Program:- 7

WAP to implement Single Linked List with the following Operations:

(a) Sort the linked list

(b) Reverse the linked list

(c) concatenation of two Linked lists

(d) implement Stack and Queue using Linked list Representation.

(a) Sorting of Linked List

→ Algorithm

a) define a node Current which will point to head

b) define another node index which will point to node next to current

(c) compare data of Current and index node. If Current's data is greater than index's data then, swap the data b/w Them.

(d) Current will point current. next and index will point to index, next

e. Continue this process until the entire List is sorted.

i-e
```
void sortList(self){
 current = self->head;
    index = none;
 if (self -> head == None){
        return;
   }
   else {
      while (Current != None); {
        index = current -> next;
      }
   while(index!= Node){
   if(Current -> data > index.data;
     ~~index. data = temp.~~
        temP = Current. data;
        current.data = index.data;
         index.data = temp;
         index = index. next;
        Current = Current. next;
```

Φ→ Reversing the linked list

```
void Reverse( struct Node** head_ref)
{
    struct Node* prev = NULL;
    Struct Node* current = *head_ref;
     struct node* next = null;
      while ( current != NULL) {
            next = current ->next;
             current -> next = prev;
               prev = current;
                current = next;
            }
            *head_ref = prev;
```

Algorithm: 1. Initialize three points prev as NULL, curr as head and next as NULL.

2. Iterate through the linked list. In loop, do following

③ Before changing next to current, store next node.

④ next = curr ->next

⑤ Now change next to current

⑥ This is where actual reversing happens

Current -> next = prev

⑦ move prev and current one step forward.

prev = curr ;

curr = next;

→ ~~void~~ Concatenation of two Linked list :-

```
void Concatenate ( Struct node* a , Struct node *b)
{
        if(a->next == NULL)
           a -> next = b;
       else
           Concatenate ( a -> next , b);

}.
void Concatenate (struct node* a, struct node* b) {
      if (a != NULL && b != NULL)
        {
            if (a-> next == NULL)
                  a-> next= b;
          else   concatenate (a-> next, b) ;
        }
        else
          {
            Printf ("Either a or b is NULL\n");
            }
        }
```

→ Implementation of Queue and Stack in Linked List :-

Stack :-

→ i~~nsert rear()~~ ~~→ Push~~()

→ ~~insert-front()~~

→ push()

→ pop()

~~(Push)~~

```
~~void~~   #define Max 10
      typedef. Struct {
               int key;
               } element;
      typedef struct stack *Stackpointer
        typedef Struct {
               element data;
               stack pointer link;
               } Stack;
           Stack pointer top [Max];

      top [i] = NULL ; 0 ≤ i < Max ~~stacks~~.
```

```
void push (int i, element item)
  {
      stack pointer temp;
      Malloc (temp, sizeof (*temp));
        temp → data = Item;
          temp → link = top[i]
            top [i] = temp;
      }

 void pop (int i)
  {
   stack pointer temp = top[i]
       element item;
        if (!temp)
         return stack Empty();
          Item = tem→data
           top[i] = tem→link;
            free (temp);
             return Item;
      }.
```

queue:

```
    front[i] = NULL , 0 ≤ i < Max .
      Front[i] = NULL if the ith queue is empty.
 /* add to rear of the linked queue
 void addq (i, item)
    { queue pointer temp;
      malloc (temp, sizeof (*temp));
         temp → data = item;
           temp → link = NULL;
             if (front[i])
                    rear [i] → link = temp;

             else .
                 front [i] = temp;

           rear [i] = temp.
```

```
void deleteq (int i){ /* Delete from the front of a linked queue
   queuepointer  temp = front[i];
     element item;
     if (!temp)
        return queueEmpty();
      item = temp -> data;
      front[i] = temp -> link;
      free (temp);
     return item;
  }.
```