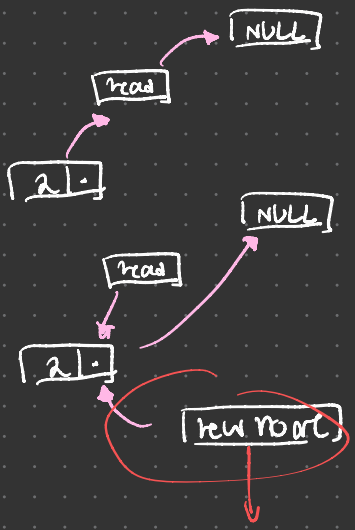
```
class LinkedList{
    Node *head = nullptr; //this variable is private
    //size, omitted
```


```
myList.insert( value: 2,  position: 0);
```

& calls this function

```
void LinkedList::insert(DataType value, int position) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. create new node
    Node *newNode = new Node(value);
    //step 3. if special case, when position = 0
    //insert as the new head
    //3.1 when head == nullptr, or size == 0
    if(position == 0){
        if(head == nullptr){
            head = newNode;
            return;
        }
        else{ //3.2 when size > 0
            newNode->next = head;
            head = newNode;
            return;
        }
    }
```

pointer node

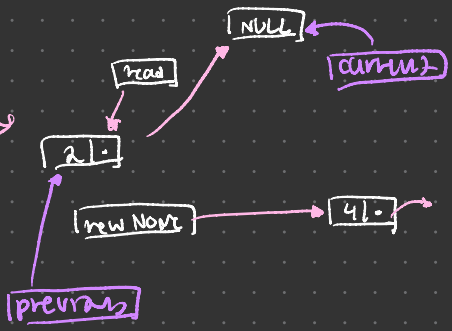


disappears bc its a local variable, disappears after a function

```
myList.insert( value: 4,  position: 1);
```

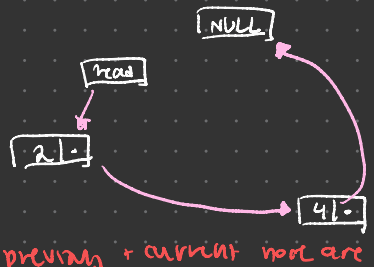
```
void LinkedList::insert(DataType value, int position) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. create new node
    Node *newNode = new Node(value); //* new node is a pointer (left-side), right-side is the node

    //step 3. if special case, when position = 0
    //insert as the new head
    //3.1 when head == nullptr, or size == 0
    if(position == 0){
        if(head == nullptr){
            head = newNode;
            return;
        }
        else{ //3.2 when size > 0
            newNode->next = head;
            head = newNode;
            return;
        }
    }
```

→ step is skipped




```
//step 4. when position > 0, size > 0
//iterate through the list to find the position
//with an additional pointer for the address of the previous node
Node *previous = head;
Node *current = head->next;
int i = 1;
while (i < position) {
    previous = current;
    current = current->next;
    if (current == nullptr) {
        break; // reach the end
    }
    i++;
}
//step 5. insert the node between previous and current
newNode->next = current;
previous->next = newNode;
```

skipped se

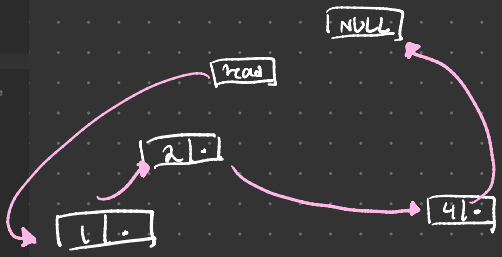


previous + current node are skipped

```
myList.insert( value: 1,  position: 0);
```

```
void LinkedList::insert(DataType value, int position) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. create new node
    Node *newNode = new Node(value); //* new node is a pointer (left-side), right-side is the node

    //step 3. if special case, when position = 0
    //insert as the new head
    //3.1 when head == nullptr, or size == 0
    if(position == 0){
        if(head == nullptr){
            head = newNode;
            return;
        }
        else{ //3.2 when size > 0
            newNode->next = head;
            head = newNode;
            return;
        }
    }
```

this line will run bc head is not empty



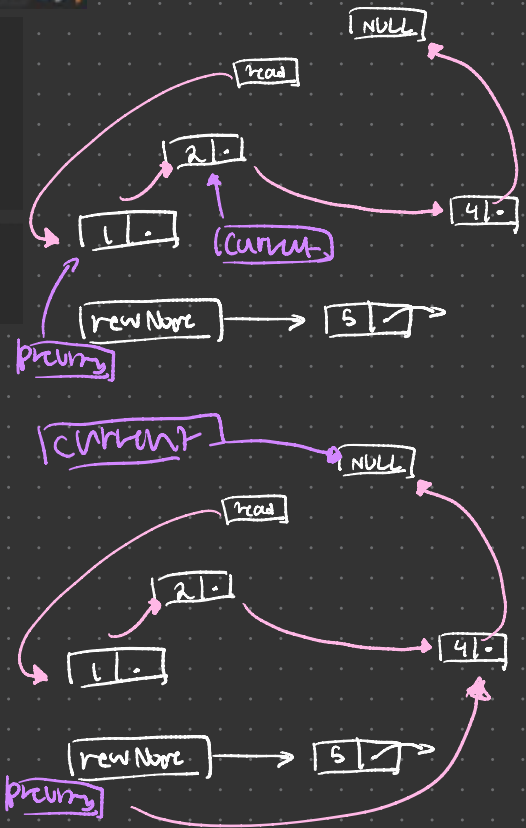

```
myList.insert( value: 5,  position: 3);
```

```
void LinkedList::insert(DataType value, int position) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. create new node
    Node *newNode = new Node(value); //* new node is a pointer (left-side), right-side is the node

    //step 3. if special case, when position = 0
    //insert as the new head
    //3.1 when head == nullptr, or size == 0
    if(position == 0){
        if(head == nullptr){
            head = newNode;
            return;
        }
        else{ //3.2 when size > 0
            newNode->next = head;
            head = newNode;
            return;
        }
    }
```

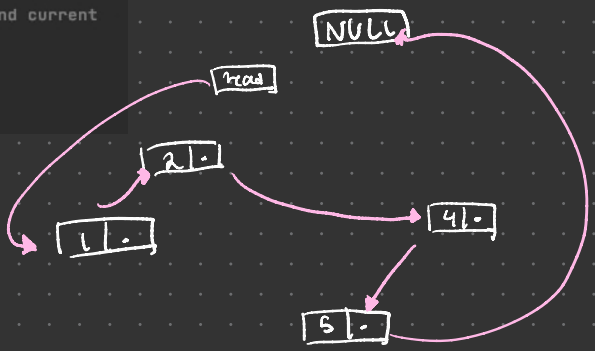
does not run

```
    //step 4. when position > 0, size > 0
    //iterate through the list to find the position
    //with an additional pointer for the address of the previous node
    Node *previous = head; //need to know address of previous node
    Node *current = head->next;
```

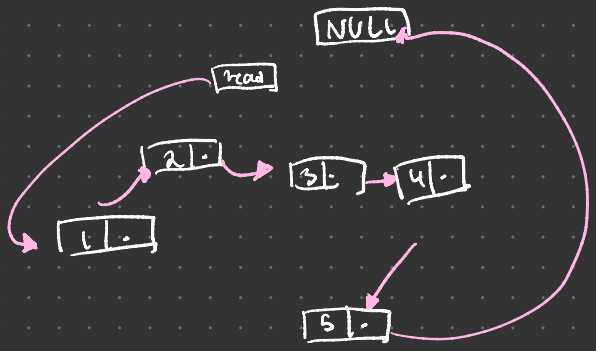
```
int i = 1;
while (i < position) {
    previous = current;
    current = current->next;
    if (current == nullptr) {
        break; // reach the end
    }
    i++;
}
```



```
//step 5. insert the node between previous and current
newNode->next = current;
previous->next = newNode;
```

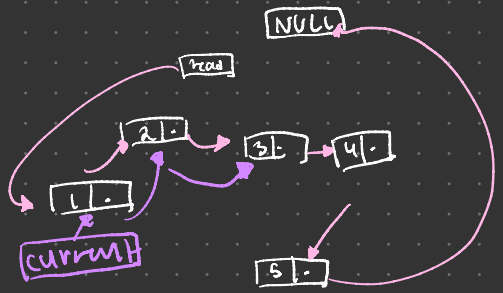


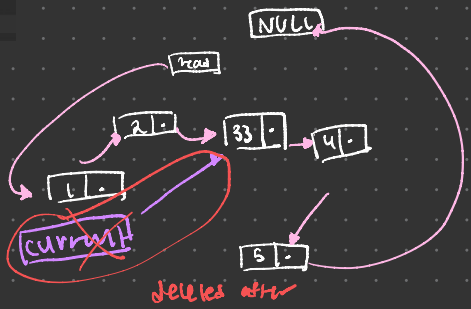

```
myList.insert( value: 3,  position: 2);
```




```
myList.replace( position: 2,  value: 33);
```

```
void LinkedList::replace(int position, DataType value) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. iterate a pointer to the position
    Node *current = head;
    int i = 0;
    while (i < position) {
        current = current->next;
        if (current == nullptr) {
            break; // reach the end
        }
        i++;
    }

    //step 3. replace the value
    current->data = value;
}
```

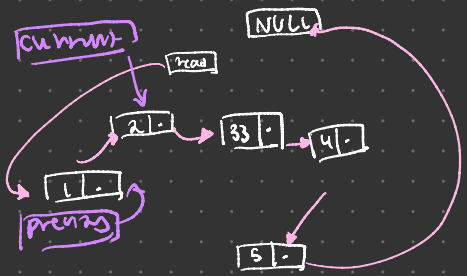




```
myList.remove( position: 3);
```

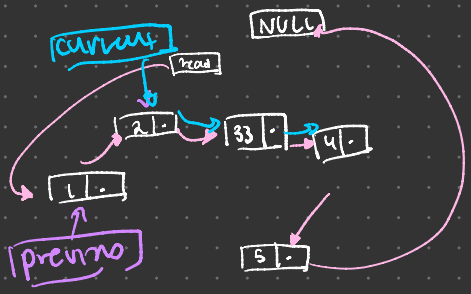
```
void LinkedList::remove(int position) {
    //step 1. check position validity, compare with 0 and size,
    //if not valid, quit, report error, or force into a valid value
    //omitted in this demo, assuming it is valid

    //step 2. special case, if position == 0, remove head
    if(position == 0){
        Node *temp = head;
        head = head->next;
        delete temp;
        temp = nullptr;
        return;
    }
```

skipped



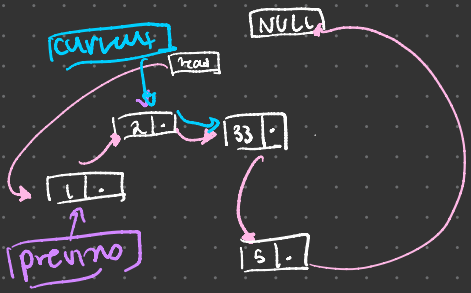

```
//step 3. iterate a pointer to the position
//and another pointer for the node address previous of it
Node *previous = head;
Node *current = head->next;
```

```
while (i < position) {
    previous = current;
    current = current->next;
    if (current == nullptr) {
        break; // reach the end
    }
    i++;
}
```

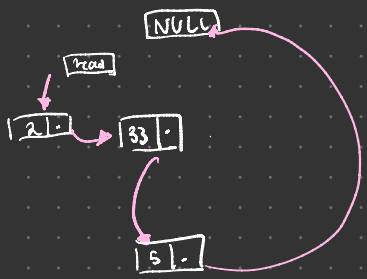



```
//step 4. remove the node and properly link the remaining nodes.
previous->next = current->next;
```

```
delete current;
current = nullptr;
```




```
myList.remove( position: 0);
```



FW of the same