Cosc 2P03 Fall 2014
Assignment#1

(Due date for assignment is Friday Sept. 19th,4:00 p.m. ., Late date Friday Sept. 22nd, 4:00 p.m.)

This assignment will exercise your knowledge of 1P03 material. It is not a difficult assignment, but may be a little tedious.

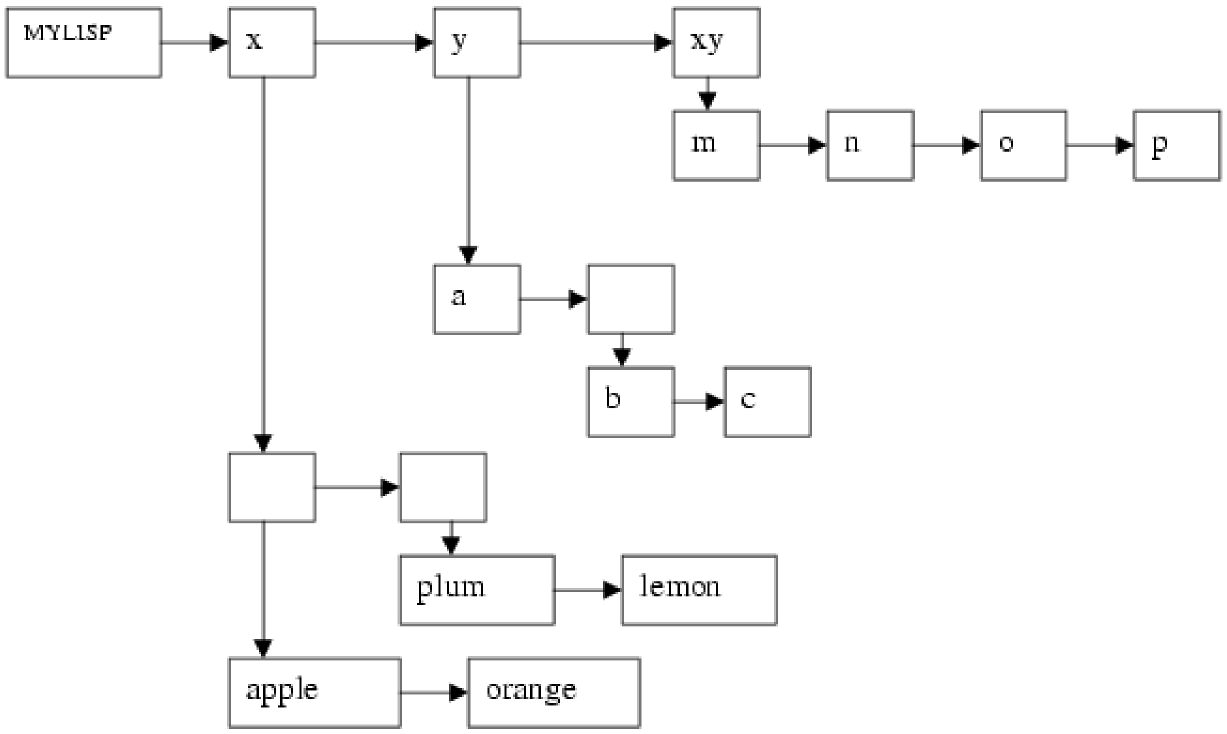
LISP is a Language used for list processing and is often associated and used in the artificial intelligence community to implement AI type problems. Mainly LISP is a symbolic processing language which manipulates symbols as opposed to traditional languages like JAVA which manipulates objects. LISP is an acronym for **LIS**t **P**rocessing but often this acronym is associated with **L**ost **I**n **S**tupid **P**arenthesis (this will become apparent). The list in LISP is the corner stone in which statements are defined. A list is a sequence of items or elements enclosed in a pair of parentheses. Where each element can either be an atomic element (meaning a basic element) or another list. For example:

(a) -- single element **a**
(a b c) -- 1 list with **a b c** as elements.
(a (b c)) -- 1 list 2 elements where the second element is a list.
((apple orange) (plum lemon)) -- 1 list with 2 sublists, the first **apple orange** the second **plum lemon.**

We can define variables associated with each list. So like any language we can declare **x** to represent say **((apple orange) (plum lemon))**, thus if we refer to **x** we refer to **((apple orange) (plum lemon))**. We will create a data structure which will be able to store LISP lists. This data structure will consist of a linked list of linked lists. The main section of the data structure will contain 1 link for each variable defined. For example:

x = ((apple orange) (plum lemon))
y = (a (b c))
xy=(m n o p)

Then the data structure would look as follows.



Notice that atomic elements are placed directly into the node, while a list is defined as an empty node which points to the sublist. In order to store the lists we can use the LISP command setq. For example:

setq x '((apple orange) (plum lemon))

Note that the ' (apostrophe) is needed to indicate that **((apple orange) (plum lemon))** is literal. You will need to code this command to accept a literal or other variables. For example:

setq xy y --would make a copy of what **y** represents and assign it to **xy** if **y** exists, otherwise nothing happens. If xy does not exist it is created, otherwise the new assignment supper-seeds the old.

setq x nil --would delete x from the data structure.

The command line version is used since this is traditionally the way LISP interpreters worked, from the command line.

Now we need to be able to print the contents of a variable. Here we will use a command **print n.** Where **n** represents the variable to be printed if it exists. If n is omitted then all variables declared are printed. For Example:

```
print x
==> ((apple orange) (plum lemon))   --Prints the value of x

print abc
==> NIL  --Prints NIL since the variable abc does not exist

print
==> ((apple orange) (plum lemon))
==> (a (b c))
==> (m n o p)
```

An online lisp implementation

If you want to play a bit with lisp here is an online lisp interpreter, http://www.ugcs.caltech.edu/~rona/tlisp/, you can copy and paste the following into the "Lisp Input"

```
(setq a '(b c d))
(print a)
```

then hit evaluate

Some suggestions:

1. A LISP list can have an infinite number of nested sublists, for example the following list is legal (((((b))))) hence the data structure must be able to support this.
2. Since we are dealing with balanced brackets we could parse a list using a recursive method (an assignment requirement). This method would then store each section of the list in the data structure as exemplified above.
3. Note that brackets are not stored within the data structure. A bracket is really just a link with a right and down pointer but no data. Thus we can distinguish between a node which contains data from one which represents a sublist by querying the down pointer. This pointer is NULL with data.
4. Printing a variable should also be recursive (an assignment requirement), since this would take into account the organization of the data structure. As we descend we print a left bracket, as we exit the right bracket is printed.
5. Be conscious of mismatched parenthesis, since we know that for each left parentheses a matching right should exist. Bad commands and or data should not crash your program.
6. Final suggestion, a recursive method is used for **setq** and **print.** These 2 methods are very similar, once one is coded the other will follow directly. The setq method is about 25 +- lines of code, if your method goes over 35 lines, you are doing it wrong. You will need to get your head wrapped around the structure of a list. Think ONION shell.

You should test your program with as many variants as possible to ensure it is bug free. It will be your responsibility to ensure that bad commands, data, or types of valid input does not crash your program. The markers will be testing your solution with their own data. You are to submit the input and output obtained from your tests.

Submission Requirements:

- **Cover Sheet** completely filled out, available from: "http://www.cosc.brocku.ca/coverpage" Note: your assignment will not be marked unless one is submitted with the assignment on the assignment due date.
- **Commented and properly documented** source code listing, use Java Doc style.
- Listing of any input you used to test your program.
- Listing of your output which reflects the input.
- Source code is to be Java.
- Electronic submission, run the script "submit2p03" from sandcastle.

- Statement on coversheet with following information.
    - Platform, e.g. Mac, PC, Commodor 64, my Java enabled wrist watch.
    - Compiler Version, e.g. Java 1.6, Java 1.7 e.g.

Good Luck