CEng242 Homework 3

Due 13^{th} April 2003

Most of the imperative languages provide array types which forms a mapping between two types. However in this mapping, index (or source) type is restricted to a subrange of discrete ordered primitive types.
In this homework, you will implement an abstract data type in ML which implements a set of mappings on two arbitrary data types. This datatype: '(α, β) `MapSet`' will implement:
(α, β) `MapSet`$= \mathcal{P}(\alpha \mapsto \beta)$
Which is a set of mappings.
That means each value of the `MapSet` stores the mappings (corressponding β typed values) for a subset of α. This datatype should provide all following interfaces:

`toMapSet` ***`maplist cmpfunc`*** `:` α`*`β `list -> (`α`->`α`->int) -> (`α`,`β`) MapSet`
This function is the constructor for `MapSet`. It first gets a list of tuples containing key-value pairs in the map. Second parameter is a function comparing two α values. This function will be used to guarantee that there is at most 1 mapping for an α typed key. Also it is used for ordered operations on mapping keys. It returns a negative value if first α is less than the second, a positive value if it is larger than the second and it returns 0 if two values are equal (like `strcmp()` in C).

`get` ***`mapset key`*** `:` `(`α`,`β`) MapSet -> `α `-> `β
This function is the selector for this datatype. For a given value of α typed ***`key`*** value it returns the value of this mapping if available. If the value is not available, that is there is no mapping for ***`key`*** in ***`mapset`*** it raises an exception named `NotFound`.

`set` ***`mapset key value`*** `:` `(`α`,`β`) MapSet -> `α `-> `β `-> (`α`,`β`) MapSet`
This function returns a new `Mapset` with ***`key`*** mapping is set/updated to the ***`value`*** and the other elements remain same with the ***`mapset`***.

`delete` ***`mapset key`*** `:` `(`α`,`β`) MapSet -> `α `-> (`α`,`β`) MapSet`
This function returns a new `MapSet` with ***`key`*** is deleted and remaining elements are same with ***`mapset`***. If ***`key`*** does not exist in ***`mapset`***, it silently returns the same `MapSet`.

`mapsetA` `++` ***`mapsetB`*** `: ( (`α`,`β`) MapSet * (`α`,`β`) MapSet ) -> (`α`,`β`) MapSet`
This infix operator function returns the union of two `MapSet` values. Resulting set should not contain duplicate keys. In case of same key existing in both mapsets, the value in the resulting mapset is taken from the second mapset (***`mapsetB`***).

`mapsetA` `**` ***`mapsetB`*** `: ( (`α`,`β`) MapSet * (`α`,`β`) MapSet ) -> (`α`,`β`) MapSet`
This infix operator function returns the intersection of two `Mapset` values. Resulting set contains the keys existing in both mapsets. The map values are taken from the second mapset (***`mapsetB`***).

`mapsetA` `--` ***`mapsetB`*** `: ( (`α`,`β`) MapSet * (`α`,`β`) MapSet ) -> (`α`,`β`) MapSet`
This infix operator function returns the difference of two `Mapset` values. Resulting set contains the keys that exist in the first mapset but not in the second mapset. The map values are preserved as they are in the first mapset (***`mapsetA`***).

`keys` ***`mapset`*** `:` `(`α`,`β`) MapSet -> `α `list`
This function returns a sorted list of all keys in the mapset. This list consists of key values and should be in ascending order according to the comparison function given in the constructor.

`pairs` ***`mapset`*** `:` `(`α`,`β`) MapSet -> (`α`*`β`) list`
This function returns a sorted list of all key-value pairs in the mapset. This list should be in ascending order accoring on key values as in the `keys` function. constructor.

`mapAll` *mapset func* : (α,β) `MapSet -> (`α`->`β`->`γ`) -> `γ `list`
: This function applies *func* to all key-value pairs in the mapset and returns the list of return values. Resulting list should contain results of key values in ascending order.

`exception NotFound`
: This is just an exception for `get` function. Declare it as it is and in `get`, just use '`raise NotFound`'. expression.

Put definition of all these function in an `abstype` declaration. Follow this instructions:

1. You can choose any internal data representation for `MapSet`. This representation should be hidden by the `abstype`.
2. Only export the requested definitions in the topmost scope. Hide all auxilary definitions in a `local` block.
3. Infix operators should have the same precedence. You can put definitions like: `infix **;` in order to use infix syntax.
4. Assume that type specification is not ambiguous. That is you will not be given: `toMapSet [] (fn x=> fn y=> x-y)` Where the type for β is underspecified. Instead it will be: `toMapSet ([]:(int*string) list) (fn x=> fn y=> x-y)`
5. In the constructor `toMapSet` if multiple keys exists in the input list, the later value overwrites the former one.
6. Assume both comparison functions are identical in the infix operator. When they are called with different comparison functions result is inpredictable. That is you can use and return comparison function of either `mapsetA` or `mapsetB`.

Sample run (Some of the ML outputs are indicated in italic):

```
fun strcmp a b = if (String.< (a,b)) then ~1
                 else if (String.> (a,b)) then 1
                      else 0;
 val strcmp = fn : string -> string -> int
val a = toMapSet ([]:(string*int) list) strcmp;
 val a = - : (string,int) MapSet
val a = set a "bugs bunny" 7;
val a = set a "road runner" 2;
val a = set a "coyote" 3;
val a = set a "tweety" 1;
val a = set a "sylvester" 4;
val a = set a "bugs bunny" 1;
 val a = - : (string,int) MapSet
val b = toMapSet [("bugs bunny",4),("duffy duck",10),("coyote",2)] strcmp;
 val b = - : (string,int) MapSet

get a "bugs bunny";
 val it = 1 : int

get a "mickey mouse";
 uncaught exception NotFound
   raised at: ......

val a = delete a "road runner";
```

```
 val a = - : (string,int) MapSet

val c = a ++ b;
 val c = - : (string,int) MapSet

val d = a ** b;
 val d = - : (string,int) MapSet

val e = a -- b;
 val e = - : (string,int) MapSet

keys a;
 val it =["bugs bunny", "coyote", "sylvester", "tweety"] : string list

pairs b;
 val it =[("bugs bunny",4),("coyote",2),("duffy duck",10)] : (string*int) list
keys c;
 val it =["bugs bunny", "coyote", "duffy duck", "sylvester", "tweety"] : string list
keys d;
 val it =["bugs bunny", "coyote"] : string list
keys e;
 val it =["sylvester", "tweety"] : string list

fun attach s n = s ^ ":" ^ (Int.toString n);
 val attach = fn : string -> int -> string

mapAll a attach;
 val it =["bugsbunny:1", "coyote:3", "sylvester:4", "tweety:1" ] : string list
```

Your submission will be again in a single file. Starting from this homework, late submission will be graded as:

- before deadline: over 100
- 1 day late: over 80
- 2 days late: over 60
- 3 days late: over 30
- Later: 0