

Introduction to Programming I

Mid-Semester Examination, Semester I

1st October, 2004

Duration: 3 hours

Marks: 25 marks

Note: Whenever a question asks for a Haskell function, write down the most general type of the function before the actual function definition, including any dependency on type classes.

1. Write a Haskell function *threebest* that takes as an input a list l and returns a list containing the three largest elements in l . The list l may have duplicate values, so by the “three largest elements” we mean the first three elements that appear when l is sorted in descending order. For instance, *threebest* [10, 3, 9, 7, 9] is [10, 9, 9]. The function should take time $O(n)$ where n is the length of the list.
(4 marks)
2. Define the following functions in Haskell.
 - *applyEach*, that takes a list of functions $[f_1, f_2, \dots, f_n]$ and applies each of them to a given value v , returning the list of values $[f_1(v), f_2(v), \dots, f_n(v)]$. For instance, assuming *factorial* is already defined:
applyEach [(+3),factorial]5 \Rightarrow [8, 120].
 - *applyAll* that takes a list of functions $[f_1, f_2, \dots, f_n]$ and a value v and returns the value $f_1(f_2(\dots(f_n(v))\dots))$. For instance:
applyEach [(+3),factorial]5 \Rightarrow 123.(4 Marks)
3. We can write *concat* in terms of ++ using either *foldl* or *foldr* as follows:
 - *concatl* = *foldl* (++) []
 - *concatr* = *foldr* (++) []

Which of the following is preferable? Why?
(2 marks)

4. Write a function *endgame* that takes as input an integer n and returns the first n positive integers whose last digit is 3 but which are not multiples of 3. For instance:
endgame 7 \Rightarrow [13, 23, 43, 53, 73, 83, 103]
 (2 Marks)

5. Consider the following function that computes x^n :

power :: *Float* -> *Int* -> *Float*

power *x* 0 = 1.0

power *x* *n* = *n**(*power* *x* (*n*-1))

For inputs x and n , this function calls itself $O(n)$ times. To reduce the number of recursive calls, we can observe that when n is an even number, $2k$, then x^{2k} can be rewritten as $(x \cdot x)^k$, and when n is odd, $n = 2k + 1$, then $x^n = x \cdot (x \cdot x)^k$.

- Write an improved function *fastpower* to compute x^n using these observations.
- How many times does *fastpower* call itself as a function of its inputs as x and n ?

(4 Marks)

6. Explain whether the following Haskell definitions are equivalent:

times1 :: *Int* -> *Int* -> *Int*

times1 0 _ = 0

times1 0 _ = 0

times1 *nm* = *n*+(*times1* *n*(*m* - 1))

times2 :: *Int* -> *Int* -> *Int*

times2 0 _ = 0

times2 _ 0 = 0

times2 *nm* = *m*+(*times2* (*n* - 1)*m*)

(3 Marks)

7. A collection of non-empty lists l_1, l_2, \dots, l_k is said to be a partition of a list l if $l == l_1 ++ l_2 ++ \dots ++ l_k$. Write a Haskell function *partition* that takes as input a list l and returns all partitions of l .

(3 Marks)

8. The built-in function *concat* “dissolves” one level of brackets in a list. Suppose we want to extend *concat* to a function *flatten* that dissolves all but the outermost level of brackets in a list. Here are some examples of how *flatten* should work:

• *flatten* [1, 2, 3, 4, 5] = [1, 2, 3, 4, 5]

• *flatten* [[1, 2], [], [3, 4, 5]] = [1, 2, 3, 4, 5] = *concat* [[1, 2], [], [3, 4, 5]]

• *flatten* [[[[]], [1, 2]], [[]], [[3, 4], [5]]] = [1, 2, 3, 4, 5] = *concat* (*concat* [[[[]], [1, 2]], [[]], [[3, 4], [5]]])

As the third example suggests *flatten* can be thought of as a repeated application of *concat*.

Is it possible to define *flatten* in Haskell? Explain your answer.
(3 marks)
