[Important: Do not use existing higher order functions, list comprehension, or any other advanced features of Haskell not covered in class prior to release of the assignment. You must provide signature for every function. Whenever meaningful, use type variables instead of types. Finally, ensure that the parameters passed to the functions are of the correct types and are passed in the correct order.]

Part 1 Lambda Calculus and Currying

Total: 20 Points

[Note: Use "\" for the lambda character in your answers]

Problem 1 [10 Points]. Using the definitions of boolean constants and operators presented in class, show that the following evaluates to false. Show all your steps for a perfect mark. Always evaluate the "outer" applications first (i.e., use lazy evaluation), but continue to evaluate further until you obtain false.

```
and true (not true)
```

Problem 2 [5 + 5 Points]. Define the higher-order library function curry that converts a function on pairs into a curried function, and, conversely, the function uncurry that converts a curried function with two arguments into a function on pairs.

Part 2 Type System

Total: 40 Points

Problem 3 [40 Points]. Declare a data type MyFloat -- for handling floating point numbers -- where each number is a (mantissa, exponent) pair of integers representing the floating point number mantissa x 10exponent, so that a decimal point is assumed to exist just to the left of the leftmost digit of mantissa. For example, (329, 23) would represent 0.329 x 1023. Both the mantissa and the exponent can be positive or negative.

Carefully pick Haskell's built-in type classes which this type should be an instance of. You should define (and when possible overload) simple arithmetic and comparison operations on these floating numbers (at least: *, /, +, -, negate (negation), <=, >=, <, >, ==). Also define functions whole and fraction to extract the part of the represented number to the left of the decimal point (as an Int) and to the right of the decimal number (as a standard Haskell floating point number), respectively. For example, for (329, 2), whole should return 32, and fraction should return 0.9.

Part 3 Lists

Total 40 Points

Problem 4 [10 Points]. Write a polymorphic function, shuffle, which takes two lists l1 and l2 representing decks of cards, and returns the perfectly shuffled contents of the lists. In other words, the returned list contains the first element of l1, followed by the first element of l2, followed by the second element of l1, and so on. If one of the lists ends before the other, the other list’s remaining elements are simply added to the end.

Problem 5 [5 Points]. Write a polymorphic function, split, which takes as parameters a list and an Integer n, indicating the splitting point. It splits the contents of the provided list into two lists, the first with the first n elements of the list (in order), and the second with the remaining elements (again, in order). These two lists are then returned as a pair.

Problem 6 [15 Points]. Write a function, nshuffle, which takes two integers c and n. It first generates two lists, each of size c. One list contains c instances of the character ‘b’ (for black) and the other contains c instances of the character ‘r’ (for red). Then, it carries out n number of perfect shuffles, splitting each shuffled list into two equally sized lists before the next shuffle. For the shuffling and splitting, you must use functions shuffle and split written for the previous two problems. The function returns the final outcome of the n shuffles in the form of a single list. You may find it useful to define local functions for parts of this computation.

Problem 7 [10 Points]. Write a function, consecutive, which takes a list of characters, and returns an integer indicating the largest number of consecutive identical characters in the list.

Submission:

Create a directory with your nsid as its name.

For Problem 1, put your answer in a text file named lambda.txt and place it in the directory.

For the remaining problems, place your functions in a Haskell source file called programs.hs. Create a second text file programs.txt to show transcripts of your testing of the functions. Place both files in the directory.

Once you have everything in your directory, create a zip file for the entire directory. If your nsid is <your_nsid> name the zip file <your_nsid>.zip. When opened, it should create a directory called <your_nsid>.

You may submit multiple times before the deadline, so you are advised not to wait till the last minute to submit your assignment to avoid system slowdown. You are encouraged to submit completed parts of the assignment early. Late submissions will not be accepted.t