Оглавление

[Лекция 1 Тема “Понятия итертивного и “ 08.02.2020 2](#_Toc32049868)

# Лекция 1 Тема “Понятия итеративного и декларативного программирования“ 08.02.2020

Существуют стиль программирования в повествовательном наклонении при котором программа просто совокупность утверждений этот стиль называется Декларативное программирования. Наиболее важное разновидность этого стиля это функциональные и логическое программирования.

Программирования в итеративном стили мы должны точно сообщить компьютеру как решить проблему.

В декларативном программирования мы сообщаем компьютеру что собой представляет проблема.

Алгоритм = Логика + Управления

Transform{n:nats(Vertexs[n]Vertex,indexes:[]nat<n,m:matrix):[]vertex = forench(I in indexes)

transformMatrix(m,vertex[i])

В хаскле нет переменых состояний и циклов

# Лекция 2 Тема “Рекурсивные функции и абстракции списков” 22.02.2020

Определения функций

Foo(названия)::Int(принимает) -> Int(возврашает)(типы)

Foo::(String,Int) -> Int

Foo::[Char] -> Int

Тело для первой

Foo x(формальный параметр) = x + 2

Вызываем

Foo 3(фактический параметр)

Определения функции с помощью сопоставления с образцом:

Образец

XOR::Bool -> Bool -> Bool

XOR True False(аргументы) = True(возврат)

XOR False True = True

XOR - - = False

Что может являтся образцом

1.Числа

2.Константы(True False и тд)

3.Названия параметров

4.Списки в которых элементы также являются образцами

Foo [1,-,x] = True

5.Оператор : с образцами слева и справа

Head (x:x) = x

Tail (-:xs) = xs

6.Оператор + слева образец а справа натуральное число

Bar (n + 1) = n

Bar 8

7

RepeatFirst (x:x)=(x:x:x)

RepeatFirst[1,2,3]

[1,1,2,3]

Принцип DRY –

RepeatFirst s@ (x:x) = x:s

As – образец

## Рекурсивные функции

F x = f x

F 2

Принципы работы рекурсивных функций

1.Параметр рекурсивного вызова должен быть более прост чем параметр основного вызова.

2.Имеется не рекурсивное определения для основного базового случая.

Fact::Integer -> Integer

Fact 0 = 1

Fact n = n \* Fact(n-1)

Fact (n+1) = (n+1) \* fact n

Fact s@ // не успел

Fact n = n \* Fact(n-1)

Как работает

Fact 5 - > 120

5\*fact 4 - > 4 \* fact 3

3 \* Fact 2 - > 2 \* fact 1 - > 1 \* fact 0 Вычисления не выпольнается пока не дойдем до конца

Возведения в степень

Pow::Integer -> Integer -> Integer

Pow – 0 = 1

Pow valve n =

valve x pow valve (n-1)//Небыстро

fastpow::Integer -> Integer -> Integer

fastpow – 0 = 1

fastpow valve n

if odd n then value \* fastpow value (n-1)

else r \* r

where

r = fastpow valve ( n div 2)

Существует два вида рекурсий прямая и хвостовая.

В прямой есть два прохода вниз и вверх

Хвостовая – добавления накопителя и сделать через один проход

Fact’::Integer->Integer->Integer

Fact’ 0 acc = acc

Fact’ n acc = fact’(n-1)(acc \* n)

Fact::Integer->Integer

Fact n = fatc’n 1

## Рекурсивные функции на списках

Length::[a](Любой тип)->Integer

Length [] = 0

Length (\_:xs)

1+length xs

Функция удаления элементов

Drop::Integer->[a]->[a]

Drop 0 xs = xs

Drop (n+1) [] = []

Drop (n+1) (\_:xs) = drop n xs

Функция разворота

Revers::[a]->[a]

Revers [] = []

Revers (x:xs) = revers xs ++[x]

Функция обьединения списков

(++)::[a]->[a]->[a]

[] ++ ys = ys

(x:xs) ++ ys = x:(xs ++ys)

Concat(x:xs) ys = x: concat x sys//без ++

Сортировка вставкой

Insert::a->[a]->[a]

Insert x [] =[x]

Insert x (y:ys) = if x<=y then x y:ys

Else y insert x ys

Insort::[a]->[a]

Isort [] = []

Isort (x:xs) = insert x (insort xs)

Сортировка слиянием

//Функция слияния

Merge:: Ord a=> [a]->[a]->[a]

Merge [] [] = []

Merge [] ys = ys

Merge xs [] = xs

Merge (x:xs) (y:ys) = if x <= y then x:merge xs (y:ys)

Else y:merge(x:xs) ys

//Функция сортировки

margeSort:: Ord a=>[a]->[a]

mergeSort [x] = [x]

mergeSort xs = merge (mergeSort left)

(mergeSort right)

Where

Half = (length xs) div 2

Left = take half xs

Right = drop half xs

## Взаимная рекурсия

Есть две функции и они друг от друга зависят

Even::Int -> Bool

Even 0 = True

Even (n+1) = odd n

Odd::Int -> Bool

Odd 0 = False

Odd (n+1) = even n

## Абстракции списков

[x^2|x<-[1..5]]

[1,4,9,16,25]

[(x,y)|x<-[1,2,3]

y<-[4,5]]

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

First::[(a,b)]->[a]

First ps =[x](x,\_)<-ps]

Length xs = sum[1|\_<-xs]

[x|x<-[1..10],even x]

# Лекция 3 Тема “” 07.03.2020

Опоздал

…

## Функция о работе с арифмитическими уравнениями

Data expr = num int |

Var String

Add Expr Expr

MUL Expr Expr

Instance Show Expr where

Show = formatExpr False

FormatExpr:: Bool->Expr->String//вывод

formatExpr \_ (num n) = show n

formatExpr \_(var s) = s

formatExpr True eo(ADD \_ \_ ) = “(” ++ formatExpr False ++ “)”

formatExpr \_ (Add e1 e2) = formatExpr e1 ++ “+” ++ formatExpre false e2

format Expr\_ (MUL e1 e2) = formatExpr True e1 ++ “\*” ++ formatExpr True e2

eval::Expr ->Int//вычисления

eval (num n ) = n

eval (var s) = head [n|(k,v)<-varLis,k,s]

eval (Add e1 e2) = eval e1 + eval e2

eval e1 \* eval e2

diff::Expr

diff (var s) x|x = s

diff (Add e1 e2) = Add(diff e1)(diff e2)

diff(Mul e1 e2) = Add(Mul e1(diff e2 x)

Функции вышего порядка

В ФЯ функции видут себя так же как и другии типы

1.Имеют тип

2.Могут быть результатом работы других функций

3.Могут использовать другии функции в качестве параметров

Функция использующая другии функции в качестве параметров называется функцией вышего порядка.

Quads:: Num a =>[a]->[a]

Quads [] = []

Quads (x:xs) = (x^x) quads xs

Quads::(a->a)->[a]->[a]

Quads \_[] = []

Quads f (x:xs) = (f x):quads f xs

Map:(a->b)->[a]->[a]

Map f xs = [f x]x<-xs

Filter::(a->bool)=>[a]->[a]

Filter p xs = [x|x<-xs,p x]

Карирования функций(можно подать не все параметры)

…

Композиция функций

(.)::(b->c)->(a->b)->(a->c)

f.g

читайется как f композиция с g

Это функция которая принимает аргумент x приминяет к ниму функцию g а затем к результату применяет функцию f

Odd’’ = not.even

Twice’’ f = f.f

Sumsqreven = sum.map(^2).(filter even)