Lab 4

Problem 1

- Factorial

```
(defun factorial (num)
    (if (= 0 num)
        1
        (* num (factorial (- num 1)))
    )
)
```

- Test Cases:

```
(print (factorial 1))
; 1

(print (factorial 10))
; 3628800

(print (factorial 15))
; 1307674368000
```

Problem 2

- Fibonacci

```
(defun fibonacci (num)
    (defun fib (num)
        (cond
            ((= num 0) 0)
            ((= num 1) 1)
            (t (+ (fib (- num 1)) (fib (- num 2))))
        )
    )
    (if (= 0 num)
        nil
        (append (fibonacci (- num 1)) (list (fib num)))
    )
)
```

- Test Cases:

```
(print (fibonacci 11))
; (1 1 2 3 5 8 13 21 34 55 89)

(print (fibonacci 3))
; (1 1 2)

(print (fibonacci 21))
; (1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
4181 6765 10946)
```

Problem 3

- Is Member

```
(defun is_member (elem lst)
    (cond
        ((null lst) nil)
        ((eql elem (car lst)) lst)
        (t (is_member elem (cdr lst)))
    )
)
```

- Test Cases:

```
(print (is_member 4 '(1 (2 3) 4 5 6)))
; (4 5 6)

(print (is_member 4 '(1 (2 3) 4.0 5 6)))
; NIL

(print (is_member '(2 3) '(1 (2 3) 4 5 6)))
; NIL

(print (is_member 7 '(1 (2 3) 4 5 6)))
; NIL
```

Problem 4
- Trim Head

```
(defun trim_head (lst n)
    (cond
        ((null lst) nil)
        ((zerop n) lst)
        (t (trim_head (cdr lst) (- n 1)))
    )
)
```

- Test Cases:

```
(print (trim_head '(1 2 3 4 5 6) 3))
; (4 5 6)

(print (trim_head '(1 2 3) 3))
; NIL

(print (trim_head '(1) 3))
; NIL

(print (trim_head nil 3))
; NIL
```

Problem 5
- Trim Tail

```
(defun trim_tail (lst n)
    (defun trim_head (lst n)
        (cond
            ((null lst) nil)
            ((= 0 n) lst)
            (t (trim_head (cdr lst) (- n 1)))
        )
    )
    (reverse (trim_head (reverse lst) n))
)
```

- Test Cases:

```
(print (trim_tail '(1 2 3 4 5 6) 3))
; (1 2 3)

(print (trim_tail '(1 2 3) 3))
; NIL

(print (trim_tail '(1) 3))
; NIL

(print (trim_tail nil 3))
; NIL
```

Problem 6

- Count Atoms

```
(defun count_atoms (lst)
    (let (
        (head (car lst))
        (tail (cdr lst))
    )
        (if (null lst)
            0
            (+
                (if (atom head)
                    1
                    (count_atoms head)
                )
                (count_atoms tail)
            )
        )
    )
)
```

- Test Cases:

```
(print (count_atoms '(1 2 nil (three 4) 5 (6 (seven 8) nine) 10)))
; 11

(print (count_atoms nil))
; 0

(print (count_atoms '(1 2 3)))
; 3
```

Problem 7

- Add

```
(defun add (num1 num2)
    (if (= 0 num2)
        num1
        (add (1+ num1) (1- num2))
    )
)
```

- Test Cases:

```
(print (add 5 7))
; 12

(print (add 7 5))
; 12
```

Problem 8

- Reverse

```
(defun my_reverse (lst)
    (cond
        ((null lst)
            nil
        )
        (t
            (append (my_reverse (cdr lst)) (list (car lst)))
        )
    )
)
```

- Test Cases:

```
(print (my_reverse nil))
; nil

(print (my_reverse '(1 2 3 4 5)))
; (5 4 3 2 1)

(print (my_reverse '(1 2 (3 4) 5)))
; (5 (3 4) 2 1)
```

Problem 9

- Is Member

```
(defun is_present (elem lst)
    (let (
        (head (car lst))
        (tail (cdr lst))
    )
        (cond
            ((null lst)
                nil
            )
            ((listp head)
                (is_present elem head)
            )
            ((eql (car lst) elem)
                t
            )
            (t
                (is_present elem (cdr lst))
            )
        )
    )
)
```

- Test Cases:

```
(print (is_present nil '(1 2 3 4 5)))
; nil

(print (is_present nil '(1 2 () 3 4 5)))
; T

(print (is_present 3 '(1 2 3 4 5)))
; T

(print (is_present 3 '(1 2 (3 4) 5)))
; T

(print (is_present 6 '(1 2 3 4 5)))
; NIL
```

Problem 10

- Squash

```
(defun squash (lst)
    (let (
        (head (car lst))
        (tail (cdr lst))
    )
        (cond
            ((null lst)
                nil
            )
            ((atom head)
                (append (list head) (squash tail))
            )
            ((listp head)
                (append (squash head) (squash tail))
            )
            (t
                (squash tail)
            )
        )
    )
)
```

- Test Cases:

```
(print (squash '(1 nil (two 3) 4 (5 (6 7) 8) 9 nil)))
; (1 nil two 3 4 5 6 7 8 9 nil)
```