MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1998

Problem Set 0

- Issued: Wednesday, September 9
- To Be Completed By: Monday, September 14
- Reading:
  - Read "6.001 General Information", handed out at the first recitation (September 9). Be especially sure to read the 6.001 policy on collaborative work.
  - Scan the 6.001 Home Page on the web at http://mit.edu/6.001. All handouts (including this one) can be found there. The web page also contains announcements, Scheme software, problem sets, documentation, advice on where to get help, and other useful information. You should make a habit of looking at this page at least once a week during the semester.
  - Textbook reading: In general, the lectures will assume that you've read the appropriate sections of the text **before** coming to lecture. Your first reading assignment: sections 1.1 and 1.2 before lecture on September 15.
- Recitation Assignments: posted on the 6.001 website at http://mit.edu/6.001 by 9am Friday, September 11. Will also be posted outside the 6.001 laboratory (5th floor of Building 34) and outside of the tutorial area near the rooms 36-115 through 36-121. Start attending your assigned recitation on Friday, September 11.

The purpose of Problem Set 0 is to familiarize you with the Scheme programming environment and the resources available to you. The format of this problem set is different from the other problem sets that will be handed out this term. Subsequent problem sets will consist of two sorts of activities: tutorial preparation and written programming assignments.

Problem Set 0 is organized into two parts:

- "Getting Started" is designed to get you set up with Scheme. Please go through this section as soon as possible – no later than this weekend.
- "Your Turn" consists of some Scheme expressions you are to have Scheme evaluate. You will be emailing the results of evaluation and the answers to some other administrative questions to your tutor and recitation instructor.

1. Getting Started

The purpose of this section is to get you started using Scheme as quickly as possible. Start by looking at the subject web page labeled "Using Scheme in 6.001". You'll see there that the 6.001 Scheme system runs in the 6.001 Lab (34-501), on some unix platforms (Linux-Athena and NetBSD-Athena), and on PCs running GNU/Linux, Windows NT4, or Windows 95 (sorry, no Macs or Windows 98 or general Athena); and that you can move among the different implementations as you require. If you have a PC capable of running Scheme, we suggest that you install it there, since it will be convenient for you to work at home a lot of the time. The 6.001 lab is probably the best place to work if you want help, since that is staffed by knowledgeable and friendly Lab Assistants.

Remember, whether you're working with your machine or MIT's, the source of all useful information is the 6.001 web page: **http://mit.edu/6.001**

Starting scheme

The first thing to do is to get an implementation of Scheme and start using it:

- *In the 6.001 lab ...* A good way to get started learning Scheme is to do this problem set in the Lab, where there are LAs to help you, and then install Scheme on your own PC and run through the section again to verify that your home system works just like the one in the lab. Find a free lab computer and log in with the username `u6001`. You will need to use a floppy disk in order to save your files.[1]
- *On your home computer ...* follow the instructions on the Web page for downloading and installing Scheme. *Important:* The version of Scheme we are using is Scheme 7.5a. If you have an earlier version, it is important that you get the new one. Once you've installed the Scheme system you should be able to simply start it and work through this problem set. If you are using your own computer, you'll also have to download the code for each weekly assignment, which you'll find on the web page. (For this problem set, there is no code to download.)

Learning to use Edwin

In 6.001, we use a text-editing system called *Edwin* which is a Scheme implementation of the Emacs text editor. Edwin is virtually identical to Emacs. Even if you are familiar with Emacs, you will find it helpful to spend about 15 minutes going through the on-line tutorial, which you start by typing `C-h` followed by `t`. (`C-h` means "control-`h`": to type it, hold down the CTRL key and then type `h`. Then release the CTRL key before typing `t`.) You will probably get bored before you finish the tutorial, but at least skim all the topics so you know what's there. You'll need to gain reasonable

[1] The lab system encourages you to use a floppy disk if you want to save your files. There is no permanent storage for students on the 6.001 lab system. You can also transfer your files to your Athena directory over the network. See the 6.001 web page on problem sets for information on how to transfer files to and from the lab: **don't use FTP!**

facility with the editor in order to complete the problem sets.[2] To get out of the tutorial, type `C-x k`, which will kill the buffer.

Files and Buffers

Edwin allows you to read and edit existing files (`C-x C-f filename`), and create and write files (`C-x C-w filename` or `C-x C-s`). The text within these files are contained in Edwin objects called "buffers". The name of the current buffer can be found at the base of the Edwin window. Notice that Edwin starts up in a buffer called `*scheme*`. What distinguishes the Scheme buffer from other buffers is the fact that underlying this buffer is a Scheme evaluator (more about scheme evaluation later!)

One could simply type all one's work into the `*scheme*` buffer, but it is usually better to store versions of your code in a separate buffer, which you can then save in a file. The most convenient way to do this is to split the screen to show two buffers at once—the `*scheme*` buffer and a buffer for the file you are working on. You will need know how to split the screen (`C-x 2`) and how to move from one half to the other (`C-x o`). Choose a filename that ends in `.scm` for your code buffer, and then type `C-x C-f` *filename* (for example, `C-x C-f myps0work.scm`). The half of the screen your cursor is in will now be a buffer for this new file. You can type in this buffer and evaluate expressions in this buffer(more on this in the next section!). The results of expression evaluation will appear in the `*scheme*` buffer. If an error occurs during evaluation you must go to the `*scheme*` buffer to deal with the error.

In addition to the `*scheme*` buffer, Edwin also provides a special read-only buffer called the `*transcript*` buffer which is automagically maintained and keeps a history of your interactions with the Scheme evaluator. You can extract pieces of this buffer to document examples of your work for problem sets (`C-space`, `C-w`, `C-y`). To go to this buffer type `C-x b *transcript*`. You can go back to any buffer with the `C-x b` command.

Scheme Expression Evaluation

Scheme programming consists of writing and evaluating *expressions*. The simplest expressions are things like numbers. More complex arithmetic expressions consist of the name of an arithmetic operator (things like $+, -, *$) followed by one or more other expressions, all of this enclosed in parentheses. So the Scheme expression for adding 3 and 4 is `(+ 3 4)` rather than $3 + 4$.

While in the `*scheme*` buffer[3], try typing a simple expression such as

```
1375
```

Typing this expression inserts it into the buffer. To ask Scheme to evaluate it, we type `C-x C-e`, which causes the evaluator to read the expression immediately preceding the cursor, evaluate it

[2] If you're running Scheme on your own PC and want a little mindless diversion, you can adjust the screen size, position and color. Click on the shield icon at the upper left of the Edwin window to adjust the font and the background color. You can change the foreground color using the command `M-x set-foreground-color`.

[3] If you're not currently in the `*scheme*` buffer and don't know how to go to this buffer, go back and learn more about Edwin.

(according to the kinds of rules described in lecture) and print out the result which is called the expression's "value". Try this.

An expression may be typed on a single line or on several lines; the Scheme interpreter ignores redundant spaces and line breaks. It is to your advantage to format your work so that you (and others) can read it easily. It is also helpful in detecting errors introduced by incorrectly placed parentheses. For example, the two expressions

```
(* 5 (+ 2 (/ 4 2) (/ 8 3)))

(* 5 (+ 2 (/ 4 2)) (/ 8 3))
```

look deceptively similar but have different values. Properly indented, however, the difference is obvious.

```
(* 5
   (+ 2
      (/ 4 2)
      (/ 8 3)))

(* 5
   (+ 2
      (/ 4 2))
   (/ 8 3))
```

Edwin provides several commands that "pretty-print" your code, indenting lines to reflect the inherent structure of the Scheme expressions (see Section B.2.1 of the *Don't Panic* manual). Make a habit of typing `C-j` at the end of a line, instead of RETURN, when you enter Scheme expressions, so that the automatic indentation takes place. `Tab` and `M-q` are other useful Edwin formatting commands.

While the Scheme interpreter ignores redundant spaces and carriage returns, it does not ignore redundant parentheses! Try evaluating

```
((+ 3 4))
```

Scheme should respond with a message akin to the following:

```
;The object 7 is not applicable.
;Type D to debug error, Q to quit back to REP loop:
```

Type `q` for now. In the next problem set, we'll learn that typing `d` invokes the debugger which is a useful tool. If you are interested in finding out more about the debugger now, look in the *Don't Panic* manual!

We've already seen how to use `C-x C-e` to evaluate the expression preceding the cursor. There are also several other commands that allow you to evaluate expressions: `M-z` to evaluate the current definition, or `M-o` to evaluate the entire buffer (this does not work in the `*scheme*` buffer). You may mark a region and use `M-x eval-region` to evaluate the marked region. Each of these commands will cause the Scheme evaluator to evaluate the appropriate set of expressions. Note that you can type and evaluate expressions in either buffer, but the values will always appear in the `*scheme*` buffer. See the *Don't Panic* manual for more details.

2. Your Turn

There are two things that you need to do for this part: evaluate scheme expressions and answer some documentation/administrative questions. The answers for both parts should be emailed to your tutor and recitation instructor; instructions for how to do so appear in a subsequent section of this problem set handout.

Expressions to Evaluate

Below is a sequence of Scheme expressions. Can you predict what the value of each expression would be when evaluated? Go ahead and type in and evaluate each expression in the order it is presented.

```
-37

(* 8 9)

(> 10 9.7)

(- (if (> 3 4)
       7
       10)
   (/ 16 10))

(* (- 25 10)
   (+ 6 3))

+

(define double (lambda (x) (* 2 x)))

double

(define c 4)

c

(double c)

c

(double (double (+ c 5)))

(define times-2 double)

(times-2 c)

(define d c)
```

```
(= c d)

(cond ((>= c 2) d)
      ((= c (- d 5)) (+ c d))
      (else (abs (- c d))))

(define applyto3 (lambda (x) (x 3)))

(applyto3 double)

(applyto3 (lambda (z) (* z z)))
```

In your email, include an excerpt from the `*transcript*` buffer of these expressions and the resulting values of their evaluation.

Documentation and Administrative Questions

Explore the 6.001 webpages to find the answers to the following questions:

1. According to the *Don't Panic* manual, what is the stepper and how do you invoke it?
2. According to the *Guide to MIT Scheme*, which of the words, in the scheme expressions you just evaluated above, are "special forms"?
3. After referring to the course policy on collaboration, complete the following two sentences:
   "If you work with other students, ..."
   "On your homework paper, you should write ..."
4. Locate the list of announcements for the class. What is the September 8th entry about?

In your email, include a brief, but warm and enthusiastic greeting to your tutor and to your recitation instructor. Also, say which Scheme implementation you are using: 6001 Lab or your own machine (with which operating system?).

Emailing Your Tutor and Recitation Instructor

If you are using the 6.001 lab, typing `C-x m` will open a mail buffer in which to compose your message. The system will ask for your own email address, so the recipient will know who sent the mail. You may insert work by copying from other buffers. To send your completed message type `C-c C-c`. If you're using your own machine, you can save your work in a file and then mail this using your favorite mailer.

If you don't know who your tutor and recitation instructor are, look on the 6.001 web page. The point here is not only to make sure you can send mail, but also to enable your tutor to add your email return address to the recitation section email list.

Congratulations! You have reached the end of Problem Set 0!