COP 3502 9/22/21 9^{30} am

1. Insert end LL
2. Insert in order LL
3. Delete
4. Circular
5. Doubly LL

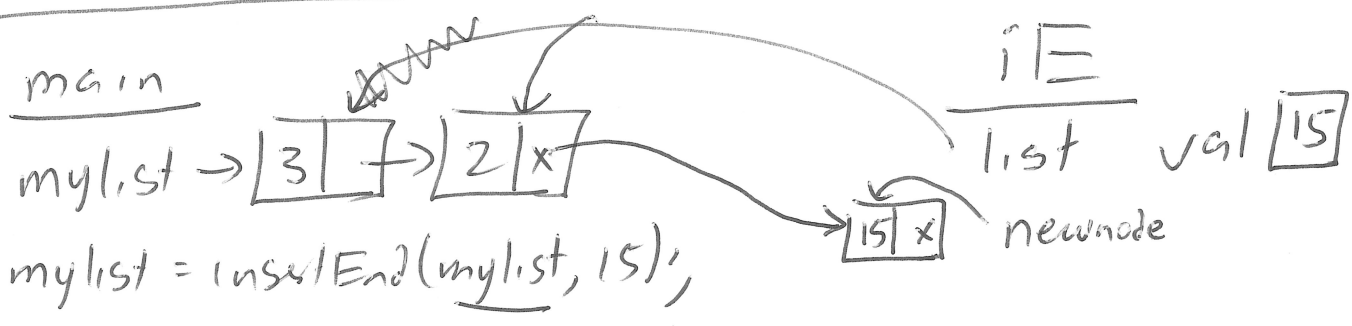
front / head (other names I use)

↑

```
node* insertEnd(node* list, int val) {
    node* newnode = createNode(val);
    if (list == NULL) return newnode;
    while (list->next != NULL)
        list = list->next
    list->next = newnode;
    return savefront;
}
```

→ `node* savefront = list;`

`x = y` // "make x point to what y is pointing to"



```
void insertEnd(node** ptrToList, int val) {
    node* newnode = createNode(val);
    if (*ptrToList == NULL){
    *ptrToList = newnode;
        return;
    }
}
```

main

~~mylist~~ | 0x2345

~~0x0000~~

mylist | 0x2345 | ~~0x0000~~ | 0x7777

~~0x4444 STRUCT~~

iE

ptrToList

newnode

| 3 | x |

0x7777

newnode | 0x7777

Insert In Order

savelist

newnode | 4 | x

main

mylist → | 2 | ~~0x0000~~ | → | 7 | → | 15 | x |

mylist = insertInOrder(mylist, 14, 4, 20)

Special case
NULL ret node
not NULL → attach newnode
rest list

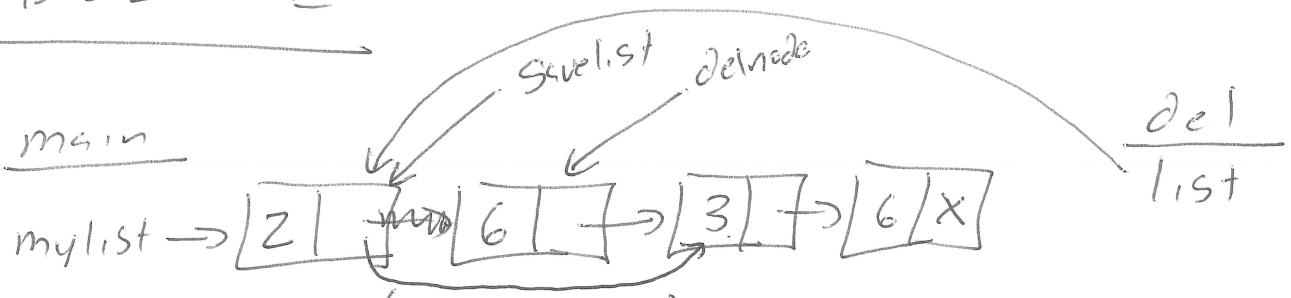
iIO

list

```
while ( list->next != NULL && val > list->next->data)
    list = list->next;
```

```
newnode->next = list->next;
list->next = newnode;
```

DELETE




```
mylist = del (mylist, 6);
```

(1) Iterate list to node before del value.

(2) `node* delnode = list->next;`

(3) `list->next = delnode->next;`

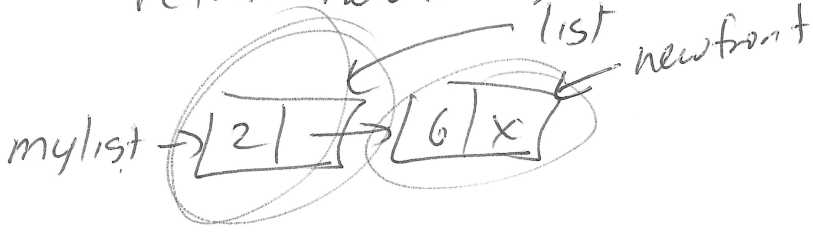
(4) `free (delnode);`

(5) `retrun savelist;`

In the case delnode is NOT in the front!

To DEL 1st node

```
node* newfront = list->next;
free (list);
return newfront;
```



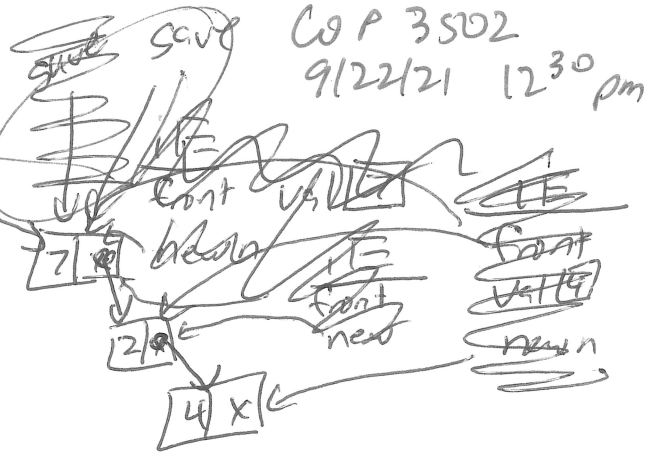
main

```
mylist = del (mylist, 2);
```

COP 3502
9/22/21 12:30 pm

① Insert End

main
mylist → X
mylist = insertEnd(mylist, 7);
mylist = insertEnd(mylist, 2);
mylist = insertEnd(mylist, 4);




```
node* insertEnd(node* front, int val) {
    node* newn = createNode(val);
    if (front == NULL) return newn;
    node* savelist = front;
    while (front->next != NULL)
        front = front->next;
    front->next = newn;
    return savelist;
}
```

---

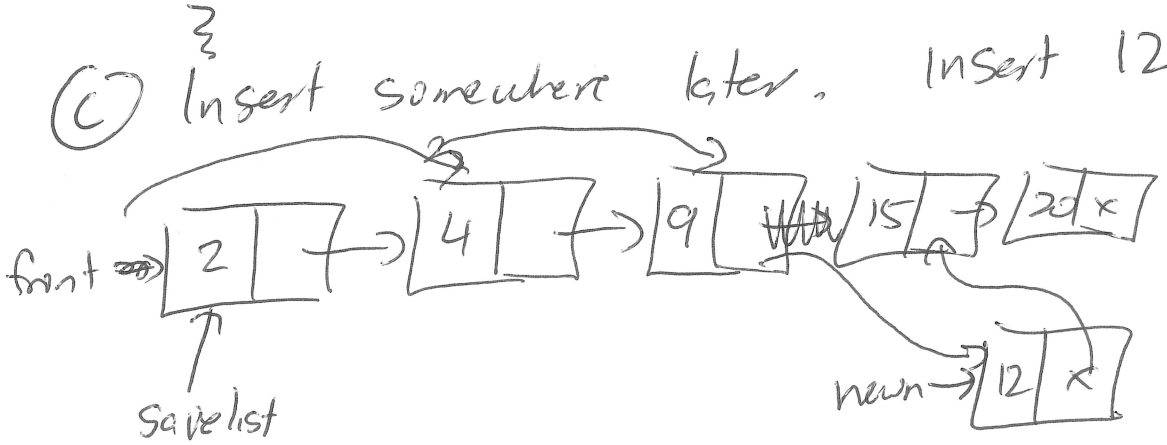
② Insert In Order

ⓐ NULL list → just return ptr new node.

ⓑ Insert in front
if (val < front->data) {

}

ⓒ Insert somewhere later. Insert 12



```
while (front->next != NULL &&
       val > front->next->data)
    front = front->next;
newn->next = front->next;
front->next = newn;
return savelist;
```

Delete

(1) take in ptr front list
(2) value to delete
(3) if value not in list, not change list, return ptr to front list.
(4) if it IS in list, delete the 1st occurrence of value + return the new front of the list.

In code

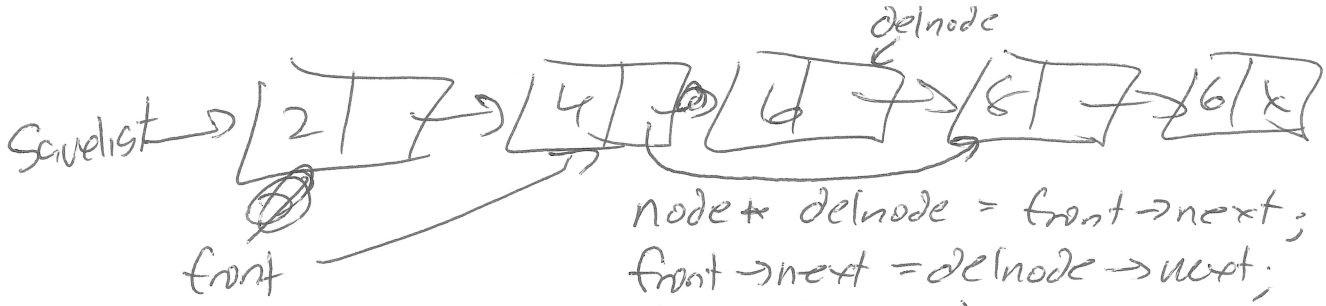
(a) NULL case - return NULL
(b) 1st node needs to be deleted


```
node* newfront = list->next;
free(list);
return newfront;
```

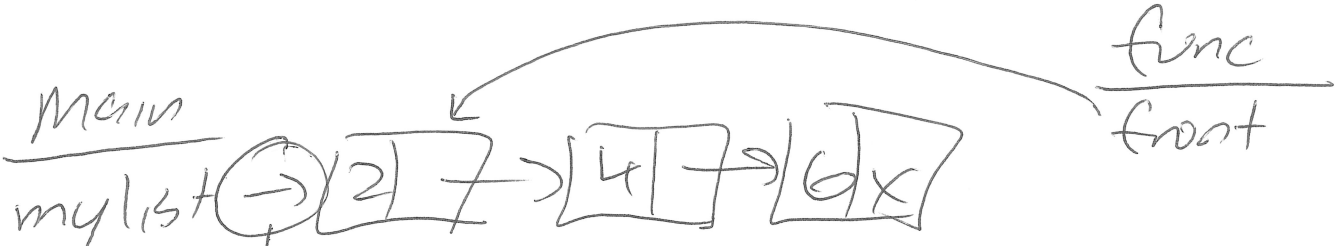
(c) either not in list, 1st occurrence isn't in the front

Iterate to the node right before the node to delete.

```
while (front->next != NULL && front->next->data
       != delval)
    front = front->next
```




```
node* delnode = front->next;
front->next = delnode->next;
free(delnode);
return savelist;
```



front CAN NOT CHANGE

WHERE this arrow points!

(that's why these functions return a ptr to the front of the list.)

Some things we can do

* (1) Reverse LL (recursive soln recommendation)

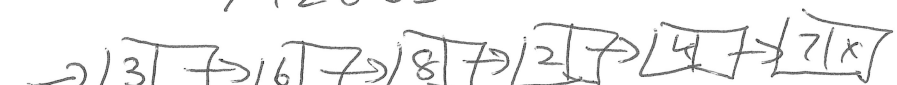
(2) Write functions manipulate list
- a) structural (move front to end)
- b) change values (double each value)

(3) Store strings

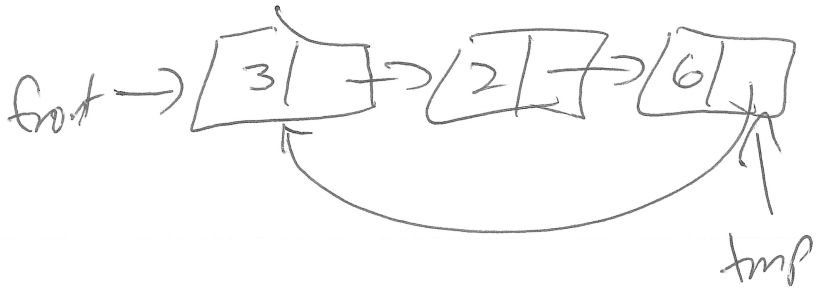
write strlen, strcat, strcpy, strcmp, etc.

(4) Store big int 742863

ADD, SUBTRACT, ...



Circular LL



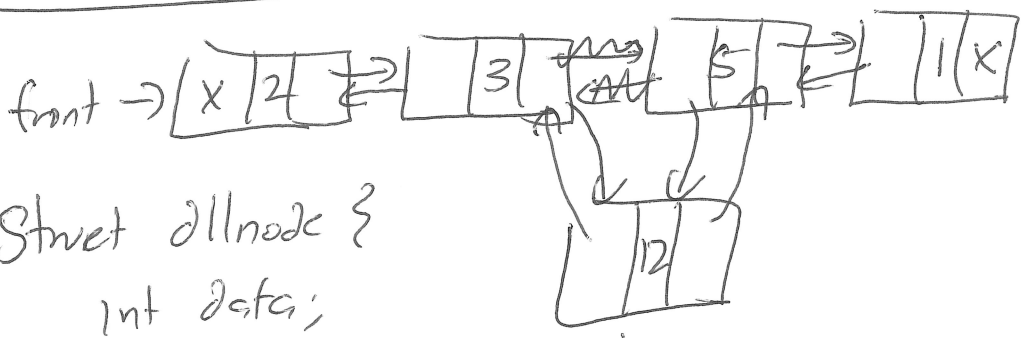
KEY ISSUES

Lots of stuff same
How do I detect the end?

```
if (tmp->next == front)
// I'm at last node
```

* Insert front is annoying because you have to change the last node.

Doubly LL




```
typedef Struct dllnode {
    int data;
    struct dllnode* prev;
    struct dllnode* next;
} dllnode;
```

move 4 ptrs NOT 2!

KEY ISSUES

Can go backwards BUT I have to repatch twice as many ptrs.

Example CD.c

Good example to make sure you understand list syntax