20/December/23

__/__/____

Linked-List

- doesnot store at contiguous location,
- use in Stacks / Queues.

Example :-

```
int x = 3;
int* y = &x;
cout << y << endl ; {// Print the pointer
                        to the y with the
                        reference to x}
```

Memory

32 bit | 64 bit

{ int → 4 byte } data { int → 4 byte }
{ * → 4 byte } pointer { * → 8 byte }

8 byte | 12 byte

connect array 2 22

2 , 5, 8, 7

[2] [5]

↳ move

Insertion :-

```
Node* InsertAtHead(Node* head, int val){
    return new Node(val, head);
}

Node* InsertAtTail(Node* head, int val){
    if(head==NULL){
        return new Node(val);
    }
    Node* temp = head;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = new Node(val);
    return head;
}
```

__/__/____

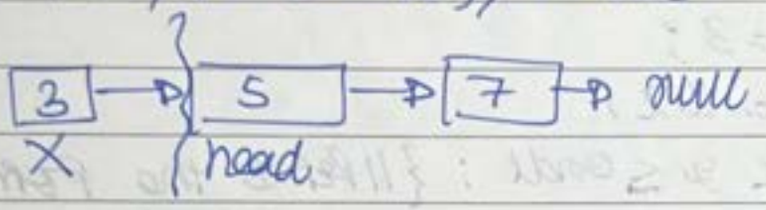
```
Node* insertAtK(Node* head,
int el, int k){
if(head==NULL){
    if(k==1){
        return new Node(el);
    }
    else return NULL;
}
if(k==1){
    return new Node(el, head);
}
int cnt=0;
Node* temp=head;
while(temp!=NULL){
  cnt++;
  if((cnt==k){
    Node* newNode=
      new Node(el, temp->next);
      temp->next=newNode;
      break;
  }
  temp=temp->next;
}
return head;
}
```

```
Node* insertBeforeValue(Node*
head, int el, int val){
if(head==NULL){
      return head;
}
if(head->data==val){
  return new Node(el, head);
}
Node* temp=head;
while(temp->next!=NULL){
  if(temp->next->data
      ==val){
    Node* newNode=
      new Node(el, temp->next
    temp->next=newNode;
    break;
}
temp=temp->next;
}
return head;
}
```

Traversal → O(N)
length → O(N)
Search of an element → O(N), O(1), O(N/2)
Deleting a node in LL →
Inserting node in LL →

⊕ Delete a node in Linked List

{Head, Tail, K^{th} element}, value}




```
RemoveK (){
  if (head == NULL){ return NULL; }
  if (K==1){
     Node* temp = head;
     head = head->next;
     free (temp);
     return head;
  }
  int cnt=0;
  Node* temp = head;
  Node* prev = NULL;
  while (temp != NULL){
     cnt++;
     if (cnt ==k){
        prev->next = prev->next->next;
        free(temp);
        break;
     }
     prev = temp; temp = temp->next;
  } return head; }
```

Deletion :-

- {remove head from LL}

```
Node* removehead (Node* head){
    if(head==NULL) return head;
    Node* temp = head;
    head = head->next;
    free(temp);
    return head;
}
```

- { remove tail from LL}

```
Node* removeTail (Node*head){
    if (head==NULL or head->next==NULL){
        return NULL;
    }
    Node* temp =head;
    while ((temp->next)->next != NULL){
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    return head;
}
```

__/__/____

{ Removing kth Position element in the LL }

```
Node * removekth(Node* head, int k){
    if(head == NULL){ return NULL; }
    if ( k==1){
        Node* temp = head;
        head = head->next;
        free(temp);
        return head;
    }
    int cnt=0;
    Node* prev =NULL;
    Node* temp = head;
    while ( temp != NULL){
        cnt++)
        if(cnt == k){
            free(temp);
            prev->next =(prev->next)->
                        next;
            break;
        }
        prev = temp;
        temp = temp->next;
    }
    return head;
}
```

+ { Remove the value element in the LL }

```
Node* removeEl ( Node* head, int el ){
    if ( head == NULL ) { return NULL; }
    if ( head -> data == el ) {
        Node* temp = head;
        head = head -> next;
        free (temp);
        return head;
    }
    Node* prev = NULL;
    Node* temp = head;
    while ( temp != NULL ){
        if ( temp -> data == el ) {
            free (temp);
            prev -> next = (prev -> next) -> next;
            break;
        }
        prev = temp;
        temp = temp -> next;
    }
    return head;
}
```