Sonoma State University
Computer Science Department
CS 460 – Fall 2023 - Watts

Exercise 4

Date Due: Friday, 1 December 2023; 6:59 a.m.

Part 1. Create a new directory for Exercise 4 files. In this part of the exercise, several new elements PL460 will be introduced:

```
let     read     eof?     list     append     #t     #f
```

1. The PL460 “let” command allows the programmer to define and initialize local variables. The syntax of the let command is illustrated below.

a. Enter the following PL460 code into a file called Ex4-1a.pl460:

```
;; This program will be used to illustrate use of the "let" command

(define (main)
      (let ((value1 2) (value2 5))
             (display "value1 --> ")
             (display value1)
             (newline)
             (display "value2 --> ")
             (display value2)
             (newline)
             (display "value1 + value2 --> ")
             (display (+ value1 value2))
             (newline)
      )
)

(main)
```

Run the program using the pl460 interpreter. (You can copy it from your Exercise 3 directory.) Is the output what you expected?

b. Copy Ex4-1a.pl460 to a new file called Ex4-1b.pl460. Add a second let statement to the main function:

```
;; This program will be used to illustrate use of multiple "let"
;; commands

(define (main)
      (let ((value1 2) (value2 5))
             (display "value1 --> ")
             (display value1)
             (newline)
             (display "value2 --> ")
```

```
                (display value2)
                (newline)
                (display "value1 + value2 --> ")
                (display (+ value1 value2))
                (newline)
        )
        (let ((value1 10) (value2 15))
                (display "value1 --> ")
                (display value1)
                (newline)
                (display "value2 --> ")
                (display value2)
                (newline)
                (display "value1 + value2 --> ")
                (display (+ value1 value2))
                (newline)
        )
)

(main)
```

Run the program using the pl460 interpreter. Is the output what you expected?

c. Copy Ex4-1a.pl460 to a new file called Ex4-1c.pl460. Add a second let statement nested within the first let statement in the main function:

```
;; This program will be used to illustrate use of nested "let"
;; commands

(define (main)
   (let ((value1 2) (value2 5))
        (display "value1 --> ")
        (display value1)
        (newline)
        (display "value2 --> ")
        (display value2)
        (newline)
        (display "value1 + value2 --> ")
        (display (+ value1 value2))
        (newline)
        (let ((value1 (+ value1 value2))
                                 (value2 (* value1 value2)))
             (display "value1 --> ")
             (display value1)
             (newline)
             (display "value2 --> ")
             (display value2)
             (newline)
             (display "value1 + value2 --> ")
             (display (+ value1 value2))
             (newline)
```

```
            )
        )
    )

    (main)
```

2. The PL460 “read” command allows the programmer to read a white space delimited string from the standard input (stdin) stream. The syntax of the read command is illustrated below.

a. Enter the following PL460 code into a file called Ex4-2.pl460:

```
;; This program will be used to illustrate use of the "read" command

(define (main)
        (let ((value1 (read)) (value2 (read)))
                (display "value1 --> ")
                (display value1)
                (newline)
                (display "value2 --> ")
                (display value2)
                (newline)
                (display "value1 + value2 --> ")
                (display (+ value1 value2))
                (newline)
        )
)

(main)
```

b. Run this program using the pl460 interpreter. When the program pauses for input, enter the values 34 and -24. Is the output what you expected?

c. Run this program using the pl460 interpreter. When the program pauses for input, enter the values abc and xyz. Is the output what you expected?

3. The PL460 “eof?” predicate allows the programmer to test for the end of input.

a. Enter the following PL460 code into a file called Ex4-3.pl460:

```
;; This program will be used to illustrate use of the "eof?" command

;; The readList function will read values from the input stream.
;; When the end-of-file marker is reached it will stop reading and
;; return the values as a list.

(define (readList)
          (let ((value (read)))
                (if (eof? value)
                        '()
                        (cons value (readList))
                )
        )
)

(define (main)
        (let ((alist (readList)))
                (display "The input list --> ")
                (display alist)
                (newline)
        )
)

(main)
```

b. Run this program using the pl460 interpreter. When the program pauses for input, enter white space separated values of your choice. When you get tired of entering values, press enter and ctrl-d to send an end-of-file marker. Is the output what you expected?

c. Enter the following data into a file called “input”

```
    123 abc 45.6 x 37/2 y
            "Hello world" 'goodbye
```

Run this program using the pl460 interpreter and the input file:

```
    ./pl460 Ex4-3.pl460 < input
```

Is the output what you expected?

4. The PL460 “list” command allows the programmer to turn an “atom” into a list.

a. Enter the following PL460 code into a file called Ex4-4.pl460:

```
;; This program will be used to illustrate use of the "list" command

(define (main)
        (let ((alist '(a b 23 45)))
                (display "The list --> ")
                (display alist)
                (newline)
                (display (cons "first" alist))
                (newline)
                (display (cons alist "last"))
                (newline)
        )
)

(main)
```

Run the program using the pl460 interpreter. Is the output what you expected?

Modify the line:
```
        (display (cons alist "last"))
```
to:
```
        (display (cons alist (list "last")))
```

Run the program again. How did the output change?

5. The PL460 “append” command allows the programmer to concatenate 2 lists.

   a. Enter the following PL460 code into a file called Ex4-5.pl460

```
;; This program will be used to illustrate use of the "append" command


(define (main)
    (let ((alist '(a b 23 45)) (blist '(x y -34 -1/2)))
        (display "alist --> ")
        (display alist)
        (newline)
        (display "blist --> ")
        (display alist)
        (newline)
        (display (cons alist blist))
        (newline)
        (display (cons blist alist))
        (newline)
        (display (append alist blist))
        (newline)
        (display (append blist alist))
        (newline)
    )
)

(main)
```

   b. Add the following lines to the end of the main function:
```
        (display (append alist blist))
        (newline)
        (display (append blist alist))
        (newline)
```

   Run the program again. How did the output change?

6. The PL460 “#t” and “#f” literals can be used as return values or as part of a conditional statement.

   a. Enter the following PL460 code into a file called Ex4-5.pl460

```
;; This program will be used to illustrate use of the "#t" and
;; "#f" logical literals

(define (main)
        (let ((value1 (read)) (value2 (read)))
                (display "value1 --> ")
                (display value1)
                (newline)
                (display "value2 --> ")
                (display value2)
                (newline)
                (display "value1 < value2 --> ")
                (if (< value1 value2)
                        (display #t)
                        (display #f)
                )
                (newline)
        )
)

(main)
```

Run the program using the pl460 interpreter. When the program pauses for input, enter 7 and 10. Is the output what you expected?

Run the program using the pl460 interpreter. When the program pauses for input, enter 10 and 7. Is the output what you expected?

Run the program using the pl460 interpreter. When the program pauses for input, enter "seven" and "ten". Is the output what you expected?

Part 2. In this part of the exercise, you will be writing additional PL460 functions.

1. <u>Copy the function "reverse" that you created for Exercise 3 to a new file called *lastname*E4.pl460. You may wish to use other functions from Exercise 3 as you complete Exercise 4. That is a very good plan; however, please do not copy all of your Exercise 3 functions to Exerise 4 – just reverse and any others you decide to use.</u>

2. *Execute the list reversing procedure you wrote for Exercise 3 using '(a (b c) d) as input. Does it return (d (c b) a)? Or does it return (d (b c) a)?* Write a new procedure called **`all_reverse`** that recursively reverses nested lists. Execute your new procedure using '(a (b (c d)) (e f)) as input. It should return ((f e) ((d c) b) a).

Recursive sorts. When looked at from a procedural paradigm perspective, the recursive sorts mergesort and quicksort are of the form:

```
recursively-sort (list L)
{
        if (size of L > 1)
        {
              split L into 2 parts: L1 and L2
              recursively-sort L1
              recursively-sort L2
              combine sorted lists L1 and L2 into a sorted list L
        }
    }
```

From a functional paradigm perspective, they are of the form:

```
    (define (recursively-sort L)
          (if (size of L > 1)
                (combine (recursively-sort (half of L))
                           (recursively-sort (other half of L))
                )
          )
    )
```

3. Using PL460, write 2 sorts that illustrate an understanding of the functional paradigm:

   a. Write a function called **`mergesort`** that uses the Merge Sort algorithm to sort its single list argument into ascending order. You will probably need to create a helper function (or functions) to split the list into 2 parts. You will also need a merge procedure.

   b. Write a function called **`quicksort`** that uses the Quick Sort algorithm to sort its single list argument into descending order. You will probably need to create a helper function (or functions) to split the list into 2 parts – those elements < than the pivot and those >= to the pivot. This is in excellent opportunity for using the let function. You will also need to write an append procedure.

4. Write a recursive function called **`numbers_only`** that will remove all non-numeric values from a list.
   a. The input to the function should be a list
   b. The function should return a list containing only numeric values
   c. If the input list contains sub-lists, the numeric values from the sub-lists should be included in the returned list.

   Examples:
```
    (numbers_only '(1 2 3)) ==> (1 2 3)
    (numbers_only '(1 b 3)) ==> (1 3)
    (numbers_only '(1 (2 3 f 4) 5 (a (6 (7 8) d 9)))) ==>
                                                (1 2 3 4 5 6 7 8 9)
```

5. Test your sorts
   a. Run each of your sorts using the list '(1 5 3 6 8 92 –1 0 4 5 3).
   b. Run each of your sorts using the list '(a b d e c t s).
   c. How do your sorts work with these input lists? Modify your sorts to call your `numbers_only` function so that they sort only lists of numeric values.

Date Due: Date Due: Friday, 1 December 2023; 6:59 a.m.

To Turn In: Place your well documented procedures in a file called *lastname*E4.pl460 and drop the file into the cs460drop folder.