VECTORS :

→ Dynamic array where full time size is not fixed when array is created

Data structures → mechanism to store to process data

→ push data (add data to the Array)

→ Pop (Removing the last element in array)

→ peek (Tell us the element at the end of list)

Push function in the VECTOR

(Pseudocode)

OUTCOME "Push(VECTOR* student, int item)

If array is full:

→ Created a temp array & used dynamic allocation to allocate memory

- Tested is allocation was good

→ We copied data from old array to new array

→ delete old array

→ add item to the created array

Analyse what is going with Copying

Copy 1 item take 1 unit of time

if n item in old array, total time taken is n


```
for (int i = 0; i < total; i++) {
    newArray[i] = oldArray[i]
}
```

Push function

$f(n) = O(n)$

With Creating new Arry

→ We are doubling the capacity of array.

Example oldArry 6 integers (4 bytes integer)

space in bytes = 24 bytes.

Double capacity is? 48 bytes

if we want to add only one item (20 bytes not used)

48 bytes (old arrays)

Double memory will 96 bytes (new arr)

empty storge

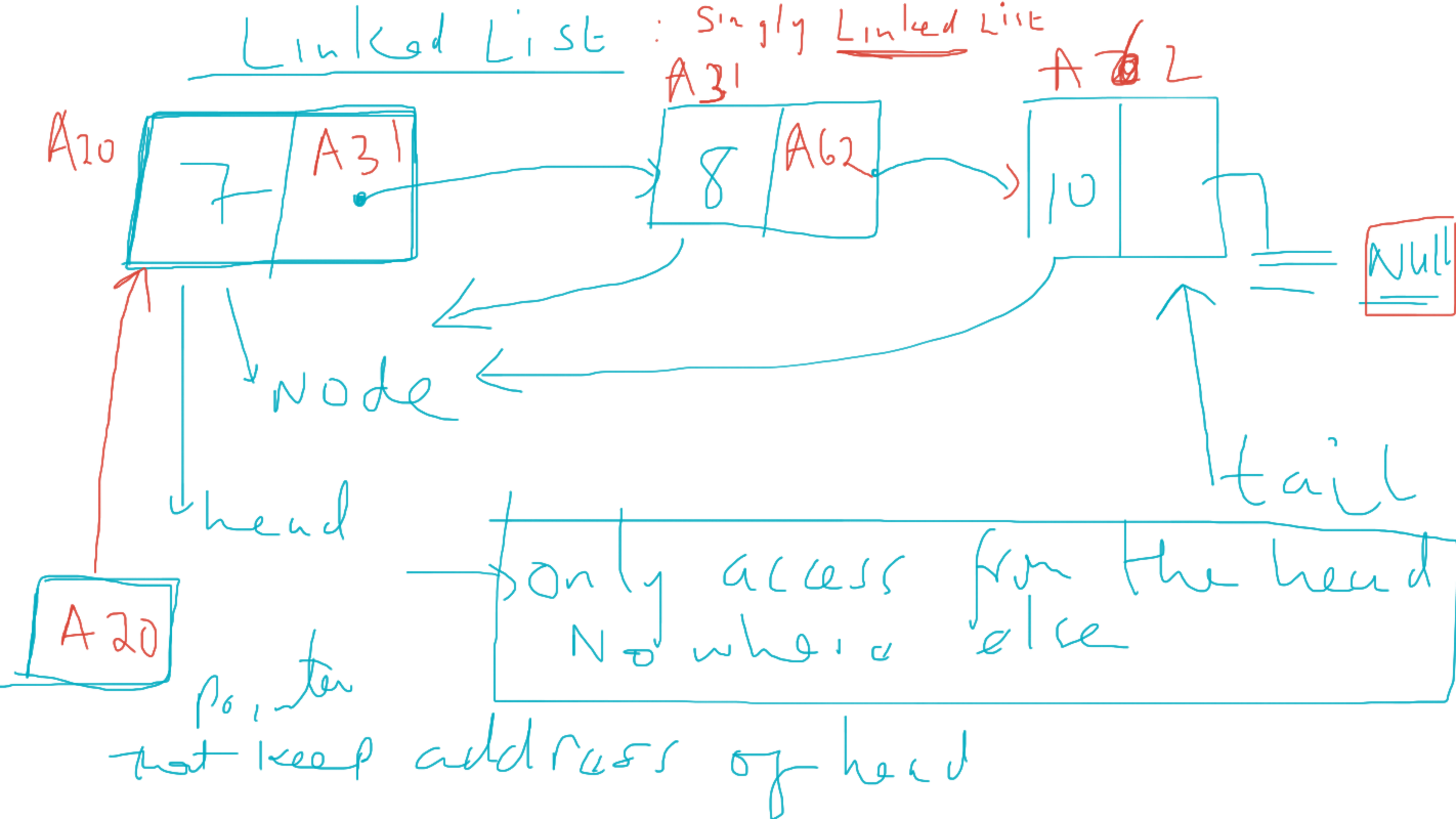
after adding one item } 144 bytes

-1s

What can we do to still store sequence of items without wasting storage

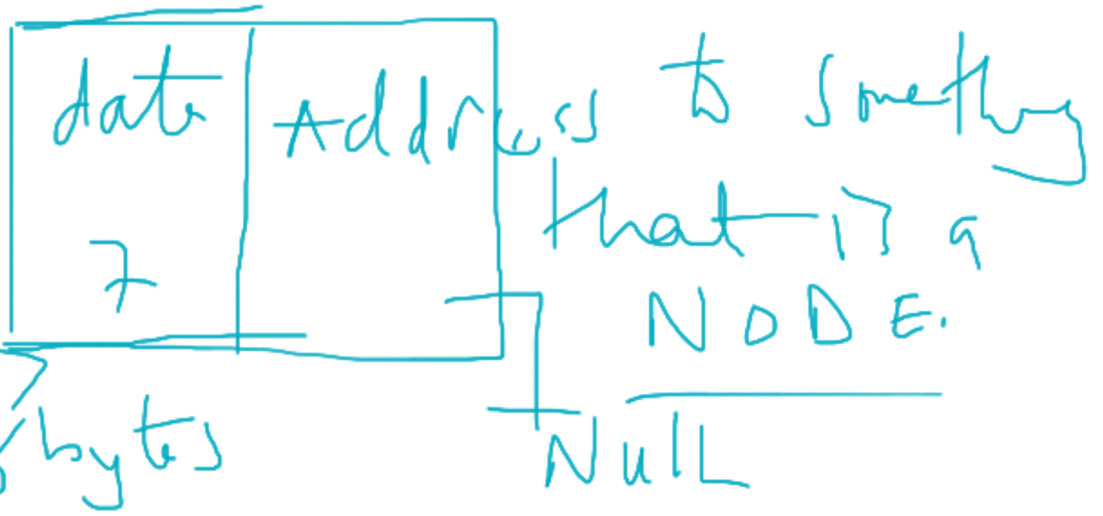
```
capacity = 10;
int *array = (int*) malloc(sizeof(int) * capacity);
```

Linked List : Singly Linked List



typedef struct node NODE; // forward declaration

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



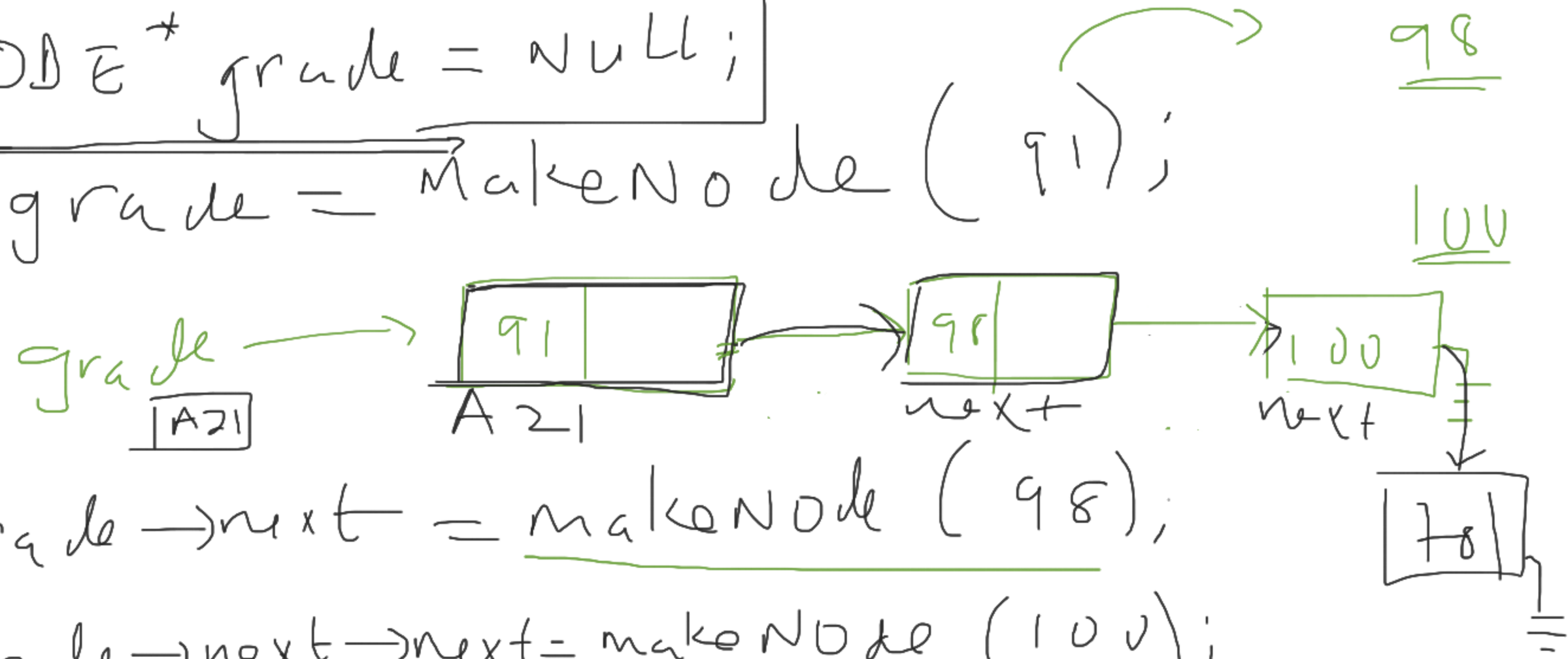

```
NODE* new = (NODE*) malloc (sizeof(NODE));
new -> data = 7;
new -> next = NULL;
```

```
NODE* MakeNode(int item) {
  NODE* new = (NODE*) malloc(sizeof(NODE));
  if (new == NULL) {
      return NULL;
  }
  new -> data = item;
  new -> next = NULL;

  return new;
}
```

NODE * grade = NULL;

grade = MakeNode(91);

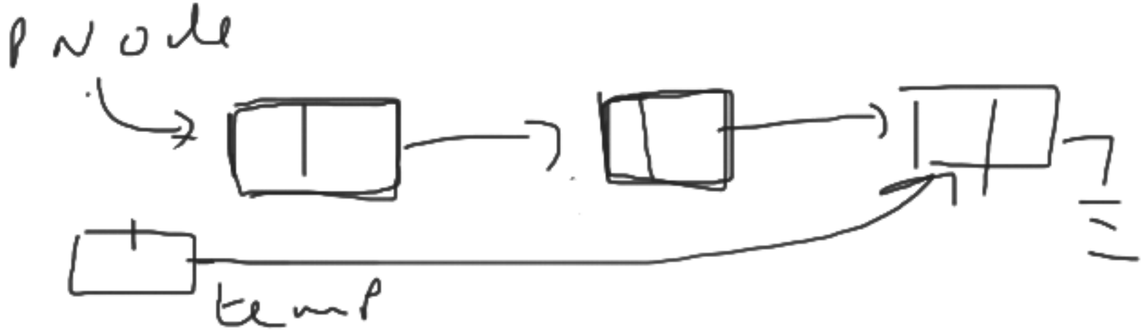


grade->next = makeNode(98);

grade->next->next = makeNode(100);

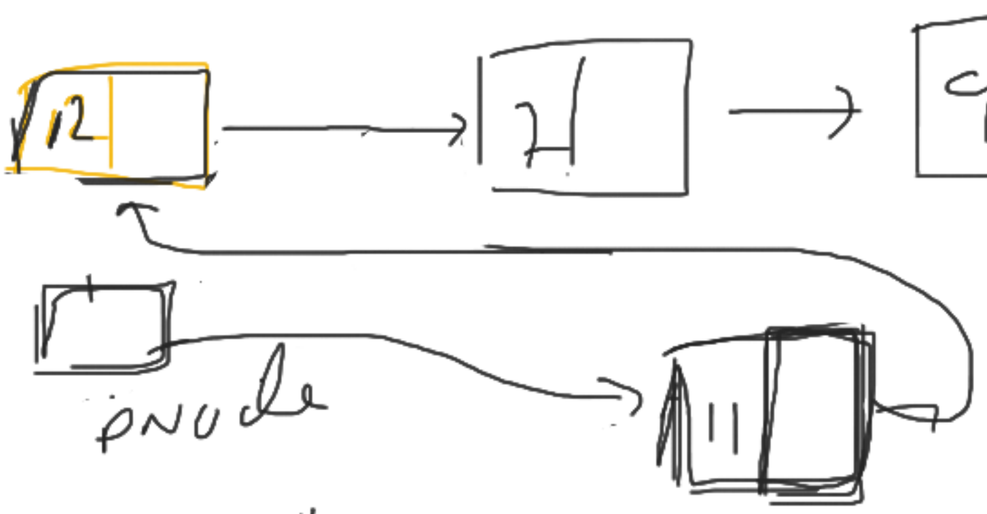
grade->next->next->next = makeNode(78);

```
void InsertTail (Node* pNode, int item) {
    Node * temp = pNode;
    if temp == NULL {
        pNode = makeNode(item);
        return;
    }
    while ( temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = makeNode(item);
    return;
}
```



O(n)

void InsertHead(NODE * pNode, int item){



⟹ Always Insert at the beginning

NODE * newNode = MakeNODE(item);

If (pNode == NULL){

pNode = newNode

return

}

newNode → next = pNode;

pNode = newNode;

} return;

} O(1)

How do we free all memory used by Linked list