Lab 0 - *MiniLab*

Due: *Tue, Sept 2nd, 2014 @ 11:59 pm*

Parallel and Sequential Data Structures and Algorithms 15-210 (Fall '14)

1 Welcome to 15-210!

Welcome to 15-210! In this very first (zeroth?) lab, we'll introduce you to the environment you will be working with for subsequent labs, as well as get you acquainted with the 210 Sequence library and its documentation. Remember to get started early and get any issues sorted out before *ParenLab* comes out next week!

Note that for this lab only, you will not be allowed to use late days. The last possible time you can submit is 11:59 pm on Tuesday, Sep 2nd.

2 Files

2.1 Extracting Files

After downloading the assignment tarball from Autolab, extract the files by running:

```
tar -xvf minilab-handout.tgz
```

from a terminal window. Some of the files worth looking at are listed below. You should only modify the files denoted by *, as these will be the only ones handed in by the submission script.

1. `Makefile`
2. `BARESEQUENCE.sig`
3. `Testing.sml`
4. * `MkPartialSequence.sml`
5. * `MkSeqFun.sml`

Additionally, you should create a file called:

```
written.pdf
```

which contains the answers to the written parts of the assignment.

2.2 Lab Handout Structure

The handout directory of every lab in 15-210 will follow this layout. In particular, files that you will either need to inspect or edit are in the root of the handout directory, and everything else is in `support/`. Future labs will also include the `lib/` subdirectory, where our library files are located. Files that end in `.cm` are for compilation with CM (the SML compilation manager) – you may ignore these files.

3 Submission

3.1 Autolab

We will be using Autolab (`https://autolab.cs.cmu.edu/15210-f14`) for homework submissions and grading. Each lab will have several grade columns, corresponding to different parts of the assignment.

When you submit your work to Autolab, the autograder will grade your code and fill in the appropriate grade columns. Most labs have two suites of tests: *public* and *private*. The results of the public tests are visible immediately after you submit. The results of the private tests will not be visible until after the due date. For more details, see section 4.4.2.

If you submit multiple times, we will consider only your *most recent* submission.

3.2 Submission Instructions

Every lab's handout directory includes a `Makefile`, which executes the `submit` script in `support/`. This script automatically packages up *only* the files marked with a `*` in section 2.1, as well as your `written.pdf`, and submits it to Autolab.

To submit your assignment to Autolab, open a terminal, `cd` to the `minilab` folder, and run:

```
make
```

Alternatively, run `make package`, open the Autolab webpage and submit the `handin.tgz` file via the "*Handin your work*" link.

After submitting, always, *always* verify that the files you submitted are indeed the versions that you intended to submit. Every semester, a handful of students will submit the wrong lab's `written.pdf`, blank files, or old versions of their files. **We will not entertain "*Oh no, I submitted the wrong file!*" emails or Piazza posts after the due date**. We will adopt a *zero-tolerance* approach to this, and you will receive the grade of your most recent submission. *You have been warned*.

4 The 210 Sequence Library

4.1 Implementing Sequence Functions

In 15-210, we will be coding almost exclusively in SML. We assume that you are already proficient in SML syntax, its type and module system (structures, functors, etc.), and its build system (CM.make, the SML/NJ REPL, etc.). If you are unsure or rusty, please attend the **SML help session** we will have in the first week of the semester (see Piazza for details).

A very important skill in 210 is being proficient with our library functions. The library docs on the course website (`http://www.cs.cmu.edu/~15210/docs/`) show both the functions available for use, as well as the work and span of various implementations. Learning how to use these functions while keeping within the cost bounds provided to you is an absolutely vital skill in this class.

Let's begin by implementing various sequence functions.

Task 4.1 (50%). In `MkPartialSequence.sml`, implement the sequence functions listed below. Their types, descriptions, and costs can be found on the course website under *Library Docs*. **Your implementations must meet the cost bounds given for** `ArraySequence`**.**

The functor `MkPartialSequence` takes a structure `BareSeq` as input. `BareSeq` contains a few primitive sequence functions which you will need for your implementations. You may assume that the cost bounds of `BareSeq` match those given for `ArraySequence`.

Many of these functions can be implemented in just a few lines (or less!). As an example, `rev` has already been completed for you.

(0%) `val rev : 'a seq -> 'a seq`

(3%) `val map : ('a -> 'b) -> 'a seq -> 'b seq`

(3%) `val enum : 'a seq -> (int * 'a) seq`

(3%) `val mapIdx : ((int * 'a) -> 'b) -> 'a seq -> 'b seq`

(3%) `val append : 'a seq * 'a seq -> 'a seq`

(3%) `val take : 'a seq * int -> 'a seq`

(3%) `val drop : 'a seq * int -> 'a seq`

(3%) `val showl : 'a seq -> 'a listview`

(3%) `val showt : 'a seq -> 'a treeview`

(5%) `val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a`

(5%) `val iterh : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b`

(3%) `val iter : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b`

(3%) `val toList : 'a seq -> 'a list`

For the following two functions, we do not require that your implementations match the cost bounds described in the library documentation. You may implement them as efficiently (or inefficiently, but be reasonable...) as you wish. However, **you are not allowed to use the** `list` **type**.

(5%) `val flatten : 'a seq seq -> 'a seq`

(5%) `val filter : ('a -> bool) -> 'a seq -> 'a seq`

4.2 Using Sequence Functions

Now that you've implemented a subset of the functions in our `SEQUENCE` signature, let's practice using them! In the following tasks, **you are not allowed to use the `list` type**.

Task 4.2 (10%). In `MkSeqFun.sml`, implement the function

```
val allPerms : 'a seq -> 'a seq seq
```

where `(allPerms S)` evaluates to the sequence containing all permutations of `S`, in no particular order. For example, `(allPerms` $\langle 1,2,3\rangle$`)` should evaluate to some permutation of the sequence

$$\langle\langle 1,2,3\rangle, \langle 1,3,2\rangle, \langle 2,1,3\rangle, \langle 2,3,1\rangle, \langle 3,1,2\rangle, \langle 3,2,1\rangle\rangle$$

For full credit, your implementation must have $O(n^k)$ span for some constant k. (This may also be referred to as *polynomial* span). There are no requirements for work bounds, however your code will timeout on Autolab if it is taking an unreasonable amount of time.

You should assume the cost bounds of functions in the structure `PartSeq` match those given for `ArraySequence` in the documentation.

Task 4.3 (10%). In `MkSeqFun.sml`, implement the function

```
val permSort : ('a * 'a -> order) -> 'a seq -> 'a seq
```

where `(permSort cmp S)` sorts `S` with respect to `cmp`. Your implementation must be **brute force**:

1. Generate all permutations.
2. Eliminate those that are not sorted.
3. Select one of the remaining sequences.

4.3 Testing

In this course you will be expected to test your code extensively. However, our testing methodology is different from that of 15-150 – we do not test your code at compile time. It is very important that you do not have any testing code in the files you submit. Sometimes, however, we may ask you to submit test cases.

To test your MiniLab code, you can either write your own tests, or play with your code in the SML/NJ REPL (Read–Eval–Print Loop). In `Testing.sml`, we have set up the necessary structures for you. You can access your code within `structure PS : PARTIALSEQUENCE` and `structure SF : SEQFUN`.

- To write your own tests, put code in the structure `Tester` within `Testing.sml`. You can access these functions at the REPL. For example, to invoke the function `testMap`:

  ```
  - CM.make "sources.cm";
  ...
  - Tester.testMap ();
  ```

- To play with your code in the REPL, simply use the structures PS and SF. For example, here is how you would create two sequences, call append, and then call allPerms:

```
- CM.make "sources.cm";
...
- val s1 = PS.fromList [1,2,3,4];
val s1 = {ary=[|1,2,3,4|],idx=0,len=4} : int ?.BareArraySequence.seq
- val s2 = PS.fromList [5,6,7,8];
val s2 = {ary=[|5,6,7,8|],idx=0,len=4} : int ?.BareArraySequence.seq
- val perms = SF.allPerms (PS.append (s1, s2));
...
```

  You could also open PS and SF for easy access to their internals.

```
- CM.make "sources.cm";
...
- open PS;
...
- open SF;
...
- val s = allPerms (fromList [1,2,3,4]);
...
```

4.4 Grading

4.4.1 Style

Style grading for this course will be **pass / fail**. Please read over our style guide (`http://www.cs.cmu.edu/~15210/resources/style.pdf`) if you're not sure what constitutes good style. Remember, good coding style *is an art*, so use the early labs to quickly get in shape!

If you fail style grading, then you must fix the issues outlined in the comments on Autolab and show your new solution to a TA. Do not attempt to resubmit to Autolab – simply show your modified code to the TA.

More details can be found on the home page of the website (`http://www.cs.cmu.edu/~15210/`) under *Style Grading* and *Regrade Deadline*.

4.4.2 Public / Private Tests

For each section, we have set aside some fraction of points for private tests, which we will turn on after the due date. These could compose of the trickier edge cases, so getting full credit for the public tests does not necessarily mean your code is correct. *So test your code thoroughly!*

5 Written Questions

For all tasks in this section, write your answers in `written.pdf`.

5.1 Academic Integrity

The following questions are "warm-up" questions, intended to make sure that you have read and understood the academic integrity policy (see `http://www.cs.cmu.edu/~15210/`) of the course, as well as found the tools that we've set up to communicate with you.

Task 5.1 (1%). Describe the picture posted as an instructor's note on Piazza. Be creative!

Task 5.2 (3%). In each of the following situations, have the students followed or broken the collaboration policy? Briefly justify your answers with a sentence or two.

1. Ludmilla, Joaquin, and Alexander have lunch together after 210 lecture. Over lunch, they discuss the homework assignment released earlier in the week, including their progress so far. After lunch, all three go to different classes and don't think about the 210 homework until that evening.

2. While working on 213 homework near his friends from 210, Jon has a moment of insight into the 210 homework assignment. He becomes excited, and tells his friends what he thought of. Ishmael hasn't gotten that far in the homework, so he doesn't quite understand what Jon is talking about. Nonetheless, Ishmael copies down what he thinks are the key bits of what he heard to look at when he gets that far.

3. Yelena has been working on the 210 homework but can't figure out why her solution isn't compiling. She asks Abida to work through a couple of toy examples of functor syntax together, and they do so with a text editor and the SML REPL. Afterwards, Yelena gets her homework to compile and keeps working towards a solution.

Task 5.3 (1%). If you hand in your homework 25 hours after the posted due date, and you have no late days remaining, what is your maximum possible score? (Assume full score is 100%)

5.2 Asymptotics

Let's begin by recalling the definition of big-O:

Definition 5.1. Consider functions $f : \mathbb{N} \to \mathbb{R}_+$ and $g : \mathbb{N} \to \mathbb{R}_+$. We say that $f \in O(g)$ if and only if there exist constants $N_0 \in \mathbb{N}$ and $c \in \mathbb{R}_+$ such that for all $n \geq N_0$, $f(n) \leq c \cdot g(n)$.

Task 5.4 (5%). Rearrange the list of functions below so that it is ordered with respect to O (i.e. for every f at index i, every f' with index less than i satisfies $f' \in O(f)$). You don't need to prove anything; just state the ordering by number. For example, if you think this list is already ordered, just say "1234567".

1. $f(n) = 36n^{52} + 15n^{18} + n^2$
2. $f(n) = 2n^{1.5}$
3. $f(n) = (n^n)!$
4. $f(n) = 43^n$
5. $f(n) = 210n$
6. $f(n) = \lg(\lg(\lg(\lg(n))))$
7. $f(n) = n^{\lg(n^2)}$
8. $f(n) = n^{n!}$

Task 5.5 (10%). Carefully **prove** each of the following statements, or provide a counterexample and **prove** that it is in fact a counterexample. You should refer to the definition of big-O. Remember that verbose proofs are not necessarily careful proofs.

1. O is a transitive relation on functions. That is to say, for any functions f, g, h, if $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.
2. O is a symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$, then $g \in O(f)$.

5.3 Pseudocode: Sequence Notation

Throughout the lecture notes, we use mathematical notation to describe code. Some basic documentation for this pseudocode is available on the website (`http://www.cs.cmu.edu/~15210/notation.html`). Review the syntax, and complete the following exercises.

Task 5.6 (5%). In each of the following, assume you are given a sequence S and $n = |S|$. Convert each algorithmic description into sequence notation.

1. Create a sequence of length $\left\lfloor \frac{n}{2} \right\rfloor$ that contains only the odd indices of S.
2. Create a sequence of length $\frac{n \cdot (n+1)}{2}$ that contains all non-empty contiguous subsequences of S.

Task 5.7 (5%). Convert the following code into sequence notation. Then give a high level algorithmic description of what the code is doing. Keep in mind that while a one-to-one correspondence between code and pseudocode is possible, it might not be desirable.

```
fun mystery (A, B) =
  let
    fun f S x = (length (filter (fn y => x = y) S)) > 0
  in
    filter (not o f B) A
  end
```