

## CS 357: Declarative Programming Homework 6 (Spring '09)

The following problems should be done in Haskell. All functions should have a type declaration identical to the ones listed below. You are expected to test your code on *at least* the test case(s) given with each problem. If your functions don't have the type declaration (with the matching type) or fail on any of the provided test case(s), that problem will receive *no credit*.

1. Write a function *numbers* which takes a list of integers as its argument and returns the integer which has those numbers as digits. For example,

```
*Main> :t numbers
:t numbers
numbers :: [Int] -> Int
*Main> numbers [1..4]
1234
```

Write *numbers* using a tail-recursive helper function which should be defined inside of a *let* expression or using *where*.

2. The function *fib* takes an integer *n* as its argument and returns the *n*-th element of the Fibonacci series. Write *fib* using a tail-recursive helper function defined inside a *let* expression or using *where*.

```
*Main> :t fib
:t fib
fib :: [Int] -> [Int]
*Main> fib 10
89
```

3. The function *deleteDuplicates* takes a list as its argument and returns a list with all duplicate elements deleted. For example,

```
*Main> :t deleteDuplicates
:t deleteDuplicates
deleteDuplicates :: (Eq a) => [a] -> [a]
*Main> deleteDuplicates [1,2,3,4,5,1,2]
[1,2,3,4,5]
*Main> deleteDuplicates "It was the best of times"
"It washebofim"
```

4. The function *select* takes a predicate and two lists as arguments and returns a list composed of elements from the second list in those positions where the predicate, if applied to the element in the corresponding position of the first list, would return *True*.

```

*Main> :t select
:t select
select :: (t -> Bool) -> [t] -> [a] -> [a]
*Main> select even [1..26] "abcdefghijklmnopqrstuvwxy"
"bdfhjlnprtvxz"
*Main> select (<= 'g') "abcdefghijklmnopqrstuvwxy" [1..26]
[1,2,3,4,5,6,7]

```

5. The function *permutations* takes a list as its argument and returns a list of lists representing the permutations of the elements of its argument. For example,

```

permutations "abc"
["abc", "acb", "bac", "bca", "cab", "cba"]
*Main> :t permutations
:t permutations
permutations :: (Eq a) => [a] -> [[a]]
*Main>

```

6. Without using the *reverse* function from the prelude, write a function *palindrome* which returns *True* if its list argument equals its reverse and *False* otherwise. For example,

```

*Main> :t palindrome
:t palindrome
palindrome :: (Eq a) => [a] -> Bool
*Main> palindrome "abba"
palindrome "abba"
True

```

7. The Goldbach conjecture states that any even number greater than two can be written as the sum of two primes. Using list comprehensions, write a function *Goldbach*, which given an even number *n* returns a list of pairs of prime numbers which sum to *n*. Note: You will have to write a function which tests an integer for primality and this should be written as a list comprehension also. For example,

```

*Main> goldbach 6
[(3,3)]
*Main> :t goldbach
:t goldbach
Int -> [(Int,Int)]
*Main>

```