ML Built-in Functions

Since ML is a functional programming language, many of its built-in functions are concerned with **function application** to objects and structures.

In ML, built-in functions are **curried** \(\rightarrow\) they expect their arguments as a sequence of objects separated by spaces **NOT** as a tuple.
The map Function

The map function accepts two parameters: a function and a list of objects. It will apply the given function to each object on the list.

Example:

- map (fn x => x + 2) \([1,2,3]\);
  val it = \([3,4,5]\) : int list

also works with built-in functions and operators such as the negation function ~ : int -> int

- map ~ \([1,2,3]\);
  val it = \([~1,~2,~3]\) : int list
The map Function

map can also be applied to a list of structures.

- map (fn (a,b) => a + b) [(1,2),(3,4)];
val it = [3,7] : int list
The foldr Function

The foldr function works similar to the map function, but instead of producing a list of values it only produces a single output value.

Syntax:
foldr <binary function> <initial value of output> <list>

Semantics:
- foldr f c [x1, x2, ..., xn-1, xn];
is the same as saying
- f(x1, f(x2, .... f(x-1,f(xn,c))...));

foldr starts at the rightmost object
xn of the list with initial value c

foldr folds a list of values into a single value starting with the rightmost element.
The foldr Function

Example:

- foldr (fn (a,b) => a+b) 2 [1,2,3];
  → fn(1,fn(2,fn(3,2)));

val it = 8 : int
The foldl Function

You guessed it! Works exactly the same as the foldr function except that it start computing at the leftmost element:

- `foldl f c [x1, x2, ... , xn-1, xn];`
  is the same as saying
  - `f(xn, f(xn-1, .... f(x2,f(x1,c))...));`

Example:

- `foldl (fn (a,b) => a+b) 2 [1,2,3];`
  => `fn(3,fn(2,fn(1,2)));`
  val it = 8 : int
In most cases foldr and foldl will produce the same results, but consider the following:

- foldr (fn (a,b) => a^b) "ef" ["ab","cd"];
  => fn("ab",fn("cd","ef"))
  => "ab"^("cd"^"ef")
  => "ab"^"cd"^"ef"
  => "abcdef"
  val it = "abcdef" : string

- foldl (fn (a,b) => a^b) "ef" ["ab","cd"];
  => fn("cd",fn("ab","ef"))
  => "cd"^("ab"^"ef")
  => "cd"^"ab"^"ef"
  => "cdabef"
  val it = "cdabef" : string

foldr and foldl will only produce the same results if the mapped function is commutative.
We can create new functions from curried library functions using partial evaluation:

- val listinc = map (fn x => x+1);
  val listinc = fn : int list -> int list
- listinc [1,2,3];
  val it = [2,3,4] : int list
Recursion and Curried Functions

(* original non-curried function *)
fun filter ([ ],e) = [ ]
  | filter (x::xs,e) = if x < e then x::filter(xs,e) else filter(xs,e);

(* curried function in traditional notation *)
fun filtercl [ ] = (fn e => [ ])
  | filtercl (x::xs) = (fn e => if x < e then x :: filtercl xs e else filtercl xs e);

(* curried function in short hand notation *)
fun filterc [ ] e = [ ]
  | filterc (x::xs) e = if x < e then x :: filterc xs e else filterc xs e;

Note: all parentheses are mandatory in the above examples.
Homework

Assignment #7 – see website

Midterm Monday 11/2 – covers chaps 1 through 9
**Week 1**
Chapter 1: Programming Languages
features of languages, classes of languages
Chapter 2: Defining Program Syntax
grammars, derivations, formal definition of languages, sentences

**Week 2**
Chapter 3: Where Syntax Meets Semantics
parse trees as semantics, ambiguous grammars
Chapter 4: Language Systems
structure of IDE/compiler, difference between compiler/interpreter

**Week 3**
Chapter 5: A First Look At ML
basic expression, tuples, lists
Chapter 6: Types
** a type is a set of values **

**Week 4**
Chapter 7: A Second Look At ML
patterns
Chapter 8: Polymorphism
overloading, parameter coercion, parametric polymorphism, subtype polymorphism

**Week 5**
Chapter 9: A Third Look At ML
higher-order programming: *** functions as parameters or return values ***
Consider the function

fun foo (a:string) = (fn (b:string) => (a,b));

What is the value and type of the following computations:
1. foo "100" "101";
2. val q = foo "happy"; q "really happy";

Rewrite this function in the abbreviated curried style.
Convert the following function

```
fun pow(b,m) = if m = 0 then 1 else b*pow(b,m-1);
```

1. to a function using patterns
2. to a function using currying
3. to function using patterns and currying
Write a curried function \textit{hdmap} that takes a function and a list of integers and applies the function to the first element of the list. If the list is empty return \sim 1,

\[
\text{hdmap} = \text{fn : (int -> int)} \rightarrow \text{int list -> int}
\]

Show that your function works by computing: \text{hdmap (fn x => x + 1)} [3,4]