

## CS 3721: Programming Languages Lab

Lab #01 Solutions: Basic Scheme: learn a new language.

First, start the DrScheme programming environment. You can download various versions of *DrScheme* from <http://www.drscheme.org> or use the environment installed on the department Linux machines (by running “`drscheme`”). The web site, <http://www.drscheme.org>, has comprehensive documentation both about *DrScheme* and about the *Scheme* language itself. Various reference material for Scheme and Dr. Scheme can be found at <http://download.plt-scheme.org/doc/>; the Revised<sup>5</sup> Report on the Algorithmic Language Scheme is the official definition of Scheme.

After the *DrScheme* environment starts, it will open a window with two panels: the top panel (called the *definition window*) is for entering programs and saving commands to files; the bottom *interactive window* is for experimenting with Scheme commands without saving them. (Your commands in the interactive window can use definitions in the definition window.) You should periodically save your code in the *definition window* into a file so that your work is not lost.

Dr. Scheme implements several variations of the Scheme language. For the purpose of our class, you should use the language level, “Teaching Languages: Intermediate Student with lambda.” (You can choose the language level when you first start “`drscheme`” or by clicking menu item “language:choose language”).

Goals of this lab:

- Interact with, write and debug programs in Dr. Scheme; Understand, read, and write simple expressions in Lisp/Scheme’s prefix notation; Use the operator, *define*, to give symbolic names to expressions; Write numeric expressions using basic operators (e.g., +, -, /, sqrt, and =); Understand Lisp/Scheme lists and be able to use the *cons*, *car*, *cdr*, *null?*, and *pair?* operators.

You can work on the questions in small groups of 1-3 individuals each. However, you have to **write up your solution on your own**. Also, you should list the names of each individual in your group.

**Due date: January 29, 3:30pm.** At the beginning of the next recitation.

In this lab, we start with the most primitive concepts in Scheme.

1. First is the atom, syntactically, an *atom* can be any string of characters that does not contain any parentheses or spaces, and it will be treated as a single atomic unit when building up data structures. An atom can be a variable name, a number, or a symbol. At the interactive window, try entering some atoms (i.e., symbols and numbers).

*You may need to use '... or (quote ...) for symbols, for example, 'programming or (quote programming).*

Please list 5 atoms.

**Solution:**

for numbers:

> 5

```

> 7.2
> 0
for symbols:
> 'giants
> (quote @xy)
> 'a

```

2. Lists are collection of atoms and/or other lists, enclosed in parentheses and separated by spaces. A list may also be empty. At the interactive window, try to build some lists. *Hint: Use '... or (quote ...)' to protect the lists from being treated as expressions when they are evaluated, (for example, '(+ 2 3) evaluates to a list which contains the elements +, 2 and 3; whereas (+ 2 3) evaluates to the number, 5.) Note that the quote is not the part of the list.* Please write an example of an empty list, a simple list of atoms, and a list containing nested list.

**Solution:**

```

> '() ; empty list
> (quote ()) ; empty list
> '(a cs3721 3) ; a simple list of atoms
> (quote (a cs3721 3)) ; a simple list of atoms
> '(1 (a b) x (3 (z y) 4) c) ; nested list
> '(1 (a b)) ; nested list

```

3. The syntax of Scheme operations (Prefix notation).

Every Scheme program contains a sequence of expressions. Each expression can be a single constant value, a variable, an operation which operates on a collection of values (other expressions), or a special form. The syntax of an operation has the following format: `'( op exp1 exp2 ... )'` — that is, each expression starts with `'(`, followed by an operator, then followed by one or more operands, and finally ends with `)'`. In the interactive window, try to evaluate the result of  $1903 + 249 \div 7.8$ . Write down your Scheme expression. What is the result of the expression? You can similarly apply comparison operations such as `<`, `>`, `<=`, `>=` to numbers, and apply *and*, or operators to boolean values (which are either *true* or *false*). Write down a scheme expression to evaluate  $(\sqrt{2+7} \leq 3) \wedge (5 > 3)$ .

**Solution:**

Scheme expression for  $1903 + 249 \div 7.8$  :

```

> (+ 1903 (/ 249 7.8))
1934.923076

```

Scheme expression for  $(\sqrt{2+7} \leq 3) \wedge (5 > 3)$  :

```

> (and (<= (sqrt (+ 2 7)) 3) (> 5 3))
true

```

4. In Scheme, to declare a new variable, use operator *define*, which takes two operands, the name of the variable, and the value of the variable. In the interactive window, define a variable *x* which has the value 5. Write down your definition. What happens if you try to redefine the value of *x*? Also, define a variable *myName* which is your

first name. After making these definitions, type  $x$  and  $myName$ , respectively, in the interactive window, and observe the results.

**Solution:**

```
> (define x 5)
> (define x 7)
define: cannot redefine name: x
```

```
> (define myName 'baris)
> x
5
> myName
'baris
```

5. Scheme provides three operators ( $cons$ ,  $car$ , and  $cdr$ ) for dynamically building lists ( $cons$ ) and for pulling lists apart ( $car$  and  $cdr$ ). In the interactive window, try to rebuild the non-empty lists (including the nested list) that you built in question 2 using the  $cons$  operator. Save your lists in variables  $list1$  and  $list2$  respectively. Then try to pull apart your lists using  $car$  and  $cdr$ . Write down your Scheme expressions.

**Solution:**

For the list '(a cs3721 3) :

```
> (define list1 (cons 'a (cons 'cs3721 (cons 3 '))))
```

For the nested list '(1 (a b)) :

```
> (define list2 (cons 1 (cons (cons 'a (cons 'b '())) '())))
```

(a b)  
((a b))  
(1 (a b))

```
> (car list1)
'a
> (cdr list1)
(list 'cs3721 3)
> (car (cdr list1))
'cs3721
```

```
> (car list2)
1
> (cdr list2)
(list (list 'a 'b))
> (car (cdr list2))
(list 'a 'b)
> (car (car (cdr list2)))
'a
```

6. In addition to the numerical comparisons, Scheme provides various test operators (predicates) that return boolean values, true or false. The two examples of these

test operators are *null?* and *pair?*. *null?* tests whether the given list is empty or not. *pair?* tests whether the given list is a non-empty list or not. First, apply these predicates on an empty list. Then, apply them on the lists *list1* and *list2* that you built in question 5. Write down your Scheme expressions.

**Solution:**

```
> (null? '())
true
> (pair? '())
false
> (null? list1)
false
> (pair? list1)
true
> (null? list2)
false
> (pair? list2)
true
```