

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
; )
```


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