CSc 372, Fall 2006
Assignment 1
Due: Thursday, September 14 at 23:59:59

Problem 1. (5 points) `ftypes.sml`

Write five functions named `f1`, `f2`, `f3`, `f4` and `f5` having the following types, respectively:

```
f1: ('a * int) * (int * 'b) -> 'b * 'a list

f2: int list list list -> 'a list list list

f3: 'a * (unit list * unit) -> 'b * 'a

f4: 'a * (''b * ''c) * ''b * 'a -> ''b * ''c list * 'a

f5: 'a list list * 'a list * 'a -> char
```

You may NOT:

> Use `bool`, `char`, `int`, `real` or `string` literals. For example, the definition `fun f("x") = 5` is not acceptable for two reasons: it uses a `string` literal and an `int` literal.
>
> Use any explicit type specifications. For example, `fun f(a:int) = a` is not acceptable.
>
> Define any additional functions.
>
> Use `let` expressions, compound expressions (`(e1; e2; e3 ...)`) or the "as" pattern matching construct. (We haven't covered the latter two.)

Examples:

```
- use "ftypes.sml";
[loading messages...]

- f1;
val it = fn : ('a * int) * (int * 'b) -> 'b * 'a list
- f2;
val it = fn : int list list list -> 'a list list list
- f3;
val it = fn : 'a * (unit list * unit) -> 'b * 'a
- f4;
val it = fn : 'a * (''b * ''c) * ''b * 'a -> ''b * ''c list * 'a
- f5;
val it = fn : 'a list list * 'a list * 'a -> char
```

Note that the task at hand is to create functions that cause the ML type deduction system to determine a particular type. The functions will not be run, only loaded, and need not perform any meaningful computation or even terminate. It is acceptable for functions to produce non-exhaustive match warnings.

Problem 2. (6 points) `to_b.sml`

Write a function `to_b(n)` of type `int -> string` that returns a string of ones and zeroes corresponding to the bit pattern represented by the integer `n`, which is assumed to be non-negative.

Examples:

```
- to_b 5;
val it = "101" : string

- to_b 1;
val it = "1" : string

- to_b 127;
val it = "1111111" : string

- to_b 1025;
val it = "10000000001" : string
```

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

Write a function `strings_to_s(L, separator)` of type `string list * string -> string` that concatenates the strings in `L` into a single string with `separator` between each string from `L`. Don't overlook the restrictions cited below.

Examples:

```
- strings_to_s(["a","bc","def"], ".");
val it = "a.bc.def" : string

- strings_to_s(["a","bc"], ", ");
val it = "a, bc" : string

- strings_to_s(["a","bc","def", "g", "h"], "");
val it = "abcdefgh" : string

- strings_to_s(["test"], "...");
val it = "test" : string

- strings_to_s([], "...");
val it = "" : string

- strings_to_s(["","","x","",""], "...");
val it = "......x......" : string
```

Restrictions for `strings_to_s`:

- You may use no integer values (literal or otherwise) in your solution.
- You may not use if-then-else.
- You may call no functions other than `strings_to_s`.

Problem 4. (8 points) `cpfx.sml`

Write a function `cpfx`, of type `string list -> string`, that produces the common prefix, if any, among a list of strings.

If there is no common prefix or the list is empty, return an empty string. If the list has only one string, then that string is the result.

Examples:

```
- cpfx ["abc",  "ab", "abcd"];
val it = "ab" : string

- cpfx ["abc",  "abcef", "a123"];
val it = "a" : string

- cpfx ["xabc",  "xabcef", "axbc"];
val it = "" : string

- cpfx ["obscure","obscures","obscured","obscuring"];
val it = "obscur" : string

- cpfx ["xabc"];
val it = "xabc" : string

- cpfx [];
val it = "" : string
```

Problem 5. (12 points) `paired.sml`

Write a function `paired(s)` of type `string -> bool` that returns `true` iff (if and only if) the parentheses in the string `s` are properly paired.

Examples (with properly paired parentheses):

```
- paired "()";
val it = true : bool

- paired "(a+b)*(c-d)";
val it = true : bool

- paired "(()()(()))";
val it = true : bool

- paired "((1)(2)((3)))";
val it = true : bool

- paired "((()(()()((((()))))((()))))))";
val it = true : bool
```

Examples with failures:

```
- paired "(";
val it = false : bool

- paired ")";
val it = false : bool

- paired "())";
val it = false : bool

- paired "(a+b)*((c-d)";
val it = false : bool

- paired ")(";
val it = false : bool
```

Note that you need only pay attention to parentheses:

```
- paired "a+}(/.$#${)[[[";
val it = true : bool

- paired"a+}(/.$#$([])[";
val it = false : bool
```

Problem 6. (12 points) `splits.sml`

Consider splitting a list into two non-empty lists and creating a 2-tuple from those lists. For example, the list `[1,2,3,4]` could be split after the first element to produce the tuple `([1],[2,3,4])`. In this problem you are to write a function `splits(L)` of type `'a list -> ('a list * 'a list) list` that for the list `L` produces a list of tuples representing all the possible splits of the list.

Examples:

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

- splits [1,2,3,4];
val it = [([1],[2,3,4]),([1,2],[3,4]),([1,2,3],[4])]
  : (int list * int list) list

- splits ["x", "y", "z"];
val it = [(["x"],["y","z"]),(["x","y"],["z"])]
  : (string list * string list) list

- splits [true, false];
val it = [([true],[false])] : (bool list * bool list) list

- length(splits(m_to_n(1,50)));  (* m_to_n is on slide 76 *)
val it = 49 : int
```

In order to be split, a list must contain at least two elements. If `splits` is called with a list that has fewer than two elements, the exception `short_list` should be raised:

```
- splits [1];

uncaught exception short_list

- splits [];
Warning: type vars not generalized because of
   value restriction are instantiated to dummy types (X1,X2,...)

uncaught exception short_list
```

Don't worry about that "`type vars not generalized...`" warning.

Problem 7. (15 points) `street.sml`

In this problem you are to write a function `street(L)` that prints a crude representation of the buildings along a street, as described by `L`, a list of `int * int * char` tuples, each of which represents a building. The elements of the tuple represent the width, height, and character used to create the building, respectively.

Consider this example:

```
- street([(3,2,#"x"),(2,6,#"y"),(5,4,#"z")]);

   yy
   yy
   yyzzzzz
   yyzzzzz
xxxyyzzzzz
xxxyyzzzzz
----------
val it = () : unit
```

The street is composed of three buildings. As specified by the first tuple, the first building has a width of three, a height of two, and is composed of "`x`"s. The second tuple specifies a width of two, a height of six, and that "`y`"s be used for the second building. The third building has a width of five, a height of four, and is made of "`z`"s. Note that a blank line appears above the buildings and a line of hyphens (minus signs, not underscores) provides a foundation for the buildings.

Note that this function differs from all the other functions on this assignment. It produces printed output and returns `unit`. The other functions simply return the value of interest.

Open spaces may be placed between buildings by using buildings of zero height:

```
- street([(3,0,#"a"),(3,4,#"b"),(1,0,#"c"),(5,7,#"d"),(2,0,#"e")]);

        ddddd
        ddddd
        ddddd
   bbb  ddddd
   bbb  ddddd
   bbb  ddddd
   bbb  ddddd
--------------
val it = () : unit
```

Note that the foundation (the line of hyphens) extends to the left of the `"b"` building and to the right of the `"d"` building because of the zero-height `"a"` and `"e"` buildings.

You may assume that: (1) at least one building is specified (2) a building width is always greater than zero (3) a building height is always greater than or equal to zero.

Additional examples:

```
- street([(2,5,#"x")]);

xx
xx
xx
xx
xx
--
val it = () : unit


- street([(5,0,#"x")]);

-----
val it = () : unit

- street;
val it = fn : (int * int * char) list -> unit
```

Again, keep in mind that `street` produces output (use the `print` function) and returns `unit`.

Problem 8. (15 points) `editstr.sml`

For this problem you are to write a function `editstr(s, mods)` that applies a sequence of modifications (`mods`) to the string `s` and returns the resulting string. Here is the type of `editstr`:

```
- editstr;
val it = fn : string * (string * string * string) list -> string
```

Note that `mods` is a list of tuples. Here is a tuple that specifies that every blank is to be replaced with an underscore:

```
("replace", " ", "_")
```

The other type of modification is "mapping":

```
("map", "aeiou", "AEIOU")
```

The above tuple specifies that every occurrence of `"a"` should be replaced with `"A"`, `"e"` with `"E"`, etc. A tuple such as `("map", "aeiouAEIOU", "**********")` specifies that all vowels should be replaced with asterisks.

Here is an example of a call that specifies a sequence of two modifications, first a replacement and then a mapping:

```
- editstr("just a test", [("replace", " ", "_"),
                          ("map", "aeiou", "AEIOU")]);
val it = "jUst_A_tEst" : string
```

For `"replace"`, the second element of the tuple is expected to be a one-character string. The third element, the replacement, is a string of any length. For example, we can remove `"o"`s and triple `"e"`s like this:

```
- editstr("toothsomeness", [("replace", "o", ""),
                            ("replace", "e", "eee")]);
val it = "tthsmeeeneeess" : string
```

Another example:

```
- editstr("5203-3100-1230", [("map", "123456789", "xxxxxxxxx"),
                             ("replace", "x", "")]);
val it = "0-00-0" : string
```

Any number of modifications can be specified. If the modifications list is empty, the original string is returned.

```
- editstr("test", []);
val it = "test" : string
```

The exception `bad_spec` is raised to indicate any of three error conditions:

- The type of modification is something other than `"map"` or `"replace"`.
- For `"replace"`, the length of the string being replaced is not one.
- For `"map"`, the two strings are not the same length.

Here are examples of each, in turn:

```
- editstr("test", [("x", "y", "z")]);

uncaught exception bad_spec

- editstr("test", [("replace", "test", "z")]);

uncaught exception bad_spec

- editstr("test", [("map", "x", "yz")]);

uncaught exception bad_spec
```

Problem 9. (6 points) `answers.txt`

For two points per question, 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) At first glance the list `[[], [[]]]` may appear to be invalid—the type of the first element is `'a list` and the type of the second element is `'a list list`. In fact, the list is valid. Why are these two types considered to be compatible?

(b) Slide 75 says that the `[...]` construct used to create lists is "syntactic sugar". Could we view it the other way, could ML programmers do without `::` and use only `[...]`?

(c) In the implementation of `maxint` below the third case contains two calls to `maxint`. Without using a `let` or another function, rewrite it so that it contains only one call to `maxint`.

```
fun maxint([]) = raise Empty
  | maxint([x]) = x
  | maxint(x1::x2::xs) = if x1 > x2 then maxint(x1::xs)
                                    else maxint(x2::xs);
```

(d) (1 point extra credit) In a previous semester a 372 assignment asked students to find and cite a reference to a description of some obscure programming language. The grade was inversely proportional to how many other students cited the same language. One student devised a scheme that essentially guaranteed his language selection to be unique. What did he do?

(e) (1 point extra credit) Estimate how long it took you to complete this assignment. Other comments about the assignment are welcome, too—I appreciate all feedback, good or bad.

Corrections and FAQs

Ideally, assignment write-ups (like this one) should be perfectly clear, free of ambiguities, and have zero defects. That goal is rarely achieved. The website will most likely have a page of corrections and FAQs for each assignment, starting with this one.

As a rule, to reduce e-mail volume, only critical issues will warrant a broadcast on the mailing list. For example, an error that reverses meaning, such as "...must include..." instead of "must not include..." would be reported on the mailing list, in addition to appearing in the corrections. Clarification of some vague wording would probably generate an entry in the corrections/FAQs but not an e-mail broadcast.

Remember that the first report of a given bug in an assignment write-up is worth one point of extra credit.

Deliverables and `turnin`

The set of final products for a project is sometimes called the *deliverables*. The deliverables for this assignment are these files: `ftypes.sml`, `to_b.sml`, `strings_to_s.sml`, `cpfx.sml`, `paired.sml`, `splits.sml`, `street.sml`, `editstr.sml` and `answers.txt`. That list can be found on `lectura` in `/home/cs372/fall06/a1/delivs`.

The `turnin` program on `lectura` must be used to submit your solutions for grading. The upshot of this is that even if you're not developing your solutions on `lectura`, you'll still to transfer them to `lectura` in order to submit them with `turnin`.

If you're not familiar with `turnin`, take a look at *`turnin` Notes* on the website.

Use the `turnin` tag `372_1` to submit your solutions. (That's *underscore one*.)

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

```
turnin 372_1 `cat /home/cs372/fall06/a1/delivs`
```

Note that those are backquotes that surround the `cat` command. The shell first runs that `cat` command. The output of that command—the names of the files in `a1/delivs`—is substituted where the backquoted `cat` command appeared. `turnin` is then run with the resulting command line arguments. (If you put `echo` in front of `turnin` the shell will print the command to be run instead of actually running it.)

I often scan submitted solutions prior to the deadline to look for trends like misinterpretations or students solving a problem the hard way. I encourage you to turn in work in progress.

Late submissions

Files turned in after the deadline are usually not examined unless the student sends mail and makes a case for an extension. If you turn in a file late and simply hope for the best, you'll probably find that your graded assignment has a blank spot where you hoped your late submission would appear.

If a submission is only a little bit late, you only need a little bit of an excuse to earn a sufficient extension. See the syllabus for details about what typically does and doesn't constitute grounds for an extension.

`turnin` errors, corrections, and retests

If you don't submit a solution for a problem you'll of course get a zero on it. Once in a while a student will complete a solution but forget to turn it in, and be surprised to find a zero on their graded work. If that happens to you, let me know.

As a rule, there is no option to correct an error in a solution once the deadline has passed. A program may be only one character away from working, > instead of >=, for example, but there is no option to make that change and have the program retested.

The only case in which submitted work can be corrected and retested is if the student had every reason to believe a solution was correct but for some reason it didn't work as expected when it was tested. The most common cause of this is a last-minute cosmetic misfix of some sort. If you fall victim to something like

that, let me know.

Miscellaneous

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—an assignment is not a take-home exam.

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

You should be able to solve all problems using only those elements of ML covered on slides 1-114.

Think in terms of using function cases with pattern matching when possible instead of if-then-elses. My initial set of solutions for these problems has a total of seven `"if"`s.

You may use elements of ML that we haven't studied, with one exception: you may not use any of ML's imperative elements. (We haven't studied any of the imperative elements.)

If you wish, you may incorporate code from lectures in your solutions. If you put those in a file, perhaps `misc.sml`, be sure to `turnin` the file `misc.sml`, too.

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

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. (See the syllabus for the details.)