1 Welcome to 15-210!

Hello and welcome to 15-210! In this very first (zeroth?) lab, we just want you to get used to the environment that you will be working in for the subsequent labs, and get you acquainted with the 210 sequence library and its documentation. You'll also prove some basic asymptotic properties to get you started thinking analytically. This should be a relatively simple lab, so get started early and get any issues sorted out before *ParenLab* comes out next week!

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. `support/BARESEQUENCE.sig`
3. `Tests.sml`
4. * `MkPartialArraySequence.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 need to modify are in the root of the handout directory, and those that are either for reference or helpers for testing are in `support/`. Every handout will also include the `lib/` subdirectory, where our library files are located. Other files like `sources.cm` and `support/*.cm` are for when you run `CM.make` in SMLNJ; you may ignore these files.

3 Submission

3.1 Autolab

We will be using Autolab (`https://autolab.cs.cmu.edu/15210-s14`) for the duration of this course, for the handing-in, autograding, and style grading of labs, as well as for overall grade management. Each lab will have several *grade columns*, corresponding to the different collections of sections in the lab.

When you submit your work to Autolab, the autograder will populate only the columns with *public* tests available. Any *private* tests will be run only after the lab's due date. For more detail about tests, see section 5.5.2. In addition, style grades manually entered by your TAs will also be shown here.

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 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/#grading`) of the course, as well as found the tools that we've set up to communicate with you.

Task 4.1 (1%). Describe the picture posted as an instructor's note on Piazza. Be creative!

Task 4.2 (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 4.3 (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 The 210 Sequence Library

5.1 Implementing Sequence Functions

In 15-210, we will be using SML exclusively to implement the algorithms that you will learn in lectures, recitations, and labs. Seeing as 15-150 is a prerequisite for this class, we are going to assume that you are already proficient in SML syntax, its type system (including functors, structures, etc.), and its build system (CM.make, running SMLNJ, etc.). If you are unsure, 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 (e.g. the `SEQUENCE` signature), as well as their work and span. Learning how to use these functions while keeping within the cost bounds provided to you is an absolutely vital skill in this class. Even being comfortable navigating the documentation page above will help by decreasing your lookup time!

To kick you off, here are some simple SML functions that we want you to write, to warm up the relevant parts of your brain after a relaxing break from SML.

Task 5.1 (37%). In `MkPartialArraySequence.sml`, implement the following sequence functions, using the functions in the `BARESEQUENCE` signature in `support/BARESEQUENCE.sig`. Their types, descriptions and costs can be found on the course website, on the "Library Docs" page. **Note: your implementation must meet the cost bounds stated by the documentation.**

The functor `MkPartialArraySequence.sml` takes, as input, the structure `BareArraySequence` which ascribes to `BARESEQUENCE`. You may assume that the cost bounds of the functions in this implementation match those stated in our documentation.

As an example, `rev` has been done for you.

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

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

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

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

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

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

(7%) `val iterh : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b`

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

In this course (and in `BareArraySequence`), the `'a seq` type is implemented using an SML `array` (`http://www.standardml.org/Basis/array.html`), wrapped in a *record* type:

```
{ ary : 'a array, idx : int, len : int }
```

A *record* is just a labeled tuple. **You should NOT use the record type in this lab**; this is just an interesting tidbit, and it is not imperative that you fully understand it early on in the course (for more details, see `https://piazza.com/class/hp3ascm5zmb49c?cid=8`).

Task 5.2 (3%). `iter` and `iterh` are often misunderstood functions. To make sure you understand how they work, step through the following call to `iterh`, **using your implementation of the function**, until it evaluates to a value. Make sure not to skip any steps – this should be a familiar exercise from 15-150.

Note: this is the only time in 210 that you will get credit for stepping through your code in this manner, since *this is in no way an analysis*; it is just to help with your understanding of the function.

Let `s` be the sequence $\langle 1, 2, 3, 4 \rangle$. Step through the call:

```
iterh (fn (st, x) => st ^ (Int.toString x)) "" s
```

5.2 Using Sequence Functions

Now that you've implemented a subset of the functions in our `SEQUENCE` signature, let's practice using some of them. Part of the difficulty of the rest of the labs is figuring out which library functions you can use to help you solve a particular problem / algorithm, and whether it meets the cost bounds provided. Note that these functions in this section are *designed to be simple*, and so are much easier than those in future labs.

Task 5.3 (10%). In `MkSeqFun.sml`, implement the function:

```
val allHarmonics : int -> real seq
```

Where $n > 0$, and `allHarmonics n` evaluates to the sequence $\langle H_1, \ldots H_n \rangle$, where H_i is the ith harmonic number: the sum of the reciprocals of the first i natural numbers, or $H_i = \sum_{k=1}^{i} \frac{1}{k}$. Note that H_0 is not included; **your function should raise the exception `BadHarmonics` when any $n \leq 0$ is passed into it**.

Since the purpose of this task is to practice using sequences, **you are not permitted to use the `list` type**. For full credit, your implementation must have $O(n)$ work and span.

For example, `allHarmonics 6` evaluates to

$$\langle 1.0, 1.5, 1.83333333333, 2.08333333333, 2.28333333333, 2.45 \rangle$$

Task 5.4 (10%). In `MkSeqFun.sml`, implement the function:

```
val groupedHarmonics : int -> int -> (int * real seq) seq
```

Where `groupedHarmonics n k` evaluates to $\langle (0, G_0), (1, G_1), \ldots, (m-1, G_{m-1}) \rangle$, where G_0 is a `real seq` that contains the first k harmonic numbers, G_1 contains the next k, ..., continuing until we have assigned each of the first n harmonic numbers to exactly one of the G_i (m is the number of groups).

To be mathematically precise, each of $G_0, \ldots, G_{m-2}$ must have exactly k harmonic numbers, while G_{m-1} must have at most k. The concatenation G^* of $G_0, \ldots, G_{m-1}$ must be a `real seq` containing the first n harmonic numbers in order (and thus $|G^*| = n$).

For example, `groupedHarmonics 6 3` evaluates to

$$\langle (0, \langle 1.0, 1.5, 1.83333333333 \rangle), (1, \langle 2.08333333333, 2.28333333333, 2.45 \rangle) \rangle$$

while `groupedHarmonics 5 2` evaluates to

$$\langle (0, \langle 1.0, 1.5 \rangle), (1, \langle 1.83333333333, 2.08333333333 \rangle), (2, \langle 2.28333333333 \rangle) \rangle$$

Note: In section 5.3, you will implement the function `printGroups`, which will help in your debugging of `groupedHarmonics`. You may want to skip ahead there before coming back to this task.

5.3 Debugging in SML

A very important debugging skill that will help you through tough labs is **printing**. Printing things in SML can sometimes be a pain; but trust us, for each lab, taking the time to write utility debugging functions to see *just what your function is returning* will save you hours of suffering later.

Task 5.5 (5%). Back in `MkPartialArraySequence.sml`, implement the sequence library function:

```
val toString : ('a -> string) -> 'a seq -> string
```

Where `toString elemToStr s` evaluates to a string representation of `s`. This representation begins with the character '<' and then the results of applying `elemToStr` to each element of `s` in ascending index order, interleaved with ',' (no space). It then ends with '>'. For example:

```
val s = fromList [(0, 1), (1, 2), (2, 3)]
fun pairToStr (a, b) =
    "(" ^ (Int.toString a) ^ ", " ^ (Int.toString b) ^ ")"
val str = toString pairToStr s
(* str should be exactly the string "<(0, 1),(1, 2),(2, 3)>" *)
```

You may find the function `String.concatWith` useful. This function is part of the `STRING` signature in the SML Basis library. See its documentation (`http://www.standardml.org/Basis/string.html`) for more details.

Task 5.6 (5%). Now in `MkSeqFun.sml`, implement the function:

```
val printGroups : (int * real seq) seq -> unit
```

Where `printGroups G` prints the string representation of G, the output of the `groupedHarmonics` function. Use SML's built-in printing function `val print : string -> unit` to do so. Also make sure to print a newline character "\n" at the end.

5.4 Testing

In this course you will be expected to test your code extensively. In addition to the autograder on Autolab, we have also prepared a testing structure `Tester` in `support/Tester.sml`, so that you may test your code before submitting. You do not have to submit test cases for this lab, but you will often have to for subsequent labs. However, it is still in your best interest to test your code thoroughly before submission.

It is very important that there is no testing code in any other file, other than `Tests.sml` when you submit your solution to Autolab. Note that unlike the 15-150 testing methodology, our testing structure **does not test your code at compile time.** In `Tests.sml`, add your tests to the appropriate list (look at the sample test cases for reference).

Note that while you may test your code on your local machine, it might be best to work on an Andrew machine, since there might be differences between your local machine and what Autolab's environment. With that said, after defining your test cases, run the following from a shell:

```
$ smlnj
Standard ML of New Jersey v110.xx
- CM.make "sources.cm";
...
- Tester.testPartSeq ();
...
- Tester.testSeqFun ();
...
- Tester.printGroups n k;
...
```

Note that `Tester.testPartSeq` and `Tester.testSeqFun` compare your implementations to reference solutions, using the test cases you put in `Tests.sml`, and respond with passing or failing feedback. However, `Tester.printGroups n k` simply runs `printGroups (groupedHarmonics n k)` using your implementations for both functions. Use this to visually test both functions.

5.5 Grading

5.5.1 Style

In this lab, we've set aside some points for your coding style. Please refer to the *SML Style Guide* we've written (`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!

Task 5.7 (5%). You don't have to submit anything for this task; your TAs will grade your code and award these points accordingly after the due date.

5.5.2 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!*

6 Asymptotics

For this problem, let's recall the definition of big-O:

Definition 6.1. 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 6.1 (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 6.2 (15%). 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 $g \in O(f)$.
3. O is an anti-symmetric relation on functions. That is to say, for any functions f and g, if $f \in O(g)$ and $g \in O(f)$, then $f = g$.