CSC330 Midterm Exam. Spring 2004 (15 pts)

A language that doesn't affect the way you think about programming is not worth knowing - Alan Perlis

Read the questions carefully. If you can't answer a question move on and come back to it later. Use the empty space (including the back side) between exam questions to write your answers. You can use the last 2 empty pages as scratch paper. If you need more paper let me know. The total number of points is 15.

Good luck .
George Tzanetakis

1 Part 1 (1 pt)

Consider this piece of ML code. What are the values of x and y after it is compiled and executed ?

```
val a = 10;
fun f x = a + x;
fun g a = a + f(a);
val a = 20;
fun f x = a + x;
val (x,y) = (f(g(20)),g(f(20)));
```

2 Part 2 (2pt)

List comprehensions are a notation used in mathematics and Haskell to describe lists with specific properties. They have the form $(exp \mid x \leftarrow xs)$. For example $(x * x | x \leftarrow [4,5,6,7,8];\ x\ even)$ is the list [16,36,64].

Question 1 (1pt) Express the above example of a list comprehension in ML using map and filter with appropriate functions as arguments.

Question 2 (1pt) Write a function comprehension with type $('a \rightarrow 'b) \rightarrow ('a \rightarrow bool) \rightarrow 'a\ list \rightarrow 'b\ list$ that takes as arguments a function corresponding to the comprehension expression, a predicate, and a list. The function returns the resulting list. Call this function with appropriate arguments so that the result is the list of the above example.

3 Part 3 (2pt)

What is the value of mystery(tasty1) and mystery(tasty2) ?

```
datatype pizza = Crust
       | Cheese of pizza
       | Onion  of pizza
       | Sausage of pizza
       | Anchovy of pizza;

fun mystery(Crust)      = Crust
  | mystery(Cheese x)   = Cheese(mystery x)
  | mystery(Onion  x)   = Onion(mystery x)
  | mystery(Sausage x)  = mystery(x)
  | mystery(Anchovy x)  = Crust;


val tasty1 = Onion(Cheese (Sausage (Cheese ((Crust)))));
val tasty2 = Onion(Cheese (Anchovy (Cheese ((Crust)))));
```

4 Part 4 (4 pts)

This part is based on the following function:

```
fun f x [] = [[x]]
  | f x (y::ys) = [(x::y::ys)] @ (map (fn x => y::x) (f x ys));
```

Question 1 (1 pt) What is the type of f ?

- **A** $'a \rightarrow\ 'a\ list \rightarrow\ 'a\ list\ list$
- **B** $('a\ *\ 'a\ list)\ \rightarrow\ 'alistlist$
- **C** $'a\ \rightarrow\ 'a\ list \rightarrow\ 'a\ list$
- **D** $'a\ list -> \ 'a -> \ 'a\ list$

Question 2 (1 pt) Write f in a non-curried form using a 2-tupple as the argument

Question 3 (2 pt) What is the value of

```
f 5 [1,2,3,4]
```

5 Part 5 (1pt)

Using foldl write a function joinlength that takes as argument a string list and that returns a tuple with all the strings joined and the length of the list. For example:

```
joinlength[‘‘ab’’, ‘‘cd’’, ‘‘ef’’]
is (‘‘abcdef’’, 3)
```

6 Part 6 (1pt)

A naive recursive implementation of the function length for computing the length of a list is the following:

```
fun nlength [] = 0
  | nlegnth (x::xs) = 1 + nlength xs;
```

Write a tail recursive (iterative) version of nlength called ilength that accumulates the result as an additional argument. Write a function length that behaves exactly like nlength but uses the locally defined (using a let declaration) function ilength to compute the length of a list.

7 Part 7 (1pt)

This part is based on the following functions

```
fun power1(x,0) = 1
  | power1(x,n) = x * power1(x,n-1);

fun sum(x,y) = x + y

fun power2(x,n) = let fun ipower(x,0,k) = k
                        | ipower(x,n,k) = ipower(x,n-1,x*k)
                  in
                      ipower(x,n,1)
  end;

val v1 = sum(power1(2,3) + power1(2,3));
val v2 = sum(power2(2,3) + power2(2,3));
```

Based on this code, answer the following questions with a short text justification :
1) Using call-by-value, which calculation is more efficient v1 or v2 ?
2) For the computation of v1, which is more efficient call-by-name or call-by-need ?

8 Part 8 (3pt)

Consider the following Scheme function:

```
(define (foo f l)
  (cond ((null? L) '())
        ((not (f (car l))) (cons (car l) (foo f (cdr l))))
        (else (foo f (cdr l)))))

(define (bar l) (= (length l) 2))
```

Question 1(1pt) What is the result of

```
(foo bar '((1 2) (1 2 3) (1 2 3 4) (1 2 3 4 5) ))
```

Question 2(1pt) Write these functions in ML

Question 3(1pt) How would you write the function call of question 1 using the ML functions. Explain in your own words what foo does.

That's it enjoy your reading break.