Programming Assignment #2 – Due Monday, 27 April 2009

Problem Statement:
For this programming assignment, you will gain some more experience programming in Haskell using modules, lists, list functions, list comprehensions, simple data types, higher-order polymorphic functions, and simple type classes.

Simple Functional DNA Computing in Silica:
DNA consists of four bases, Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). A sequence of bases forms a single strand of DNA, e.g., [A,T,A,G,G,C] is such a sequence. These bases may chemically bond to form base pairs according to a pairing relation known as Watson-Crick complementarity: (A,T), (T,A), (C,G), (G,C). Two strands can combine to form the well-known double-helix if the bases in a sequence (or subsequence) of a given strand are Watson-Crick complements of another sequence in a strand. Given a strand of DNA that we call a template, we can manufacture its complement strand by walking the sequence from one end to the other generating the corresponding complement base in the complement strand for each base in the template strand. For example, given the template strand [A,T,T,A,G,G,C], the complement strand is [T,A,A,T,C,C,G]. Given two strands that are perfect complements, we can combine them to form a double-helix: [(A,T),(T,A),(T,A),(G,C),(G,C),(G,C),(C,G)].

Module – define a Haskell module FunDNA that will contain all of the code for this assignment and export the public data types, data constructors and functions.

Datatypes – define Haskell types for each of Base, BasePair, Strand, Helix and Tube deriving appropriate type classes if necessary implementing instances of those type classes (e.g., Eq, Enum, Ord, Bounded, Read, Show). Consult the Haskell Standard Prelude documentation for definitions of these type classes and examples on how to define your own instances. A Tube consists of Strands that are always kept sorted.

Random Base – an instance of the Random type class (import the Random module from the Haskell library) for the Base type that works with a function randSequence that generates an infinite strand of bases chosen pseudo-randomly given an integer seed value. Also create a function probSequence that generates an infinite sequence based on a list of pairs giving the probability of occurrence of each base. For example, the list of pairs: [(A, 0.25), (T, 0.45), (C, 0.25), (G, 0.05)].

DNA Computing Functions – implement the following functions that operate on the data types previously described:

toBase :: Char -> Base. Convert a base letter to a base type.
fromBase :: Base -> Char. Convert from a base type to a letter.
toStrand :: String -> Strand. Convert from a String of base letters to a Strand data type.
fromStrand :: Strand -> String. Convert from a Strand to a String.
isWcc :: Base -> Base -> Bool. Check if two bases are equal.
wcc :: Base -> Base. Compute the Watson-Crick complement of a base.
wccStrand :: Strand -> Strand. Compute the Watson-Crick complement of a Strand.
wccHelix :: Strand -> Helix. Given a Strand, generate a Helix.
makeHelix :: String -> Helix. Given a String of base letters, make a Helix.
templateStrand :: Helix -> Strand. Given a Helix, extract the template Strand.
complementStrand :: Helix -> Strand. Given a Helix, extract the complement Strand.
flipStrand :: Strand -> Strand. Reverse the order of bases.
willAnneal :: Strand -> Strand -> Bool. Determine whether two strands will perfectly anneal (i.e., every base is a wcc).
perfectAnneal :: Strand -> Strand -> Helix. Given two strands that are WCC, create the corresponding Helix, else the empty Helix.
prefixAnneal :: Strand -> Strand -> Helix. Anneal two Strands up to the first pair of bases that are not wcc.
replicateStrand :: Int -> Strand -> Tube. Replicate a Strand n times returning a Tube of identical Strands.
randSequence :: Int -> Strand. Generate a random sequence of bases.
probSequence :: [(Base, Float)] -> Strand. Generate a random sequence of bases given a list of probability distributions for each base.
filterTube :: Tube -> [Strand] -> Tube. Given a Tube and a list of Strands, generate a new Tube that does not include the Strands matching any of the Strands in the filter list.

filterLength :: Integer -> Tube -> Tube. Given a length and a Tube, extract all Strands of that length into a new Tube.

strandLengths :: Tube -> [Integer]. Given a Tube, generate a list of the lengths of all Strands in the tube (gel electrophoresis).

Here are some examples of how these functions are used:

FunDNA> toBase 'A'
A
FunDNA> fromBase A
'A'
FunDNA> toStrand "ACTG"
[A,C,T,G]
FunDNA> fromStrand [A,C,T,C]
"ACTG"
FunDNA> wcc A
T
FunDNA> wcc G
C
FunDNA> isWcc A T
True
FunDNA> isWcc A C
False
FunDNA> wccStrand [A,T,C,G]
[T,A,G,C]
FunDNA> wccHelix [A,T,C,G]
[(A,T),(T,A),(C,G),(G,C)]
FunDNA> makeHelix "ACTG"
[(A,T),(C,G),(T,A),(G,C)]
FunDNA> willAnneal [A,C,T,G] [T,G,A,C]
True
FunDNA> willAnneal [A,C,T,T] [T,G,A,C]
False
FunDNA> perfectAnneal [A,C,T,G] [T,G,A,C]
[(A,T),(C,G),(T,A),(G,C)]
FunDNA> perfectAnneal [A,C,T,G,T] [T,G,A,C,G]
[]
[(A,T),(C,G),(T,A),(G,C)]
FunDNA> replicateStrand 3 [A,C,T,G]
FunDNA> replicateHelix 2 $ makeHelix "ATCG"
FunDNA> filterLength 4 someTube
[all sequences of length 4 in someTube]

Submission Requirements:
You are to hand in at the start of class on the due date a complete source code listing of all source files with your name and email address in a comment line at the top of each file. Use the enscript command to print your listing: enscript -2Gr -C -Ehaskell FunDNA.hs. You are also to submit a gzipped electronic copy of the source file(s) for the program to the course TA via email. Please use the filename: cs209-program-02-firstname.lastname.