COMP-301 PS-1

Problem-1:

1-) (+ 5 3 4) = 5+3+4= 12

2-) (/ 6 2) = 6/2= 3

3-)

(+ (* 2 4) (- 4 6)) = (+ 8 -2) = 8 – 2 = 6

4-)

```
(define a 3)  ; a= 3
(define b (+ a 1)) ; b= (+ 3 1)   b= 4
(+ a b (∗ a b)) = (+ 3 4 (* 3 4)) = (+ 3 4 12) = 3+4+12= 19
(= a b) = (= 3 4)= #f (Since 3 is not equal to 4, this statement returns #f)
```

5-)

```
(define a 3)  ; a= 3
(define b (+ a 1)) ; b= (+ 3 1)   b= 4
(if (and (> b a) (< b (∗ a b))) b a)
```

(> b a) = b>a= 4>3= #t

(< b (* a b)))= b < a*b = 4 < 3*4 = 4 < 3*4 = 4 < 12 = #t

(and #t #t)= #t

So, this program gives the value of b, which is 4, as the output.

6-)

```
(define a 3)  ; a= 3
(define b (+ a 1)) ; b= (+ 3 1)   b= 4


 (cond ((= a 4) 6)

    ((= b 4) (+ 6 7 a))

(else 25))
```

The value of b is equal to 4. So, b=4 will return #t. Since b=4 is true, the cond statement will return (+ 6 7 3) at the end.

(+ 6 7 3) = 6+7+3= 13+3 = 16

7-)

```
(define a 3)  ; a= 3
(define b (+ a 1)) ; b= (+ 3 1)   b= 4


(+ 2 (if (> b a) b a))
```

b>a= 4>3= #t

Since the condition in the if statement is true, the second number in the addition will be equal to the value of b (the value of b is equal to 4).

(+ 2 4)= 4+2= 6

8-)

```
(define a 3)  ; a= 3
(define b (+ a 1)) ; b= (+ 3 1)   b= 4


(* (cond ((> a b) a)

       ((< a b) b)

           (else -1))
        (+ a 1))
```

a>b= 3>4= #f
a<b= 3<4= #t

Since a<b returns true, cond statement will return the value of b (the value of b is equal to 4).

(* b (+ a 1))= (* 4 (+ 3 1)) = (* 4 4)= 4*4= 16

Problem-2:

Part-A:

```
(define idx_getter
   (lambda (my_list elemInd)
    (cond((null? my_list) '())
        ((= elemInd 0) (car my_list))
        (else (idx_getter (cdr my_list) (- elemInd 1))))))
```

Part-B:

```
(define recurFibo (lambda (n)
   (if(<= n 0) 0  (cond
    ((= n 1) 1)
    (else
     (+ (recurFibo (- n 1))
       (recurFibo (- n 2))))))))
```

Part-C:

```
(define (primeness_control_helper_func? my_num divisorNum)
  (if (= my_num divisorNum) #t
  (if (= (custom_remainder_func my_num divisorNum) 0) #f
    (primeness_control_helper_func? my_num (+ divisorNum 1)))))

(define ( primeness_control? my_num)
  (if(<= my_num 1) #f
  (if (= my_num 2 ) #t
   (primeness_control_helper_func? my_num 2 ) ))
)

(define (custom_remainder_func firstNum secondNum)
(- firstNum (* (floor (/ firstNum secondNum)) secondNum ))
)
```