CSc 372, Fall 2006
Assignment 2
Due: Thursday, September 28 at 23:59:59

Problem 1. (3 points) `zip.sml`

Write a higher-order function `zip F L1 L2` which, like zipping a zipper, zips together the lists `L1` and `L2` using the function `F`. Here is an example:

```
- fun pair(x,y) = (x,y);
val pair = fn : 'a * 'b -> 'a * 'b

- zip pair [1,2,3] (explode "abcdef");
val it = [(1,#"a"),(2,#"b"),(3,#"c")] : (int * char) list
```

The first tuple was formed by calling `pair(1, #"a")`, the first elements in `[1,2,3]` and `explode "abcdef"`, respectively. Likewise, the second tuple is the result of calling `pair` with the second element in each of the lists. It is not required that the lists be of equal lengths—the zipping simply stops when the shorter list has been consumed.

Note the type of `zip`:

```
- zip;
val it = fn : ('a * 'b -> 'c) -> 'a list -> 'b list -> 'c list
```

More examples:

```
- zip (fn(a,b) => a*2 + b*3) [1,2,3,4,5] [10, 20];
val it = [32,64] : int list

- zip op^ (map str (explode "abc")) (map Int.toString (iota 5));
val it = ["a1","b2","c3"] : string list

- zip op+ [] [1,2];
val it = [] : int list
```

It's ok to use recursion for this problem.

This is a very well known problem but it's a good exercise so I strongly recommend you resist the temptation to look for a solution on the net or in some book.

Problem 2. (3 points) `unzip.sml`

Write a function `unzip(L)` that separates a list of 2-tuples into two lists. The first list holds the first element of each tuple. The second list holds the second element of each tuple. **Don't overlook the problem restrictions following the examples.**

```
- unzip;
val it = fn : ('a * 'b) list -> 'a list * 'b list

- unzip [(1,10),(2,20),(3,30)];
val it = ([1,2,3],[10,20,30]) : int list * int list

- unzip [("just","testing")];
val it = (["just"],["testing"]) : string list * string list
```

Don't worry about the result you might get with an empty list:

```
- unzip [];
stdIn:7.1-7.9 Warning: type vars not generalized because of
   value restriction are instantiated to dummy types (X1,X2,...)
val it = ([],[]) : ?.X1 list * ?.X2 list
```

RESTRICTIONS for `unzip.sml`:

Your solution must be non-recursive.

Your solution may only use `map`, the composition operator, and these two functions:

```
fun first(x, _) = x
fun swap(x,y) = (y,x)
```

Problem 3. (2 points) `repl.sml`

Write a function `repl(x,n)` that produces a list consisting of `n` copies of `x`. Assume that `n` is greater than zero.

```
- repl;
val it = fn : 'a * int -> 'a list

- repl("a", 5);
val it = ["a","a","a","a","a"] : string list

- repl(3, 10);
val it = [3,3,3,3,3,3,3,3,3,3] : int list
```

RESTRICTION: Your solution must look like this:

```
fun repl(x,n) = List.tabulate(...);
```

Replace the `...` with code that makes it work. Ten characters should be about enough but it can be as long as you need.

You'll need to look at the documentation for `List.tabulate` to see what it does. Go to the Resources

page on the website and follow the link to *Man pages for the SML Basis Library*. You'll find `tabulate` on the page for *The List structure*.

Problem 4. (2 points) `doubler.sml`

Consider a function named `doubler` that duplicates each value in an `int list`:

```
- doubler;
val it = fn : int list -> int list

- doubler [5,1,3,7];
val it = [5,5,1,1,3,3,7,7] : int list

- doubler [];
val it = [] : int list

- doubler [1];
val it = [1,1] : int list
```

RESTRICTION: Your solution must look like this:

```
val doubler = foldr ...;
```

That is, you are to create `doubler` using a partial application of `foldr`.

Replace the `...` with code that makes it work. Thirty characters should be about enough but it can be as long as you need.

Note that `doubler` is specified as operating on an `int list` because attempting to create an `'a list -> 'a list` version produces the *"type vars not generalized..."* warning. Your solution should force the type by somewhere (anywhere) using an explicit type specification, like `(x:int)` instead of just `x`.

Problem 5. (5 points) `repeat.sml`

Write the `repeatValues` function described on slide 192. Start by copying the code `InfList` code from the preceding slides.

IMPORTANT:
THERE IS A RESTRICTION FOR ALL PROBLEMS THAT FOLLOW
READ THIS CAREFULLY

All of the following problems must be solved without using any explicit recursion. You may use functions in the ML Basis which could be recursive but **the code you `turnin` for these problems must not exhibit recursion**. Note that there is both direct and indirect recursion. With direct recursion, a function `f` calls `f`. With indirect recursion, `f` might call `g` and then `g` might call `f`. **BOTH DIRECT AND INDIRECT RECURSION ARE ABSOLUTELY FORBIDDEN. AN INFRACTION MAY RESULT IN A SCORE OF ZERO FOR THE PROBLEM.**

You can use code from the slides or developed in lecture but only if it is not recursive. For example, you can't use `maxint` or `member` from slide 83, or `repl` from slide 103.

There are two exceptions: you can use `m_to_n`, in curried form, and `reduce`.

One example of code that (almost) meets this restriction is the `optab` example, on slide 177. I say almost because it uses a function `xrepl`, mentioned on slide 165, whose definition is not shown. With a non-recursive `xrepl`, `optab` meets the restrictions.

Problem 6. (3 points) `cpfx.sml`

Shortly after assignment 1 is due the instructor's solution for `cpfx` will appear in `/home/cs372/fall06/a1/cpfx.sml`. The `cpfx` function is recursive. Rewrite that function to be non-recursive but still make use of the `cpfx1` function.

Problem 7. (10 points) `mfilter.sml`

Write a function `mfilter` that accepts a list of integer ranges and produces a function that when applied to a list of integers produces the integers that do not fall into any of the ranges.

```
- mfilter;
val it = fn : (int * int) list -> int list -> int list

- mfilter [(3,7)] (iota 10);
val it = [1,2,8,9,10] : int list

- mfilter [(10,18), (2,5),(20,20)] (iota 25);
val it = [1,6,7,8,9,19,21,22,23,24,25] : int list

- mfilter [] (iota 3);
val it = [1,2,3] : int list

- val f = mfilter [(5,20)];
val f = fn : int list -> int list

- f (iota 21);
val it = [1,2,3,4,21] : int list
```

```
- f (m_to_n 10 15);
val it = [] : int list
```

Assume that for a range `(x, y)` that `x <= y`. You won't see a range like `(10,1)`. As you can see above, ranges are inclusive. A range like `(1,3)` blocks `1`, `2`, and `3`.

Remember: No explicit recursion!

Problem 8. (10 points) `nnn.sml`

The behavior of the function you are to write for this problem is best shown with an example:

```
- nnn [3,1,5];
val it = ["3-3-3","1","5-5-5-5-5"] : string list
```

The first element is a `3` and that causes a corresponding value in the result to be a string of three `3`s, separated by dashes. Then we have one `1`. Then five `5`s.

Note that if a string is long, ML does not show the full value—it shows part and then uses a `#` to indicate that some is not shown:

```
- nnn [100,1,2,3];
val it =
["100-100-100-100-100-100-100-100-100-100-100-100-100-100-100-1
00-10#",
   "1","2-2","3-3-3"] : string list

- size (hd it);
val it = 399 : int
```

However, the size of 399 is correct—100 copies of `"100"` and 99 copies of `"-"` to separate them.

The type of `nnn` is `int list -> string list`. Assume that the values are greater than zero.

Remember: No explicit recursion!

Problem 9. (10 points) `exsufs.sml`

Consider the following entries that specify the spelling of a word and spelling of forms of the word with suffixes:

```
program,s,#ed,#ing,'s
code,s,d,@ing
```

If a suffix begins with a pound sign (#) it indicates that the last letter of the word should be doubled when adding the suffix. If a suffix begins with at-sign (`@`), it indicates that the last letter of the word should be dropped when adding the suffix. In other cases, including the possessive (`'s`), the suffix is simply added. The above two entries represent the following words:

```
program
programs
programmed
programming
program's
```

```
code
codes
coded
coding
```

Write a function `exsufs(L)` that accepts a list of entries and prints the all the forms, including the word with no suffix.

```
- exsufs;
val it = fn : string list -> unit

- exsufs ["adrift", "mob,s,#ed,#ing", "groove,s,d,@ing"];
adrift
mob
mobs
mobbed
mobbing
groove
grooves
grooved
grooving
val it = () : unit

- exsufs [];
val it = () : unit
```

A word may have any number of suffixes with an arbitrary combination of types.

Assume that the entries are well formed. You won't see things like zero-length suffixes.

The order of the output must match the order shown above.

Remember: No explicit recursion!

Problem 10. (20 points) `pancakes.sml`

In this problem you are to print a representation of a series of stacks of pancakes. Let's start with an example:

```
- pancakes;
val it = fn : int list list -> unit

- pancakes [[3,1],[3,1,5]];
     ***
***   *
 *  *****
val it = () : unit
```

The list specifies two stacks of pancakes: the first stack has two pancakes, of widths `3` and `1`, respectively. The second stack has three pancakes. Pancakes are always centered on their stack. A single space separates each stack. Pancakes are always represented with asterisks.

Here's another example:

```
- pancakes [[1,5],[1,1,1],[11,3,15],[3,3,3,3],[1]];
                          ***
      *   ***********    ***
  *   *       ***        ***
***** * ************** *** *
val it = () : unit
```

There are opportunities for creative cooking:

```
- pancakes [[7,1,1,1,1,1],[5,7,7,7,7,5],[7,5,3,1,1,1],
            [5,7,7,7,7,5], [7,1,1,1,1,1],[1,3,3,5,5,7]];
*******  *****  *******  *****  *******     *
   *     *******  *****  *******     *       ***
   *     *******   ***   *******     *       ***
   *     *******    *    *******     *      *****
   *     *******    *    *******     *      *****
   *      *****     *     *****      *     *******
val it = () : unit
```

Assume that there is at least one stack. Assume all stacks have at least one pancake. Assume all widths are greater than zero. The smallest order you'll see is this:

```
- pancakes [[1]];
*
val it = () : unit
```

To avoid problems with centering, pancake widths must be odd. If an even width is specified, the exception `even_cake` is raised:

```
- pancakes [[3,1],[3,1,4]];

uncaught exception even_cake
```

Remember: No explicit recursion!

Problem 11. (20 points) `group.sml`

For this problem you are to write a program that reads a text file and prints the file with a bar of dashes inserted whenever the first character on a line differs from the first character on the previous line. Additionally, the lines from the input file are numbered.

```
% cat group.1
able
academia
algae
carton
fairway
hex
hockshop
%
```

```
% sml
...
- group "group.1";
1 able
2 academia
3 algae
----------
4 carton
----------
5 fairway
----------
6 hex
7 hockshop
val it = () : unit
```

Note that the only the lines from the input file are numbed—the separators are not numbered.

Another example:

```
% cat group.2
   September 2006
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
% sml
...
- group "group.2";
1    September 2006
----------
2 Su Mo Tu We Th Fr Sa
----------
3                1  2
4  3  4  5  6  7  8  9
----------
5 10 11 12 13 14 15 16
6 17 18 19 20 21 22 23
----------
7 24 25 26 27 28 29 30
val it = () : unit
```

Yes, numbered lines of digits make it a little confusing. The first two lines are considered different because the first characters of each are a blank and an `"S"`, respectively—don't look at that `"S"` in September.

Two more examples:

```
% cat group.3
test
test
test
%
```

```
% sml
...
- group "group.3";
1 test
2 test
3 test
val it = () : unit
^D
%
% cat group.4
~/ch/a2 1033 $ cat group.4
a
b
a
b
a
a
b
a
a
a
b
% sml
...
- group "group.4";
1 a
----------
2 b
----------
3 a
----------
4 b
----------
5 a
6 a
----------
7 b
----------
8 a
9 a
10 a
----------
11 b
val it = () : unit
```

Note that in the `group.4` output when the line numbers grow to two digits the output is shifted a column to the right. That's ok.

The separator lines are ten dashes (minus signs).

The "grep" on slide 149 has an example of how to read from a file but do one thing different: Use `TextIO.inputAll` instead of `TextIO.input`. (`TextIO.input` seems to not read the full contents of larger files.) Assume that the input file exists.

As mentioned in class, a problem with chopping up a string of file contents with `String.tokens` is that empty lines—consecutive newlines, essentially—are deleted as a group:

```
- String.tokens (fn(c) => c = #"\n") "x\n\n\ny\n";
val it = ["x","y"] : string list
```

The `wc` command would report the corresponding file has four lines but we get a list with only two elements. Simply assume there are no empty lines in the input file.

Assume that there is at least one line in the input file. (A one-line file will produce no separators, of course.)

The `group.[1234]` files shown above can be found in `/home/cs372/fall06/a2` on `lectura`.

Remember: No explicit recursion!

Problem 12. (7 points) `answers.txt`

Create a text file named `answers.txt` with answers to the following questions. **DO NOT submit a Word document, PDF, rich text file, etc.**—I want plain text. Just use whatever editor you're using to edit your ML source code to type up your answers.

(a) (2 points) Given `swapArgs` and `curry` from the slides, explain what the following expression is doing:

```
val f = swapArgs (curry String.sub) 0
```

Also, show an example of how it might be used. What's a good name for the value that's created?

(b) (1 point) Given the `doubler` function from earlier in the write-up, what's a quick way to make `quadrupler`, which produces four copies of each value in a list?

(c) (4 points) Consider this sequence of calls in ML:

```
(drawLine(canvas, 0, 0, w, 0);
 drawLine(canvas, 0, 0, 0, h);
 drawLine(canvas, w, 0, w, h);
 drawLine(canvas, 0, h, w, h))
```

The code is repetitious—`drawLine` and `canvas` are each specified four times. By making a slight revision to `drawLine` and using `map`, the same work could be accomplished with code in which the names `drawLine` and `canvas` appear only once each. Describe a revision to `drawLine` and write a revised version of the above code in which `drawLine` and `canvas` appear only once. (No, you can't simply use a `val`!)

(d) (1 point extra credit) Haskell B. Curry has the rare distinction of having both his first and last name honored in separate ways by later generations. (The Haskell language and curried functions.) Name another person, in any field of study, who has been similarly honored.

Deliverables

The deliverables for this assignment are these files: `zip.sml`, `unzip.sml`, `repl.sml`,

`doubler.sml`, `repeat.sml`, `cpfx.sml`, `mfilter.sml`, `nnn.sml`, `exsufs.sml`, `pancakes.sml`, `group.sml` and `answers.txt`. That list can be found on `lectura` in `/home/cs372/fall06/a2/delivs`.

Use the `turnin` tag `372_2` to submit your solutions.

The following command, which uses *command substitution*, submits all the deliverables.

```
turnin 372_2 `cat /home/cs372/fall06/a2/delivs`
```

Note that those are backquotes that surround the `cat` command.

Corrections and FAQs, late submissions, `turnin`, retests, etc.

Refer to the write-up for the first assignment for details on these topics and similar ones.

Miscellaneous

When developing code that performs a complex series of transformations on data I recommend the technique shown in class on September 12 with "grep", where each expression was first placed in the `in ... end` and then, once working, moved up into the `let...in`, bound to a name with `val`, and the process repeated for the next computation. If you missed that, see me or Poorna during office hours and we'll show it to you.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem before you ask for help with it. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems—assignments are not intended to kill you. Try to think in terms of spending no more than fifteen hours on this assignment. (Ten hours a week times two weeks minus five hours for lectures.) Seek the help of Poorna and me as needed to meet your time budget.

As stated in the syllabus, a ten point homework problem corresponds to one point on your final average.

You may use elements of ML that we haven't studied, with one exception: you may not use any of ML's imperative elements <u>except</u> expression sequencing. (We haven't studied any of the imperative elements except for expression sequencing: *`(e1, e2, ..., eN)`*.)

If you wish, you may incorporate code from lectures in your solutions.

Feel free to use comments to document your code as you wish but note that no comments, not even your name, are required.

Solutions will be scored only on correctness and compliance with the cited restrictions. If you're worried about whether a solution complies with the restrictions, mail it to `cs372-staff` and we'll look it over, assuming time permits. (Avoid the last-minute rush!) **<u>BE VERY CAREFUL that you don't get the addresses mixed up and mail your solution to the whole class.</u>**

If you're developing elsewhere than `lectura`, be sure that you're using v110.57 of `sml`.

I hate to have to mention it but keep in mind that cheaters don't get a second chance. If you give your code to somebody else and they turn it in, you'll both likely fail the class, and more. (See the syllabus for the details.)