Showed in Class

PRACTICE EXAM

IN SCHEME AND PROLOG CREATE A PROGRAM THAT REMOVES DUPLICATES

PROLOG:

```
remove_dups([],[]).                          % Nothing in List

remove-dups([E|A],R):-                       % is a member / is duped
        member(E,A)
        remove-dups(A,R)                     % concatenating Element to Result
                                             % Because it is not duped

remove-dups([E|A],[E|R]):-                   % is not a member
        not(member(E,A))
        remove-dups(A,R)
```

SCHEME: Shown in class & in labs

List(a) '(a) '()

```
(define (remove-dups L)
  (letrec (Let (L sort(L))                   ; only have 1 elem
    (cond
      ((null? L) '())
      ((null? (cdr L)) L)                    ; last item, return item
      ((equal? (car L) (car (cdr L))) (remove-dups (cdr L)))
      (else (cons (car L) (remove-dups (cdr L))))
    )
  )
)
```

I will double check this later & will fill in later

cadr = car cdr

start/head — head of tail

a unique eleme — Answer

READ THE SCHEME FUNCTION BELOW, WHAT IS IT DOING? COMPLETE THE RECURSIVE CALL.

L1 L2

```
(define (union L1 L2)
 (cond
        ((null? L1) L2)
        ((null? L2) L1)
        ((not (member (car L1) L2)) (union (cdr L1) L2))
        (else (cons (car L1) (union (cdr L1) L2 )
)
)
```

— if head of L1 is not unique to L2

when car L1 is unique

—Zep. from foobar N Grill

-Joey:I XD

descending

CREATE A PROLOG FUNCTION THAT DOES THE FOLLOWING:

TAKES 2 LISTS, AND MERGES THE CONTENTS SO THAT IT IS IN ASCENDING ORDER

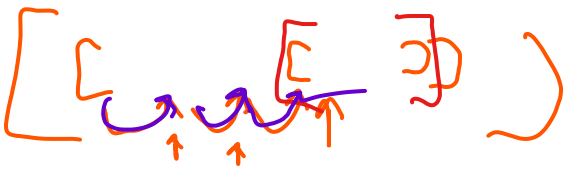
EXAMPLE:

merge([2,6,13,15,20],[4,7,9,17,21,34,56],L).
L = [2,4,6,7,9,13,15,17,20,21,34,56].

```
merge([],L,L).
merge(L,[],L).

merge([H1|T1],[H2|T2],[H1|R]) :-
        H1 =< H2,
        merge(T1,[H2|T2],R).

merge([H1|T1],[H2|T2],[H2|R]) :-
        H1 > H2,
        merge([H1|T1],T2,R).
```



CREATE A PROLOG FUNCTION THAT DOES THE FOLLOWING:

other questions that invole slicing

slice([2,6,12,10,1,7],1,3,L).
L=[6,12,10]. in

if num elements from given index is less than given number, give rest of list (instead of out of bounds)

n/n List

```
slice(_,_,0,[]).
```
index number

```
slice([],_,_,[]).
```

```
// Loop to the correct index
Slice([[H|T],I,N,R):-
    N>0,
    I>0,
    I1 is I-1,
    Slice(T,I1,N,R).
```

decrement

[H|T]
[H|R]

```
//@ index zero, if len L is > N
Slice([H|T],0,N,[H|R]):-
    N>0,
    length([H|T],len),
    Len > N,
    N1 is N-1,
    Slice(T,0,N1,R).
```

```
//@ index zero, if length < N, Return Rest of list
Slice(L,0,N,L):-
    N>0,
    length(L,len),
    len =< N.
```

CREATE A SCHEME FUNCTION THAT PERFORMS THE SAME TASK

(define (slice L index n)

```
(cond
 [ ((equal? n 0) '()) ; if n = 0, return empty
 [ ((null? L) '()) ; if empty list
 [ ((equal? i 0) (> (length L) n))
      (cons (car L) (slice (cdr L) 0 (- n 1)))
 [ (and (equal? i 0) (< (length L) N) L)
 [ (else (slice (cdr L) (- i 1) N))
```

(let [1, 2, 3, 4, 5, 6])

(cdddr L)

cdr cdr cdr = [4, 5, 6]

(cdar L) 2, 3, 4, 5, 6

TIPS

- **REVIEW CHAPTER ON STRUCTS W GO FUNCTIONS** Know how they work Know how to build funcs for them
- **STUDY SETOF AND BAGOF** w/n prolog

(test '(1 2 2 3 4 5))

WHAT DOES THIS FUNCTION DO

takes every elem & checks # time they repeat

```
(define (test L)
        (test1 L 0)
)

(define (test1 L m)
        (if (null? L)
                m
        else    (if (> (test-helper L (car L)) m)
                        (test1 (cdr L) (test-helper L (car L)))
                        (test1 (cdr L) m)
                )
        )
)

(define (test-helper L c)
        (cond
                ((null? L) 0)
                ((equal? (car L) c) (+ 1 (test-helper (cdr L) c)))
                (else (test-helper (cdr L) c))
        )
)
```

checking the max # of occurences of every element w/in L

if cond ten + occurs w/in L increment

(cons recursive

WHAT DOES THIS FUNCTION DO

```
sibling([john, mike, linda]).
sibling([jane, maria, harry]).
sibling([kelly, brian, nina, james]).
//sorted from oldest to youngest

setof(H, sibling([H|T]),length(T)>1,B).
B=[john,james,kelly]

setof(H, sibling([H|T]),length(T)>2,B).
B=[kelly]
```

3 > 2

if first sibling has more than 1 sibling

if first sibling has more than 2 other siblings (greater than 2)

easy & short

n=0

```
(let loop ((n 4) (i 2)) (if (zero? n) i (loop (- n 1) (+ i 1))))
```

n=4
i=2
i=2+4=6

=> 6

head L

```
(let* ((L1 '(1 2 3 4)) (L2 (car L1))
       (F (lambda (L) L))) (cons (F L2) L1))
```

name

=> (1 1 2 3 4)

```
((lambda (x y) (* (* x y) (* x y))) 2 3)
```

6 6

x = 2
y = 3

=> 36

```
(let ((A '(1 2))) (cond ((null? (cdr A)) '(1))
                        ((list? (cadr A)) '(2))
                        ((list? (cddr A)) '(3))))
```

No

No

A = (1,2)
(1) ()

atom returns

checks if empty list is a list

car cdr
A = (2)
= ()

=> (3)

* car returns atom

car ((1)) → (1)

```
(cons (car '((1))) '((8)))
```

L

=> ((1) ((8)))

'(1 2 3) '(4) = '(1 2 3 4)

'((1)

ignore 1st layer of Brackets in Second element but not first

```
> (q5 '(1 2 3 4))
'(2 1 3 4)
> (q5 '(1 2))
'(2 1)
> (q5 '(4))
'(4)
> (q5 '())
'()
```

take head of the tail replace "swap" in the head of List

null? (cdr L)

cdr cdr = cddr

```
(define (q5 L)
  (cond
    ((null? L) L)
    ((null? (cdr L)) L)
    (else (let a (car L)) (let b (cadr L))
          (cons (cons (b) (a)) (cddr L))
    ))
```

cadr is taking head of tail

(else (cons (cadr L) (cons (car L) (cddr L))))

The function `two-last` is to evaluate to the two last elements from a list. (For lists shorter than two, the list is unchanged).

Example:

```
> (two-last '(1 2 3 4 5))
'(4 5)
```

Complete the function definition below:

```
(define (two-last L)
  (cond
    [(null? L) L)
    ((null? (cdr L)) L)
    ((null? (cddr L)) L)
    (else (two-last (cdr L)))
```

only occur @ start

decrements till that point

Question 7 Binary Search Tree in Scheme [3 marks]

Consider the following function that searches a binary tree

```
(define (tree-search x t)
      (cond
        ((null? t) #f)
        ((equal? x (car t)) #t)
        ((< x (car t)) (search x (cadr t)))
        ((> x (car t)) (search x (caddr t)))
        (else #f)))
```

The function returns true if the specified element is found in the tree, and returns false otherwise.

```
> (tree-search 101
       '(101 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
#t
> (tree-search 85
       '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
#f
```

Modify the function `tree-search` such that it returns the subtree routed at the element (i.e., the element will become the root).

```
> (tree-search2 101
       '(101 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
'(101 (83 () (97 () ())) ())

> (tree-search2 85
       '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
'()
```

Please see the next page!

```
package main

import "fmt"

type Reader struct {
     name string
     books []Book
}

type Book struct {
     title string
}

func main() {
     jack:= Reader{"Jack", make([]Book,0,5)}
     nBooks:= jack.Add(Book{"Lord of the rings"})
     fmt.Printf("Number of books read= %d\n", nBooks)
     nBooks= jack.Add(Book{"The name of the rose"})
     fmt.Printf("Number of books read= %d\n", nBooks)
     fmt.Println(jack.books)
}
```

Define a method `Add` that enables the addition of a book into the list of books to read. The program then should produce the following output:

```
Number of books read= 1
Number of books read= 2
[{Lord of the rings} {The name of the rose}]
```

Please see the next page!

Question 10 Go Routines and Channels [4 marks]

Below is a Go function which distributes a computation on the elements of an array by parallelizing the loop in two competing threads.

```
package main

import "fmt"

func calcul(tab []int, ch chan int) {
     for i, v := range tab {
          ch <- i + v
     }
     close(ch)
}

func main() {
     ch := make(chan int)
     x := []int{1, 2, 3, 4, 5, 6}
     go calcul(x[:3], ch)
     go calcul(x[3:], ch)
     for {
          val, ok := <-ch
          if !ok {
               break
          }
          fmt.Printf("%d ", val)
     }
}
```

0+4=4
1+5=6
2+6=8

0+1=1
1+2=3
3+2=5

Unfortunately, as written this program exits too early. For example:

Run 1:
```
4 6 8 Success: process exited with code 0.
```

Run 2:
```
4 6 1 3 5 Success: process exited with code 0.
```

Fix the problem by using two channels.

Please see the next page!

```
func calcul(tab []int, ch chan int) {
	for i, v := range tab {
		ch <- i + v
	}
	close(ch)
}

func main() {
	chA := make(chan int)
	chB := make(chan int)
	x := []int{1, 2, 3, 4, 5, 6}
	go calcul(x[:3], chA)
	go calcul(x[3:], chB)

	for {
		select { case valA, ok := <-chA :
			if ok { Printf("%d", valA)
		}
		case valB, ok := <-chB;
			if ok { Printf("%d", valB)
		}
	}
}
```