Exam Stuff

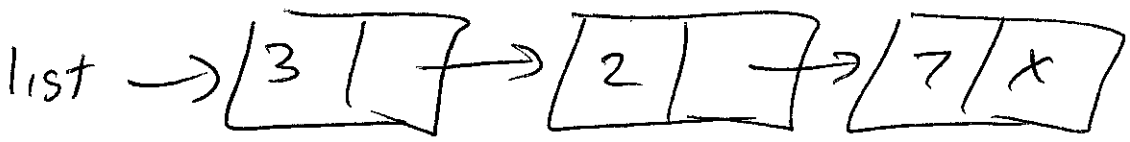
Spr16 = 60% Spr17 = 47% ???

(1) ADVICE: WRITE 3 PRACTICE PROGRAMS A WEEK!

(2) COME TO CLASS ALL THE TIME (MINUS EMERGENCIES)

2/21/13

Linked List




```
struct node {
    int data;
    struct node* next;
};
```

(1) Add to ~~Front~~ Front
(2) Add to Back
(3) Print
(4) Add in order
(5) Sum all values
(6) Count freq a particular #
(7) Check if sorted

(8) Delete
(9) Reverse

Structural Changes (harder)

Non-structural (easier)

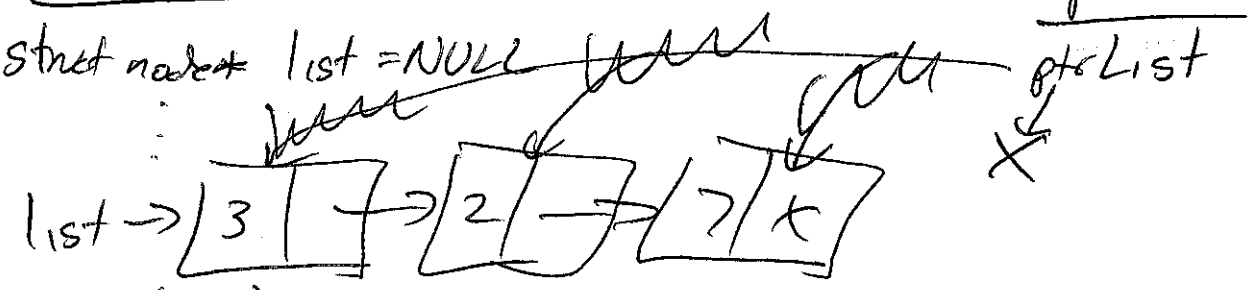
Print

main

```
struct node* list = NULL;
```

print

ptrList




```
print(list);
```

3 2 7

```
void print(struct node* ptrList) {
    while (ptrList != NULL) {
        printf("%d ", ptrList->data);   // (*ptrList).data
        ptrList = ptrList->next;
    }
}
void printRec(struct node* ptr) {
    if (ptr != NULL) {
        printf("%d ", ptr->data);
        printRec(ptr->next);
    }
}
int sum(struct node* ptr) {
    if (ptr == NULL) return 0;
    return ptr->data + sum(ptr->next);
}
int freq(struct node* ptr, int value) {
    int res = 0;
    while (ptr != NULL) {
        if (ptr->data == value) res++;
        ptr = ptr->next;
    }
}
```

printRec(3 2 7)
3
printRec(2 7)
2
printRec(7)
7
printRec(x)

```
int isSorted(struct node* ptr) {
    if (ptr == NULL) return 1;
    while (ptr->next != NULL) {
        if (ptr->data > ptr->next->data)
            return 0;
        ptr = ptr->next;
    }
    return 1;
}
```

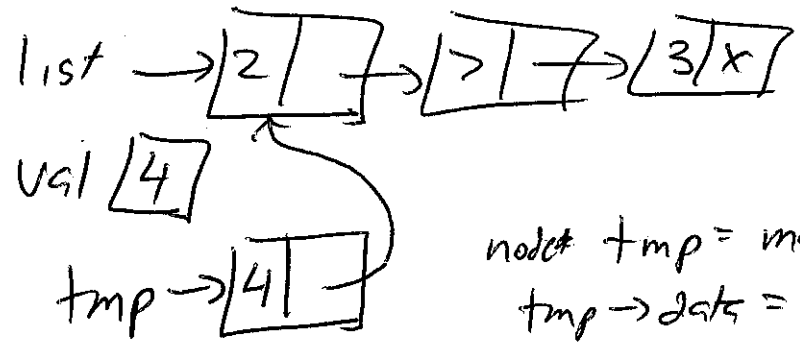
// NULL ptr error is NULL → something

// In our class
for (int i=0; i<n; i++)
occasionally I do
for (int i=0; i<n-1; i++)

2/21/17 (L)

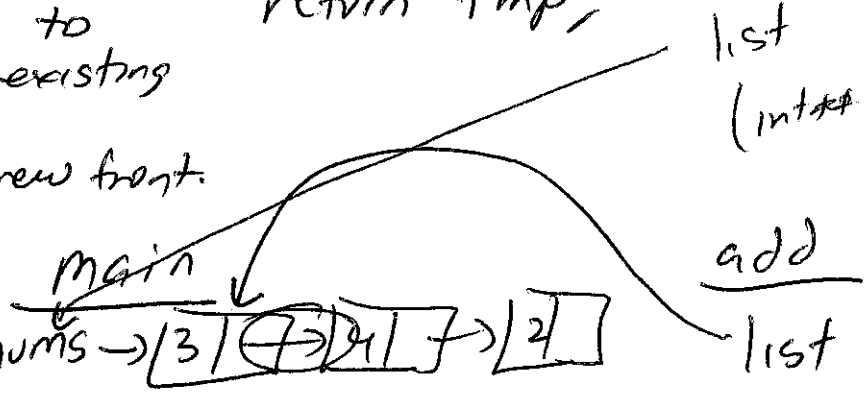
Linked Lists

1) add to front
2) add to back
3) add in order
4) delete
5) reverse
6) circular LLs
7) doubly LLs



1. create new node
2. copy val into it
3. link its next to front of the existing list.
4. return ptr to new front.

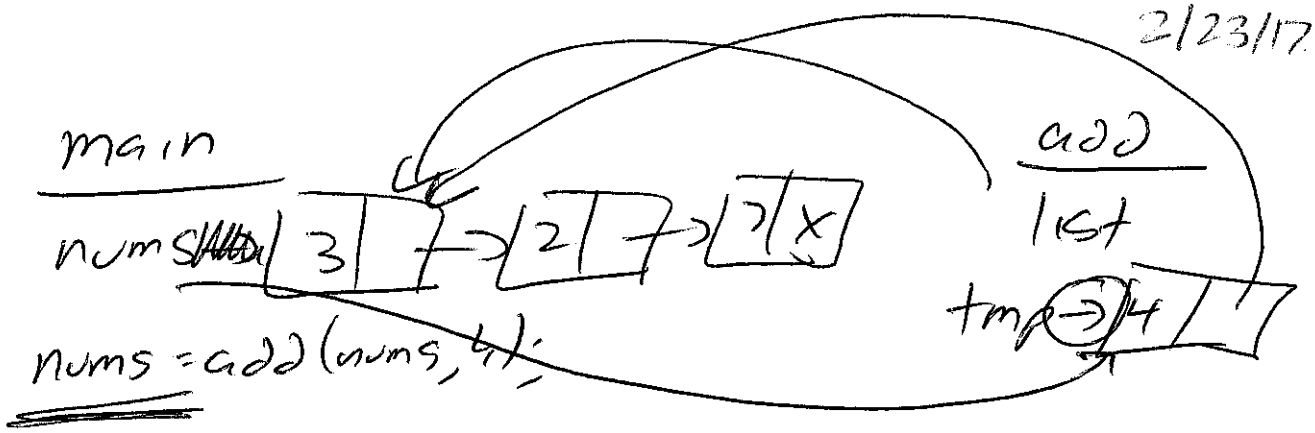
```
node* tmp = malloc(sizeof(node));
tmp->data = val;
tmp->next = list;
return tmp;
```

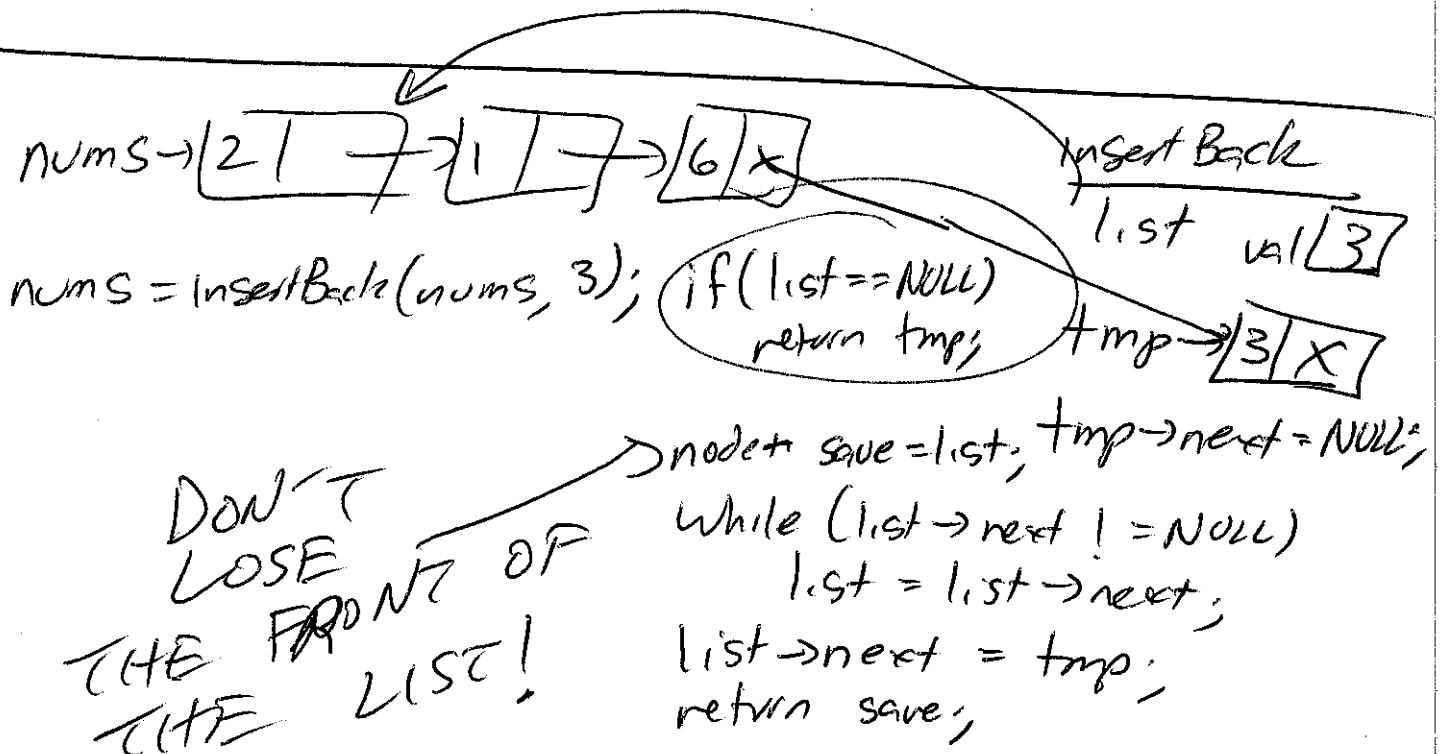


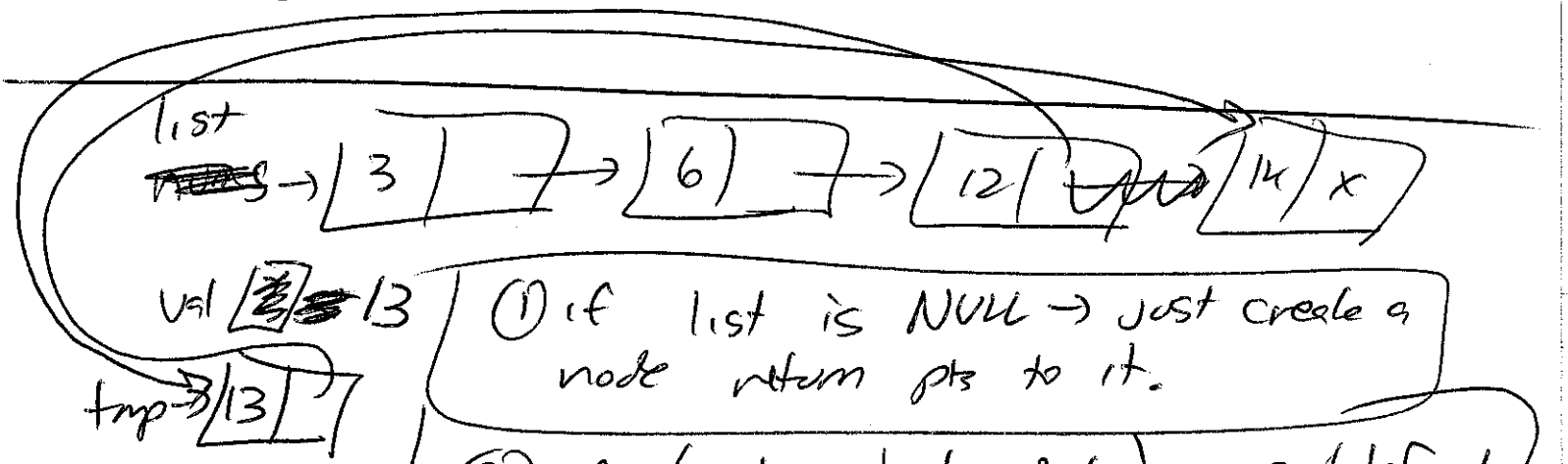

```
node* nums = NULL;
nums = add(nums, 4);
```

CAN the add function MOVE the pointer nums? NO

list is limited to changing values in nodes + changing next ptrs in nodes.







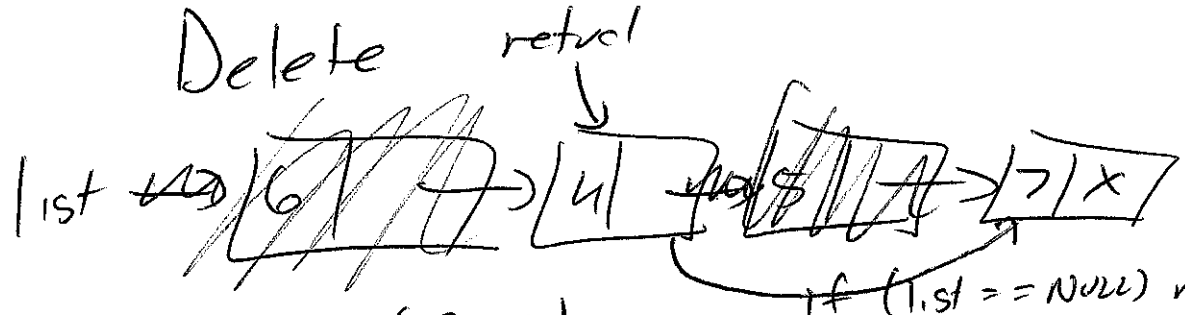
① if list is NULL → just create a node return ptr to it.

② if (val < list->data) insert to front

③ node* iter = list; while (iter->next != NULL && val > iter->next->data)
iter = iter->next;
tmp->next = iter->next;
iter->next = tmp;

works for all inserts except to the front!

Delete



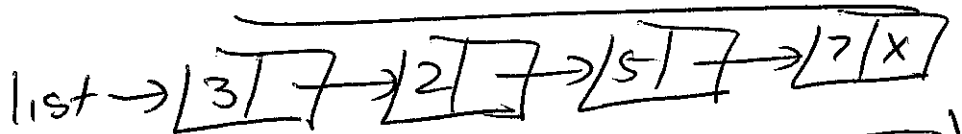
① delete 6 (front)

nums = delete(nums, 6);

② delete 5 (middle/end)

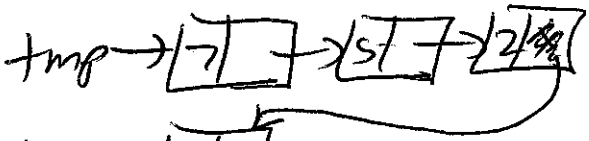
```
if (list == NULL) return NULL;
if (delval == list->data) {
    node* retval = list->next;
    free(list);
    return retval;
}
node* iter = list;
while (iter->next != NULL &&
       delval != iter->next->data)
    iter = iter->next;
node* freeptr = iter->next;
iter->next = freeptr->next;
free(freeptr);
return list;
```

Linked List Reverse



recursively (3 X , (2 → 5 → 7 X))

rev



① reverse rest list

② iterate to end of new list

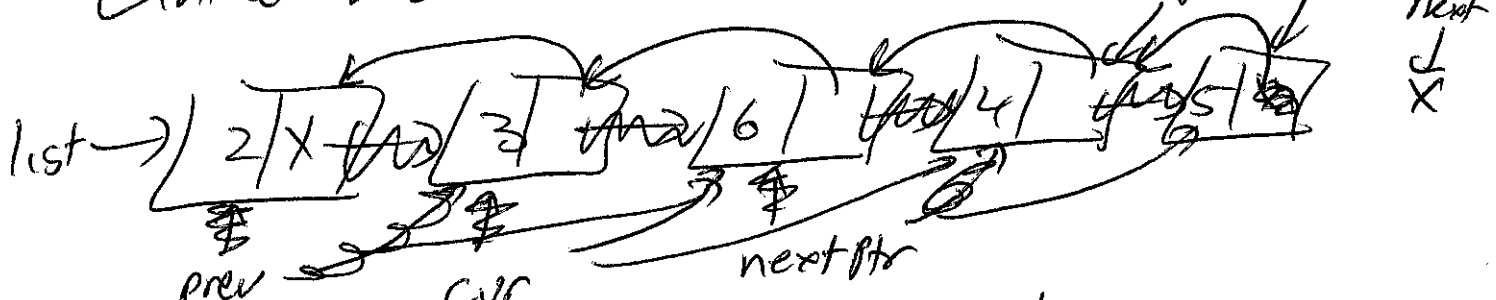
③ ~~add~~ attach end of reversed list to 1st item in original list

④ return new front

link 2 → 3

return ~~temp~~ tmp

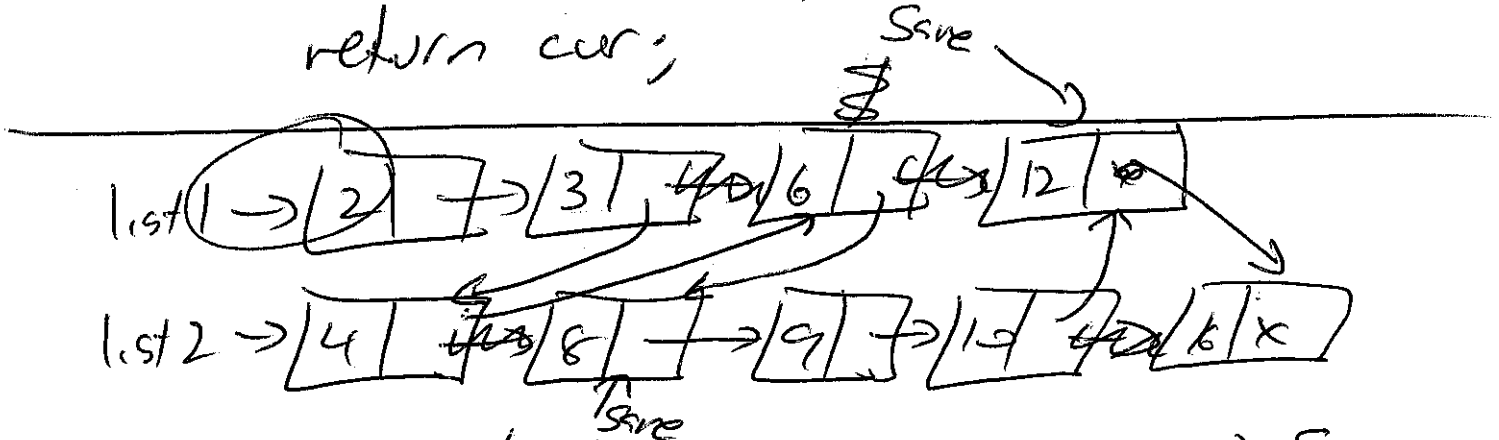
Linked List Reverse Iterative O(n)



while (nextPtr != NULL) { in a loop

```
cur->next = prev;
prev = cur;
cur = nextPtr;
nextPtr = nextPtr->next;
}
```

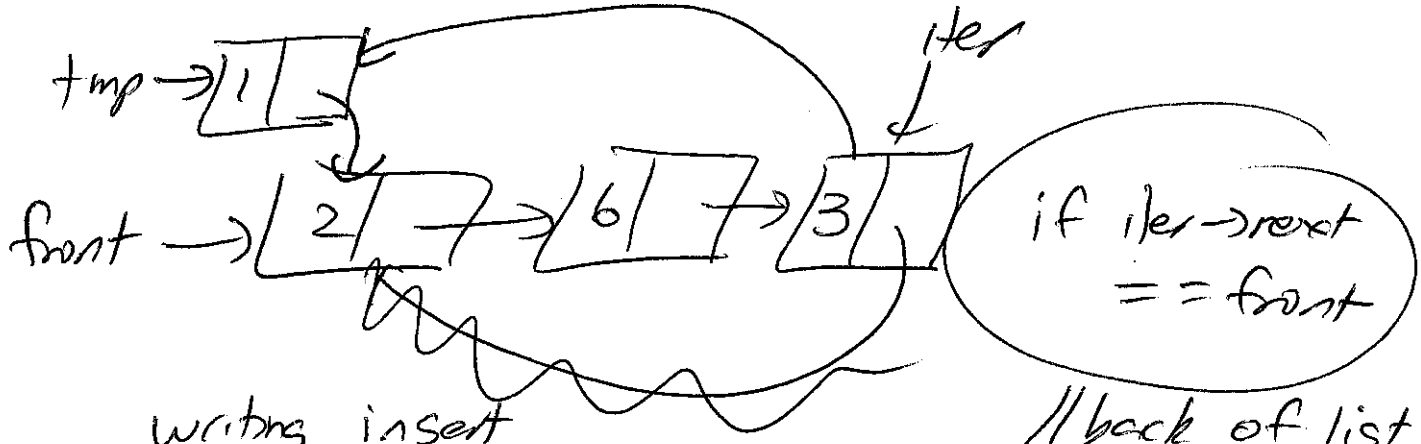
```
cur->next = prev;
return cur;
```




```
node* merge(node* list1, node* list2) {
```

Linked List Variants

(1) Circular LL

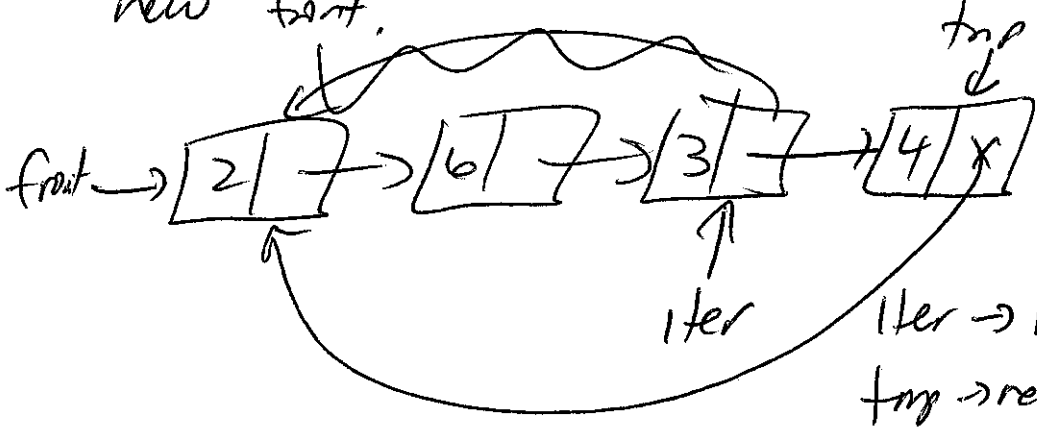


writing insert

detecting end of the list

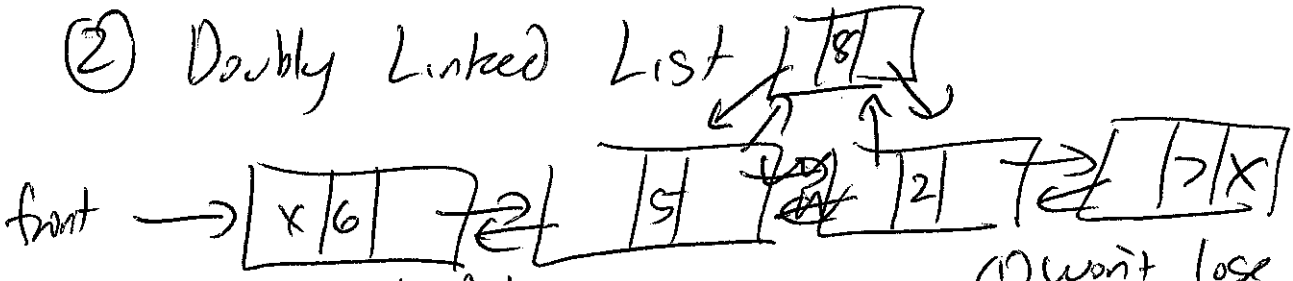
If inserting into the front or the end some changes have to be made

→ iter to end list, repatch the end to the new front.



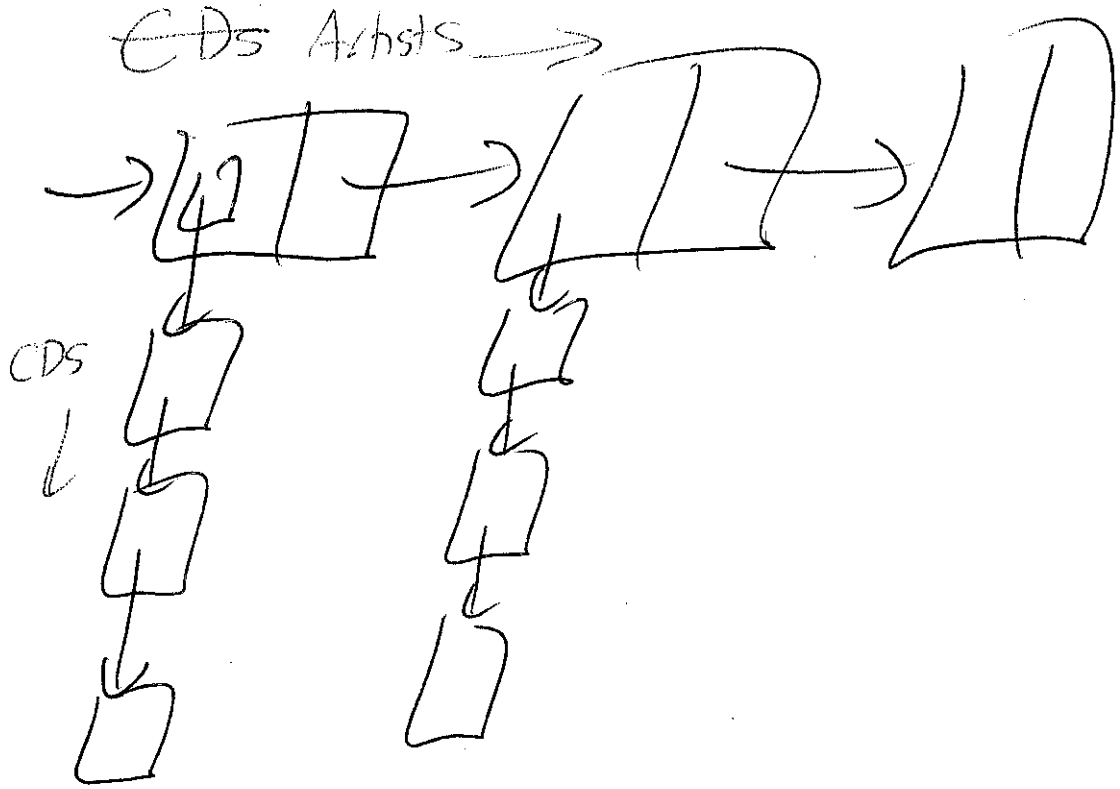

```
iter -> next = tmp;
tmp -> next = front;
return front;
```

(2) Doubly Linked List




```
int data;
struct node* prev;
struct node* next;
```

(1) won't lose list
(2) annoying to maintain more ptrs


~~CDs~~ Artists
CDs