PLC: Homework 8 [75 points]

Due date: Wednesday, March 29, 10:30pm
3 point bonus if you turn in by Tuesday, March 28, 10:30pm

About This Homework

The goal of this homework is to explore a little untyped functional programming using elisp, the version of the LISP programming language supported by the emacs text editor. As usual, the first thing you should do for this homework is copy the files from the `hw/hw8` subdirectory of the class repository, to a new `hw8` subdirectory of your personal repository, similarly to previous homeworks.

Partners Allowed

For this homework, you may work by yourself or with one partner (no more). See the instructions for `hw2` for how to create the `ack.txt` and `partner.txt` files that are required if you work with a partner

All files should go in a hw8 subdirectory (which you create) of your personal repository. Please call the subdirectory exactly that. Do not call it `Homework 4` or `Hw8` or other variations, or we will penalize you 5 points.

How to Turn In Your Solution

Make sure you have done a "Subversion add" on the `hw8` directory you created in your personal repository, and on the `.agda` files you have to submit for this assignment, and then do a "Subversion commit" on your personal repository directory to submit them. You can commit as many times as you want up to the deadline. If you commit after the deadline, we will use the last existing version before the deadline. Work submitted after the deadline will not count.

How to Check Your Solution

As for previous homeworks, you might want to make sure you have correctly submitted your solution via Subversion. Again, just go to the URL for your personal repository using a web browser, log in with your HawkId and password, and you can see what has been submitted.

How To Get Help

You can post questions in the `hw8` section on Piazza, or elsewhere on Piazza. See the course's Google Calendar, linked from Piazza, for the locations and times for office hours, including evening Skype office hours for Prof. Stump.

1 Reading

No assigned reading this week.

Basics of elisp programming

All coding in this assignment will be in elisp. emacs (and also the internet) provides quite a bit of good documentation about elisp. We will also be looking at this in class March 21 and 23. Some useful commands for basic elisp programming:

- **Control-h k** tells you what elisp command is executed for the keystroke you enter right after typing Control-h k.
- **Control-h f** gives you documentation about the elisp function you name.
- **Contrl-h v** gives you documentation about an elisp variable you name.
- **Meta-:** (type "meta", which is often bound to the "alt" key, or else the escape key always works for meta; and then type a colon): this lets you enter an elisp expression to be evaluated.
- **Control-Meta-x** If your cursor is sitting in a function definition in a `.el` file (within emacs), then this will actually add that function definition to elisp, so you can then call it using Meta-:.

To define a new elisp function, the syntax is

```
(defun NAME(ARGS) DESCRIPTION CODE)
```

where `NAME` is the name of your function, `ARGS` is a list (separated just by spaces) of any input variables to the function, `DESCRIPTION` is a double-quoted string giving a description of the function, and `CODE` is then the code for the function.

2 Basic elisp programming [35 points]

In a file called `basic.el`, write functions to do the following (of course you should test them in a sample buffer). I am giving some hints about how to do these tasks, by referencing existing elisp functions you can use. (These functions are already defined in elisp; use **Control-h f**, as described above, to learn more about these functions.)

1. `goto-middle` which should cause your cursor to jump to the middle character of the buffer. You can move the cursor using the function `goto-char` and you can find the beginning and ending characters of the buffer using `(point-min)` and `(point-max)` respectively. [10 points]
2. `switch-halves` which should switch the first half of your buffer's text with the second half. You can use the function `kill-region` to cut a region of text, and `yank` to paste it back. [10 points]

3. `capother` which should capitalize every other character in the buffer, starting from the beginning `(point-min)` to the end of the buffer `(point-max)`. To capitalize a region of text, you can use `upcase-region`. [15 points]

3 Extending an XML navigation mode [40 points]

Thanks to some amazing software called `se-mode` ("structured editing mode") written by PLC alum Carl Olson, we can pretty easily turn emacs into an IDE for a language, if we have a backend parser for that language. For this problem, we will do this for basic XML. First, compile `mainx.agda`. This is the backend tool that will parse XML and send to the frontend (emacs mode) an abstract description of the parse tree. The `se-mode` code in the frontend will assemble a parse tree internally from that abstract description, and then provides commands for the user to navigate through the parse tree. To see what `mainx` is sending to the frontend, you can run `mainx` and just type in the name of a `.xml` file like `AUFLIA.xml` (provided with `hw8`). You will see a dump of a bunch of information.

For this problem, you will just be modifying `xmlnav-mode.el`. First, you need to configure emacs so it knows about this new mode. To do that, copy the `se-mode` directory and all its files into your `hw8` directory. It is important that you have an `se-mode` directory as a subdirectory of your `hw8` directory. Put `xmlnav-mode.el` and the `mainx` executable in your `hw8` directory (not the `se-mode` subdirectory). Then open your `.emacs` file (type Control-x Control-f in emacs, and then type `~/.emacs`). Add the following commands somewhere in the `.emacs` file (where `PATH-TO-HW8-DIRECTORY` should be the path on your computer to your `hw8` directory) and then save it, and restart emacs:

```
(add-to-list 'load-path "PATH-TO-HW8-DIRECTORY")
(require 'xmlnav-mode)
```

Next, open the `xmlnav-mode.el` file and change the first line so that it has the path to your `hw8` directory. Finally, open the file `AUFLIA.xml`, included with the `hw8` files. If all goes well, you should be able to type "Meta-s" (alt-s or escape-s), and you should see "(xmlnav navi)" displayed in the mode line towards the bottom of your emacs window. You can then use "p", "n", "f", and "b" to navigate through the text following the structure of the parse tree.

3.1 Showing errors [15 points]

Modify `xmlnav-mode-update-mini` in `xmlnav-mode.el` so that if the current selected parse-tree node has an error attribute, the error is shown as a message. To do this:

- `(se-mode-selected)` will return the current selected node, if any.
- `se-term-data`, given a node, will return the attributes associated (by the backend) with that data.
- The `assoc` function will let you look for an `'error` attribute in the attributes returned by `se-term-data`.

- if there is an `'error` attribute, you can use `message` to print out the data which is part of that attribute. The attribute is a pair that looks like `'error . m`. To access the second component of a pair in elisp, you use the `cdr` function.

You should see an error at the very last end tag of `AUFLIA.xml`, which does not match the start tag.

3.2 Hiding elements [25 points]

Modify `xmlnav-mode-toggle-hidden` so that it toggles whether the selected node (if there is one) is invisible or not. To get, set, or clear the invisibility property, respectively, you use `get-text-property`, `put-text-property` and `remove-text-properties`. There is good documentation about text properties here:

`https://www.gnu.org/software/emacs/manual/html_node/elisp/Text-Properties.html`

If there is no `'invisible` property for the currently selected node, you should set the `'invisible` to `'xmlnav-hide`. I have already added code in `xmlnav-mode.el` to tell emacs to display invisible text with an ellipsis (`...`) if the `'invisible` property is set to `'xmlnav-hide`. If there is a `'invisible` property for the currently selected node, you should remove it with `remove-text-properties`. After this is done, call `se-navigation-mode 1` to restart the `navi` mode (which shuts off automatically if the buffer is modified), and also call `se-mode-select` with the node that was selected at the start of the call to `xmlnav-mode-toggle-hidden`, so it will be reselected.