DARSHAN NINGTHOUJAM
IBM19CS041

LINKED LIST APPLICATIONS

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
 int data;
 struct node *next;
};
void insertAtEnd(struct node **head, int d){
 struct node *temp, *n;
 if (*head == NULL){
   temp = (struct node *)malloc (sizeof (struct node));
   temp => data = d;
   temp->next = NULL;
   *head = temp;
 }
 else {
    temp = *head;
    while (temp->next != NULL){
      temp = temp->next;
    }
    n= (struct node*) malloc (sizeof(struct node));
    n-> data = d;
    n-> nxt = NULL;
    temp-> nxt = n;
  }
}
void reverse (struct node **head){
   struct node *prev, *cur, *nxtl;
  cur = *head;
  prev = NULL;
  nxtl = NULL;
```

```
if (*head == NULL){
 printf ("Empty LIST \n");
     return;
}
 while (cur != NULL){
  nextL = cur -> next;
 cur -> next = prev;
  prev = cur;
  cur = nextL;
  }
  *head = prev;
}

void concat (struct node **headL, struct node **head2) {
  if (*headL == NULL){
    *headL = *head2;
    return;
  }
  if (*headL = ~~*head~~ NULL) {
    *head2 = *headL;
    return;
  }
  struct node *temp = *headL;
  while (temp -> next != NULL){
    temp = temp -> next;
    }
    temp -> next = *head2;
  }
```

```
struct node* merger(struct node *a, struct node *b)
{
  if (a==NULL){
    return b;
    }
    if(b==NULL){
    return a;}
  struct node* c = NULL;
  if (a->data < b->data)
    {
     c= a;
     c-> next = merger(a-> next, b);
    }
    else {
     c = b;
     c-> next = merger (a, b-> next);
    }
    return c;
}
struct node * MidPoint (struct node *head) {
   if (head == NULL || head->next == NULL){
     return head;
    }
    struct node *fast = head->next;
     struct node *slow = head;
    while (fast != NULL && fast ->next != NULL){
    fast = fast-> next ->next;
     slow = slow -> next;
    {
    return slow;
}
```

```
Struct node * MergeSort (struct node * head){
  if(head == NULL || head -> next == NULL) {
    return head;
    }
    Struct node * mid = MidPoint (head);
    Struct node * a = head;
    Struct node * b = mid -> next;
    mid -> next = NULL;
    a = MergeSort(a);
    b = MergeSort(b);
    Struct node * c = merge(a, b);
    return c;
  }
void display (struct node *head)
  {while (head != NULL){
     printf ("%d :->", head -> data);
     head = head -> next;
    }
    printf ("\n");
  }
  int main()
  {struct node * head1 = NULL, * head2 = NULL, * head3 =
   NULL, * head4 = NULL, * ans = NULL;
   int data, n;
    printf (" SORTING--\n");
    prin%f ("Enter the list to be sorted (Enter -1 to stop)\n");
    scanf ("%d", &data);
```

```
while (data != -1){
  insertAtEnd (&head1, data);
  scanf ("%d", &data);
}
printf ("List before sorting:");
display (head1);
ans = MergeSort(head1);
printf ("List After Sorting:");
display(ans);
printf ("\n -- REVERSE :- \n");
printf ("Enter the list to be reversed (-1 to stop): \n");
scanf ("%d ", &data);
while (data != -1){
  insertAtEnd (&head2, data);
  scanf ("%d ", &data);
}
printf ("List before reversing\n");
display (head2);
reverse (&head2);
printf ("List after reversing:");
display (head2);
printf ("\n CONCATENATION \n");
printf ("Enter the first list (Enter -1 to stop): \n");
scanf ("%d ", &data);
while (data != -1) {
  insertAtEnd (&head3, data);
  scanf ("%d ", &data);
}
```

```
printf("Enter the second list (Enter -1 to stop):\n");
scanf("%d", &data);
while (data != -1){
  insertAtEnd(&head4, data);
  scanf("%d", &data);
}
printf("First List");
display(head3);
printf("Second List");
display(head4);
concat(&head3, &head4);
printf("Concatenated List:");
display(head3);
return 0;
}
```