The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, SPRING SEMESTER 2011-2012

FUNCTIONAL PROGRAMMING

Time allowed TWO hours

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced.

Answer QUESTION ONE and THREE other questions

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

ADDITIONAL MATERIAL: Haskell Standard Prelude

Question 1 (Compulsory)

Select ONE answer for each section.

There is no negative marking for incorrect answers.

a) The expression `["False","True"]` has type:

i) `[a]`
ii) `[Bool]`
iii) `[String]`
iv) `[Bool,Bool]`
v) `[String,String]` (3 marks)

b) Which of the following is an invalid list in Haskell:

i) `[[1,2,3,4]]`
ii) `[1,[2,3],4]`
iii) `[[1,2],[3,4]]`
iv) `[[1],[2,3],[4]]`
v) `[[1],[2],[3],[4]]` (3 marks)

c) Evaluating `[(x,y) | x <- [1,2], y <- [1,2]]` gives:

i) `([1,2],[1,2])`
ii) `[(1,2),(1,2)]`
iii) `[(1,1),(2,2)]`
iv) `[(1,1),(1,2),(2,1),(2,2)]`
v) `[(1,1),(2,1),(1,2),(2,2)]` (3 marks)

d) A function of type `(Int -> Int) -> Int`:

i) Takes a function as its argument
ii) Takes two arguments one at a time
iii) Takes a pair of arguments
iv) Returns a function as its result
v) Returns a pair of results (3 marks)

e) Evaluating `sum [x | x <- [1..10], even x]` gives:

i) An error
ii) `10`
iii) `25`
iv) `30`
v) `55` (3 marks)

f) Evaluating `zip [1,2] ['a','b','c']` gives:

i) An error
ii) `([1,2],['a','b','c'])`
iii) `[(1,'a'),(2,'b')]`
iv) `[(1,'a'),(2,'b'),(2,'c')]`
v) `[(1,'a'),(2,'b'),(3,'c')]` (3 marks)

g) Which of the following statements about Haskell is true:

i) Function application brackets to the left
ii) All programs are guaranteed to terminate
iii) All lists must be of finite length
iv) Recursive functions must have a base case
v) Type errors can occur at run-time (3 marks)

h) The expression `Node (Leaf 1) (Leaf 2)` is a value of the datatype:

i) `data Tree = Node | Leaf Int`
ii) `data Tree = Leaf Int | Node Int Int`
iii) `data Tree = Leaf Tree | Node Int Int`
iv) `data Tree = Leaf Int | Node Tree Tree`
v) `data Tree = Leaf Tree | Node Tree Tree` (4 marks)

Question 2:

a) What are the types of the following expressions? (5 marks)

```
"Haskell"
[False,True,False]
(False,'a')
(['a','b'],[False,True])
filter
```

b) Define the following library functions using recursion: (8 marks)

```
product :: [Int] -> Int
length  :: [a] -> Int
reverse :: [a] -> [a]
map     :: (a -> b) -> [a] -> [b]
```

c) Using the definition

```
fac   :: Int -> Int
fac 0  = 1
fac n  = n * fac (n-1)
```

show how the expression `fac 3` is evaluated. (4 marks)

d) Given the alternative definition

```
fac       :: Int -> Int
fac n      = accum 1 n

accum     :: Int -> Int -> Int
accum x 0  = x
accum x y  = accum (x*y) (y-1)
```

in terms of an auxiliary function `accum` that uses an extra argument to accumulate the result, show how `fac 3` is now evaluated. (4 marks)

e) Explain using your answers to the two previous parts why the original definition for `fac` is inefficient in terms of memory usage, and how the alternative definition resolves this problem. (4 marks)

Question 3:

a) Give concise English definitions for the following terms: (5 marks)

Recursive function

Curried function

Higher-order function

Polymorphic function

Overloaded function

b) Give one example function definition from the Haskell Standard Prelude for each of the five terms that are listed above. You may not use the same example function for more than one answer. (5 marks)

c) The sum of the integers between 1 and 100 can be expressed using the list comprehension notation as `sum [x | x <- [1..100]]`. Express each of the following using a list comprehension: (8 marks)

Sum of the squares of the integers between 1 and 100

Product of the even integers between 1 and 100

List of all pairs of integers between 1 and 100

Sum of the products of all pairs of integers between 1 and 100

d) Suppose you are given a function `divides :: Int -> Int -> Bool` that decides if one integer is divisible by another, e.g. `divides 15 2` is `False` and `divides 15 3` is `True`. Using a list comprehension, define a function `divisors :: Int -> [Int]` that returns the divisors of a natural number. For example, `divisors 15` should give `[1,3,5,15]`. (4 marks)

e) A natural number greater than one is *prime* if its only divisors are one and itself. Using `divisors`, define a function `prime :: Int -> Bool` that decides if a natural number is prime or not. (3 marks)

Question 4:

Suppose that you have been invited to write an article for a professional computing magazine on the *benefits that Haskell brings to programmers.* Write a short article on this topic, illustrating each of the benefits that you mention with a small example written in Haskell. (25 marks)

Question 5:

a) Complete the missing parts `???` in the following definition for a recursive function that inserts an integer into the correct position in a sorted list. For example, `insert 3 [1,2,4,5]` should give `[1,2,3,4,5]`. (6 marks)

```
insert          :: Int -> [Int] -> [Int]
insert x []      = ???
insert x (y:ys) = if x <= y then
                     ???
                  else
                     ???
```

b) Show how your definition evaluates `insert 3 [1,2,4,5]`. (4 marks)

c) Using `insert`, complete the following definition for a recursive function that sorts a list of integers using *insertion sort*, in which the empty list is already sorted, and any non-empty list is sorted by inserting the head into the list that results from sorting its tail. (4 marks)

```
isort         :: [Int] -> [Int]
isort []       = ???
isort (x:xs)   = ???
```

d) Show how your definition evaluates `isort [3,2,1]`. There is no need to show how each application of `insert` is evaluated. (5 marks)

e) Complete the missing parts in the following definition for a recursive function that implements *quicksort*, in which the empty list is already sorted, and any non-empty list is sorted by sorting the tail values `<=` the head, sorting the tail values `>` the head, and appending the resulting sorted lists on either side of the head value. (6 marks)

```
qsort         :: [Int] -> [Int]
qsort []       = ???
qsort (x:xs)   = qsort ??? ++ ??? ++ qsort ???
```