Assignment #1: ML Introduction

Due: January 13, 2010 (11:59pm)

Overview

This assignment is intended to serve as an introduction to the ML programming language. This assignment is not meant to cover the entire language, but should prepare you for the remaining programming assignments.

Note: Though you might be asked to write a function, you are always allowed to write additional “helper” functions to simplify the implementation of the required function. This is good design in most languages, and is extremely important in this language. Also, the examples given below show only the resulting values and not their types.

Your solutions **may not** use mutation. This means that your solution cannot have variables of reference type, nor can it use data structures provided by the ML libraries that use mutation. The purpose of this exercise is for you to familiarize yourself with the language, not to program in Java using the ML syntax.

Part 1

number_of: ”a → ”a list → int

Write a function `number_of` that takes a value, `v`, of any equality type, a list of values, `L`, of the same type, and that returns the number of times `v` appears in the list `L`. For example,

```
- number_of 1 [];
0
- number_of 4 [1, 2, 3, 4];
1
- number_of 5 [1, 2, 3, 4];
0
- number_of 4 [1, 4, 3, 4];
2
```

Note: You should use patterns in the definition of this function.

Part 2

pair_swap: (’a * ’b) list → (’b * ’a) list

Write a function `pair_swap` that takes a list of pairs and return a list of pairs. The returned list matches the input list except that the elements of each pair have their positions swapped.

```
- pair_swap [(1,2), (3,4), (5,6)];
[(2,1),(4,3),(6,5)]
```

Part 3

weave: ’a list → ’a list → ’a list

Write the `weave` function that takes two lists and returns a list that represents the weaving (or interleaving) of the input lists. Specifically, if the input lists are [a_1, a_2, ... a_n] and [b_1, b_2, ... b_n], then the output list is [a_1, b_1, a_2, b_2, ... a_n, b_n]

If a natural weaving cannot be done (i.e., if one list is exhausted before with more than one value remaining in the other), this function must raise an `ImbalancedWeaving` exception. You must define this exception.

```
- weave [1,2,3] [4,5,6];
[1,4,2,5,3,6]
- weave [] [];
[]                 (* note that this will give a warning about generalization *)
- weave [1,2] [3];
[1,3,2]
- weave [1,2,3] [3];
```

```
uncaught exception ImbalancedWeaving
- weave [1] [2,3];
uncaught exception ImbalancedWeaving
```

Part 4

file_subst: string → (char * char) list → unit

Write a function `file_subst` that takes a filename as a string and a list of characters pairs. This function must open the named file, read the contents of the file, and echo the characters to the screen. Any occurrence of a character in the first position of a pair must echoed as the character in the second position of the pair.

For example, an invocation such as `file_subst "inputFile" [(#"a", #"b"), (#"b", #"z")]` will echo the contents of `inputFile` with all occurrences of the character `a` replaced by the character `b`.

Your function need only find the first occurrence of the character in the list (i.e., if there are multiple pairs with the same character in the first position, only the first such pair is used).

You may want to use `TextIO.openIn`, `TextIO.input1`, `TextIO.print`, `TextIO.endOfStream`, `isSome`, and `valOf`.

Part 5

Consider the following datatype:

```
datatype 'a ThingCollection =
     OneThing of ('a * 'a ThingCollection)
   | TwoThings of ('a * 'a * 'a ThingCollection)
   | ManyThings of ('a list * 'a ThingCollection)
   | Nothing
;
```

This datatype represents a collection of things; things may be of any type, but a collection can contain only things of the same type.

number_of_things: 'a ThingCollection → int

Write the function `number_of_things` that takes a ThingCollection value and returns an integer count of the number of "things" in that collection.

```
- number_of_things Nothing;
0
- number_of_things (OneThing (7, Nothing));
1
- number_of_things (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, Nothing))));
5
```

Part 6

number_of_OneThing: 'a ThingCollection → int

Write the function `number_of_OneThing` that takes a ThingCollection and returns the number of OneThing "nodes" in the collection (i.e., this counts the number of OneThing constructors).

```
- number_of_OneThing Nothing;
0
- number_of_OneThing (OneThing (7, Nothing));
1
- number_of_OneThing (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, Nothing))));
1
- number_of_OneThing (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, OneThing (99, Nothing)))));
2
```

Part 7

number_of_XThing: ('a ThingCollection → bool) → 'a ThingCollection → int

Write the function `number_of_XThing` that generalizes the previous function (`number_of_OneThing`) such that it counts the number of "Thing" nodes, where the type of "Thing" is determined by a predicate passed to the function (i.e., the function determines if which nodes should be counted).

For example, using the generalized function, the function for the previous problem could be written as:

```
fun number_of_OneThing things = number_of_XThing (fn (OneThing _) => true | _ => false) things;
```

Specifically, `number_of_XThing` return the number of "Thing" values that satisfy the given predicate.

```
- number_of_XThing (fn (OneThing _) => true | _ => false) Nothing;
0
- number_of_XThing (fn (OneThing _) => true | _ => false) (OneThing (7, Nothing));
1
- number_of_XThing (fn (OneThing _) => true | _ => false)
  (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, Nothing))));
1
- number_of_XThing (fn (OneThing _) => true | _ => false)
  (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, OneThing (99, Nothing)))));
2
```

Part 8

number_of_TwoThings: 'a ThingCollection → int

Write the function `number_of_TwoThings` that takes a ThingCollection and returns the number of TwoThings values in the collection. You *must* define this function in terms of `number_of_XThing`. This means that your definition will consist of only a call to `number_of_XThing`; this function will not be recursive.

```
- number_of_TwoThings Nothing;
0
- number_of_TwoThings (OneThing (7, Nothing));
0
- number_of_TwoThings (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, Nothing))));
1
- number_of_TwoThings (OneThing (7, ManyThings ([1, 2], TwoThings (1, 2, OneThing (99, Nothing)))));
1
```

Part 9

map_thing_collection: ('a → 'b) → 'a ThingCollection → 'b ThingCollection

Write the function `map_thing_collection` that takes a function and a ThingCollection. `map_thing_collection` applies the given function to the value stored in each node of the original collection (and each value in the list stored within a `ManyThings` value). This function behaves much like the standard `map` function for lists.

```
- val C = (OneThing (7, ManyThings ([4, 3], TwoThings (10, 8, OneThing (99, Nothing)))));
- map_thing_collection (fn x => x + 1) C;
OneThing (8,ManyThings ([5,4],TwoThings (11,9,OneThing (100,Nothing))))
- map_thing_collection (fn x => x * x) C;
OneThing (49,ManyThings ([16,9],TwoThings (100,64,OneThing (9801,Nothing))))
- map_thing_collection (fn x => x > 7) C;
OneThing (false, ManyThings ([false,false],TwoThings (true,true,OneThing (true,Nothing))))
```

Part 10

flatten_collection: 'a ThingCollection → 'a ThingCollection

A `'a ThingCollection` value is constructed from `OneThing`, `TwoThings`, `ManyThings`, and `Nothing` constructors to store a collection of `'a` values. Write a function `flatten_collection` that takes a `'a ThingCollection` as an argument and that returns a `'a ThingCollection` that is constructed from a single `ManyThings` and a single `Nothing`. In other words, this function extracts all of the `'a` values from the collection and places them, in the same order, in a single `ManyThings` constructed value.

```
- flatten_collection (OneThing (7, ManyThings ([4, 3], TwoThings (10, 8, OneThing (99, Nothing)))));
ManyThings ([7,4,3,10,8,99],Nothing)
```

Logistics

- Strive for simplicity in your programming. Write short helper functions.
- Be certain that you can do each part of this assignment as you will use these features in later assignments. Ask lots of questions.
- Grading will be divided as follows.

Part	Percentage
1	10
2	10
3	10
4	10
5	10
6	10
7	10
8	10
9	10
10	10

- Get started **now** to avoid the last minute rush.