MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 – Structure and Interpretation of Computer Programs
Spring Semester, 2005

Project 0

- Issued: Tuesday, February 1
- To Be Completed By: Friday, February 4 6:00pm
- Reading:
    - Read "6.001 General Information" found at the MIT server. Be especially sure to read the 6.001 policy on collaborative work, and to read the detailed information on collaboration on the MIT server.
    - Browse the 6.001 web pages. All handouts (including this one) can be found there. The web page also contains announcements, Scheme software, projects, 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. Note in particular that the calendar outlining the schedule of lectures may change during the term as the course content is adjusted, and you should check this regularly to know what is coming.
    - Scan the *Don't Panic* manual, which is an introductory guide to using Scheme in 6.001. This can be found under the "Scheme Documentation" link on the tools section. You will likely consult this manual often over the next few weeks as you are getting used to the Scheme computing system.
    - Textbook reading: In general, the lectures will assume that you've read the appropriate sections of the text **before** listening to lecture or coming to recitation.

The purpose of Project 0 is to familiarize you with the Scheme programming environment and the resources available to you. The format of this project is different from the more substantive ones you will see later in the term. Those projects focus on extended experience in creating and testing code, and integrating such code with existing computational frameworks. As a consequence, those projects will take a significant amount of time to complete. This project is really just a warm up, and is intended to get you up to speed on the mechanics of using Scheme and the course resources. We expect that this project should not take much time, and while we understand that some of the things we ask you to do seem fairly obvious, we want to make sure you get the mechanics down right. Don't worry; the later projects and problem sets will involve much more extensive thinking and coding!

1. Getting Started

The purpose of this section is to get you started using Scheme as quickly as possible. You'll see there that the 6.001 Scheme system runs in the 6.001 Lab on Linux,

and on your own machine if you are running MIT server-Linux, GNU/Linux, Windows 95, NT, 2000 or XP. We do not support Macintoshes. Furthermore, if you use the MIT server, you can only run Scheme from an IBM or a Dell computer. Finally, you cannot run Scheme from a dialup MIT server machine. 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. The 6.001 lab is probably the best place to work if you want help, however, 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 tools section.

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 project 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 in a similar fashion to that in the lab. The 6.001 machines are on the right side as you enter the lab, the left side belongs to 6.004. Both sets of machines use the MIT server logins:
    - *The MIT Server Linux:* Find a free lab computer running an MIT server login screen, and log in with your MIT server username and password. In this case, you will have access to your usual MIT server home directory and customizations. To start Scheme and Edwin type

```
add 6.001
6001-scheme
```

- *On your home computer...* Follow the instructions on the Web page for downloading and installing Scheme (either Linux or Windows versions). *Important:* The version of Scheme we are using is Scheme 7.5.1. 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 project, there is no code to download.)

Learning to use Edwin

When you start Scheme (either by using the commands listed above for the MIT server Linux, or by clicking on the appropriate 6.001 icon on your home machine if you installed Scheme on it) you are interacting with a text-editing system called *Edwin*, which is a Scheme implementation of the Emacs text editor. You should not think of Edwin as just another text editor – instead, Edwin will be your primary interface to the Scheme system. If you write code in your favorite text editor and then try to cut and paste into Scheme, you will end up making your life miserable. Edwin is integrated with Scheme

(e.g. you can directly execute a Scheme expression in the middle of editing it) and it is very important that you become comfortable editing and working in Edwin.

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 will need to gain facility with the editor in order to complete the problem sets and projects. To get out of the tutorial, type `C-x k Return`, which will kill the buffer.

General Emacs commands

For reference information on using Emacs, see the MIT server, which contains links to a reference card for Emacs and more extensive documentation on Emacs. There are a few Edwin specific commands, you can see many of them by typing `C-h m` at the top level of Edwin. Below we discuss some of the commands you are likely to use as you interact with Scheme and Edwin.

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 is 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 connected to this buffer is a Scheme interpreter (more about how to activate the interpreter 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 to 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 myproj0work.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 projects (see the `C-space`, `M-w`, and `C-y` commands). 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, 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 interpreter to read the expression immediately preceding the cursor, evaluate it (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. You will find 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 several things that you need to do for this part: evaluate some simple Scheme expressions, use the provided tools to manipulate these expressions, and answer some documentation and administrative questions. The answers for all these parts should be **submitted electronically on the tutor**, using the `Submit Project Files` button. Remember that this is Project 0; when you have completed all the work and saved it in a file, you should upload that file and submit it for Project 0.

Part 1: Preparing your project material for submission

Create a new buffer (use the `C-x b` command, and give the prompt a new name). **At the top of the buffer, include your name, your TA's name, your section, the project number, and the part of the project you are answering** (yes, we know this seems silly, but it is amazing how many otherwise bright MIT students forget this!). Use this buffer as file (which you will need to save by using the `C-x C-s` command) into which you will insert your answers to the following parts.

Note that if you want to see your directories, you can use `(pwd)` inside a Scheme environment to list the name of the current directory to which you are connected, `(cd "foo")` to go to directory "foo", and `(cd "~foo")` to go to "foo" underneath the main directory.

Of course, since Edwin behaves like Emacs, you can also use `M-x dired` to list the entries in your directory.

Part 2: 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. **Extract the relevant parts out of the `*transcript*` buffer and copy this into your buffer that you made in part 2.1** (you will need to find the relevant Edwin/Emacs commands to do this; check out the `M-x set-mark` command, the `C-w` command, the `M-w` command and the `C-y` command).

```
-37

(* 3 4)

(> 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))))
```

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

comments that document your work. Remember that your real-life flesh and bones tutor will be reading and grading your project. Commenting your code and otherwise explaining your work is an important part of your project submission -- it is a good habit to get into NOW! For example:

```
;;; PART 2



;;;
;;; The following test cases explore the evaluation of simple
expressions.
;;;

(* 7 8)
;Value: 56
```

Note that later on you will be using the on-line tutor to submit small segments of code for evaluation by the tutor, as part of each weekly problem set. We **strongly suggest** that you still use Edwin to develop and test your code. Once it is working to your satisfaction, then you can cut and paste the expressions into the tutor window for submission.

Part 3: Pretty printing

As you begin to write more complicated code, being able to read it becomes very important. To start building good habits, type the following simple expression into Scheme (with interspersed Edwin commands as shown):

```
(define abs C-j (lambda (a) C-j (if (> a 0) C-j a C-j (- a))))
```

Show a copy of how this actually appears in your buffer. Do the same thing, but in place of each `C-j`, use the `Enter` key followed by the `Tab` key. What difference is there in the result?

Part 4 (Optional): Real printing

If you want to print your a hardcopy of your work (at the end of this project), you will need commands to do this. These commands are issued from an interface to the operating system. If you logged in at the 6.001 Lab, an `xterm` window opened up. Inside that window, you can issue commands for printing. To see what the printer queue looks like, use `lpq` (this automatically selects the first available printer among those physically in the lab).

To print, assume that you have saved your work in ~/u6001/work/project0l.scm. Then go

to `xterm` and type the following

```
cd ~/u6001/work    (to connect to this directory)
ls                 (make sure the file you want is in fact there)
```

```
lpr project0.scm
lpq               (to check the status of your print job)
```
To remove a job, use `lprm job-number`.

If you are working on the MIT server (as opposed to in the 6.001 Lab), you may use `cview printers` to get a list of available printers. This can be done from an `xterm` window, or you can invoke `M-x shell` inside Edwin/Emacs, which will create a shell window within the editor, to which you can type Linux commands. To queue a print job, use

`lpr -P` **`printername filename`**
or if you prefer

`lpr -P` **`printername`**`2` **`filename`**
which will print things two-sided rather than single-sided.

Part 5: Documentation and Administrative Questions

Explore the 6.001 webpages to find the answers to the following questions. Add your answers to these questions to your project 0 solutions write-up.

1. **According to the *Don't Panic* manual, how do you invoke the stepper? What is the difference between the stepper and the debugger?**

2. **According to the *Guide to MIT Scheme*, which of the words in the scheme expressions you evaluated in Part 2 above are "special forms"?**

3. **Referring to the course policy on collaboration, describe the policy on the use of "bibles."**

4. **List three people with whom you might collaborate on projects this term. You are not required to actively collaborate with these people, we just want you to start thinking about what collaboration means in this course.**

5. **Locate the list of announcements for the class. What do these announcements say about recitation attendance during the first week of classes?**

6. **What are the three methods for controlling complexity described in the learning objectives section of the course objectives and outcomes? List one example from each category.**

7. **What does the *MIT Scheme Reference Manual* say about treatment of upper and lower case in expressions?**

8. **What are the Edwin commands for creating a new file, and for saving a file? What is the difference between the `*scheme*` buffer and the `*transcript*` buffer?**

Submission

Once you have completed this introductory project, your file should be **submitted electronically on the tutor**, using the `Submit Project Files` button. Remember that this is Project 0; when you are have completed all the work and saved it in a file, upload that file and submit it for Project 0.

Congratulations! You have reached the end of Project 0!