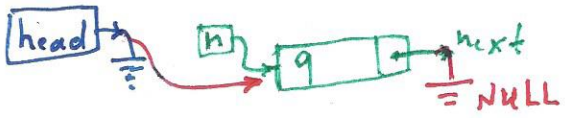
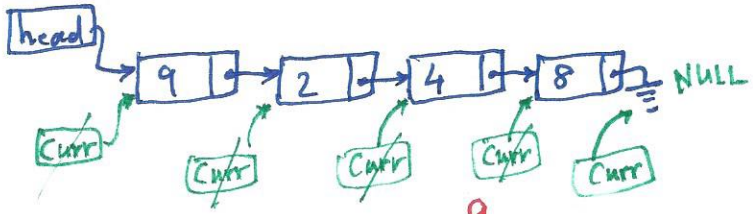
```
    { head=n;
      n->next=NULL;
    }
  }//end addNode
```



To print the Contents of a linked list (traversing a linked list):

```
Void display()
{ Node *Curr;
  Curr=head;
  While (Curr!=NULL)
  { Cout << Curr->data <<endl;
    Curr=Curr->next;
  }
}//end display
```



9
2
4
8

Week 6, Sat:

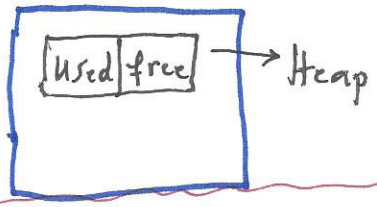
Heap: At runtime every program maintain an area of Memory known as **heap.**

Dynamic Data structures such as a linked list inhabit in heap.

Heap is an area of allocated memory (e.g. add a Node).

" " " " " free " When we delete a Node.

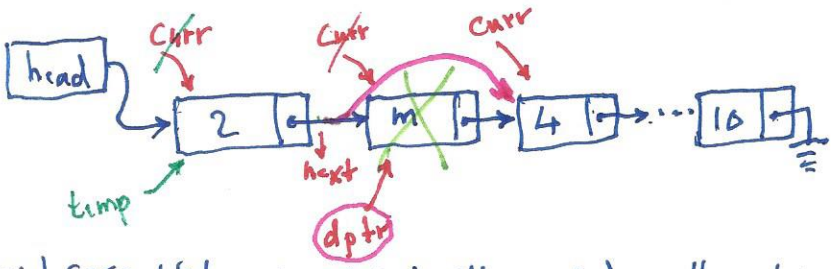
"RAM" →

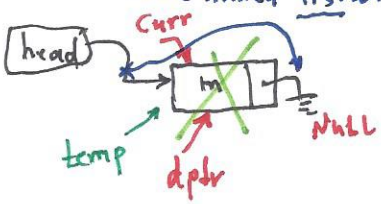


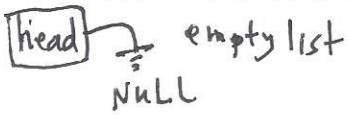
To delete a Node:

```
void List:: DeleteNode(int m)
{ Node *dptr = nullptr; //or NULL → address 0
  Node *temp=head;
  Node *Curr; → Curr=head;
  While (Curr!=NULL && Curr->data !=m)
  { temp = Curr;
    Curr = curr->next;
  }
  if (curr==NULL)
```

```
{ Cout<<m<<" not in list \n";
  delete dptr;
}
else { dptr=Curr;
       Curr=Curr->next;
       temp->next=Curr;
       if (dptr==head) // Special case, We have linked list with one Node or the node is the first Node in the list.
       { head=head->next;
         temp=NULL;
       }
       delete dptr;
       Cout<<"The value "<<m<<"was deleted.";
     }//end else
}// end DeleteNode
```

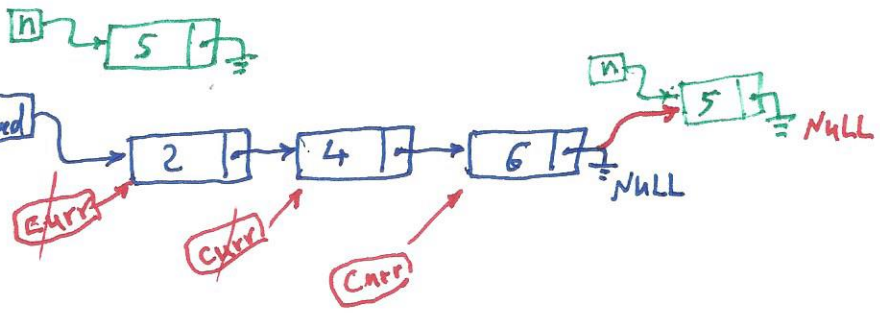


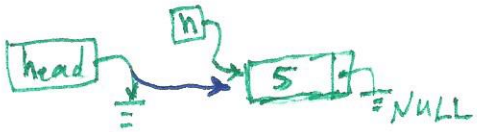




Adding a Node at the end of the list: or append

```
Void List::append(int m)        -> 5
{ Node *n=new Node; //  n -> [data | next]
  Node *Curr; // Curr ->
  n->data=m; // 5
  n->next=NULL;
  if (head!=NULL)
  { Curr=head;
    while(Curr->next!=NULL)
    { Curr=Curr->next;}
    Curr->next=n;
  }
  else
    { head=n;} // When we an empty list.
}//end append
```





Example: To Create linked list Class

```
#include <iostream>
using namespace std;

class LinkedList
{
    struct Node   // To Create Nodes
    { int data;
      Node *next;
    };
  private:
        Node *head;

  public:  // Constructor
        LinkedList()
        { head = NULL;
        }
    // Add the function addNode
    void addNode(int m)  // adding a Node at the head of a list
    {  // write the body which We had befor
    }
    // Add display()
    void display()
    {  // write body of function, we had.
    }
    // Add delete a Node
    void DeleteNode(int m)
    {  // Add your code here, we had it today
    }
    // Add append function
    void append(int m)
    {  // write the code, we had it today
    }
    ...
};  // end class
```

```
// Driver for class LinkedList                    → ADT (Abstract Data Type)
int main( )
{   LinkedList list1 ; // using defaut Constructor
    list1.addNode ( 100 ); → // we add 100 at the beginning
    list1.addNode ( 200 );
    list1.addNode ( 300 );
    list1.display ( ) ;  → output : 300
                                    200
                                    100
    list1. DeleteNode ( 200 );
    list1. append ( 400 );
    list1.display ( ) ; → output : 300
                                   100
                                   400
    . . .
} //end main
```


```
Void List:: insertNode ( int num )  → 25
{  Node *n = new Node;
   n->data = num; //25
   n->next = NULL;
   Node *curr ;
   Node *preNode; // previous Node
   if ( head == NULL) // empty list
   { head = n; }
   else { Curr = head;
          preNode = NULL;
          while ( Curr != NULL && Curr -> data < num)
          { preNode = curr ;
            Curr = Curr -> next;
          }
          if (preNode == NULL) // only one Node in the list
          { head = n ;
            n -> next = curr;
          }
          else // insert after preNode
          { preNode -> next = n;
            n -> next = curr;
          }
   } //end else
} //end func.
```