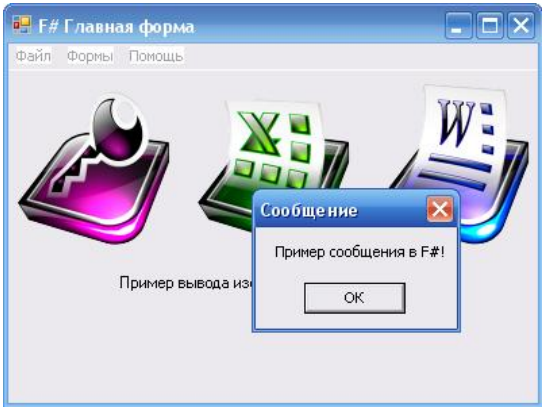


Рисунок 5.7. Пример сообщения

Создадим ещё три дополнительных формы и организуем переход через меню (см. рис. 14).

```
//Форма_1
let Form1 = new Form(Text = «Дочерняя форма №1» ,Width = 400, Height = 300)
let Edit1 = new TextBox(Text=«10»)
let Edit2 = new TextBox(Top=20, Text=«5»)
let button1 = new Button(Text=«+», Top=50, Width=25, Height=25)
let button2 = new Button(Text=«-», Top=50, Left=30, Width=25, Height=25)
let button3 = new Button(Text=«*», Top=50, Left=60, Width=25, Height=25)
let button4 = new Button(Text=«/», Top=50, Left=90, Width=25, Height=25)
Form1.Controls.Add(Edit1)
Form1.Controls.Add(Edit2)
Form1.Controls.Add(button1)
Form1.Controls.Add(button2)
Form1.Controls.Add(button3)
Form1.Controls.Add(button4)
let Summ_ = MessageBox.Show(string(int(Edit1.Text)+ (int(Edit2.Text))), «Сумма») |>ignore
let Minus_ = MessageBox.Show(string(int(Edit1.Text) –(int(Edit2.Text))), «Разность») |>ignore
```

```
let Umnoj_ = MessageBox.Show(string(int(Edit1.Text) * (int(Edit2.Text))), «Умножение») |>ignore
let Del_ = MessageBox.Show(string(int(Edit1.Text)/ (int(Edit2.Text))), «Деление») |>ignore

let _ = button1.Click.Add(Summ)
let _ = button2.Click.Add(Minus)
let _ = button3.Click.Add(Umnoj)
let _ = button4.Click.Add(Del)

//Форма_2
let Form2 = new Form(Width= 400, Height = 300, Text = «Дочерняя форма №2»)
let ProgressBar1 = new ProgressBar(Dock=DockStyle.Top)
let ScrollBar1 = new TrackBar(Top = 50, Maximum = 100, Width = 400)
let Button1 = new Button(Dock=DockStyle.Bottom, Text=«Перейти на форму 3»)
Form2.Controls.Add(ProgressBar1)
Form2.Controls.Add(ScrollBar1)
Form2.Controls.Add(Button1)
let Change _ = ProgressBar1.Value <- ScrollBar1.Value
let _ = ScrollBar1.ValueChanged.Add(Change)
//Форма_3
let Form3 = new Form(Text = «Дочерняя форма №3» ,Width = 400, Height = 300)
let Cal = new MonthCalendar()
let Button2 = new Button(Dock=DockStyle.Bottom, Text=«Нажми меня»)
Form3.Controls.Add(Cal)
Form3.Controls.Add(Button2)
let MsgDay_ = MessageBox.Show («Сегодня «+string DateTime.Today.Now, «Дата») |>ignore
let _ = Button2.Click.Add(MsgDay)
//Вызов форм
let opForm1 _ = do Form1.ShowDialog()
let opForm2 _ = do Form2.ShowDialog()
let opForm3 _ = do Form3.ShowDialog()
//
let _ = miForm1.Click.Add(opForm1)
let _ = miForm2.Click.Add(opForm2)
let _ = miForm3.Click.Add(opForm3)
let _ = Button1.Click.Add(opForm3)
```

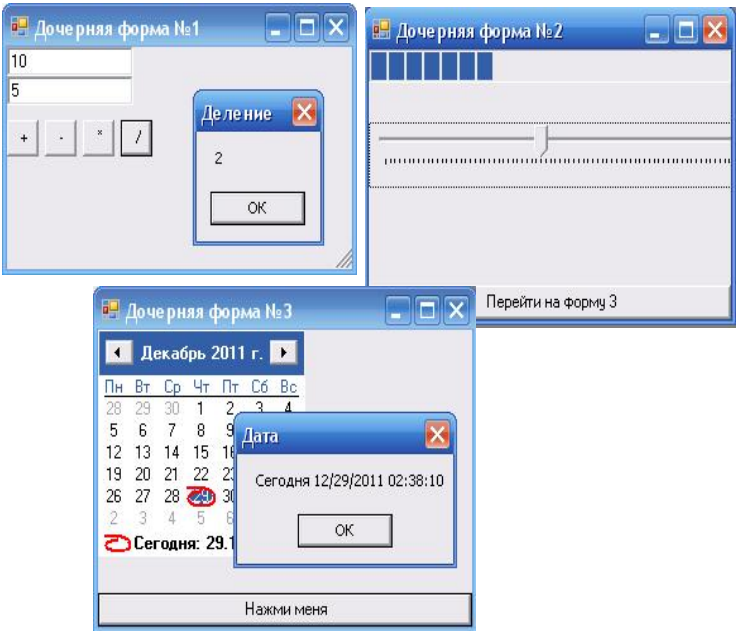


Рисунок 5.8. Дополнительные формы

Задания

1. Разработать программу, состоящую из трех форм. На главной форме находится меню (MainMenu), через которое осуществляется переход на другие формы.
2. Разработать программу решающую квадратное уравнение, состоящую из одной формы. Ответ выводится в RichTextBox.
3. Разработать программу, реализующую простые вычисление над двумя действительными числами (сложения, вычитания, деления, умножения). Учесть все возможные ошибки.
4. Разработать программу, состоящую из главной формы, рисунка и одной кнопки. По нажатию на кнопку меняется рисунок.
5. Разработать программу, которая по выбору месяца выдавала сообщение со временем года. Главная форма, а на ней выпадающий список с месяцами, сообщение появляется при нажатии на кнопку.

6. Разработать приложение, состоящее из одной формы, на которой размещены кнопка и ползунок. При изменении положения ползунка меняется ширина кнопки.

7. Разработать программу, состоящую из главной формы на которой размещены два флажка и одна кнопка. По нажатию на кнопку появляется сообщение «установлен первый флажок», «установлен второй флажок», «установлены оба флажка».

8. Разработать программу, состоящую из одной формы, на которой размещены: поле для ввода, кнопка и список элементов. После того как ввели текст в поле ввода нажатием на кнопку он добавляется в список.

9. Разработать программу, на главной форме разместить поле для ввода и индикатор хода загрузки. По мере ввода текста в поле ввода заполняется индикатор.

10. Разработать приложение, вычисляющее площадь прямоугольника.

Контрольные вопросы

1. Сколько существует способов разработки форм?
2. Основные компоненты библиотеки WinForms?
3. Как осуществляется добавление компонентов на форму?
4. Какие ссылки необходимо добавить в проект для работы с библиотекой WinForms?
5. Какое свойство получает или задаёт путь или URL адрес изображения?
6. Существует ли визуальный дизайнер форм в F#?
7. Какие свойства отвечают за положения объекта на форме?
8. Как изменить консольное приложение на Windows приложение?
9. Назовите основные свойства, которые использует кнопка?
10. Как добавляется событие на нажатие кнопки?

Лабораторная работа № 6. «Библиотека WPF»

Цель работы: Научиться создавать графический интерфейс, добавлять объекты и обрабатывать события с применением библиотеки WPF.

Практическая работа

Разработаем простую программу, состоящую из главной формы и дочерней формы, с помощью библиотеки WPF.

Запускаем Visual Studio 2010 и создаем проект — Приложение F#, проект приложения для командной строки (рис. 6.1).

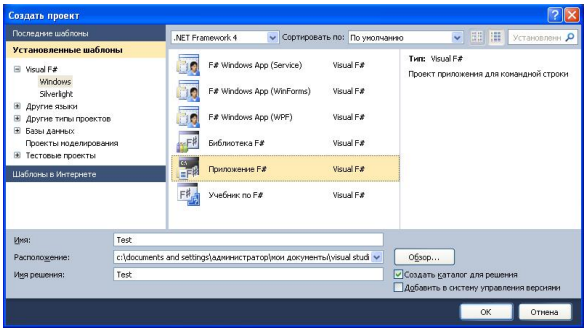


Рисунок 6.1. Создание проекта

После создания проекта, изменяем его тип через Проект -> Свойства, меняя в типе выходных данных Консольное приложение на Windows-приложение (рис. 6.2), сохраняем изменения и закрываем вкладку.

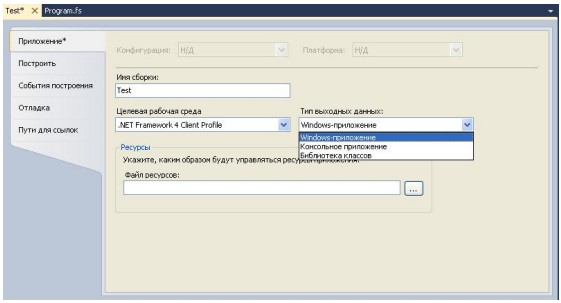


Рисунок 6.2. Изменение типа приложения

Теперь необходимо добавить в проект ссылки на требуемые библиотеки: WindowsBase, PresentationCore, PresentationFramework, System.Xaml (рис. 6.3). Это осуществляется через контекстное меню на пункте Ссылки в Обозревателе решений.

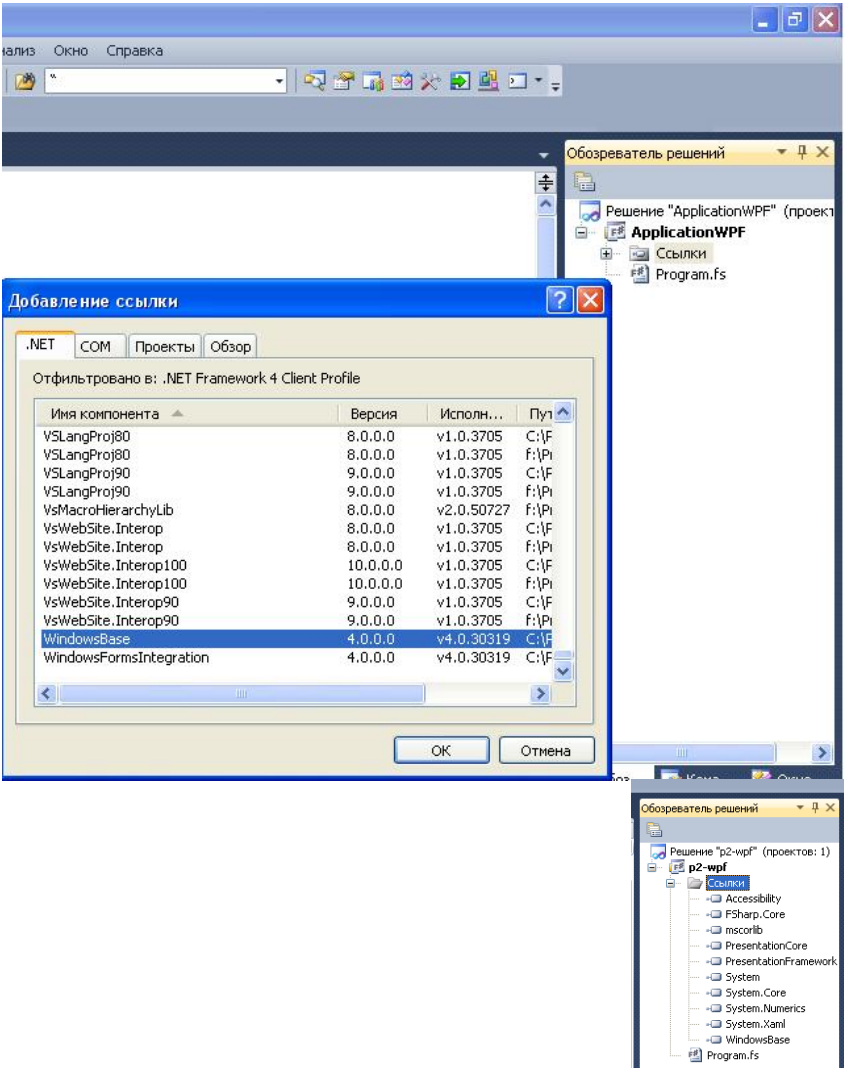


Рисунок 6.3. Добавление ссылок в проект

Теперь внесем текст программы. Создадим главную форму, на которой разместим элементы управления в виде двенадцати кнопок.

```
open System
open System.Windows
open System.Windows.Controls
open System.Windows.Markup
//Главная форма
let mwXaml = "
<Window
xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
      xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
      Title='F# WPF' Height='280' Width='320'>
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width='156*' />
         <ColumnDefinition Width='163' />
      </Grid.ColumnDefinitions>
      <GroupBox Header='[Варианты окна:]' Height='89' HorizontalAlign-
ment='Left' Name='groupBox1' VerticalAlignment='Top' Width='313'
Grid.ColumnSpan='2'>
         <Grid>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width='38*' />
               <ColumnDefinition Width='453*' />
            </Grid.ColumnDefinitions>
            <Button Content='Tool Window' Grid.ColumnSpan='2' Height='23'
HorizontalAlignment='Left' Margin='6,6,0,0' Name='button1' VerticalAlign-
ment='Top' Width='138' />
            <Button Content='Single Border Window' Height='23' Horizonta-
lAlignment='Left' Margin='6,35,0,0' Name='button2' VerticalAlignment='Top'
Width='138' Grid.ColumnSpan='2' />
            <Button Content='3D Border Window' Height='23' HorizontalAlign-
ment='Left' Margin='131,6,0,0' Name='button3' VerticalAlignment='Top'
Width='138' Grid.Column='1' />
            <Button Content='None Border Window' Height='23' Horizonta-
lAlignment='Left' Margin='132,35,0,0' Name='button4' VerticalAlignment='Top'
Width='138' Grid.Column='1' />
         </Grid>
      </GroupBox>
```

```
      <GroupBox Header='[Положение окна:]' Height='92' HorizontalAlign-
ment='Left' Margin='0,95,0,0' Name='groupBox2' VerticalAlignment='Top'
Width='313' Grid.ColumnSpan='2'>
         <Grid>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width='74*' />
               <ColumnDefinition Width='227' />
            </Grid.ColumnDefinitions>
            <Button Content='Move Top' Height='23' HorizontalAlignment='Left'
Margin='6,6,0,0' Name='button5' VerticalAlignment='Top' Width='138'
Grid.ColumnSpan='2' />
            <Button Content='Move Bottom' Height='23' HorizontalAlign-
ment='Left' Margin='6,35,0,0' Name='button6' VerticalAlignment='Top'
Width='138' Grid.ColumnSpan='2' />
            <Button Content='Move Left' Height='23' HorizontalAlignment='Left'
Margin='81,6,0,0' Name='button7' VerticalAlignment='Top' Width='137'
Grid.Column='1' />
            <Button Content='Move Right' Height='23' HorizontalAlign-
ment='Left' Margin='81,35,0,0' Name='button8' VerticalAlignment='Top'
Width='137' Grid.Column='1' />
         </Grid>
      </GroupBox>
      <Button Content='Справка' Height='23' HorizontalAlignment='Left' Mar-
gin='12,219,0,0' Name='button9' VerticalAlignment='Top' Width='90' />
      <Button Content='Автор' Height='23' HorizontalAlignment='Left' Mar-
gin='110,219,0,0' Name='button10' VerticalAlignment='Top' Width='90'
Grid.ColumnSpan='2' />
      <Button Content='Выход' Height='23' HorizontalAlignment='Left' Mar-
gin='60,219,0,0' Name='button11' VerticalAlignment='Top' Width='90'
Grid.Column='1' />
      <Button Content='Create new Window' Height='23' HorizontalAlign-
ment='Left' Margin='12,193,0,0' Name='button12' VerticalAlignment='Top'
Width='286' Grid.ColumnSpan='2' />
   </Grid>
</Window>
"
// загрузка разметки XAML
let getWindow(mwXaml) =
   let xamlObj=XamlReader.Parse(mwXaml)
   xamlObj :?> Window

let win = getWindow(mwXaml)

//получение экземпляров элементов управления
```

```
//главная форма:
let button1 = win.FindName("button1") :?> Button
let button2 = win.FindName("button2") :?> Button
let button3 = win.FindName("button3") :?> Button
let button4 = win.FindName("button4") :?> Button
let button5 = win.FindName("button5") :?> Button
let button6 = win.FindName("button6") :?> Button
let button7 = win.FindName("button7") :?> Button
let button8 = win.FindName("button8") :?> Button
let button9 = win.FindName("button9") :?> Button
let button10 = win.FindName("button10") :?> Button
let button11 = win.FindName("button11") :?> Button
let button12 = win.FindName("button12") :?> Button
//обработка событий
button1.Click.AddHandler(fun _ _ -> win.WindowStyle <- Win-
dowStyle.ToolWindow)
button2.Click.AddHandler(fun _ _ -> win.WindowStyle <- Win-
dowStyle.SingleBorderWindow)
button3.Click.AddHandler(fun _ _ -> win.WindowStyle <- Win-
dowStyle.ThreeDBorderWindow)
button4.Click.AddHandler(fun _ _ -> win.WindowStyle <- WindowStyle.None)
button5.Click.AddHandler(fun _ _ -> win.Top <- win.Top - 10.)
button6.Click.AddHandler(fun _ _ -> win.Top <- win.Top + 10.)
button7.Click.AddHandler(fun _ _ -> win.Left <- win.Left - 10.)
button8.Click.AddHandler(fun _ _ -> win.Left <- win.Left + 10.)
button9.Click.AddHandler(fun _ _ -> let msg = sprintf "Программа предназна-
чена для иллюстрации возможностей управления формой
                                  \nНажмите на любую из кнопок в меню
'Варианты окна' для изменения вида формы;
                                  \nДля программного изменения положения
окна воспользуйтесь клавишами из меню: 'Положение окна';
                                  \nС помощью кнопки: 'Create New
Window' вы можете создать новое окно;
                                  \nКнопка 'Выход' позволяет завершить
выполнение программы."
                        MessageBox.Show(msg) |> ignore)
button10.Click.AddHandler(fun _ _ -> let msg = sprintf "Автор: Щербаков
А.Ю. \nГруппа:1881 \nE-Mail:PrincNochi@inbox.ru"
                         MessageBox.Show(msg) |> ignore)
button11.Click.AddHandler(fun _ _ -> win.Close())
// запуск приложения
[<STAThread>] ignore <| (new Application()).Run win
```

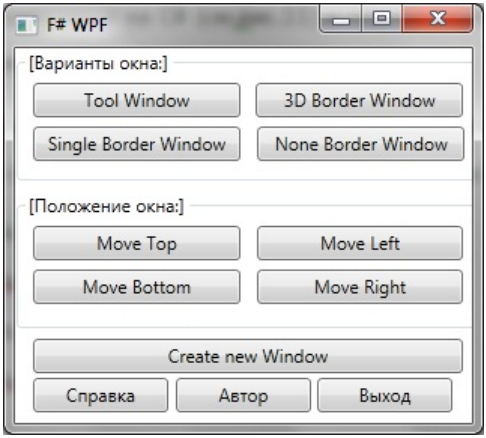


Рисунок 6.4. Вид главной формы

Добавим дочернюю форму (рис. 6.5). На ней разместим: две кнопки, поле для ввода текста, компонент для вывода текста, checkbox и три radiobutton`a.

```
//дочерняя форма
let nw= "
<Window
xmlns='http://schemas.microsoft.com/winfx/2006/xaml/presentation'
      xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
      Title='New Window' Height='221' Width='351'>
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width='156*' />
         <ColumnDefinition Width='163' />
      </Grid.ColumnDefinitions>
      <Button Content='Apply' Grid.Column='1' Height='23' HorizontalAlign-
ment='Left' Margin='76,30,0,0' Name='button1' VerticalAlignment='Top'
Width='75' IsEnabled='False' />
      <TextBox Height='23' HorizontalAlignment='Left' Margin='20,30,0,0'
Name='textBox1' VerticalAlignment='Top' Width='216' Grid.ColumnSpan='2'
IsEnabled='False' />
      <Button Content='Close' Height='23' HorizontalAlignment='Left' Mar-
gin='12,153,0,0' Name='button2' VerticalAlignment='Top' Width='305'
Grid.ColumnSpan='2' />
```

```
        <CheckBox Content='Заголовок окна:' Height='16' HorizontalAlign-
ment='Left' Margin='12,9,0,0' Name='checkBox1' VerticalAlignment='Top' />
        <RadioButton Content='RadioButton' Height='16' HorizontalAlign-
ment='Left' Margin='12,73,0,0' Name='radioButton1' VerticalAlignment='Top'
/>
        <RadioButton Content='RadioButton' Height='16' HorizontalAlign-
ment='Left' Margin='12,95,0,0' Name='radioButton2' VerticalAlignment='Top'
/>
        <RadioButton Content='RadioButton' Height='16' HorizontalAlign-
ment='Left' Margin='12,117,0,0' Name='radioButton3' VerticalAlignment='Top'
/>
        <Label Content=' ' Grid.Column='1' Height='28' HorizontalAlign-
ment='Left' Margin='96,90,0,0' Name='label1' VerticalAlignment='Top' />
    </Grid>
</Window>
"
let getWindow1(nw) =
    let xamlObj=XamlReader.Parse(nw)
    xamlObj :?> Window

let wind = getWindow1(nw)

let nb1 = wind.FindName("button1") :?> Button
let nb2 = wind.FindName("button2") :?> Button
let ncb = wind.FindName("checkBox1") :?> CheckBox
let nrb1 = wind.FindName("radioButton1") :?> RadioButton
let nrb2 = wind.FindName("radioButton2") :?> RadioButton
let nrb3 = wind.FindName("radioButton3") :?> RadioButton
let nl = wind.FindName("label1") :?> Label
let ntb = wind.FindName("textBox1") :?> TextBox

button12.Click.AddHandler(fun _ _ -> wind.Show()|>ignore)
```

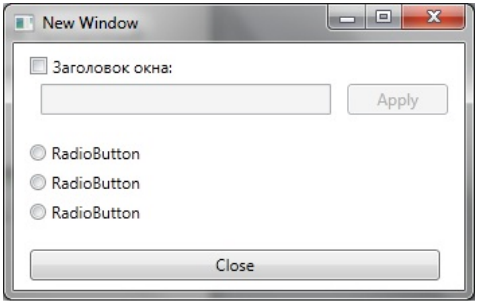


Рисунок 6.5. Вид дочерней формы

Внесем дополнения в программу. При активации флажка поле ввода текста и кнопка становятся активными и, наоборот, (при снятии флажка)

```
ncb.Checked.Add(fun _ -> ntb.IsEnabled<-true
                         nb1.IsEnabled<-true)
ncb.Unchecked.Add(fun _ -> ntb.IsEnabled<-false
                           nb1.IsEnabled<-false)
```

Допишем обработчик кнопок:

```
nb1.Click.AddHandler(fun _ _ -> wind.Title<-ntb.Text)
nb2.Click.AddHandler(fun _ _ -> wind.Hide())
nrb1.Checked.Add(fun _ -> nl.Content<-"1")
nrb2.Checked.Add(fun _ -> nl.Content<-"2")
nrb3.Checked.Add(fun _ -> nl.Content<-"3")
```

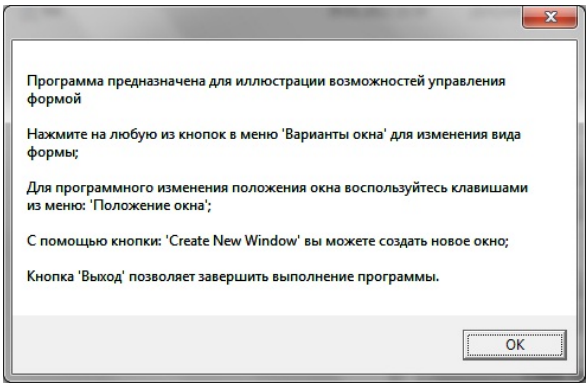


Рисунок 6.6. Пример сообщения

Задания

1. Разработать программу, состоящую из трех форм. На главной форме находится меню через которое осуществляется переход на другие формы.
2. Разработать программу решающую квадратное уравнение, состоящую из одной формы. Ответ выводится в TextBox.
3. Разработать программу, реализующую простые вычисление над двумя действительными числами (сложения, вычитания, деления, умножения). Учесть все возможные ошибки.

4. Разработать программу, состоящую из главной формы, рисунка и одной кнопки. По нажатию на кнопку меняется рисунок.

5. Разработать программу, которая по выбору месяца выдавала сообщение со временем года. Главная форма, а на ней выпадающий список с месяцами, сообщение появляется при нажатии на кнопку.

6. Разработать приложение, состоящее из одной формы, на которой размещены кнопка и ползунок. При изменении положения ползунка меняется ширина кнопки.

7. Разработать программу, состоящую из главной формы на которой размещены два флажка и одна кнопка. По нажатию на кнопку появляется сообщение «установлен первый флажок», «установлен второй флажок», «установлены оба флажка».

8. Разработать программу, состоящую из одной формы, на которой размещены: поле для ввода, кнопка и список элементов. После того как ввели текст в поле ввода нажатием на кнопку, он добавляет в список.

9. Разработать программу, на главной форме разместить поле для ввода и индикатор хода загрузки. По мере ввода текста в поле ввода заполняется индикатор.

10. Разработать приложение, вычисляющее площадь прямоугольника.

Контрольные вопросы

1. Сколько существует способов разработки форм?
2. Основные компоненты библиотеки WPF?
3. Как осуществляется добавление компонентов на форму?
4. Какие ссылки необходимо добавить в проект для работы с библиотекой WPF?
5. Существует ли визуальный дизайнер форм в F#?
6. Какие свойства отвечают за положения объекта на форме?
7. Как изменить консольное приложение на Windows приложение?
8. Назовите основные свойства, которые использует кнопка?
9. Как добавляется событие на нажатие кнопки?
10. Как вызвать дочернюю форму?

Лабораторная работа №7. «Кортежи и WinForms»

Цель работы: Создать программу, выполняющую действия над комплексными числами и обрабатывающую события с применением кортежей в библиотеке WinForms.

Практическая работа

Запускаем Visual Studio 2010 и создаем проект — Приложение F#, проект приложения для командной строки (рис. 7.1).

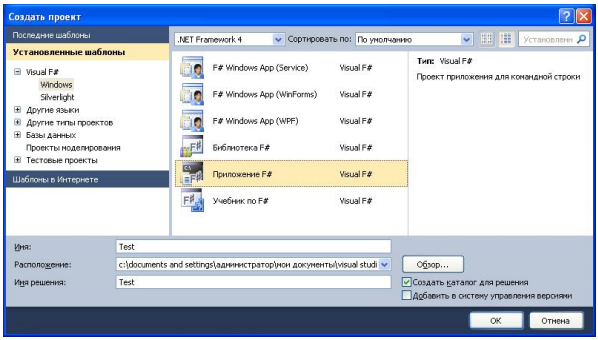


Рисунок 7.1. Создание проекта

После создания проекта, изменяем его тип через Проект — > Свойства, меняя в типе выходных данных Консольное приложение на Windows-приложение (рис. 7.2), сохраняем изменения и закрываем вкладку.

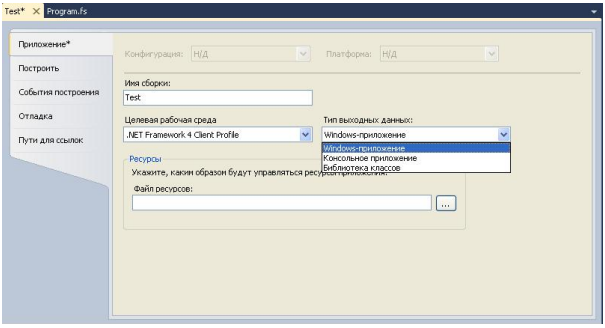


Рисунок 7.2. Изменение типа приложения

Теперь необходимо добавить в проект ссылки на требуемые библиотеки: System.Windows.Forms, System.Drawing (рис. 7.3). Это осуществляется через контекстное меню на пункте Ссылки в Обозревателе решений.

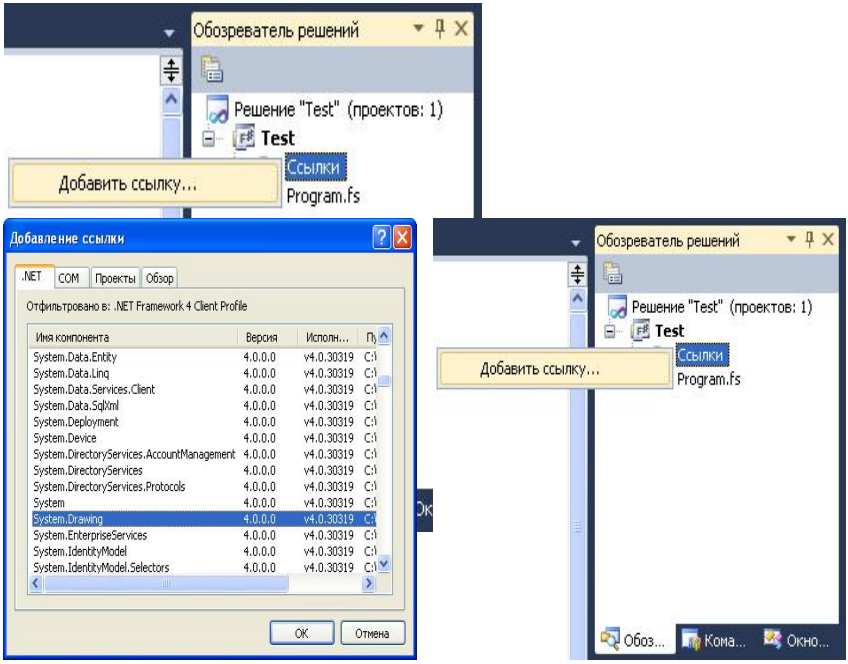


Рисунок 7.3. Добавление ссылок в проект

Теперь внесем текст программы. Создадим главную форму на которой разместим меню, 8 полей для ввода текста и 3 кнопки (рис. 7.4).

```
open System.Drawing
open System.Windows.Forms

let form = new Form(
                Width= 373, Height = 266,
                Visible = true,
                Text = "Кортежи в F#",
                Menu = new MainMenu()
                )
```

```
let button1 = new Button(Text="Сумма", Left=10, Top=9, Width=75,
Height=23);
let button2 = new Button(Text="Произведение", Left=91, Top=9, Width=108,
Height=23);
let button3 = new Button(Text="Разность", Left=205, Top=9, Width=75,
Height=23);
let button4 = new Button(Text="Деление", Left=286, Top=9, Width=75,
Height=23);
let textBox1 = new TextBox(Text="0", Left=10, Top=55, Width=55,
Height=20);
let textBox2 = new TextBox(Text="0", Left=71, Top=55, Width=55,
Height=20);
let textBox3 = new TextBox(Text="0", Left=151, Top=55, Width=55,
Height=20);
let textBox4 = new TextBox(Text="0", Left=212, Top=55, Width=55,
Height=20);
let label1 = new Label(Text="?", Left=132, Top=58, Width=13, Height=13);
let label2 = new Label(Text="a", Left=31, Top=39, Width=13, Height=13);
let label3 = new Label(Text="bi", Left=88, Top=39, Width=13, Height=13);
let label4 = new Label(Text="c", Left=171, Top=39, Width=13, Height=13);
let label5 = new Label(Text="di", Left=233, Top=39, Width=13, Height=13);
let button5 = new Button(Text="=", Left=273, Top=53, Width=32, Height=23);
let label6 = new Label(Text="0", Left=311, Top=58, Width=60, Height=13);
let label7 = new Label(Text="Модуль комплексного числа a+bi", Left=10,
Top=85, Width=250, Height=13);
let label8 = new Label(Text="a", Left=31, Top=100, Width=10, Height=13);
let label9 = new Label(Text="b", Left=88, Top=100, Width=10, Height=13);
let textBox5 = new TextBox(Text="0", Left=10, Top=120, Width=55,
Height=20);
let textBox6 = new TextBox(Text="0", Left=70, Top=120, Width=55,
Height=20);
let button6 = new Button(Text="=", Left=130, Top=119, Width=32,
Height=23);
let label10 = new Label(Text="0", Left=170, Top=122, Width=100, Height=13);
let label11 = new Label(Text="Аргумент комплексного числа", Left=10,
Top=150, Width=290, Height=13);
let label12 = new Label(Text="Im", Left=30, Top=163, Width=30, Height=13);
let label13 = new Label(Text="Re", Left=87, Top=163, Width=30, Height=13);
let textBox7 = new TextBox(Text="0", Left=10, Top=180, Width=55,
Height=20);
let textBox8 = new TextBox(Text="0", Left=70, Top=180, Width=55,
Height=20);
let button7 = new Button(Text="=", Left=130, Top=179, Width=32,
Height=23);
```

```
let label14 = new Label(Text="Градусы = 0", Left=170, Top=182, Width=100,
Height=13);

form.Controls.Add(label14);
form.Controls.Add(button7);
form.Controls.Add(textBox8);
form.Controls.Add(textBox7);
form.Controls.Add(label13);
form.Controls.Add(label12);
form.Controls.Add(label11);
form.Controls.Add(label10);
form.Controls.Add(button6);
form.Controls.Add(textBox6);
form.Controls.Add(textBox5);
form.Controls.Add(label9);
form.Controls.Add(label8);
form.Controls.Add(label7);
form.Controls.Add(label6);
form.Controls.Add(button5);
form.Controls.Add(label5);
form.Controls.Add(label4);
form.Controls.Add(label3);
form.Controls.Add(label2);
form.Controls.Add(label1);
form.Controls.Add(textBox4);
form.Controls.Add(textBox3);
form.Controls.Add(textBox2);
form.Controls.Add(textBox1);

let krt _ = label1.Text <- "+"
let _ = toolStrip1.Click.Add(krt)


//////////////////////////////////////////////////////////
let umn _ =label1.Text <- "*"
let _ = toolStrip3.Click.Add(umn)

//////////////////////////////////////////////////////////
let raz _ =label1.Text <- "-"
let _ = toolStrip2.Click.Add(raz)

//////////////////////////////////////////////////////////
let delenie _ =label1.Text <- "/"
let _ = toolStrip4.Click.Add(delenie)
```

```
//////////////////////////////////////////////////////////////

let ravno _ = if label1.Text="/" then
              let del (a, b) (c, d) =
                  let znam = c*c + d*d
                  let dei = ((a*c + b*d)/znam)
                  let mni = ((-a*d + b*c)/znam)
                  (dei , mni)
              let d1 = (float textBox1.Text, float textBox2.Text)
              let d2 = (float textBox3.Text, float textBox4.Text)
              let d3 = del d1 d2
              label6.Text <- string d3

             if label1.Text="+" then
              let summ (r1, i1) (r2, i2) =
                  let dei = r1 + r2
                  let mni = i1 + i2
                  (dei , mni)
              let s1 = (float textBox1.Text, float textBox2.Text)
              let s2 = (float textBox3.Text, float textBox4.Text)
              let s3 = summ s1 s2
              label6.Text <- string s3

             if label1.Text="-" then
              let raznost (a, b) (c, d) =
                  let dei = a - c
                  let mni = b - d
                  (dei , mni)
              let r1 = (float textBox1.Text, float textBox2.Text)
              let r2 = (float textBox3.Text, float textBox4.Text)
              let r3 = raznost r1 r2
              label6.Text <- string r3

             if label1.Text="*" then
              let umnoj (a, b) (c, d) =
                  let dei = (a*c) - (b*d)
                  let mni = (a*d) + (b*c)
                  (dei , mni)
              let u1 = (float textBox1.Text, float textBox2.Text)
              let u2 = (float textBox3.Text, float textBox4.Text)
              let u3 = umnoj u1 u2
              label6.Text <- string u3
let _ = button5.Click.Add(ravno)
```

```
let modul _ = let mo = sqrt((float textBox5.Text*float textBox5.Text)+(float
textBox6.Text*float textBox6.Text))
            label10.Text <- string mo
let _ = button6.Click.Add(modul)

let arg _ =
        let x = float textBox7.Text
        let y = float textBox8.Text
        let ar = atan(x/y)
        let grad = (ar*180.0)/3.14159265
        label14.Text <- "Градусы = "+string grad
let _ = button7.Click.Add(arg)

do Application.Run(form)
```

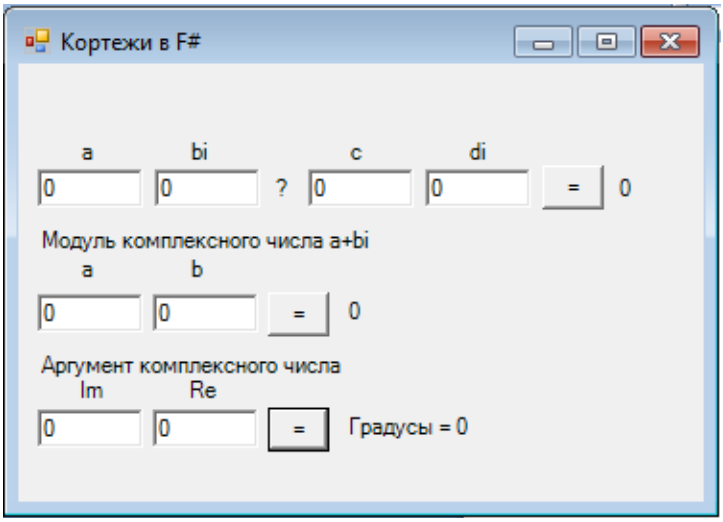


Рисунок 7.4. Вид главной формы

Внесем дополнения в программу. При нажатии правой кнопкой на «?» будет отображаться меню (рис. 7.5).

```
let contextMenuStrip1 = new ContextMenuStrip()
let toolStrip1 = new ToolStripMenuItem(«+ (Сумма)»)
let toolStrip2 = new ToolStripMenuItem(«- (Разность)»)
let toolStrip3 = new ToolStripMenuItem(«* (Произведение)»)
let toolStrip4 = new ToolStripMenuItem(«/ (Деление)»)
```

```
contextMenuStrip1.Items.Add(toolStrip1)
contextMenuStrip1.Items.Add(toolStrip2)
contextMenuStrip1.Items.Add(toolStrip3)
contextMenuStrip1.Items.Add(toolStrip4)
label1.ContextMenuStrip <- contextMenuStrip1
```

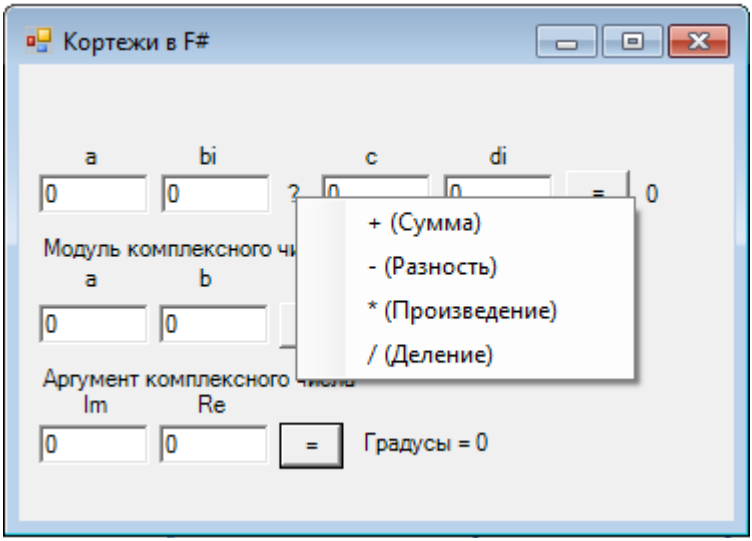


Рисунок 7.5. Пример меню

Задания

1. Разработать программу, состоящую из одной формы. На главной форме находится меню (MainMenu), через которое осуществляется действия над комплексными числами.
2. Разработать программу решающую квадратное уравнение, состоящую из одной формы. Ответ выводится в lable.
3. Разработать программу, реализующую простые вычисление над двумя действительными числами (сложения, вычитания, деления, умножения). Учесть все возможные ошибки.
4. Разработать программу, которая по выбору месяца выдавала сообщение со временем года. Главная форма, а на ней выпадающий список с месяцами, сообщение появляется при нажатии на кнопку.

5. Разработать программу, состоящую из главной формы на которой размещены два флажка и одна кнопка. По нажатию на кнопку появляется сообщение «установлен первый флажок», «установлен второй флажок», «установлены оба флажка».

6. Разработать программу, на главной форме разместить поле для ввода и индикатор хода загрузки. По мере ввода текста в поле ввода заполняется индикатор.

7. Разработать приложение вычисляющее площадь прямоугольника.

8. Разработать приложение получения отдельных значений, используя подбор шаблонов.

9. Разработать программу вычисляющий cos(x), sin(x),tn(x), с помощью библиотеки WinFoms.

10. Разработать программу для нахождения log n, при нажатии знака «равно» ответ выводим в textbox.

Контрольные вопросы

1. Сколько существует способов разработки форм?
2. Основные компоненты библиотеки WinForms?
3. Как осуществляется добавление компонентов на форму?
4. Какие ссылки необходимо добавить в проект для работы с библиотекой WinForms?
5. Кортеж — это …
6. Может ли Кортеж иметь смешенные типы?
7. Кортеж может иметь целочисленный тип?
8. Какими типами обладает Кортежи?
9. Начиная с какой версии платформы .NET кортеж является стандартным типом?
10. Как добавляется событие на нажатие кнопки?

Лабораторная работа № 8 «Массивы и WinForms»

Цель работы: изучить основные принципы работы с массивами в F#, разработки приложений на F#.

Практическая работа

Необходимо разработать программу с использованием WinForms, создать понятный пользователю интерфейс.

Разработаем простую программу, которая создаст массив квадратов чисел от 1 до 10, с помощью библиотеки WinForms.

Запускаем Visual Studio 2010 и создаем проект — Приложение F#, проект приложения для командной строки.

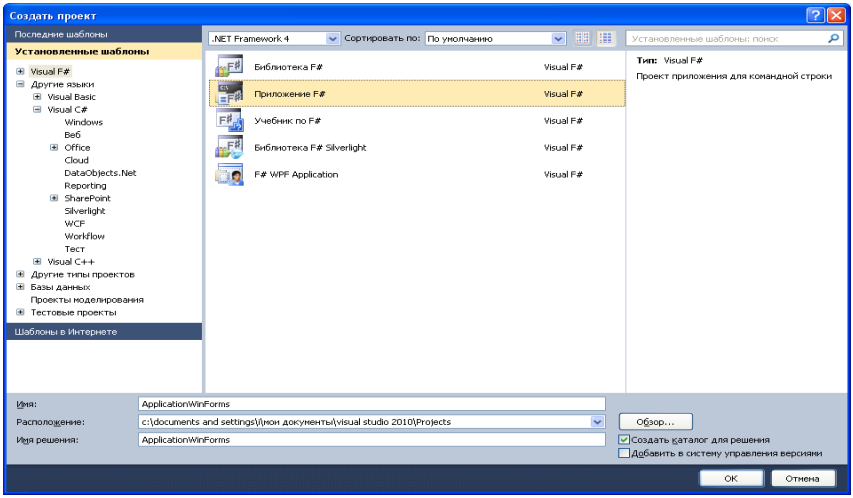


Рисунок 8.1.

После создания проекта, изменяем его тип через Проект-Свойства, меняя в типе выходных данных Консольное приложение на Windows-приложение, закрываем вкладку.

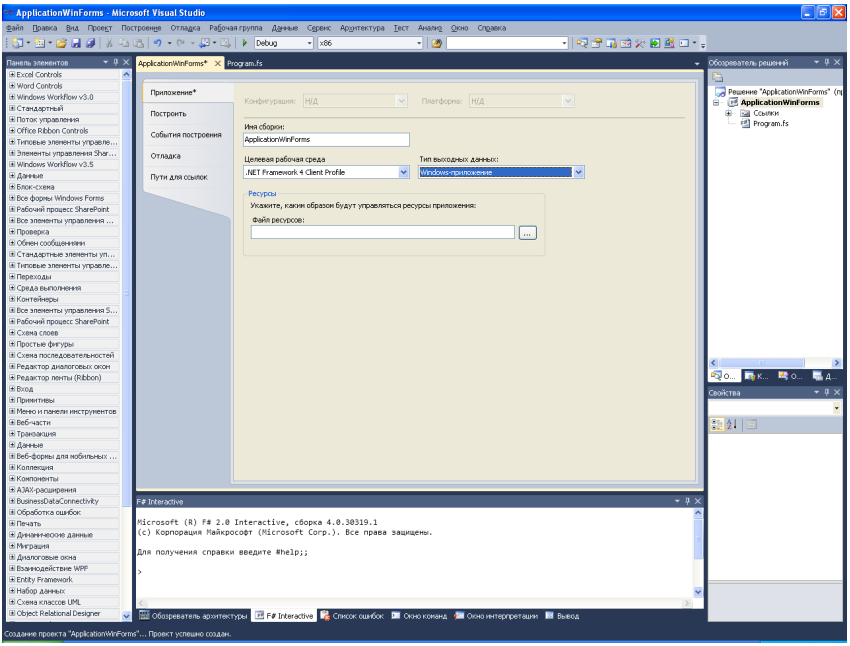


Рисунок 8.2.

Теперь необходимо добавить в проект ссылки на требуемые библиотеки: System.Windows.Forms, System.Drawing. Это осуществляется через контекстное меню на пункте Ссылки в Обозревателе решений.

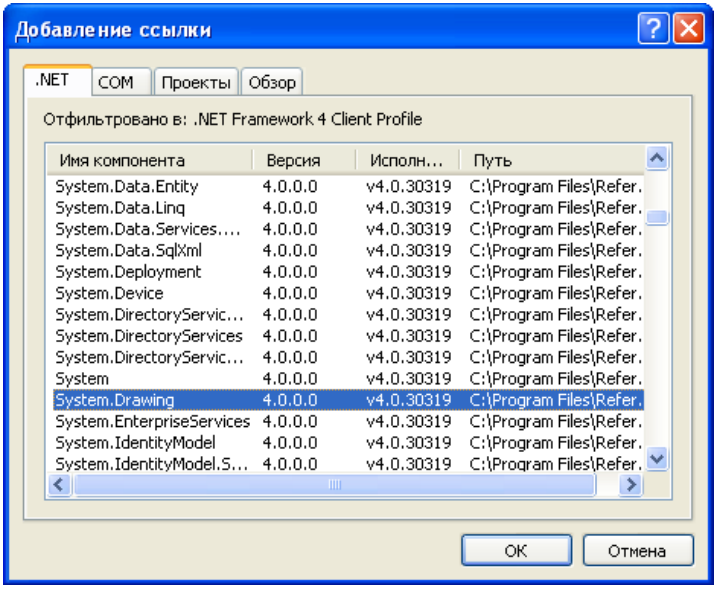


Рисунок 8.3.

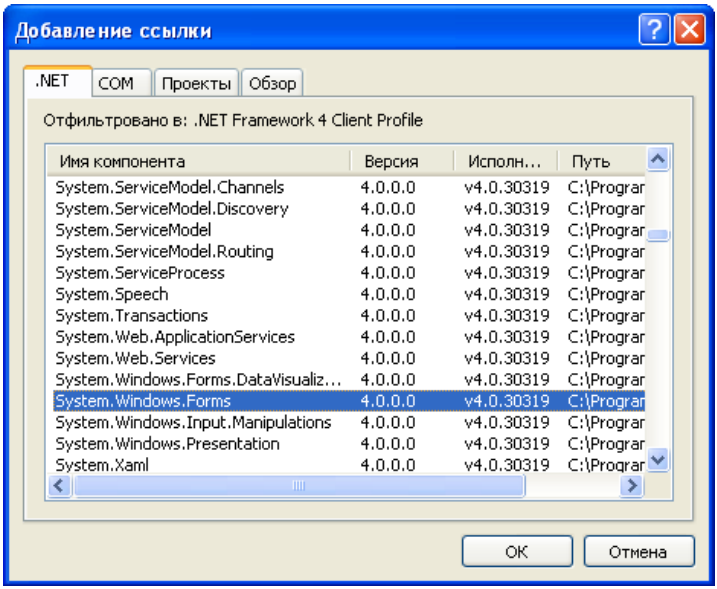


Рисунок 8.4.

Внесем текст программы.

```
open System
open System.Windows.Forms
open System.Drawing
Application.EnableVisualStyles()
let form = new Form(Text="Работа с массивами")
let label1 = new Label()
label1.Location<-new Point(100,50)
label1.Text<-"Массив 1"
label1.Width<-60
label1.Height<-12
let TextBox = new TextBox()  // Создание текстового поля для ввода информации
TextBox.Location<-new Point(170,50)
TextBox.Width<-60
TextBox.Height<-25
TextBox.Text<-""
// Создание кнопки с текстом "Вывести"
let button = new Button(Text="Вывести_1")
button.Location<-new Point(15,50)    // позиция кнопки
//Добавление обработчика события - Нажатие на кнопку
button.Click.AddHandler(fun _ _ ->
        let array = [| for i in 1 .. 10 -> i * i |]
        let run = TextBox.Text<-  (array |> Seq.map string |> String.concat ", ")
        run
        |> ignore)
//Добавление элементов на форму
form.Controls.Add(button)
form.Controls.Add(label1)
form.Controls.Add(TextBox)
// запуск формы
Application.Run(form)
```

Наглядно это выглядит на рис. 8.5.

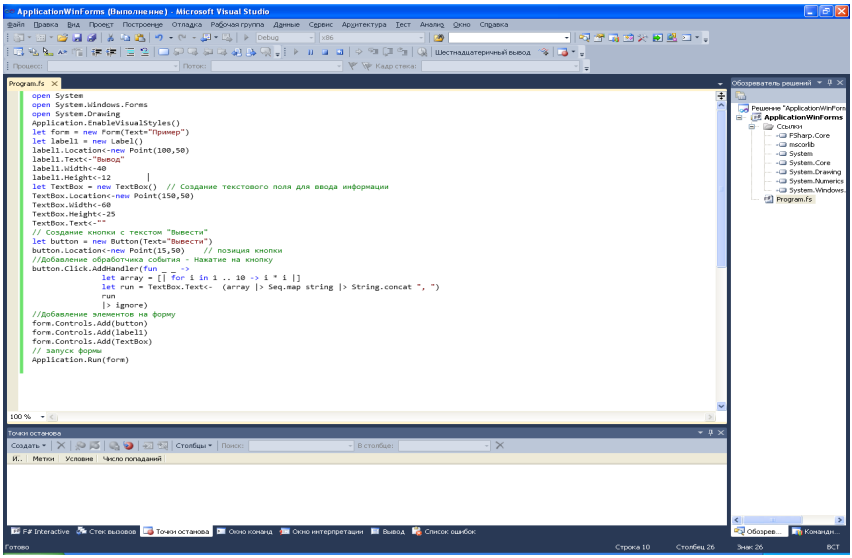


Рисунок 8.5.

Запустив приложение, увидим результат (рис. 8.6).

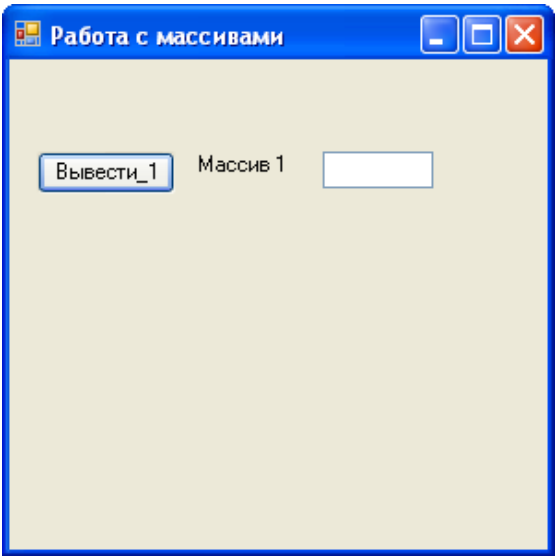


Рисунок 8.6.

Аналогично примеру дорабатываем программу, для того чтобы она работала с разными массивами. Применим различные функции, а так же прорабатываем элементы интерфейса. Код готовой программы:

```
open System
open System.Windows.Forms
open System.Drawing

Application.EnableVisualStyles()

// Создание формы с заголовком "Работа с массивом"
let form = new Form(Text="Работа с массивом")

// Создание подписи для поля ввода
let label1 = new Label()
label1.Location<-new Point(100,25)
label1.Text<-"массив 1"
label1.Width<-60
label1.Height<-12

// Создание подписи для поля вывода
let label2 = new Label()
label2.Location<-new Point(100,70)
label2.Text<-" массив 2"
label2.Width<-60
label2.Height<-12

let label3 = new Label()
label3.Location<-new Point(100,110)
label3.Text<-"массив 3"
label3.Width<-60
label3.Height<-12

let label4 = new Label()
label4.Location<-new Point(100,160)
label4.Text<-"массив 4"
label4.Width<-60
label4.Height<-12

// Создание текстового поля для ввода информации
let txtInputA = new TextBox()
txtInputA.Location<-new Point(170,25)
```

```
txtInputA.Width<-100
txtInputA.Height<-25
txtInputA.Text<-""

// Создание текстового поля для вывода информации
let txtOutputB = new TextBox()
txtOutputB.Location<-new Point(170,70)
txtOutputB.Width<-100
txtOutputB.Height<-25
txtOutputB.Text<-""

let txtOutputC = new TextBox()
txtOutputC.Location<-new Point(170,110)
txtOutputC.Width<-100
txtOutputC.Height<-25
txtOutputC.Text<-""

let txtOutputD = new TextBox()
txtOutputD.Location<-new Point(170,130)
txtOutputD.Width<-100
txtOutputD.Height<-25
txtOutputD.Text<-""

let txtOutputE = new TextBox()
txtOutputE.Location<-new Point(170,160)
txtOutputE.Width<-100
txtOutputE.Height<-25
txtOutputE.Text<-""

let txtOutputF = new TextBox()
txtOutputF.Location<-new Point(170,180)
txtOutputF.Width<-100
txtOutputF.Height<-25
txtOutputF.Text<-""

let txtOutputG = new TextBox()
txtOutputG.Location<-new Point(170,210)
txtOutputG.Width<-100
txtOutputG.Height<-25
txtOutputG.Text<-""

// Создание кнопки с текстом "Вычислить!"
let button = new Button(Text="Вывести_1")
button.Location<-new Point(15,25)    // позиция кнопки
```

```
//Добавление обработчика события - Нажатие на кнопку
button.Click.AddHandler(fun _ _ ->
        let array3 = [|for i in 1 .. 10 -> i * i|]
        //txtInputA.Text <- (array3 |> sprintf "%A") тоже вывод массива
        let run = txtInputA.Text<- (array3 |> Seq.map string |> String.concat ", ")
        run
        |> ignore)

let button1 = new Button(Text="Вывести_2")
button1.Location<-new Point(15,70)
button1.Click.AddHandler(fun _ _ ->

            let arrayOfTenZeroes : string array = Array.zeroCreate 10
            let run = txtOutputB.Text<- (arrayOfTenZeroes |> Seq.map string |>
String.concat ", ")
            run
            |> ignore)

let button3 = new Button(Text="Вывести_3")
button3.Location<-new Point(15,110)
button3.Click.AddHandler(fun _ _ ->

             let stringReverse (txtOutputC: string) =
                System.String(Array.rev (txtOutputC.ToCharArray()))
             let run = txtOutputD.Text <- ( stringReverse (txtOutputC.Text))
             run
             |> ignore)

let button4 = new Button(Text="Вывести_4")
button4.Location<-new Point(15,160)
button4.Click.AddHandler(fun _ _ ->

            let array = [| 1 .. 10 |]
                    |> Array.filter (fun elem -> elem % 2 = 0)
                    //|> Array.choose (fun elem -> if (elem <> 8) then
Some(elem*elem) else None)
                    |> Array.rev

      let run = txtOutputE.Text<- ( array |> Seq.map string |> String.concat ", ")
        run
        |> ignore)

let button2 = new Button(Text="Выход")
button2.Location<-new Point(200,235)
```

```
button2.Click.AddHandler(fun _ _ ->
            let cl = form.Close()

            cl

            |> ignore)
//Добавление элементов на форму
form.Controls.Add(button)
form.Controls.Add(button1)
form.Controls.Add(button2)
form.Controls.Add(button3)
form.Controls.Add(button4)
form.Controls.Add(label1)
form.Controls.Add(label2)
form.Controls.Add(label3)
form.Controls.Add(label4)
form.Controls.Add(txtInputA)
form.Controls.Add(txtOutputB)
form.Controls.Add(txtOutputC)
form.Controls.Add(txtOutputD)
form.Controls.Add(txtOutputE)
//form.Controls.Add(txtOutputF)
//form.Controls.Add(txtOutputG)

// запуск формы
Application.Run(form)
```

Запускаем приложение, чтобы увидеть результат.

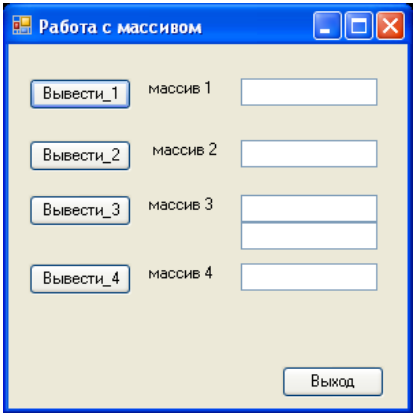


Рисунок 8.7. Главная форма программы

На форме имеется пять кнопок, четыре для вывода информации и одна для завершения работы приложения, так же на форме имеются поля ввода\вывода информации, а именно массивов.

Кнопка «Вывести_1» выводит массив из квадратов чисел от 1 до 10.

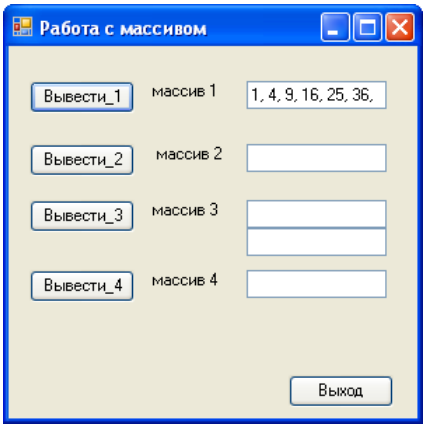


Рисунок 8.8. Вывод массива квадратов чисел

Кнопка «Вывести_2» выводит массив размерностью 10, нулевых элементов.

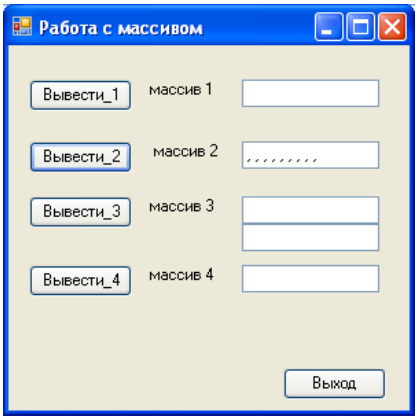


Рисунок 8.9. Вывод массива нулевых элементов

Чтобы воспользоваться кнопкой «Вывести_3» предварительно требуется ввести произвольный текст в поле для ввода, к примеру «Hello world», нажав на кнопку, увидим результат в поле, расположенном ниже «dlrow olleH».

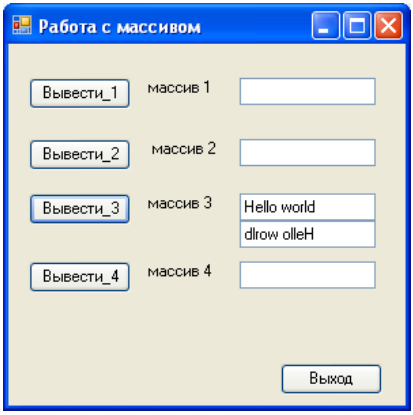


Рисунок 8.10. Вывод зеркального массива

Кнопка «Вывести_4» выводит массив с применением сразу трех функций. Вначале генерирует массив квадратов чисел от 1 до 10 как в первом случае, затем с помощью функции фильтра оставляет только те элементы, которые делятся на 2, и последним шагом является зеркально отображение массива.

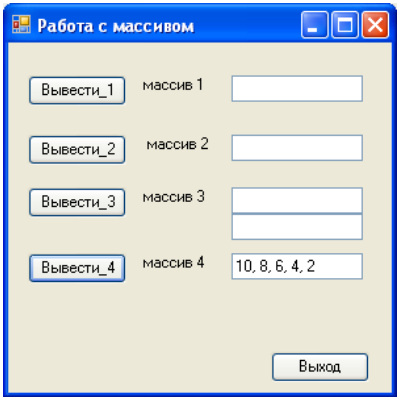


Рисунок 8.11. Результат нажатия кнопки «Вывести_4»

По нажатию кнопки «Выход» приложение завершает свою работу.

Задания

1. Создать новый массив, в котором порядок элементов существующего массива будет изменен на обратный (Привет -> тевирП).
2. Дан массив A [1; 2; 3;] и массив B [4; 5; 7] скопировать последний элемент массива B в массив A.
3. Объединить два произвольных массива в один.
4. Создать функцию фильтр для массива A [1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12] в котором элементы будут делиться на 3 без остатка.
5. Напишите программу, которая вводит с клавиатуры два непустых массива целых чисел в диапазоне от нуля до девяти, и, считая эти массивы десятичным представлением двух чисел, печатает их разность.
6. Напишите программу, которая вводит с клавиатуры два непустых неубывающих массива целых чисел, и печатает те и только те элементы, которые встречаются хотя бы в одном из массивов (объединение множеств).
7. Напишите программу, которая вводит с клавиатуры два непустых неубывающих массива целых чисел, и печатает те и только те элементы, которые встречаются в обоих массивах (пересечение множеств).
8. Напишите программу, которая вводит с клавиатуры два непустых неубывающих массива целых чисел, и печатает те и только те элементы, которые входят только в один из массивов (симметрическая разность множеств).
9. Напишите программу, заносящую в массив первые 100 натуральных чисел, делящихся на 13 или на 17, и печатающую его.
10. Напишите программу, вводящую целые коэффициенты многочлена и находящую все его рациональные корни.

Контрольные вопросы

1. Как создаются массивы в F#.
2. Как создается массив нулевых элементов.
3. Как осуществляется доступ к элементам массива .
4. Как задается диапазон в массивах.
5. На какой платформе создаются приложения в F#
6. Опишите функции создания массива.
7. Как называется библиотека для создания визуальных компонентов используемых в F# .
8. С помощью какой функции можно упорядочить массив.
9. С помощью каких функций можно получить максимальный и минимальный элемент массива.
10. Какая функция возвращает среднее значение каждого элемента в массиве.

Лабораторная работа № 9. «Списки и WinForms»

Цель работы: Создавать программу с использованием библиотеки WinForms, иллюстрирующую работу трех функций над списками.

Практическая работа

Для этого воспользуемся библиотекой WinForm, так как F# не имеет визуальной оболочки и компоненты приходится создавать вручную или вставляя код сгенерированный языком C#, где нужно предварительно разработать интерфейс.

Запускаем Visual Studio 2010 и создаем проект — Приложение F#, проект приложения для командной строки.

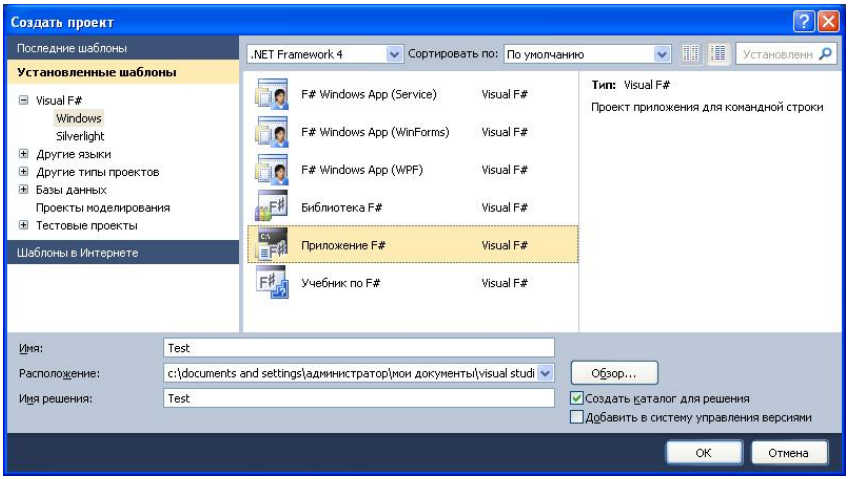


Рисунок 9.1. Создание проекта

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

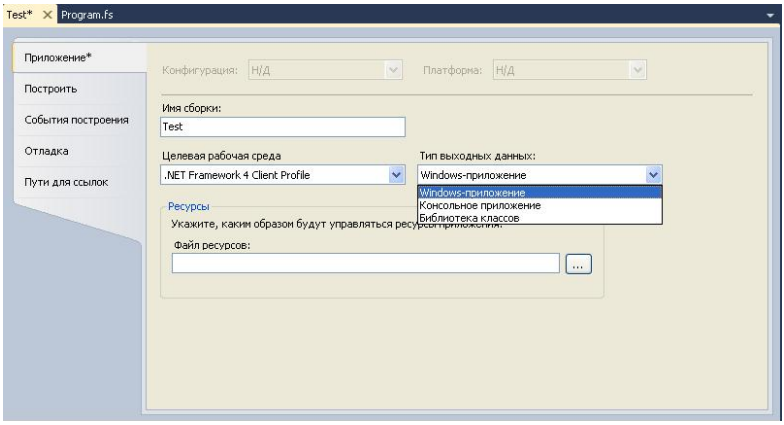


Рисунок 9.2. Изменение типа приложения

Теперь необходимо добавить в проект ссылки на требуемые библиотеки: System.Windows.Forms, System.Drawing (рис. 9.3). Это осуществляется через контекстное меню на пункте Ссылки в Обозревателе решений.

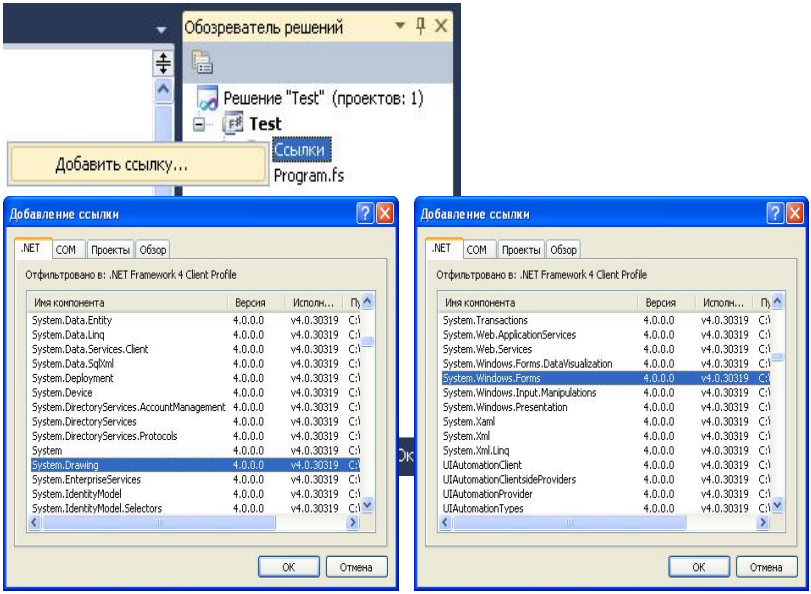


Рисунок 9.3. Добавление ссылок в проект

Далее вводим код для сформирования формы

```
open System
open System.Windows.Forms
open System.Drawing
Application.EnableVisualStyles()
let form = new Form(Width=302, Height=350,Text = "Работа со списками")
let button1 = new Button(Left=21, Top=38, Text="вывод списка", Width=96,
Height=23)
let button2 = new Button(Left=21, Top=81, Text="объединение списков",
Width=96, Height=46)
let button3 = new Button(Left=21, Top=152, Text="добавления элемента в
список", Width=96, Height=55)
let button4 = new Button(Left=21, Top=244, Text="сортировка по возраста-
нию", Width=96, Height=43)
let textBox1 = new TextBox(Left=156, Top=38, Width=114, Height=20)
let textBox2 = new TextBox(Left=156, Top=107, Width=114, Height=20)
let textBox3 = new TextBox(Left=156, Top=187, Width=114, Height=20)
let textBox4 = new TextBox(Left=156, Top=267, Width=114, Height=20)
let textBox5 = new TextBox(Left=156, Text="a,b,c",Top=81, Width=46,
Height=20)
let textBox6 = new TextBox(Left=222, Text="x,y,z",Top=81, Width=48,
Height=20)
let textBox7 = new TextBox(Left=156, Text="0",Top=152, Width=25,
Height=20)
let textBox8 = new TextBox(Left=200, Text="5,6,7,8,9,10",Top=152, Width=70,
Height=20)
let textBox9 = new TextBox(Left=156, Text="1; 4; 8; -2; 5",Top=244,
Width=114, Height=20)
        // Form1
        //
form.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
form.ClientSize = new System.Drawing.Size(302, 314);
form.Controls.Add(textBox9);
form.Controls.Add(textBox8);
form.Controls.Add(textBox7);
form.Controls.Add(textBox6);
form.Controls.Add(textBox5);
form.Controls.Add(textBox4);
form.Controls.Add(textBox3);
form.Controls.Add(textBox2);
form.Controls.Add(textBox1);
form.Controls.Add(button4);
```

```
form.Controls.Add(button3);
form.Controls.Add(button2);
form.Controls.Add(button1);
form.Text = "Работа со списками";
form.ResumeLayout(false);
form.PerformLayout();
```

Воспользуемся первым методом, сгенерировав код и описав все функции, запускаем приложение, перед нами сразу появляется главная форма.

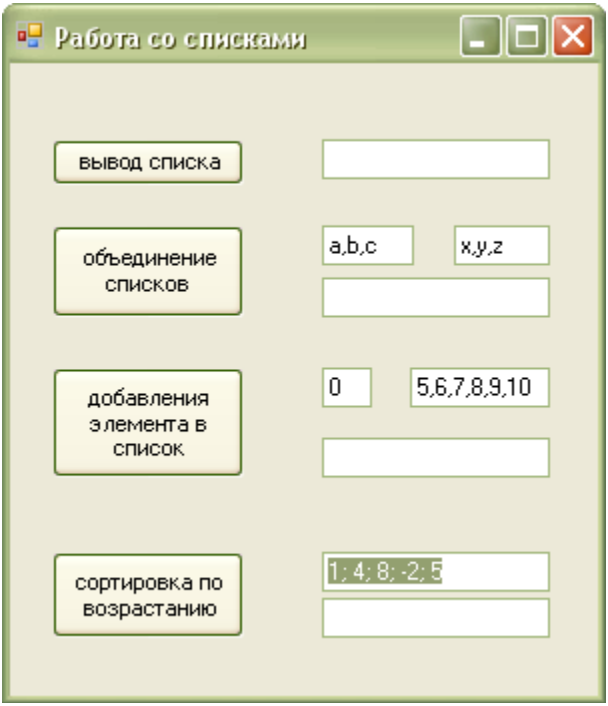


Рисунок 9.4. Главная форма программы

На форме имеется четыре кнопки для вывода информации, так же на форме представлены поля вывода информации, а именно списков.

Кнопка «вывод списка» выводит список из квадратов чисел от 1 до 10

```
button1.Click.AddHandler(fun _ _ ->
        let list1 = [for i in 1 .. 10 -> i * i ]
        let run = textBox1.Text<-  (list1 |> Seq.map string |> String.concat ", ")
            run
            |> ignore)
```

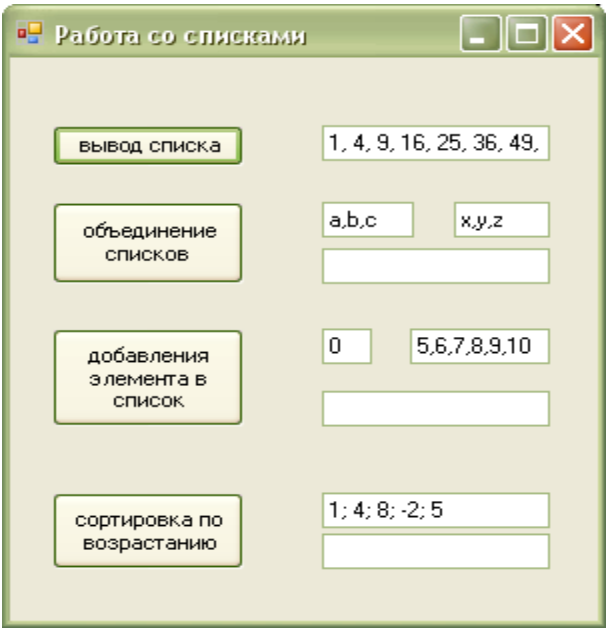


Рисунок 9.5. Вывод списка квадратов чисел

Кнопка «объединение списков» выводит новый список, состоящий из объединенных двух списков.

```
button2.Click.AddHandler(fun _ _ ->
        let list1 = ['a' ..'c' ]
        let list2 = ['x' ..'z' ]
        let list3 = list1 @ list2
        let run = textBox2.Text<-  (list3 |> Seq.map string |> String.concat ", ")
            run
            |> ignore)
```

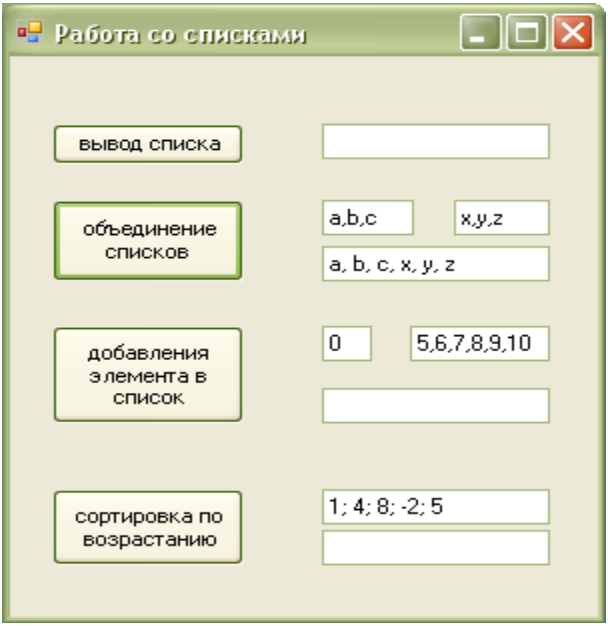


Рисунок 9.6. Вывод список новый список состоящий из объединенных двух списков

Чтобы воспользоваться кнопкой «добавления элемента в список» предварительно нужно ввести произвольный текст в поле для ввода, к примеру «2», нажав на кнопку, увидим результат в поле расположенном ниже «2, 5, 6, 7, 8, 9, 10»

```
button3.Click.AddHandler(fun _ _ ->
        let list1 = [ 5 .. 10 ]
        let list2 = int textBox7.Text :: list1
        let run = textBox3.Text<-  (list2 |> Seq.map string |> String.concat ", ")
          run
          |> ignore)
```

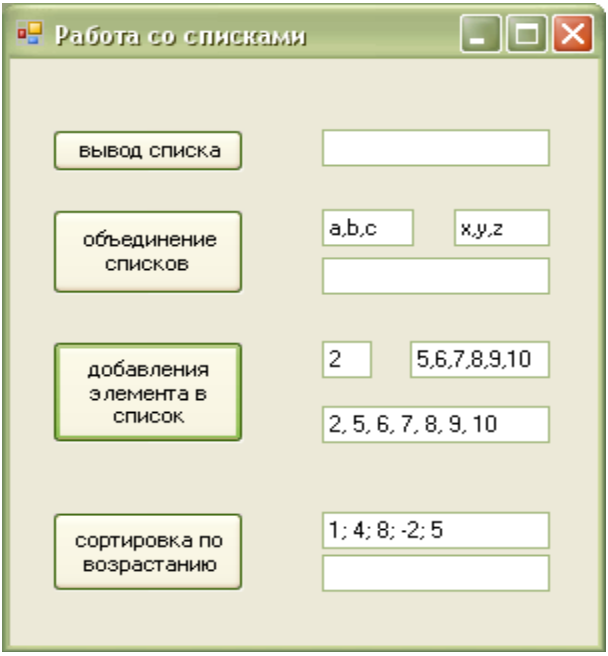


Рисунок 9.7. Вывод новый список с добавленным элементом.

Кнопка «сортировка по возрастанию» выводит список, отсортированный по возрастанию.

```
button4.Click.AddHandler(fun _ _ ->
        let list1 = List.sort [1; 4; 8; -2; 5]
         let run = textBox4.Text<-  (list1 |> Seq.map string |> String.concat ", ")
            run
             |> ignore)
```

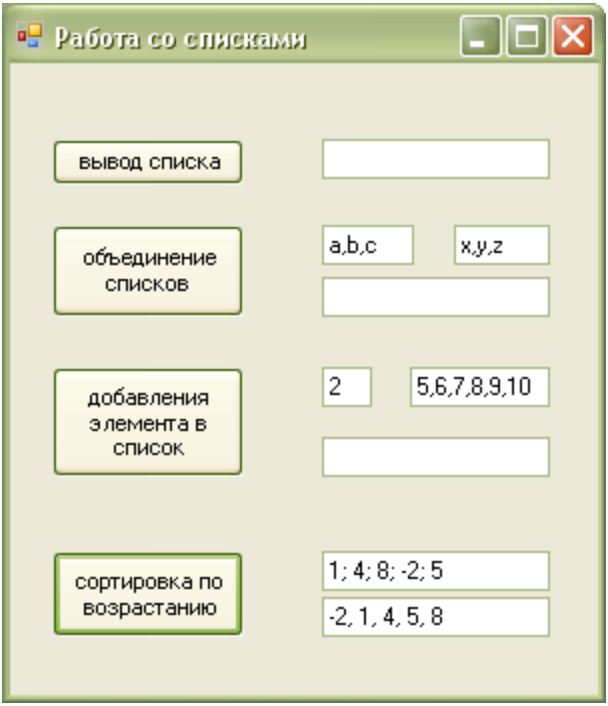


Рисунок 9.8. Результат нажатия кнопки «сортировка по возрастанию»

Задания

1. Разработать программу, которая будет выводить первый элемент списка, делящейся на 5.
2. Разработать программу, сортирующую списки.
3. Разработать программу, объединяющую несколько списков.
4. Разработать программу, которая будет искать сумму элементов в списке.
5. Разработать программу, которая будет удалять первые два элемента в списке.
6. Разработать приложение, проверяющее, удовлетворяют ли все элементы списка условию все чётные.
7. Разработать программу, которая выводит список квадратов элементов.

8. Разработать программу, которая выводит среднее значение элементов списка.

9. Разработать программу, отображающую зеркально заданный список.

10. Разработать приложение выполняющую логическую проверку над элементами списка и возвращает значение true, если какой-либо элемент удовлетворяет заданному условию.

Контрольные вопросы

1. Что такое список?
2. Способы задания списка?
3. Какие свойства поддерживает список?
4. Какие ссылки необходимо добавить в проект для работы с библиотекой WinForms?
5. Какие операции можно проводить со списками
6. Существует ли визуальный дизайнер форм в F#?
7. Бывают ли списки, в которых элементы заданы разными типами данных?
8. Как выводить списки?
9. Назвать основные функции списков?
10. Как добавляется событие на нажатие кнопки?

Список литературы

1. Bazhenov R. I., Dzikovsky F. G., Dubei O. Ya. The teaching of object-oriented approach on programming language F# // Eastern European Scientific Journal. 2014. № 2. С. 254–259.
2. Harrop J. F# for Scientists. Wiley, 2008. — 335 с.
3. Neward T., Erickson A., Crowell T., Minerich R. Professional F#2.0. Wiley, 2011. — 404 с.
4. Petricek T. Functional Programming for the Real World. Manning, 2009.
5. Pickering R. Beginning F#. A-Press, 2009. — 428 с.
6. Pickering R. Foundations of F#. A-Press, 2007. — 360 с.
7. Smith C. Programming F#. O'Reilly, 2010. — 384 с.
8. Syme D., Granicz A., Cisternio A. Expert F#. A-Press, 2007. — 609 с.
9. Лазин Е., Моисеев М., Сорокин Д. Введение в F# // Практика функционального программирования. 2010. № 5. URL: http://fprog.ru/2010/issue5/maxim-moiseev-et-al-fsharp-intro/
10. Смит К. Программирование на F#. — СПб. : Символ-Плюс, 2011. — 448 с.
11. Сошников Д. В. Видеокурс «Функциональное программирование» интернет университета информационных технологий ИНТУИТ.РУ.http://www.intuit.ru/department/pl/funcprog/
12. Сошников Д. В. Программирование на F#. — М. : ДМК Пресс, 2011. — 192 с.
13. Функциональное и логическое программирование. Курсовая работа: Методические рекомендации для студентов специальности «230105.65 Программное обеспечение вычислительной техники и автоматизированных систем» и направления «230100.62 Информатика и вычислительная техника» / Р. И. Баженов. — Биробиджан : ФГБОУ ВПО «ДВГСГА», 2012. — 27 с.
14. Шамшев А. Б., Воронина В. В. Функциональное программирование на языке F# : учебное пособие. — Ульяновск : УлГТУ, 2012. — 165 с.
15. Электронный журнал «Практика функционального программирования»: http://fprog.ru.

Оглавление

Введение 3
Лабораторная работа № 1. «Функции» 4
Лабораторная работа № 2. «Циклы» 14
Лабораторная работа № 3. «Кортежи» 22
Лабораторная работа № 4. «Списки» 27
Лабораторная работа № 5. «Библиотека WinForms» 40
Лабораторная работа № 6. «Библиотека WPF» 49
Лабораторная работа №7. «Кортежи и WinForms» 58
Лабораторная работа № 8 «Массивы и WinForms» 66
Лабораторная работа № 9. «Списки и WinForms» 79
Список литературы 88

Учебное издание

Руслан Иванович Баженов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ФУНКЦИОНАЛЬНОМУ ПРОГРАММИРОВАНИЮ

Учебно-методическое пособие

Ответственный редактор *А. Иванова*
Верстальщик *Т. Качанова*

Издательство «Директ-Медиа»
117342, Москва, ул. Обручева, 34/63, стр. 1
Тел/факс + 7 (495) 334–72–11
E-mail: manager@directmedia.ru
www.biblioclub.ru