EE473—Introduction to Artificial Intelligence
Spring 2014
Problem Set #1
Due: Friday 31–January–2014

This problem set is a warm-up exercise in using the course software. The course software is all written in SCHEME->C, an implementation of the SCHEME programming language that is itself a dialect of LISP. Documentation for SCHEME in general, and the SCHEME->C implementation in particular, is available on-line from

`http://shay.ecn.purdue.edu/~ee473/documentation.html`.

The course software adds extensions to the basic SCHEME->C implementation. These will be described in the problem set handouts as these extensions are needed.

All problem set handouts will assume that you are using one of the ECN SOLARIS machines in MSEE189 or the LINUX machines in EE306 and that you know how to use GNU EMACS, UNIX, and X WINDOWS. If you do not know how to use GNU EMACS, UNIX, and X WINDOWS, ask the TA for help.

The course software is designed to provide graphical display of your algorithms in operation. This display software requires the use of an X WINDOWS display. This means that it will not be possible for you to run this software on an ordinary ASCII terminal.

You start the course software by typing the `ee473` command at the UNIX shell prompt.

```
ecelinux13> ee473
```

This should bring up a GNU EMACS window.

All of the course software is run under GNU EMACS. You should learn how to use GNU EMACS in order to be maximally efficient in doing the problem sets for this course. GNU EMACS has on-line documentation available via the `c-h` command. In particular, the `c-h t` command runs an on-line tutorial designed to teach GNU EMACS to newcomers. To help get you started, here are a few tips. Many GNU EMACS commands consist of one or two keystrokes. Some keystrokes are ‘control’ keystrokes or ‘meta’ keystrokes. The control and meta keys operate like shift keys; you hold them down while you type other keys. In documentation, such control and meta keystrokes are indicated with a `c-` and `m-` prefix. Thus `c-h` means hold down the control key while pressing ‘h’, while `m-x` means hold down the meta key while pressing ‘x’. Note that `m-a` means hold down the meta key while pressing lower case ‘a’ and `m-A` means hold down the meta key while pressing upper case ‘A’.

Most people know where the control key is on their keyboard. Computer keyboards have had control keys since the very early days of computers. Widespread availability of keyboards with meta keys is a relatively recent phenomena, however. While the ECN keyboards do have a meta key, many people do not know where it is. Unfortunately, the key cap legend for the meta key has not been standardized, so it will vary from keyboard type to keyboard type. On ECN SOLARIS machines, it appears as a key labeled ‘◇’ to the left of the space bar. On ECN LINUX machines, it appears as a key labeled ‘Alt’ both to the left and to the right of the space bar.

Since until recently, many keyboards did not have meta keys, GNU EMACS provides a way to type meta keystrokes on keyboards that do not have meta keys. On such keyboards, to type a command like `m-x`, one types the ESC key followed by the ‘x’ key. Note that the ESC key is not a meta key. It does not operate like a shift key. Instead, it precedes another keystroke. It is a bad habit to use the ESC key. There is a reason for this. It is much more efficient to type a single-keystroke command than to type a two-keystroke command. For example, GNU EMACS provides the `m-b` command for moving the cursor backward a word at a time. If you want to move back five words, you can hold down the meta key and hit ‘b’ five times. Using the ESC key, you would have to alternate between hitting the ESC key and the ‘b’ key. Try it. You will discover that the latter is much slower and more cumbersome than the former.

Many commands use the control and meta keys simultaneously. These are indicated in the documentation using notation like `m-c-f` or `c-m-`. These are equivalent. To type this command, hold down both the meta and control key and type 'f'. This command means move the cursor forward by an entire SCHEME expression. In general, the meta-control commands are used for editing SCHEME programs. If you learn to use the meta-control commands for moving the cursor, rather than the arrow keys, you will find that you can edit SCHEME programs much more quickly and with greater convenience.

Some less-frequently-used GNU EMACS commands are two-keystroke commands. For example, the command `c-x c-s` is used to save the file you are editing. Here, the `c-x` keystroke is a prefix. In general, GNU EMACS has four prefix keystrokes, `c-x`, `c-c`, `c-z`, and `c-h`.[1] Some much-less-frequently-used GNU EMACS commands are invoked by typing their whole name. To do this, you must first type the `m-x` command. This puts the cursor in the 'mini-buffer' area at the bottom of the window. You then type the whole name of the command, followed by return, to invoke the command. Before typing return, you can edit the command name by using ordinary GNU EMACS keystrokes. In the documentation, such extended commands are indicated using notation like `m-x ee473`.

Inside the course software, a SCHEME interpreter can be invoked by typing the extended command `m-x ee473` or the two-keystroke command `c-c L`. Note that the second keystroke of the two-keystroke command is a shifted 'L'. This will give you a `*ee473*` buffer that is running a SCHEME interpreter with the course software pre-loaded. When this buffer is created you will see something like:

```
Scheme->C -- 15mar93jfb -- Copyright 1989-1993 Digital Equipment Corporation
>
```

You can switch to any other buffer at any time. You can always get back to the `*ee473*` buffer by typing `c-c L`.

The `*ee473*` buffer runs a standard SCHEME read-eval-print loop. That means you can type SCHEME expressions and evaluate them by typing return. The results are then displayed in the `*ee473*` buffer. The `*ee473*` buffer provides a number of features that make debugging SCHEME programs easier. First, there is a history mechanism. All of the SCHEME expressions that you type are saved. If you need to reexecute a previously entered SCHEME expression, you do not need to retype it. The commands `m-p` and `m-n` will cycle backward and forward through previously typed expressions. When you find the one you want, simply hit return to execute it. You can also edit the expression before hitting return.

Sometimes, the history list can grow quite long, and you wish to recall an expression that is way back in the history list. It can be tedious to type many `m-p` commands in a row. If the expression begins with a unique sequence of characters, you can type just a few characters and then cycle backward and forward through the subset of the history that begins with those characters using the commands `m-r` and `m-s`.

The `*ee473*` buffer runs a special version of the `sci` SCHEME interpreter, described in the SCHEME->C manuals, that has the course software already compiled and loaded in. For the most part, other than the addition of this pre-loaded software, the `*ee473*` buffer should behave the same as `sci`. The `*ee473*` buffer, however, has an enhanced debugger. The debugger is entered whenever the interpreter encounters an error in your program.

```
> (/ 1 0)

>>Error:  Divisor is equal to 0:  0
DEBUGGER:
>>
```

Note that the debugger prompt consists of `>>` instead of the normal `>`. You can also interrupt the execution of your program and enter the debugger with the `c-c c-c` command.

```
> (let loop () (loop))
<user types c-c c-c>
>>Interrupt:
DEBUGGER:
>>
```

[1] Experienced users will note that `c-q`, `c-u`, ESC, and several other keystrokes behave in certain ways like prefix keystrokes.

In the debugger, you can evaluate any SCHEME expression, just like at the top level. Several additional commands are available. You can type `m-b` and get a back-trace of your program. You can move up and down the call stack with the commands `m-c-n` and `m-c-p`. When you are at a particular point in the stack, expressions that you evaluate in the debugger are evaluated in the context of that stack frame. In other words, you can access the parameters and local variables of that frame by name. The commands `c-c <` and `c-c >` move to the top and bottom of the call stack. You can abort the program and return to the top level with the `m-a` command. You can continue execution after an interrupt (but not after an error) with the `m-c` command.

SCHEME->C allows SCHEME code to be either interpreted or compiled. All of the predefined course software has been compiled into the SCHEME interpreter that runs in the `*ee473*` buffer. The code that you will write, however, will be interpreted, (i.e. you need never use `scc`, the SCHEME->C compiler described in the documentation). This is to make it easier for you to debug your software. While interpreted code is slower than compiled code, since your code will be calling procedures that we have provided, and our procedures will be compiled, much of the performance penalty will be mitigated. All of the problem sets are designed so that they will not require excessive CPU time to run. Each problem set handout will give the expected amount of CPU time needed to run the program. If you exceed this estimation substantially, it is likely that you are doing something wrong.

GNU EMACS also contains a special mode for editing SCHEME programs. This mode will automatically indent your programs so that they are readable. The course software is configured so that SCHEME mode will be entered automatically when you edit a file with a `.sc` extension. You must use the course software to edit and run your programs. Using GNU EMACS will make editing SCHEME programs much easier and will allow you to properly format your programs prior to handin. We enforce these draconian measures as it makes it easier to grade your problem sets if they all conform to standard indenting conventions for SCHEME code.

The special mode for editing SCHEME programs provides two useful commands. The `c-z )` command will check for unbalanced parentheses and show you where they are. The `c-z l` will load the file that you are editing into the SCHEME interpreter that runs in the `*ee473*` buffer. You will need to use this command after you have written your program, as well as after you have made any changes, to inform the SCHEME interpreter of your changes.

The following table summarizes all of the GNU EMACS commands added by the course software.

`m-x ee473`	starts up `*ee473*` buffer
`c-z l`	loads file into SCHEME interpreter
`c-c c-c`	interrupts SCHEME interpreter
`m-a`	abort
`m-b`	back-trace
`m-c`	continue
`m-c-n`	move towards caller
`m-c-p`	move away from caller
`c-c <`	move to top of call stack
`c-c >`	move to bottom of call stack
`c-z )`	find unbalanced parentheses
`m-p`	previous history item
`m-n`	next history item
`m-r`	search previous history item
`m-s`	search next history item

For this problem set, you will implement three procedures for doing union, intersection, and set difference of sets of numbers. A *set* is a collection of *elements* where the order and multiplicity of the elements is not relevant. Sets contain at most a single copy of a given element. We will represent sets as SCHEME lists. The *union* of two sets A and B, denoted $A \cup B$, is a set that contains both the elements of A and the elements of B. The *intersection* of two sets A and B, denoted $A \cap B$, is a set that contains the elements that are in both A and B. The *set difference* of two sets A and B, denoted $A \setminus B$, is a set that contains the elements that are in A but not in B.

We want you to implement the following three SCHEME procedures:

`set-union` A B [*Procedure*]

A and B are lists of numbers that represent sets of numbers. Returns $A \cup B$. A and B may contain duplicates but the result will not.

`set-intersection` A B [*Procedure*]

A and B are lists of numbers that represent sets of numbers. Returns $A \cap B$. A and B may contain duplicates but the result will not.

`set-minus` A B [*Procedure*]

A and B are lists of numbers that represent sets of numbers. Returns $A \setminus B$. A and B may contain duplicates but the result will not.

You *must* call these procedures by these names and they must take the indicated arguments and return the indicated results in order for the GUI and the grading software to work.

Create a file `p1.sc` with the above three procedures. Then issue the `c-c L` command to create a `*ee473*` buffer and start up a SCHEME interpreter in that buffer. Then issue the `c-z l` command, when you are in the `p1.sc` buffer, to load your program into the SCHEME interpreter. Then execute the SCHEME expression `(p1)`.

```
Scheme->C -- 15mar93jfb -- Copyright 1989-1993 Digital Equipment Corporation
> (p1)
#F
```

In general, the GUI for problem-set i will be invoked by calling the procedure `p`i with no arguments. For the GUI to interface correctly with your code, the code you write must conform to the naming and calling conventions established for each problem set. For this problem set, that means that you must write procedures, named `set-union`, `set-intersection`, and `set-minus`, that each take two lists of numbers as parameters and return lists of numbers as a result.

After you issue the `(p1)` command, a window will be displayed showing two sets A and B. If you have implemented the procedures `set-union`, `set-intersection`, and `set-minus` correctly, you will also see correct values for $A \cup B$, $A \cap B$, and $A \setminus B$. You can add elements to A or B by selecting an element (by clicking on one of the buttons `0`, `1`, 2, 3, 4, or 5) and then clicking on either the symbol A or B. You can remove elements from A or B by clicking on the element that you wish to remove. As you add and remove elements from A or B, the display of $A \cup B$, $A \cap B$, and $A \setminus B$ is dynamically updated. When you are finished, you can click on the Quit button to exit the program.

Solutions to all problem sets must be handed in electronically. Run the course software on one of the ECN Solaris machines in MSEE189 or Linux machines in EE306. Load your solution into a GNU Emacs buffer and type `m-x handin`. Solutions must conform to the following standards in order to be graded:

1. The solution to each weekly problem must consist of a single file.
2. The file must be handed in using the `m-x handin` command in the course software installed on one of the ECN Solaris machines in MSEE189 or the Linux machines in EE306.
3. The file must be handed in by 5:00pm on the Friday that it is due. Thus this problem set must be handed in by 5:00pm on Friday 31–January–2014. Late handins will not be graded.
4. The file must be an ASCII text file containing no more than 79 characters per line.
5. There must be no whitespace (i.e. ASCII space, tab, or new line characters) after any opening parenthesis or before any closing parenthesis in the file.
6. The file must be properly formatted using the GNU Emacs command sequence `c-x h m-c-\` in Scheme mode prior to handin.

See `http://shay.ecn.purdue.edu/~ee473/syllabus.html` for more details on the problem-set dissemination mechanism.

Good luck and have fun!