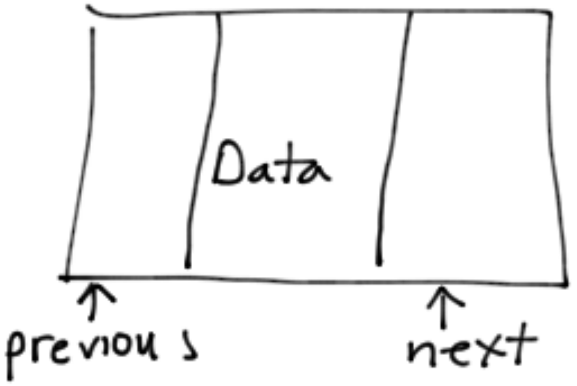
Lecture 6

1. Topic #5 - Other Linked Lists
2. Look ahead vs previous pointer
3. DLL, CLL efficiency
4. Begin Practicing Recursion!

DLL (Doubly Linked List)



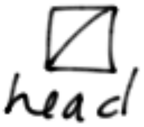
```
Struct node
{
      student peer;
      node *previous;
      node * next;
};
```

```
// first node
if (!head) //empty
{
      head = new node;
      head->peer.set(to_add);
      head->previous = NULL; ← extra (not done
      head->next = NULL;       in a LLL)
      tail = head; ← common for DLL
}
```

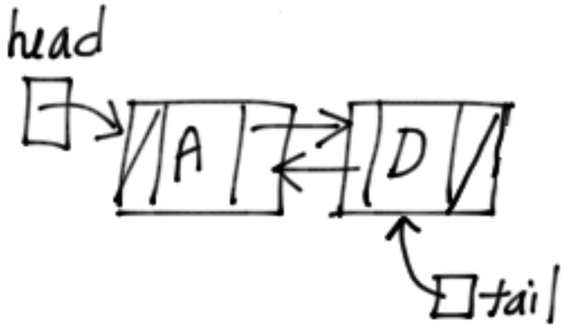
#extra operations & fetches?

— DLL —

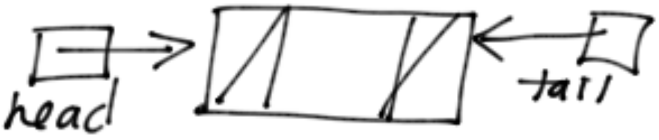
Before



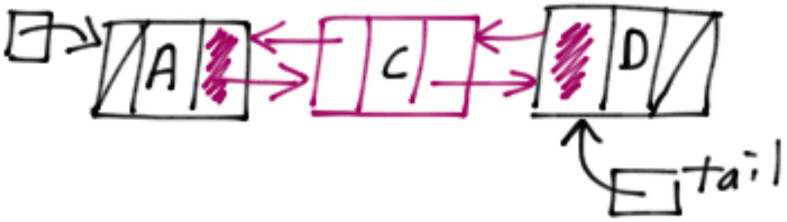
(empty)



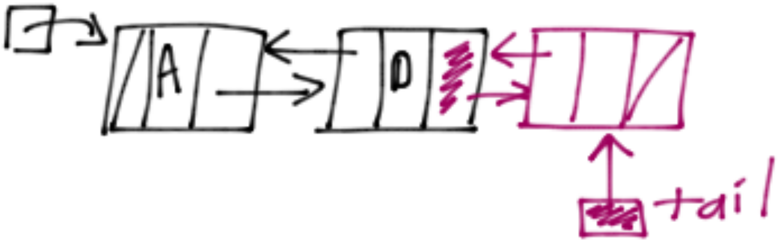
After

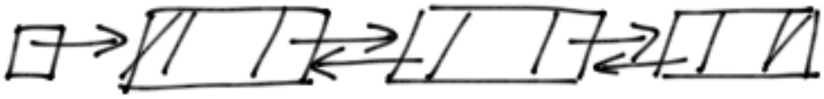


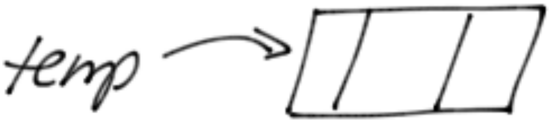
insert C (middle)



insert Z (end-special case)



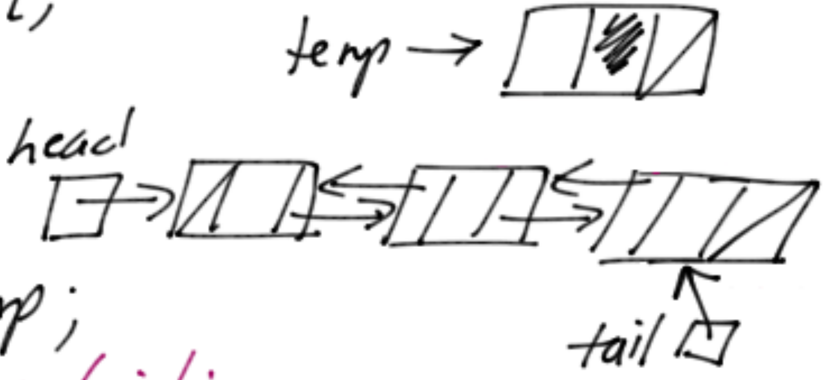




```
① current->previous->next = temp
② temp->previous = current->previous;
③ temp->next = current;
④ current->previous = temp;
```

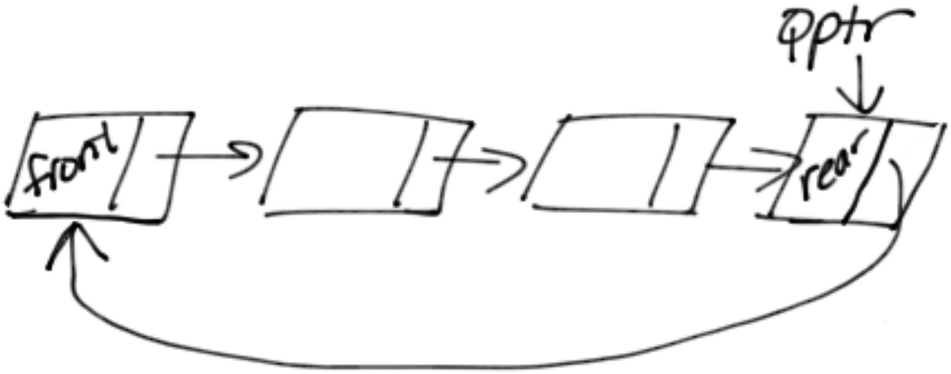
With a tail pointer — adding at end

```
node * temp = new node;
//save the data
temp->next = NULL;
```



```
① tail->next = temp;
② temp->previous = tail;
③ tail = temp;
```

** Non empty list !!!



① temp = qptr->next;

② qptr->next = NULL;

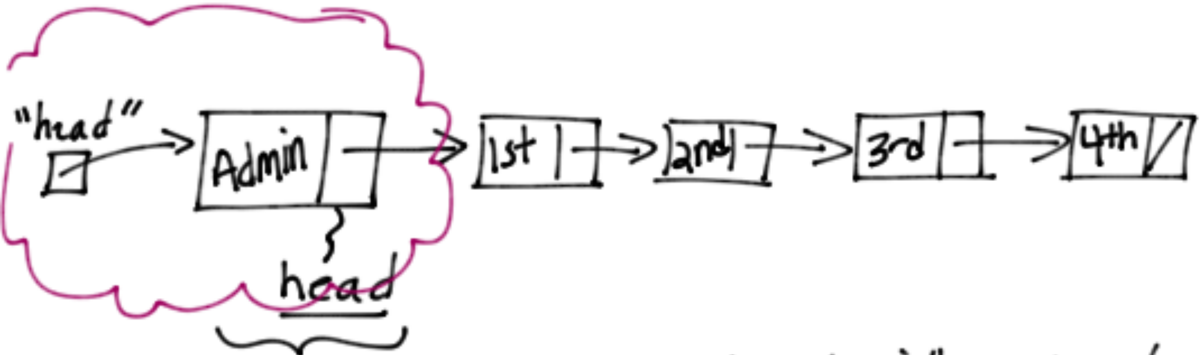
Traversal Efficiency

```
 node * current = head;
if ( head)
{    while ( current -> next)              (*current). next
               current = current -> next;
          total of 9op/fetches * 10,000
       current -> next = ____________
```

If tail pointer

```
    tail -> next = ____________
    tail = tail -> next;
                (temp)
```

Dummy head node



always exists — so head will never be NULL

Without "dummy" head node

```
// add at end
if (!head)
{   head = new node;
        :
}
else {
   node * current = head;
   while (current→next)
          current = current→next;
   current→next = new node;
      :

}
```

With

```
// skip the simple case ....... unless
// the client calls the destructor
// explicitly and causes the dummy
// node to be deallocated.
  if (!head)
         return Ø; //failure
              OR do we set up the
              dummy head node &
              the first node?!?!

  else
  {
         // same

}
```

Review Recursion

1) Always have an if statement for the stopping condition

2) Part of the function call will get us to the next step

3) Start with the simple case

```
// Add at the end
int list::Append(node *& head, data & adding)
{
    // simple case First
    if(!head)
    {   head = new node;
        head->next = NULL;
        head->data.set(adding);
    }
    else  // Now "traverse" via a function call
            return Append(head->next, adding);
} return 1;
```

because I might need to modify head <u>AND</u> it is how we will connect the nodes