END-SEMESTER EXAMINATION

Design and Implementation of Functional Programming Languages

DATE: 27th November 2003 **Maximum Marks: 100**

1. (a) What is the complexity of the function *reverse* defined below? $\boxed{\text{2+5 Marks}}$

   $reverse\ [\,] = [\,]$
   $reverse\ (x : xs) = reverse\ xs +\!\!+ [x]$

   (b) Redefine *reverse* so that its complexity decreases by more than a constant factor.

2. Define a binary tree as: $\boxed{\text{1+1+1+(6+6) Marks}}$

   $data\ Bintree\ a = Leaf\ a \mid Bnode\ (Bintree\ a)\ (Bintree\ a)$

   and a general tree as:

   $data\ Gentree\ a = Gnode\ a\ [Gentree\ a]$

   Now consider the expression $f(g(a, b), h(c), d)$

   (a) Considering each of the function and variable symbols to be a *Char*, represent the above expression as a *Gentree Char*.

   (b) Write the curried version of the same expression.

   (c) Represent the curried version as a *Bintree Char*. (Hint: In the representation of the curried version, the constructor *Bnode* should be interpreted as an application node.)

   (d) Define the following two function definitions:

   i. $curry :: Gentree\ Char \rightarrow Bintree\ Char$ converts the expression from uncurried to curried form.

   ii. $uncurry :: Bintree\ Char \rightarrow Gentree\ Char$ converts the expression from curried to uncurried form.

3. This is actually a lambda calculus question. Consider the datatypes: $\boxed{\text{8+8 Marks}}$

   $data\ NatPlus = One \mid Succ\ NatPlus$
   $data\ Nat = Zero \mid Succ\ Nat$

   and the folding function $foldn$ defined as:

   $foldn\ f\ id\ One = id$
   $foldn\ f\ id\ (Succ\ n) = f\ (foldn\ f\ id\ n)$

   Let $\overline{n}$ denote the real number corresponding to the $n :: NatPlus$. In terms of $foldn$, write the following functions from $NatPlus$ to Nat (a) $\lceil \overline{n}/2 \rceil$ (b) $\lceil log_2 \overline{n} \rceil$. Use Haskell notation.

   You can use any function whose lambda representation has been discussed during the course.

4. Find the principal type of g in: $\boxed{\text{6 Marks}}$

$$
\begin{aligned}
&g\ \ n = let\ ch = map\ g\ (f\ \ n)\\
&\qquad in\ (n, ch)\\
&\qquad where\ f\ \ 0 = [\,]\\
&\qquad\qquad f\ \ n = f\ \ (n-1) \mathbin{+\!\!+} [n-1]
\end{aligned}
$$

Assume that the arithmetic operators are Int operators and constants are Int constants.

5. Rewrite the following definition of $factorial$ to resolve overloading: $\boxed{\text{6+4 Marks}}$

$$factorial\ n = if\ \ (n == 0)\ then\ 1\ else\ n * factorial\ (n-1)$$

Also assuming a standard *default* declaration, show how the call $factorial\ 5$ will be rewritten to resolve overloading by providing the right dictionary.

6. We want to write a monadic tracing interpreter for the language whose grammar is shown below: $\boxed{\text{2+7+10 Marks}}$

$$
\begin{aligned}
&Exp \rightarrow Int \ \mid\ Exp + Exp \ \mid\ Exp\,/\,Exp \ \mid\ Position : Exp\\
&Position \rightarrow Int
\end{aligned}
$$

For a expression like $1 : 4 + 2 : (3 : (9\,/\,3) + 4 : 5)$, the output should be the string:

$$
\begin{aligned}
Position\ 1 :\ & Argument\ 1 = 4\\
& Argument\ 2 = 8\\
& Operation = +\\
& Result = 12\\
Position\ 2 :\ & Argument\ 1 = 3\\
& Argument\ 2 = 5\\
& Operation = +\\
& Result = 8\\
Position\ 3 :\ & Argument\ 1 = 9\\
& Argument\ 2 = 3\\
& Operation = /\\
& Result = 3\\
Position\ 4 :\ & Result = 5
\end{aligned}
$$

The result of the computation can be an error value instead of an integer. You can choose how to handle the error value.

(a) Define a Haskell datatype for expressing the above grammar.

(b) Define an appropriate monad for writing the interpreter.

(c) Write the interpreter in terms of the monad.

7. Lambda lift the following programs to supercombinator form. The resulting programs, when executed, should be fully lazy. $\boxed{\text{6+6 Marks}}$

(a) let

$f \;\; y = g \;\; y \;\; 4$

$g \;\; y \;\; x = y + sqrt \;\; x$

$in \;\; f \;\; 1 + f \;\; 2$

How many times is $sqrt$ 4 evaluated in the original and the lambda lifted program?

(b) let

$f \;\; x \;\; 0 = 0$

$f \;\; x \;\; n = sqrt \;\; x + f \;\; x \;\; (n - 1)$

$in \;\; f \;\; 4 \;\; 1000$

Once again answer the same question. How many times is $sqrt$ 4 evaluated in the original and the lambda lifted program?

8. (a) The correct sequence of G-code for the $HEAD supercombinator is: 8+7 Marks

EVAL; HEAD; EVAL; UPDATE 1; UNWIND

Explain with an example why the second EVAL is necessary. Also explain why the UNWIND instruction cannot be replaced by a RETURN.

(b) Consider the following fragment of a program in supercombinator form:

$\$sumints \;\; m = letrec \;\; count = \$count \;\; count \;\; m$

$\quad in \;\; \$sum \;\; count$

What will be the G-code for the supercombinator $\$sumints$?