CSC 330
Programming Languages
Assignment 2

Note 1 **This assignment is to be done individually**
Note 2 You can discuss the assignment with others, but sharing and copying code is prohibited.

A note on Academic Integrity and Plagiarim

Please review the following documents:

- Standards for Professional Behaviour, Faculty of Engineering:
  `https://www.uvic.ca/engineering/assets/docs/professional-behaviour.pdf`
- Policies Academic Integrity, UVic:
  `https://www.uvic.ca/students/academics/academic-integrity/`
- Uvic's Calendar section on Plagirism:
  `https://www.uvic.ca/calendar/undergrad/index.php#/policy/Sk_0xsM_V`

Note specifically:

> Plagiarism
> Single or multiple instances of inadequate attribution of sources should result in a failing grade for the work. A largely or fully plagiarized piece of work should result in a grade of F for the course.

You are responsible for your own submission, but you could also be responsible if somebody plagiarizes your submission.

Objectives

After completing this assignment, you will have experience with:

1. Implementing polymorphic functions.
2. Using pattern matching.
3. Defining operators.
4. Implementing a data structure.

Introduction

Jon Bentley's Programming Pearls were probably the most influential readings of my incipient career as a programmer. He wrote: *"Coding skill is just one small part of writing correct programs. The majority of the task [is]: problem definition, algorithm design, and data structure selection. If you perform those tasks well, writing correct code is usually easy. "*[1] To this day, I still think about this statement every time I am starting to write a program.

Lists are an interesting data structure. They can be used to implement many others: a tuple? use a two elements list. Do you want records? How about lists of tuples, each tuple has the name of the field, and the value of the field. A set is a list with no duplicates. The node of tree is a list that contains nodes (lists). That is one of the reasons that some languages —like SML and the Lisp family—place lists at the center stage.

In this assignment we will implement a polymorphic set data structure.

---

1. Jon Bentley, *Programming pearls: Writing correct programs*, Communications of the ACM Vol. 26, No. 12, 1983. Highly recommended reading, even today.

Your task, should you choose to accept it

We are going to implement functional sets. What are functional sets? We will have a constructor that creates an empty set, and operations to add, remove elements from the set. And we will have operations between sets: union, intersection, etc. They will be used in a manner similar to the way lists are used.

We will implement a set with a list. The list will contain the elements of the set. I recommend that you keep this elements in order because when we inspect the contents of the set, they should be in order. However, you can choose to keep them in any order you want, because, as any ADT, the internal implementation details do not matter, as long as the external API is satisfied.

The following datatype describes our set:

```
datatype 'a set = EmptySet of ('a * 'a -> order)
    | Set of 'a list * ('a * 'a -> order)
```

Note that a set of 'a type is either an `EmptySet` or a `Set`. Both `EmptySet` and `Set` include a function that that has type `('a * 'a) -> order`. This function is needed to know how to order the elements of the set (`Set` also includes a list that contains the elements of the set).

The datatype `order` is predefined in SML as:

```
datatype order = LESS | EQUAL | GREATER
```

This datatype is returned by functions such as `Int.compare` and `String.compare`:

```
- Int.compare(0,1);
val it = LESS : order
- Int.compare(1,0);
val it = GREATER : order
- Int.compare(1,1);
val it = EQUAL : order
- String.compare;
val it = fn : string * string -> order
```

The idea is that when we create a set, we need to tell the set what are the types of its elements and how to order them. This is done via the comparision function. For example, this is how we create an empty set of integers:

```
val a = EmptySet Int.compare
```

Now, if we want to add an element to our set, we will invoke the function `insert_into_set`. Its first parameter is a set, the second the value to add. The following code will add 1 and 2 to the empty set:

```
insert_into_set(insert_into_set(EmptySet Int.compare, 1), 2)
```

This notation is kind of ugly, so let us create "syntactic sugar" to make our life easier. Let's use `++` to add elements to a set, and `--` to remove elements from a set. Thus:

```
(EmptySet Int.compare) ++ 1 ++ 2 ++ 3 ++ 2
```

will result in a set of integers 1, 2 and 3. While

```
(EmptySet Int.compare) ++ 1 ++ 2 -- 1 --3
```

will result in a set with only 2 in it.

We can create a set of any data structure. For example, if we have a function called `comp_list_int` that compares two lists of integers and has the following signature:

```
val comp_list_int = fn : int list * int list -> order
```

We can then create a set of lists of integers as follows:

```
(EmptySet comp_list_int) ++ [1,2] ++ [3,2,9] ++ [1,2] ++ [] ++ [];
```

With the following result:

```
[],[1,2],[3,2,9]
```

Code provided

Download the source code from conneX. Your program will be tested in `linux.csc.uvic.ca`. The source code, Makefile and tests assume this.

Specification

Part 1

Write the following functions:

1. `is_empty_set s`: returns true if set s is empty
2. `min_in_set s`: returns the minimum value in set s. Raises exception `SetIsEmpty` if s is empty.
3. `max_in_set s`: returns the maximum value in set s. Raises exception `SetIsEmpty` if s is empty.
4. `insert_into_set(s,v)`: returns $s \cup v$
5. `in_set(s, v)`: returns true if $v \in s$
6. `union_set(s, t)`: returns the $s \cup t$. **Must be tail recursive.**
7. `intersect_set(s, t)`: returns $s \cap t$. **Must be tail recursive.**
8. `except_set(s, t)`: returns $s - t$. **Must be tail recursive.**
9. `remove_from_set(s,v)`: returns $s - v$
10. `size_set(s)`: returns the number of elements in the set.
11. `equal_set(s, t)`: returns true if $s = t$
12. `is_subset_of(s, t)`: returns true if $s \subseteq t$
13. `list_to_set(lst, f)`: returns the set created from the contents of the list, using the comparision function f.
14. `set_to_list s`: returns a list with the contents of the set, in order.
15. `str_set (s, fstr)` : returns a string with the contents of the set. It will convert each element of the set with the function `fstr`. The set contents are surrounded by braces, the elements are separated by ":" and the elements are in order.
    ```
    - str_set((EmptySet Int.compare) ++ 3 ++ 2 ++ 4, Int.toString);
    val it = "{2:3:4}" : string
    ```
16. `map_set (s, fcomp, f)` : Return a new set (that uses the fcomp comparision function). If we call this set r, then:

$$r = \bigcup_{v \in s} f(v)$$

Please note that you do not have to implement them in this order. In fact, if you are strategic you can write some functions in terms of other functions (think of the properties that sets have).

We will not test the functions with huge sets. I expect that the complexity of most functions will be $O(n)$ complexity, and a handful will be $O(n^2)$.

The union, intersection and set difference must be tail recursive functions.

Part 2: operators

Implement the following operators. Note that their precedence is already defined in the code provided.

1. s – v = remove_from_set(s,v)
2. s ++ v = insert_into_set(s,v)
3. s IDENTICAL t = equal_set(s,t)
4. s UNION t = union_set(s,t)
5. s INTERSECT t = intersect_set(s,t)
6. s EXCEPT t = except_set(s,t)
7. v IN s = in_set(s, v)
8. s IS_SUBSET t = in_set(s, t)

Part 3

Write the following polymorphic comparison function, so we can create sets of lists of any type (see `main.sml` for a similar function to compare pairs of anything.

```
comp_list_any (a: 'a list, b: 'a list, fcomp : ('a * 'a) -> order)
```

This function compares two lists of any type. `fcomp` is a function to compare two elements of the list. Use lexicographical comparison.

Your code

Modify the file `set.sml`. The functions you create must have the following type signatures:

```
val is_empty_set = fn : 'a set -> bool
val min_in_set = fn : 'a set -> 'a
val max_in_set = fn : 'a set -> 'a
val insert_into_set = fn : 'a set * 'a -> 'a set
val in_set = fn : 'a set * 'a -> bool
val union_set = fn : 'a set * 'a set -> 'a set
val intersect_set = fn : 'a set * 'a set -> 'a set
val except_set = fn : 'a set * 'a set -> 'a set
val remove_from_set = fn : 'a set * 'a -> 'a set
val size_set = fn : 'a set -> int
val equal_set = fn : 'a set * 'a set -> bool
val is_subset_of = fn : 'a set * 'a set -> bool
val list_to_set = fn : 'a list * ('a * 'a -> order) -> 'a set
val set_to_list = fn : 'a set -> 'a list
val str_set = fn : 'a set * ('a -> string) -> string
val map_set = fn : 'a set * ('b * 'b -> order) * ('a -> 'b) -> 'b set
val -- = fn : 'a set * 'a -> 'a set
val ++ = fn : 'a set * 'a -> 'a set
val IDENTICAL = fn : 'a set * 'a set -> bool
val UNION = fn : 'a set * 'a set -> 'a set
val INTERSECT = fn : 'a set * 'a set -> 'a set
val EXCEPT = fn : 'a set * 'a set -> 'a set
val IN = fn : 'a * 'a set -> bool
val CONTAINS = fn : 'a set * 'a -> bool
val IS_SUBSET_OF = fn : 'a set * 'a set -> bool
val comp_list_any = fn : 'a list * 'a list * ('a * 'a -> order) -> order
```

Tests provided

The file `main.sml` contains some tests that will clarify the behaviour of your functions. I recommend you modify this file to add your own tests. **You will not submit this file**. Make sure you test your submission with an unmodified version of this file, to make sure your program works as expected.

Restrictions

Your solution should satisfy the following restrictions:

1. Your program should generate **NO** warnings. If your program generates a single compilation warning it will receive a zero.
2. **It cannot use mutation**. Use of `ref`, `:=` and `!` is forbidden.
3. It **cannot** use any library function that requires the explicit name of its structure when called (e.g. `TextIO.inputAll`, `List.nth`, `List.filter`, etc.). For this reason you are not allowed to use the character `.` anywhere in your submission program (even in comments) Your program will receive a zero if you do. You cannot circumvent this rule using `open`.

Testing

You will need to write many tests.

Evaluation

Solutions should:

1. Compile and run in `linux.csc.uvic.ca` If a program does not compile **for any reason** it will receive a zero.
2. Satisfy the restrictions above.
3. Pass tests. Your grade will be determined based on how many tests it passes.
   (a) Be correct. They should pass all the tests we have provided.
   (b) Be in good style, including indentation and line breaks

Second submission

If you choose, you can submit this assignment a second time (the deadline will be set after the results of the first submission are published). The final grade of your assignment will be:

$$max(first\_submission, 0.5 * second\_submission)$$

You are not required to submit a second time.

Hints

1. If you have trouble dealing with type errors, start by adding types to the parameters of the functions you are writing (see the type signatures above).
2. Start by implementing the functions to insert into a set and test membership. Once you have these two functions, you can start testing. You can also get partial marks if you implement at the very least these two functions.

What to submit

1. Using conneX, submit your version of the file `set.sml`. **Do not submit any other files**.
2. You should assume that your submission was received correctly only if both of the following two conditions hold:
   (a) You receive a confirmation email from conneX.
   (b) You can download and view your submission, and it contains the correct data.
3. In the event that your submission is not received by the marker, you will need to demonstrate that both of the above conditions were met if you want to argue that there was a submission error.