Haskell Types

COMP360

“No computer has ever been designed that is ever aware of what it's doing; but most of the time, we aren't either.”

Marvin Minsky

Haskell Programming Assignment

- A Haskell programming assignment has been posted on Blackboard
- You have to write 8 simple Haskell functions
- Upload one .hs file to Blackboard by noon on Tuesday, April 4, 2017

Comments

- In Haskell comments start with two dashes and run until the end of the line

```
-- This is a comment
```

- Please put comments in all Haskell files you submit giving your name and a brief explanation of the method

Tuples

- Tuples are like list but they can have different data types
- Tuples are written with parenthesis

(5,"cow") (1,"aardvard")

- When using multiple tuples, they must have elements of the same type

Tuple Functions

- **fst** – returns the first element of a tuple
- **snd** – returns the second element of a tuple
- **zip** – given two lists, returns a list of tuples with elements from the first list matched with the second list

  zip [1,2,3] ['X', 'Y', 'Z'] returns [(1,'X'),(2,'Y'),(3,'Z')]

Data Types

- Haskell is strongly typed. Each variable and function has a specific data type
- You do not have to declare data types. Haskell infers the data type by how it is used
- You can specify the types in Haskell and often it makes your intentions clearer to the compiler

Usual Simple Data Types

- Int – whole numbers
- Integer – whole numbers that can be very large
- Double – floating point numbers
- Bool – true or false
- Char – character

- Datatype names all start with a CAPITAL letter

List and Tuples

- Both lists and tuples define types
- [Char] is a list of Char and can be called String
- A tuple specifies the type of all elements
- (Int, Char) is a tuple that has an int as the first value and a single character as the second, i.e. (5, ‘Z’)

Write a Haskell Function

- Write a simple Haskell function to reverse a list without using the built-int reverse function

myrev "Williams" will return "smailliW"

Defining Function Types

- You can define the type of a function and its parameters

```
myReverse :: [a] -> [a]
myReverse lst = if length lst == 1 then etc.
```

- This specifies that myReverse takes a list of objects and returns a list of the same type of objects

Type Inference

- The Haskell system is very good at determining the data type you intended for your program
- If your program does arithmetic with a parameter, Haskell recognizes that the parameters must be numbers

Generic Type Definitions

- You can specify the type of a function without specifying the type of the parameters
- The type specification of the built-in tail function is

```
tail :: [a] -> [a]
```

- The letter "a" is a type variable
- Type variables are usually one letter
- For tail, "a" represents an undefined type. It will work for any type
- tail is polymorphic

Haskell Practice

- Write a Haskell function that takes two arrays of numbers and returns the sum of the each pair of elements multiplied together

$$a1 * b1 + a2 * b2 + a3 * b3 + \textit{etc.}$$

```
dot :: [a] -> [a] -> a
```

Typeclasses

- A typeclass groups the different types based on functionality
- All types in a typeclass can perform a specific action

Some Haskell Typeclasses

- **Eq** is used for types that support equality testing
- **Ord** is for types that have an ordering.
- **Show** can be presented as strings
- **Read** takes a value out of a string
- **Num** are numbers
- **Integral** are whole numbers

Defining Function Types with Typeclasses

- You can specifically define the classtype of a parameter

```
sumsqr :: Num a => [a] -> a
sumsqr lst = if length lst == 0 then etc.
```

- The typeclass specified to the left of the => is called a *class constraint*
- The type definition states that the function takes a list of numerical values and returns a number

Write Haskel Type Specification

- Write Haskel type specification for these functions

trim – Remove leading spaces from a string

count - Count the occurrence of a character is a string

Possible Solutions

- Write Haskel type specification for these functions

trim – Remove leading spaces from a string

```
trim :: [Char] -> [Char]
```

count - Count the occurrence of a character is a string

```
count :: (Eq a1, Num a) => [a1] -> a1 -> a
```

Checking Types

- The **:t** command in GHCi will give you the type of data or a function

```
:t "Sample stuff"
"Sample stuff" :: [Char]
:t trim
trim :: [Char] -> [Char]
```

Practice

- Create a Haskell function to sum a list of list of numbers

sumList [[4,5,6] , [1,2]] returns 18

```
sumList :: Num a => [[a]] -> a
```

Functions as Parameters

- A function can be passed to another function as a parameter
- Consider this function which does anything to a list

```
univ  lst  func = func lst
```

then

```
univ [2,1,7] length   returns    3
univ [2,1,7] sum      returns  10
univ [2,1,7] sumsqr returns   54
```

Haskell Patterns

- You can write multiple definitions of a function
- Haskell will consider them in order and execute the first on that fits
- You can put constants as the parameters in the definition. If you call the function with that value, it will execute that function definition

Fibonacci Pattern Example

- You might have written the Fibonacci function as

```
fib n = if n <= 2 then 1 else fib (n-1) + fib (n-2)
```

- It can also be written as

```
fib 1 = 1
fib 2 = 1
fib n = fib (n-1) + fib (n-2)
```

Empty List Pattern

- You can specify an empty list as a parameter. The definition will be use if it is executed with an empty list

```
mySum :: Num a => [a] -> a
mySum [] = 0
mySum str = head str + mySum (tail str )
```

Haskell Programming Assignment

- A Haskell programming assignment has been posted on Blackboard
- You have to write 8 simple Haskell functions
- Upload one .hs file to Blackboard by noon on Tuesday, April 4, 2017