CPSC 326: Homework 8 *Due: On or Before Mon., May 5*

Goals:

- Gain practice writing basic OCaml functions
- Gain practice using pattern matching, guards, and algebraic types
- Explore logic programming (via ASP) as optional extra credit (details at end of assignment)

Instructions:

1. Use the GitHub Classroom link (posted in Piazza) to copy the starter code into your own repository. Clone the repository in the directory where you will be working on the assignment.
2. Write the functions described below in `hw8a.ml` and `hw8b.ml`. As you write each function be sure to test each one.
3. Keep track of the test cases you used for each function (and include each test in your HW writeup).
4. Create a write up as a **pdf file** named `hw8-writeup.pdf`. For this assignment, your write up should provide a short description of the tests you used for each function and any challenges and/or issues you faced in finishing the assignment and how you addressed them.
5. Submit your files to your repository. Ensure all of your code and writeup is pushed to your GitHub repo. You can verify that your work has been submitted via the GitHub page for your repo.

Additional Requirements: Note that in addition to items listed below, details will also be discussed in class and in lecture notes. You must implement the functions below *from scratch* in `hw8a.ml` and `hw8b.ml`. *Each function you write cannot use if-then-else and must used patterns with guards as appropriate.* Additionally, *you may only use the OCaml constructs we've discussed in class or as provided in the function description (if you go beyond what we've done*, you'll receive no points for the question).

Part A: (10 points) For part A, your job is to implement a single function called `eval` that takes a value of type `expr` (see below) and returns its corresponding integer value. You can think of this part of the assignment as defining a "mini" language for representing simple integer expressions (via the type `expr`). Your data type definition for `expr` consist of the following eight data constructors.

- `Int` is a basic "wrapper" for an integer value. For example: `Int 3`.
- `Plus`, `Minus`, `Times`, and `Divide` create binary expressions. For example, `Plus (Int 3, Int 4)` represents `3 + 4`, `Minus (Plus (Int 3, Int 4), Int 6)` represents `(3 + 4) - 6`, and so on.
- `Negate` is for unary minus. For example, `Negate (Int 3)` represents `-3`.
- `If` is an if-then-else expression where a non-zero integer value denotes true. For example, `If (Int 2, Int 10, Int 15)` evaluates to `10` and `If (Int 0, Int 10, Int 15)` evaluates to `15`.
- `Iterate`, which represents an expression that calls a given function f with a starting expression e a total of n times: $f_n(\cdots f_2(f_1(e))\cdots)$. The function to apply must have type `int -> int`. For example, `Iterate (3, (fun x -> x + 1), Int 2)` should return 5.

Part B: (10 points) For Part B, your job is to write a set of functions (see below) for a linked-list based, "ordererd", parametric `set` data type. The data type for `set` is given as:

```
type 'a set = Elem of 'a * 'a set
            | EmptySet
```

The functions you need to write are the following.

1. Write a function `add` that adds a value to a give set. The element being added must maintain the ascending sort order of the set and must not result in any duplicate elements. Example:

```
# let s = Elem (1, Elem (3, EmptySet)) ;;
val s : int set = Elem (1, Elem (3, EmptySet))
# add 2 s ;;
- : int set = Elem (1, Elem (2, Elem (3, EmptySet)))
# add 1 s ;;
- : int set = Elem (1, Elem (3, EmptySet))
# add 4 s ;;
- : int set = Elem (1, Elem (3, Elem (4, EmptySet)))
```

2. Write a function `member` that takes a value and a set and returns true if the value is in the given set and false otherwise. Your `member` function must be $\mathcal{O}(n)$.

3. Wrte a function `subset` that takes two sets and returns true if the first set is a proper subset of the second set. (Note: if $s_1 \subset s_2$, then every element in s_1 is in s_2 and s_2 contains at least one element that isn't in s_2). Your `subset` function must be $\mathcal{O}(n)$.

4. Write a function `equal` that takes two sets and returns true if they contain the same values and false otherwise. Your `equal` function must be $\mathcal{O}(n)$. You cannot use `subset` in your answer.

5. Write a function `union` that takes two sets and returns the union of the two sets. The resulting set must be a valid ordered set (i.e., the elements must be in ascending order and there cannot be any duplicates). Your `union` function must be $\mathcal{O}(n)$.

6. Write a function `intersect` that takes two sets and returns the intersection of the two sets. The resulting set must be a valid ordered set (i.e., the elements must be in ascending order and there cannot be any duplicates). Your `intersect` function must be $\mathcal{O}(n)$.

7. Write a function `difference` that takes two sets and returns the the elements in the first set that are not in the second set. The resulting set must be a valid ordered set (i.e., the elements must be in ascending order and there cannot be any duplicates). Your `difference` function must be $\mathcal{O}(n)$.

8. Write a function `set_map` that implements the map function for sets. The resulting set must be a valid ordered set (i.e., the elements must be in ascending order and there cannot be any duplicates). Your `set_map` function can call your other functions as needed.

Homework Submission and Grading. Your homework will be graded using the files you have pushed to your GitHub repository. Thus, you must ensure that all of the files needed to compile and run your code have been successfully pushed to your GitHub repo for the assignment. Note that this also includes your homework writeup. This homework assignment is worth a total of 20 points. The points will be allocated based on the correctness and completeness of your solution to the above functions. Note that you must

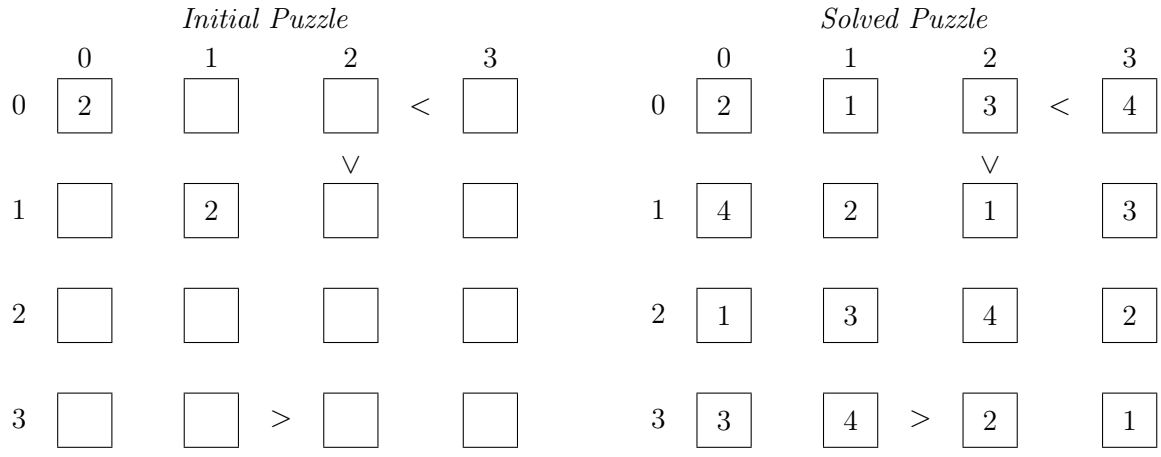
also include evidence of testing and your writeup. *Assignments without a write up and without discussion of thorough testing will not receive a grade.*

Extra Credit. The following Clingo / ASP problems are optional and will be graded as extra credit (6 points per problem solved). For each problem, you must:

- Develop at least four example inputs (beyond the ones I give). Each example input must be of a different size (increasing n) and should be relatively "interesting".
- Fill out the header for each program file you work on (`ec_1.lp` and `ec_2.lp`) with your name and any additional comments.
- Add a section in your write up describing which of the two (if any) that you attempted, explain the input programs that you developed and why you think they were useful for testing, what your general thoughts are regarding programming in this way, and any challenges / issues you had in solving the problem.

1. (6 points) *The n chairs problem:* Write a Clingo program to assign n guests to n chairs arranged around the perimter of a table. The guests should be arranged such that guests that dislike each other do not sit next to each other. The input to the program is a list of `dislike/2` facts and the number of total guests n given by a fact `guests(`n`)`. The output is a set of `at(`x`,`y`)` facts stating that guest x is seated at chair y. Assume the chairs are number 1 through n (where chair 2 is to the left of chair 1 and chair n is to the right of chair 1). See `ec_1.lp` for additional information.

2. (6 points) *Solving Futoshiki:* Write a program that implements a solver for the game "More or Less" (also called Futoshiki), which is similar to Sudoku. The game is played on an $n \times n$ board. Each cell contains a number from 1 to n such that the same number does not occur twice on a row or column. The game consists of a partially filled in board with (optinal) inequality constraints ($<$) between adjacent cells. The goal is to solve the board by filling it in without violating any of the constraints. The following is an example 4×4 game (left) and solution (right).



A game is represented by a three relations: `size(`n`)` where n denotes the size of the board; `lt(r1,c1,r2,c2)` that define inequality constraints where `r1, r2` denote rows from 0 to $n-1$ and `c1, c2` denote columns from 0 to $n-1$ (e.g., `lt(0,2,0,3)` states that cell (0,2) must be less than cell (0,3)); and `cell(r,c,v)` stating that cell (`r`,`c`) has the value `v` representing the initial cell values given by the puzzle. The goal of

your solver is to assert the missing `cell` relations to complete the puzzle. Note that not all puzzle configurations will have a single solution (you might try to come up with at least one puzzle that has multiple solutions as a test). For different puzzles to try, see https://www.futoshiki.org.