MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996-1997

Problem Set 0

Getting Started

Issued: Wednesday, September 4

Due: Solutions should be sent by email to your tutor by Tuesday, September 10

In general, problem sets will be distributed in lecture on Tuesdays. This is a practice problem set, which has two main purposes:

- to get you started interacting with the computer system you will be using throughout the term,
- to stress the idea that working together in groups is a great way to learn material in this (or any) course. As part of this exercise, your section instructor will be taking you into the lab, where we would like you to work in pairs or triples. When you complete the problem set early next week, you can either continue to work as a group, or finish things up individually.

This zeroth problem set is different from the ones for the rest of the term because there is no written material to hand in. Later this week, we will be adjusting the registrar's assignment of students into sections, so that by Friday you will be formally assigned to your section for the term. After that occurs, we *strongly* recommend that you return to the lab, assemble your answers into an Edwin buffer, and *send this buffer by email* to your tutor. Your tutor's email address will be announced at recitation.

The purpose of the exercises below is to familiarize you with the 6.001 laboratory and programming system. Spending a little time on simple mechanics now will save you a great deal of time over the rest of the semester.

In general, to work on assignments, you will need a copy of the course notes and the lab manual (*Don't Panic*), but you can start work on the exercises in this handout without that material.

1. Getting started in the lab

When you come to the lab, find a free computer and log in. (In general, you will do this using a floppy disk, but we'll cover that in Problem Set 1.) You should log in as `u6001`.

For 6.001, you will use a text-editing system called Edwin, which is an Emacs-like editor. Even if you are already familiar with Emacs, you should take some time *now* to run the Emacs/Edwin tutorial. This can be invoked by typing `C-h` followed by `t`. You will probably get bored before you finish the tutorial. (It's too long, anyway.) But at least skim all the topics so you know what's there. You'll need to gain reasonable facility with the editor in order to complete the exercises below.

Evaluating expressions

The language in which you will be working this term is Scheme, a dialect of Lisp. You will be hearing lots about this language next week, but you can already get some experience in using the language here. All Scheme procedures are built out of expressions, with the simplest expressions consisting of 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.

After you have learned something about Edwin, go to the Scheme buffer (i.e., the buffer named `*scheme*`).[1] As with any buffer, you can type Scheme expressions, or any other text into this buffer. What distinguishes the Scheme buffer from other buffers is the fact that underlying this buffer is a Scheme evaluator, which you can ask to evaluate expressions, as explained in the section of the Edwin tutorial entitled "Evaluating Scheme expressions."

Type in and evaluate (one by one) at least some of the expressions listed below. If the system gives a response you do not understand, discuss this among your group to try to clarify your uncertainty.

```
25

-37

(- 8 9)

(> 3.7 4.4)

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

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

Observe that some of the examples printed above are indented and displayed over several lines for readability. An expression may be typed on a single line or on several lines; the Scheme interpreter ignores redundant spaces and carriage returns. 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

[1] If you don't know how to do this, go back and learn more about Edwin.

```
(* 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, e.g., indents lines to reflect the inherent structure of the Scheme expressions (see Sec. B.2.1 of *Don't Panic.*).[2]

Creating a file

Since the Scheme buffer will chronologically list all the expressions you evaluate, and since you will generally have to try more than one version of the same procedure as part of the coding and debugging process, it is usually better to keep your procedure definitions in a separate buffer, rather than to work only in the Scheme buffer. You can save this other buffer in a file on a floppy disk so you can split your lab work over more than one session. (The course notes package contains two floppy disks.)

Chapter 5 of *Don't Panic* describes files in more detail. The basic idea is to create another buffer, into which you can type your code, and later save that buffer in a disk file. You do this by typing `C-x C-f` *filename*. If you already have a buffer open for that file Edwin simply switches you into this buffer. Otherwise, Edwin will create a new buffer for the file. If you give the system a name that has not yet been used, with an extension of `.scm`, Edwin will automatically create a new buffer with that file name, in `scheme` mode. In a Scheme-mode buffer some editing commands treat text as code, for example, typing `C-j` at the end of a line will move you to the next line with appropriate indentation.

Once you are ready to transfer your procedures to Scheme, you can use any of several commands: `M-z` to evaluate the current definition, `ctrl-x ctrl-e` to evaluate the expression preceding the cursor, `M-o` to evaluate the entire buffer, or by marking a region and using `M-x eval-region`. Each of these commands will cause the Scheme evaluator to evaluate the appropriate set of expressions (usually definitions). By returning to the `*scheme*` buffer, you can now use these expressions.

[2] Make a habit of typing `ctrl-j` at the end of a line, instead of RETURN, when you enter Scheme expressions, so that the cursor will automatically indents to the right place.

2. Exploring the system

The following exercises are meant to help you practice editing, and using on-line documentation.

Exercise 1: Practice with the editor Copy the results of evaluating some of the expressions in Section 1 from the *scheme* buffer into an answer buffer.

Exercise 2: More practice with the editor Find the Free Software Foundation copyright notice at the end of the Edwin tutorial. Copy it into the answer buffer.

Exercise 3: Learning to use `Info` Start up the `info` program with the Edwin command `M-x info`. `Info` is a directory of useful information. You select a topic from the menu, by typing `m` followed by the name of the topic. You can type a few characters that begin the topic name, and then type a space, and Edwin will complete the name. One of the `info` options (type `h`) gives you a brief tutorial in how to use the program. Use `info` to find the cost of a 12-inch cheese pizza from Pizza Ring, and copy this to the answer buffer. (Extra credit: Is the price listed in the file still correct?)

Exercise 4: Hacker Jargon The info `Jargon` entry is a collection of hacker terms that was eventually published in 1983 as the book *The Hacker's Dictionary*. The `New Jargon` info entry is an expanded version that was published in 1991 by MIT Press as *The New Hacker's Dictionary*. The old Jargon file is structured as an `info` file, with submenus; the new version is a single text file, which you can search. Find the definition of "terminal brain death" from the new jargon file, and find the definition of "Phase of the Moon" from either jargon file. Include these in the answer buffer.

Exercise 5: Scheme documentation The `Scheme` info entry is an on-line copy of the Scheme reference manual that is distributed with the notes. Find the description of "identifiers" in the documentation. What is the longest identifier in the list of example identifiers?

Exercise 6: Getting information at MIT The Edwin command `M-x shell` will create a Unix shell buffer, which you can use to run various Unix programs. One of the more interesting ones is `finger`. For example, typing

`finger` *name*`@mit`

will show you information from the MIT directory. Find out who the following people are at MIT and what they do: Joel Moses, Michael C. Behnke, Rosalind Williams.

Exercise 7: Getting information from around the world You can also use the finger program to query computers on the Internet, all over the world. You can finger a particular name at some location, or just finger the location (e.g., `finger @mit.edu`) to get general information. Try fingering some of the following places:

- `cs.berkeley.edu`—a computer run by the computer science department at UC Berkeley
- `idt.unit.no`—a computer at the Norwegian Institute of Technology, a part of the University of Trondheim, Norway.
- `whitehouse.gov`—the White House

See what else you can find. Include some piece of this in your answer file.

Turning in your answers

The section on "Feedback" in the *Don't Panic* manual describes how to send Email from the 6.001 lab. Send your answer buffer to your tutor. Do this by starting a mail message and using the Edwin `M-x insert-buffer` command. Don't forget to include a reply email address (or else put your name in the message) so your tutor will know that the message is coming from you.