Введение в функциональное программирование – Экзамен. Первый семестр

Олег Смирнов, Александр Полозов
oleg.smirnov@gmail.com, polozov.alex@gmail.com

16 декабря 2011 г.

Билет №1

1. **Функции высших порядков. Замыкания (2 балла).**
   Пусть имеется глобальный словарь *cache: Dictionary<'a, 'b>*, и существуют две функции работы с ним: *tryGetValue: Dictionary<'a, 'b> → 'a → bool * 'b*, достающая значение по ключу (если таковой существует в словаре, иначе возвращает false в первом элементе кортежа) и *addValue: Dictionary<'a, 'b> → 'a → 'b → unit*, добавляющая пару «ключ-значение» в словарь. Реализуйте оператор мемоизации *memo: ('a → 'b) → ('a → 'b)*, возвращающий мемоизированную версию поданной на вход функции. Мемоизированная функция сохраняет результаты всех своих вычислений и не вычисляет их повторно, если результат можно достать из кэша.

2. **Списочные комбинаторы (3 балла).**
   Реализуйте функцию $zip3 : 'a\ list \rightarrow 'b\ list \rightarrow 'c\ list \rightarrow ('a * 'b * 'c)\ list$, которая превращает три списка одинаковой длины в один список кортежей-троек:

   ```
   zip3 [1; -1; 0] [2; -2; 0] [3; -3; 0];;
   [(1, 2, 3); (-1, -2, -3); (0, 0, 0)]
   ```

   Примечание: если можно, реализуйте сразу итеративно.

3. **Списочные гомоморфизмы и технология MapReduce (6 баллов).**
   Пусть задан граф ссылок между страницами сети Интернет. На вход MapReduce подаются данные в следующем формате:

   ```
   A    B
   A    C
   B    C
   B    D
   C    B
   C    A
   C    B
   ```

   Необходимо построить список обратных ссылок, т.е для каждой страницы вывести, какие страницы ссылаются *на неё*. Например:

   ```
   A    C
   B    A, C
   C    A, B
   D    B
   ```

   Напишите решение, используя шаблоны функций *mapper* и *reducer*:

   ```
   let mapper (key: string) (value: string): seq<string *
       string> =
   // TODO
   let reducer (key: string) (values: seq<string>):
       (string * seq<string>) =
   // TODO
   ```

Билет №2

1. **Рекурсия и итерация. Хвостовая рекурсия (2 балла)**
   Напишите итеративную версию бинарного поиска в отсортированном массиве. Функция должна принимать четыре аргумента:

   - искомый элемент
   - минимальный индекс массива
   - максимальный индекс массива
   - функцию-предикат, которая проверяет, находится ли искомый элемент по заданному индексу: $isFound\ :\ int\ \rightarrow\ int\ \rightarrow\ bool$

2. **Списочные комбинаторы (3 балла).**
   Реализуйте функцию $mapi\ :\ (int\ \rightarrow\ 'a\ \rightarrow' b)\ \rightarrow\ 'a\ list\ \rightarrow\ 'b\ list$, которая преобразует один список в другой список с помощью указанной функции. В качестве первого аргумента функции передаётся номер текущего обрабатываемого элемента:

   ```
   mapi (fun i x -> (i, x)) [1; 2; 3];;
   [(0, 1); (1, 2); (2, 3)]
   ```

   Примечание: если можно, реализуйте сразу итеративно.

3. **Сканирующие пробеги, сегментированные пробеги (6 баллов).**
   Задана рекуррентная последовательность:

   $$\begin{cases} x_0 = 1 \\ x_i = i^2 x_{i-1} + C_{100}^i \end{cases}$$

   Укажите все операции, константы и покажите необходимые условия, необходимые, чтобы вычислить значения этой последовательности методом сканирующих пробегов.

Билет №3

1. **Рекурсия и итерация. Хвостовая рекурсия (2 балла)**
   Напишите итеративную версию бинарного алгоритма Евклида для поиска наибольшего общего делителя (GCD) двух чисел:

   (a) $gcd(0, n) = n$; $gcd(m, 0) = m$; $gcd(m, m) = m$

   (b) Если m, n чётные, то $gcd(m, n) = 2 * gcd(m/2, n/2)$

   (c) Если m чётное, n нечётное, то $gcd(m, n) = gcd(m/2, n)$

   (d) Если n чётное, m нечётное, то $gcd(m, n) = gcd(m, n/2)$

   (e) Если m, n нечётные и $n > m$, то $gcd(m, n) = gcd((n-m)/2, m)$

   (f) Если m, n нечётные и $n < m$, то $gcd(m, n) = gcd((m-n)/2, n)$

2. **Списочные комбинаторы (3 балла).**
   Реализуйте функцию $exists2 : ('a \to 'b \to bool) \to 'a\,list \to 'b\,list \to bool$, которая принимает функцию-предикат от двух аргументов, два списка элементов и проверяет, удовлетворяет ли любая пара из элементов данному предикату:

   ```
   exists2 (fun a b -> a = b) [1; 2; 3] [3; 2; 1];;
   true
   exists2 (fun a b -> a = b) [1; 1; 1] [0; 0; 0];;
   false
   ```

   Примечание: если можно, реализуйте сразу итеративно.

3. **Списочные гомоморфизмы и технология MapReduce (6 баллов).**
   Задан документ, состоящий из строк. Каждая строка состоит из множества слов. Необходимо построить обратный индекс, т.е. для каждого слова вывести список строк, в котором оно встречается. Напишите решение, используя шаблоны функций *mapper* и *reducer*:

   ```
   let mapper (key: string) (value: string): seq<string *
       string> =
   // TODO
   let reducer (key: string) (values: seq<string>):
       (string * seq<string>) =
   // TODO
   ```

Билет №4

1. **Функции высших порядков. Замыкания (2 балла).**
   Имея действительную функцию f, определим *сглаженную* функцию f как $x \mapsto \frac{f(x-dx)+f(x)+f(x+dx)}{3}$. Напишите оператор n-кратного сглаживания:

   $$smoothN : (float \to float) \to int \to (float \to float)$$

2. **Алгебраические типы данных (3 балла).**
   Пусть задан следующий тип данных, задающий древовидную структуру файловой системы:

   ```
   type Node =
   | File of string
   | Directory of string * Node list
   ```

   Выведите на консоль структуру дерева файловой системы с именами всех файлов и папок, по образцу:

```
|-dir1
| |-dir3
|   |-file1
|   |-dir4
|     |-file5
|     |-file6
|     |-dir5
|       |-file7
|       |-dir6
|       | |-file8
|       | |-dir8
|       |   |-file11
|       |   |-file12
|       |   |-file13
|       |   |-dir11
|       |     |-dir12
|       |       |-file16
|       |-dir7
|         |-file9
|         |-file10
|         |-dir9
|         | |-file14
|         |-dir10
|           |-file15
|-dir2
  |-file2
```

3. **Сканирующие пробеги, сегментированные пробеги (6 баллов).**
   Используя сегментированные пробеги и стандартные операции на их основе, реализуйте параллельный нерекурсивный алгоритм QuickSort (как и на лекции, рекурсия заменяется на итерации параллельной работы с несколькими сегментами).

Билет №5

1. **Функции высших порядков. Замыкания (3 балла).**
   Допустим, F# позволяет присваивать значения переменным с помощью Pascal-синтаксиса (`x := 5`). Используя замыкания, реализуйте функции $makeWeighter : (int \to bool) \to ?$. $makeWeighter$ возвращает некий «объект», который потом можно использовать с двумя другими функциями: $addValue : ? \to int \to unit$ и $getAverage : ? \to double$. Семантика поведения такова: при создании объекта в $makeWeighter$ передается предикат p. Функция $addValue$ добавляет в «базу» объекта новое переданное числовое значение-аргумент, если оно удовлетворяет этому предикату. Функция $getAverage$ позволяет в любой момент времени получить среднее элементов в базе. Пример работы:

   ```
   let weighter = makeWeighter (fun x -> x % 2 == 0)
   addValue weighter 2
   addValue weighter 13
   addValue weighter 6
   addValue weighter 8
   let x = getAverage weighter // 5.33 = (2 + 6 + 8) / 3
   ```

2. **Списочные комбинаторы (3 балла).**
   Реализуйте функцию $unzip : ('a * 'b)\ list \to 'a\ list * 'b\ list$, которая принимает список пар элементов и возвращает пару из двух списков:

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

   Примечание: если можно, реализуйте сразу итеративно.

3. **Моноидальные вычисления в деревьях (6 баллов).**
   Имеется массив данных типа int: [1; 7; 10; 2; 14; 13; 5; 5; 1; 3; 4; 42; 11; 2; 2; 9]. Постройте на них Rope, размер chunk = 2, мера $f(x) = [x$ является простым числом], моноидальная операция – XOR. Укажите все аннотации.

Билет №6

1. **Функции высших порядков. Замыкания (2 балла).**
   Напишите процедуру $makeFracFunction$ построения вычислителя цепной дроби:

$$f(x) = \cfrac{N_1(x)}{D_1(x) + \cfrac{N_2(x)}{D_2(x) + \cdots + \cfrac{N_k(x)}{D_k(x)}}}$$

   Сигнатура: $makeFracFunction : (int \to float \to float) \to (int \to float \to float) \to (float \to float)$. Она принимает на вход функции $N(i, x) = N_i(x)$ и $D(i, x) = D_i(x)$.

2. **Алгебраические типы данных (3 балла).**
   Пусть задан следующий алгебраический тип данных, описывающий арифметические выражения:

```
type Expr =
| Plus of Expr*Expr
| Minus of Expr*Expr
| Var of string
| Const of int
```

   Напишите функцию $prettyPrint : Expr \to string$, которая приводит выражение к "читабельному" виду. Функция должна расставлять скобки по мере необходимости. Пример работы:

```
prettyPrint (Plus (Const 5) (Minus (Const 7) (Const
    8)));;
"(3+(7-8))"
```

3. **Обобщение свёртки на АТД. Катаморфизмы (6 баллов).**
   Пусть задан алгебраический тип данных $Prop$, описывающий выражения логики высказываний. Тип данных включает поддержку констант, переменных, а также операций конъюнкции, дизъюнкции и отрицания.

   (a) Напишите тип данных для "контекста свёртки". Контекстом называется такой тип данных, где рекурсивные вхождения исходного типа ($Prop$ в данном примере) заменены на некоторый параметр $'a$ – вычисленные значения предыдущих свёрток:

```
type 'a propContext =
| ...
```

   (b) Напишите нерекурсивную функцию-вычислитель для выражения с использованием контекста: $eval : bool\ propContext \to bool$.

Билет №7

1. **Списочные комбинаторы (3 балла).**
   Реализуйте функцию $count : ('a \to bool) \to 'a\ list \to int$, которая принимает функцию-предикат и список элементов, и возвращает число элементов, удовлетворяющих данному предикату:

   ```
   count (fun x -> x > 0) [-1; 0; 1; 1; 1];;
   3
   ```

   Примечание: если можно, реализуйте сразу итеративно.

2. **Алгебраические типы данных (3 балла).**
   Пусть задан следующий алгебраический тип данных, описывающий выражения логики высказываний:

   ```
   type Prop =
   | True
   | Not of Prop
   | And of Prop*Prop
   | Or of Prop*Prop
   | Var of string
   ```

   Напишите функцию $prettyPrint : Prop \to string$, которая приводит выражение к "читабельному" виду. Функция должна расставлять скобки по мере необходимости. Пример работы:

   ```
   prettyPrint (Or (Var(''B''), Not(Var(''B'')))) ;;
   "(B||~B)"
   prettyPrint (And (True, True));;
   "(True&&True)"
   ```

3. **Моноидальные вычисления в деревьях (6 баллов).**
   Имеется Rope на данных типа char, длины chunk'ов неравномерны. Восемь chunk'ов – "Наша", "Таня", "громко", "плачет", "уронила", "в", "речку", "мячик". Тип данных аннотаций – числа (целые, вещественные или булевые на ваше усмотрение). Укажите такую функцию меры $f(x)$, моноидальную операцию $\circ$ и предикат p, чтобы разрезание по этому предикату произошло на пятом chunk'е; на первом chunk'е; вообще не произошло (предикат везде false).

Билет №8

1. **Рекурсия и итерация. Хвостовая рекурсия (2 балла)**
   Реализуйте рекурсивную функцию перевода системы счисления $convertBase : int \to int \to int\ list$. Задано число в десятичной системе счисления и основание целевой системы; получить список цифр числа в целевой системе.

2. **Алгебраические типы данных (3 балла).**
   Пусть задан следующий алгебраический тип данных, описывающий арифметические выражения:

   ```
   type Expr =
   | Plus of Expr*Expr
   | Times of Expr*Expr
   | Const of int
   ```

   Напишите функцию $polishPrint : Expr \to string$, которая выводит выражение в обратной польской нотации. Т.е. в таком виде, когда операнды расположены перед знаками операций. Пример работы:

   ```
   polishPrint (Plus(Times(Plus(Const(1), Const(2)),
       Const(4)), Const(3)));;
   "1 2 + 4 * 3 +"
   ```

3. **Обобщение свёртки на АТД. Катаморфизмы (6 баллов).**
   Пусть задан алгебраический тип данных, описывающий произвольный XML-документ в виде N-арного дерева. Тип данных включает поддержку элементов (тэгов), атрибутов и содержимого для тэгов. Пример документа:

   ```
   <item>
     <child attr="1">Hi!</child>
     <new_item>
       <child attr="2" another_attr="text"/>
     </new_item>
   </item>
   ```

   (a) Напишите тип данных для “контекста свёртки”. Контекстом называется такой тип данных, где рекурсивные вхождения исходного типа заменены на некоторый параметр $'a$ – вычисленные значения предыдущих свёрток:

   ```
   type 'a xmlContext =
   | ...
   ```

   (b) Напишите нерекурсивную функцию, которая преобразует XML-документ в текстовый вид, удобный для чтения: $prettyPrint : string\ xmlContext \to string$ (см. пример выше).