Haskell Lecture 3

Recursion and More

Loops in Haskell

- *There are none*
  - *No while loops*
  - *No for loops*

Triangle Numbers

- *The nth triangle number is computed by adding the integers 1 to n.*
- *In C (or Java), implement using a for loop*
- *In Haskell, implement using Recursion*

Recursive Def. of tri(n)

- *Base*
    - *if n < 0, tri(n) = -1 -- do this for completeness*
    - *if n == 0, tri(n) = 0*
- *Recursive part*
    - *tri(n) = n + tri(n-1)*
- *See tri1.hs (guards) and tri2.hs (if then else)*

Primitive Recursion

- *Define value of function at 0 (or other base)*
- *Define how to get from value at n-1 to value at n:*
    - *fun n*
    - *| n == 0 = (base value)*
    - *| n > 0 = (expression using fcn(n-1)*
- *Above is a template*
- *In class, write function for power3, (3 raised to int power n)*

General Recursion

- *Define a function in terms of base value(s) and*
- *A recursive formula calling function with smaller value(s)*
- *Fibonacci function is an example, see fib.hs*
- *This function is not a good solution to problem*

Power of 2

- *Note that if n is even, then 2^n == (2^m)^2 where m is n / 2*
- *Note that if n is odd, then 2^n == (2^m)^2 *2 where n is 2*m+1*
- *Write power2 function using these insights*
  - *Prelude functions even and odd may be useful*

Tuples

- *A pair is a 2-tuple*
- *A triple is a 3-tuple*
- *An n-tuple has n elements*
- *In Haskell, an n-tuple has n elements, the types of each element are fixed for that defined tuple*

Naming Tuple Types

- *type Person = (String, Int) -- Person is a synonym*
- *ldSngr :: Person*
- *ldSngr = ("John", 25)*
- *See tuple.hs for example uses*

List

- *The list is an undetermined size sequence of the same type of elements*
- *square brackets define a list*
- *Haskell has many list functions defined for use*
- *list.hs has examples*

Prelude List Functions p. 127

- *: - cons operator to add element to front of list*
- *++ - concatenate two lists*
- *!! - list !! n returns element n of list*
- *concat - Make a list of lists into a list*
- *length - number of elements in list*

Prelude List Functions Cont.

- *head, last - first, last element of list*
- *tail, init - all but the first, last element of list*
- *replicate - make a list of so many copies of item*
- *take - take a number of element from front of list*
- *drop - drop a number of elements from front of list*

Prelude List Functions Cont.

- *splitAt - split list at given position*
- *reverse - reverse order of elements in list*
- *zip - make a list of pairs out of a pair of lists*
- *unzip - make a pair of lists out of a list of pairs*