Модуль 1. Темы 1.1-1.3

1-1 Напишите программу, которая выяснит, какой порядок применения функций -– аппликативный или нормальный — используется в F#? То же для любого императивного языка на ваш выбор. Подумайте, для каких задач какой порядок применения будет предпочтительнее.

1-2 Напишите итеративную версию алгоритма быстрого возведения в степень по формуле

$$a^k = \begin{cases} a^{k/2} \cdot a^{k/2} & \text{если k} \vdots 2 \\ a \cdot a^{k-1} & \text{если k} \not\vdots 2 \end{cases}$$

1-3 Напишите функцию проверки числа на простоту. Реализуйте рекурсивную и итеративную версию. Хорошая скорость работы алгоритма – порядка $O(\sqrt{n})$

1-4 * Нахождение площади фигуры методом Монте-Карло заключается во вписывании фигуры в прямоугольник и последовательной генерации большого числа случайно выбранных точек из этого прямоугольника. Итоговой площадью считается доля точек, попавших в фигуру относительно их общего количества, умноженная на площадь объемлющего прямоугольника:

$$S_{\Phi} = \frac{n_{\text{внутр}}}{n_{\text{всего}}} \cdot S_{\Pi}$$

Напишите программу, вычисляющую внутреннюю площадь эллипса. В качестве генератора случайных чисел придумайте какую-то функцию, которая производит нетривиальную трансформацию своего аргумента, так что результат функции трудно предсказуем и варьируется в значительных пределах. Для реализации программы вам придется хранить начальное значение (seed) своего генератора случайных чисел, принимать от него при каждом вызове, кроме сгенерированного псевдослучайного числа, еще и новый seed. Этот seed вы должны будете хранить от вызова к вызову.

Модуль 2. Тема 2.1

2-1 (a) Неподвижной точкой функции f называется такое значение f, что $f(x) = x$. Её можно попробовать найти итеративной последовательностью приближений:

$$x, f(x), f(f(x)), f(f(f(x))), \ldots$$

Напишите процедуру поиска неподвижной точки. Найдите с её помощью значение золотого сечения ϕ.

(b) Описанный выше метод поиска неподвижной точки может начать осциллировать и не сойтись, например, если искать *sqrta* как неподвижную точку функции $x \to \frac{a}{x}$. Тем не менее, можно использовать торможение усреднением: искать неподвижную точку не функции $f(x)$, а функции $\frac{x+f(x)}{2}$. Реализуйте этот метод.

2-2 Напишите функцию fold, которая принимает список элементов, функцию-бинарный оператор *op* и начальное значение *base*, а затем применяет *op* к двум аргументам – к *base* и к первому элементу списка x_1. На следующем шаге fold применяет *op* к результату предыдущей операции и элементу x_2 и так далее, в результате вычисляя формулу:

$$op\ (\ldots(op\ (op\ base\ x_1)\ x_2)\ldots)\ x_n$$

2-3 Напишите функцию *dropWhile*, которая удаляет самый длинный префикс (начальную часть) заданного списка, состоящий из элементов, удовлетворяющих некоторому предикату. В качестве аргументов она принимает предикат и исходный список, а возвращает новый список. Например:

```
dropWhile (fun x -> x < 5) [1..10]
val it : int list = [5; 6; 7; 8; 9; 10]

dropWhile (fun x -> x > 0) [1; 2; -3; 2; 1]
val it : int list = [-3; 2; 1]

dropWhile (fun x -> x % 2 = 0) [2; 4; 1; 2; 3]
val it : int list = [1; 2; 3]

dropWhile (fun x -> x = 1) [2; 2; 1; 1]
val it : int list = [2; 2; 1; 1]

dropWhile (fun x -> x <> 0) [1..5]
val it : int list = []
```

2-4 * Напишите функцию dropWhile из упражнения 2-3, используя только функцию fold из упражнения 2-2.

Модуль 2. Тема 2.2

2-5 Реализовать списочные комбинаторы. Просто имя без комментариев означает одноименный стандартный комбинатор модуля List. *Примечание:* если можно, реализовать сразу итеративно.

(a) *init*

(b) *append* (он же @)

(c) *collect*, также известный как $flatMap$

(d) *partition*

(e) *permute*

(f) *skipWhile* – выкидывает из списка длиннейший префикс, в котором все элементы удовлетворяют заданному предикату; возвращает остальное.

```
skipWhile (fun x -> x < 3) [1..5];;
val it: int list = [3; 4; 5]
```

(g) *compressOn* — сжимает в данном списке группы подряд идущих элементов, равных согласно заданному компаратору, до одного представителя. Конкретный выбор представителей зависит от реализации. Один из вариантов работы:

```
let equalsParity x y = x % 2 = y % 2;;
compressOn equalsParity [1; 3; 4; 5; 9; 11; 6; 0;
    0];;
val it : int list = [3; 4; 11; 0]
```

(h) *pairwise* -– порождает список последовательных пар элементов.

```
pairwise [1..5];;
val it : (int * int) list = [(1, 2); (2, 3); (3,
    4); (4, 5)]
```

(i) *windowed* — обобщение *pairwise*, порождает последовательные подсписки заданной длины k.

```
windowed 3 [1..5];;
val it : int list list = [[1; 2; 3]; [2; 3; 4];
    [3; 4; 5]]
```

2-6 Написать функцию, порождающую список перестановок заданного списка в каком-либо порядке.

```
permute [1..3];;
val it : int list list = [[1; 2; 3]; [1; 3; 2]; [2; 1;
    3]; [2; 3; 1]; [3;1; 2]; [3; 2; 1]]
```

2-7 Написать оператор суперпозиции, принимающий список функций и возвращающий единственную функцию, которая принимает аргумент x и последовательно "цепочкой" применяет к нему все функции из списка, передавая выходное значение $(i-1)$-й функции на вход i-й.

2-8 Задан список списков чисел. "Вытянуть" его в один список.

```
flatten [[1..3]; [5..7]];;
val it : int list = [1; 2; 3; 5; 6; 7]
```

2-9 Реализовать функцию *flatten* для списков списков глубины вложенности более 1. Если у вас не получается, подумайте, почему.

2-10 Будем представлять матрицы по рядкам -– как списки списков одинаковой длины. Реализовать:

(a) Произведение матрицы на вектор (т.е. список).

(b) Произведение матриц.

(c) Транспонирование матрицы.

(d) * Возведение матрицы в степень k, с логарифмической идеей, как на первой лекции. Какая оценка времени работы теперь будет у алгоритма?

(e) Метод Гаусса.

2-11 Реализовать списочные комбинаторы без применения функции переворота списка:

(a) *takeWhileEnd* — возвращает длиннейший суффикс списка, в котором все элементы удовлетворяют заданному предикату.

(b) *skipWhileEnd* — выкидывает из списка длиннейший суффикс, в котором все элементы удовлетворяют заданному предикату; возвращает остальное.

Модуль 2. Тема 2.3

2-12 Пусть с помощью АТД реализовано двоичное дерево поиска (все ключи левого поддерева меньше ключа в корне; все ключи правого поддерева больше ключа в корне). Напишите для него функцию *memberOf*, которая ищет заданный элемент в дереве и возвращает ссылку на вершину с ним либо какое-то уведомление об отсутствии (например, Nil-вершину).

2-13 Выражения логики высказываний можно задать с помощью следующего АТД:

```
type proposition =
| True
| Not of proposition
| And of proposition*proposition
| Or of proposition*proposition
```

(a) Добавьте в АТД поддержку переменных.

(b) * Напишите функцию $isTautology : proposition \to bool$, которая проверяет, является ли заданое выражение тавтологией, т.е. принимает истинное значение на всех наборах значений своих переменных.

2-14 Опишите тип данных $Rope <' a,' b >$ – список, который хранит элементы двух типов, чередуя их друг за другом. Компилятор должен сам проверять каждый экземпляр Rope на корректность конструкции. Т.е. строчка

```
let value = Node("str0", Node(0, Node("str1", Node(1,
    Node(2, Nil)))))
```

скомпилироваться не должна, потому что нарушен порядок чередования.

2-15 Реализуйте для АТД *Rope* функцию *length*, которая возвращает кортеж из двух чисел: количество элементов 1-го и 2-го типа.

2-16 Для бинарного дерева реализовать следующие функции:

(a) Высота дерева.

(b) Количество листьев.

(c) Поиск элемента, удовлетворяющего предикату. В аргументах – три функции: предикат p, функция $whenFound$, которой нужно передать вершину в случае успешного поиска, и функция $whenNotFound$, которая вызывается в случае неуспешного.

(d) Правый поворот дерева.

2-17 * Реализуйте поиск удовлетворяющего элемента из п.2-16(c) итеративно, через хвостовую рекурсию.