**Lab 6**

**Problem 1**

* **Helper function: print\_spaces**

(defun **print\_spaces** (indent)

    (cond

        ((> indent 0)

            (princ '|  |)

            (print\_spaces (- indent 1))

        )

    )

)

* **Pretty Print 1**

(defun **pretty\_print** (lst)

    (defun **print\_elem** (elem indent)

        (if (atom elem)

            (princ elem)

            (print\_list elem indent)

        )

    )

    (defun **print\_list** (lst indent)

        (setq indent (1+ indent))

        (princ '|( |)

        (print\_elem (car lst) indent)

        (mapcan (lambda (elem)

                (terpri)

                (print\_spaces indent)

                (print\_elem elem indent)

            )

            (cdr lst)

        )

        (princ '| )|)

    )

    (print\_elem lst 0)

    (terpri)

)

* **Test cases:**

(pretty\_print '(a ( (b c) d e) nil f (g h) (i)) )

*; ( A*

*;   ( ( B*

*;       C )*

*;     D*

*;     E )*

*;   NIL*

*;   F*

*;   ( G*

*;     H )*

*;   ( I ) )*

(pretty\_print '(a b c) )

*; ( A*

*;   B*

*;   C )*

(pretty\_print '((a b c)) )

*; ( ( A*

*;     B*

*;     C ) )*

(pretty\_print 'a)

*; A*

(pretty\_print nil)

*; NIL*

* **Pretty Print 2**

(defun **my\_pretty\_print** (lst)

    (defun **my\_print\_elem** (elem indent)

        (if (atom elem)

            (princ elem)

            (my\_print\_list elem indent)

        )

    )

    (defun **my\_print\_list** (lst indent)

        (setq indent (1+ indent))

        (princ '|(|)

        (mapcan (lambda (elem)

                (terpri)

                (print\_spaces indent)

                (my\_print\_elem elem indent)

            )

            lst

        )

        (terpri)

        (print\_spaces (1- indent))

        (princ '|)|)

    )

    (my\_print\_elem lst 0)

    (terpri)

)

* **Test Cases**

(my\_pretty\_print 'a)

*; A*

(my\_pretty\_print nil)

*; NIL*

(my\_pretty\_print '(a b c) )

*; (*

*;   A*

*;   B*

*;   C*

*; )*

(my\_pretty\_print '((a b c)) )

*; (*

*;   (*

*;     A*

*;     B*

*;     C*

*;   )*

*; )*

(my\_pretty\_print '(a ( (b c) d e) nil f (g h) (i)) )

*; (*

*;   A*

*;   (*

*;     (*

*;       B*

*;       C*

*;     )*

*;     D*

*;     E*

*;   )*

*;   NIL*

*;   F*

*;   (*

*;     G*

*;     H*

*;   )*

*;   (*

*;     I*

*;   )*

*; )*

**Problem 2**

(defun **copy\_file** (file\_in file\_out)

    (let (

        (input\_stream (open file\_in

            :direction :input

        ))

        (output\_stream (open file\_out

            :direction :output

            :if-exists :supersede

            :if-does-not-exist :create

        ))

    )

        (do (

            (line (read-line input\_stream nil :eof) (read-line input\_stream nil :eof) )

        )(

            (eq line :eof)

        )

            (write-line line output\_stream)

        )

        (close input\_stream)

        (close output\_stream)

    )

)

* **Test Case:**

(defvar path "Year III - Sem I/FCPL - Fundamental Concepts of Programming Languages/LAB/Lab 6/")

(defvar file1 (concatenate 'string path "input.txt"))

(defvar file2 (concatenate 'string path "output.txt"))

(copy\_file file1 file2)

* **Input File: input.txt**

sample text

more text

aaaaaaaaaa

* **Output File: output.txt**

sample text

more text

aaaaaaaaaa

* **Input File: input.txt**

* **Output File: output.txt**