List Comprehensions

CSC345: Programming Languages and Paradigms

WCU
WEST CHESTER
UNIVERSITY

Today

1. Constructing Lists
2. List Comprehensions

Constructing Lists

The empty list

- Simplest possible list
- Other lists are built up from the empty list using the cons operator

```
(:)
```

- Cons takes an element and a list and produces a new list with the element prepended to the front

cons examples

```
ex1, ex2, ex3 :: [Int]
ex4 :: Bool

ex1 = 1 : []
ex2 = 5 : (1 : [])
ex3 = 7 : 5 : 4 : []
ex4 = [1,2,3] == 1 : 2 : 3 : []
```

Right Associativity

List Comprehensions

- “Distinct feature of a functional language”
- Recent adoption by python/java (other paradigms)

Big Idea: description of a list in terms of the elements of another

Parts:

1. Generator: a list that elements are drawn from
2. A test or guard (optional): used for *filtering*
   if True then keep the element
   If False then discard the element
3. Transformation: to form the elements of the result set; used for *mapping*

A Series of Examples

- The notation is a list comprehension is supposed to resemble set notation in mathematics

$$\{x \in \mathbb{N} \mid 2 < x < 20\}$$

Example 1

- “for each x drawn from [1,2,3], return x * x”

```
[x * x | x <- [1,2,3]]
```

Result:

[1 * 1, 2 * 2, 3 * 3]

[1,4,9]

Example 1a

```
ex = [2,4,7]

[2 * n | n <- ex]
```

Example 2

Mapping w/ a function call

- The function toLower is built-in to Haskell
- toLower :: Char -> Char
- Also remember that a String is really just a [Char]

```
[toLower c | c <- "Hello World!"]
```

Example 3

Creating a list of pairs of type (Int, Bool)

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

Example 4

Combining a generator with one or more tests

```
[ x | x <- [1,2,3], odd x]
```

Example 4a

Can have multiple tests

```
ex = [1..10]

[ 2 * n | n <- ex, even n, n > 3]
```

Example 5

- Can use list comprehensions in the definition of a function

- Write a function `addOrdPairs` that returns the sum of only those pairs whose first item is less than or equal to the second item

Example usage:

```
addOrdPairs [(2,3), (2,1), (7,8)]
```

[5,15]

Example 6

- Write a function `allEven` that takes a list of int's and returns whether every item is even

isPrime :: Int -> Bool

- Checks whether a positive int is prime
- *Prime definition:* a prime number n is a number whose only divisors are 1 and n

Example 7

Can have two generators

```
[ (x,y) | x <- [1,2,3], y <- [4,5] ]
```

Example 8

- The list of all possible order pairings of elements from the list [1..3]

```
[(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)]
```

Lookup Table Example

- Function `find` that returns the list of all values that are associated with a given key

Example usage:

```
> find 'b' [('a',1), ('b',2), ('c',3), ('b',4)]
[2,4]
```