UNIVERSITY COLLEGE LONDON
University of London

EXAMINATION FOR INTERNAL STUDENTS

For the following qualifications

B. A. , B. Sc.

COURSE CODE : COMP B11A

TITLE OF EXAMINATION

CompSci B11A: Introductory Programming I -- RESIT PAPER 1999

UNIT VALUE : 0.50

TIME ALLOWED : 2 hours 30 minutes

Answer Question 1 and Question 2 and any TWO other questions.

The use of Electronic Calculators is NOT permitted.

Answer Question 1 and any TWO other questions.

You may find the following definitions useful throughout this paper:

```
id x = x

const x y = x

converse f x y = f y x

length [] = 0
length (front : rest) = 1 + (length rest)

append a [] = a
append [] b = b
append (front : rest) b = front : (append rest b)

filter pred [] = []
filter pred (front : rest)
         = front : filter pred rest, if pred front
             = filter pred rest, otherwise
```

1. (a) What is a recursive function? Give example function definitions in Miranda for both a stack recursive function and an accumulative recursive function. For each function, provide an example application and give a hand evaluation of that application.

[7]

(b) What is a recursive type? Give an example of a built-in recursive type in Miranda. How can you define your own recursive type? Give an example in Miranda of a user-defined type which is recursive.

[5]

(c) Explain, with examples, what is meant by the following terms:

higher-order function

currying

partial application

Explain how the above terms are related.

[10]

[Question 1 cont. over page]
[TURN OVER]

[Question 1 cont.]

(d) Evaluate each of the following Miranda expressions. If you think that an expression gives an error, say why (if there is more than one error in an expression, explain all of them):

```
[ [ ] ] : [ [ [ ] ] ]

[ [ [ ] ] ] : [ [ ] ]

[ ] : [ ] : [ ]

((3-3)=0) & ((23 / 0) = 0)

((3-3)=0) \/  ((23 / 0) = 0)

exp1
where
exp1 = 25, if (3 < 5 < 27)
     = False, otherwise

exp2 5
where
exp2 = 3, if True
     = 5, otherwise

exp3 5
where
exp3 = id, if False
     = const 3, otherwise
```

[11]
[Total 33]

[CONTINUED]

2. Here are an auxiliary function “rcons” and three versions (“rev1”, “rev2”, and “rev3”) of a function which takes a list of items and produces a list with the same items in reverse order:

```
rcons a f b = f (a : b)

rev1 [] = []
rev1 (front : rest) = append (rev1 rest) [front]

rev2 items = foldr rcons id items []

rev3 = foldl (converse (:)) []
```

Give the type definitions for rev1, rev2 and rev3 (use polymorphic types where appropriate to give the most general type).

Provide hand evaluations for the following three applications:

```
(rev1 [1,2,3,4])

(rev2 [1,2,3,4])

(rev3 [1,2,3,4])
```

Which of the three versions is the slowest (i.e. takes the most evaluation steps)?

[Total 33]

3. (a) What are algebraic data types? Give examples of the different kinds of algebraic type and how they might be used. [5]

(b) Define a type structure to represent binary trees in which the nodes of the tree hold number values and the leaves also hold number values. [4]

(c) Define a function to find the height of a tree represented using your type, where the height of a tree is the number of nodes along the longest branch from the root to a leaf. [9]

(d) Consider the following function defined for lists. Define an analogous function on the trees represented by your type, where a function is applied to every sub-tree within a tree.

```
map_on_tails f [] = []
map_on_tails f xs = (f xs) : (map_on_tails f (tl xs))
```

[11]

(e) Define a function which will take a tree and return a tree containing at each node the height of the corresponding sub-tree in the input tree. [4]

[Total 33]

[TURN OVER]

4. (a) Provide definitions, including types, for the two functions (from the Miranda Standard Environment) called `foldr` and `foldl`. [12]

(b) What values do the following five expressions compute?

```
foldr (:) [] [1,2,3]

hd (foldr (:) [] [1..])

foldl (:) [] [1,2,3]

foldl (swap (:)) [] [1,2,3]
where
swap f x y = f y x

foldl (swap (:)) [] [1..]
where
swap f x y = f y x
```

[5]

(c) Under what circumstances are the functions `foldr` and `foldl` interchangeable? [8]

(d) What does the following function do and what is its type?

```
scan op = g
          where
          g r = (r:). rest
                where
                rest []    = []
                rest (a:x) = g (op r a) x
```

[8]

[Total 33]

[CONTINUED]

5. (a) What is the type of the following expression and what does it do?

```
(foldr (+) 0) . (foldr ((:) . (length . (converse (:) []))) [])
```

[8]

(b) The following function `foldiftrue` reduces only those elements of a list which satisfy a given predicate:

```
foldiftrue pred ff def [] = def
foldiftrue pred ff def (front : rest)
    = ff front (foldiftrue pred ff def rest), if (pred front)
    = foldiftrue pred ff def rest, otherwise
```

Give the type definition for `foldiftrue` (use polymorphic types where appropriate to give the most general type).

Rewrite foldiftrue in terms of foldr and filter.

[13]

(c) What is the most general type of each of the following functions:

```
lotsoffun a b f x = ((a f) . (b f)) x

morefun [] j  acc = acc
morefun f  [] acc = acc
morefun (x : xs) (y : ys) acc = morefun xs ys (x y acc)


mysterious [] y z
    = 42.0 + z
mysterious (front : rest) y z
    = mysterious rest y ((front . y) z)
```

[12]

[Total 33]

[END OF PAPER]