```
#include<stdio.h>
~~roo~~#include <stdlib.h>
#include <string.h>
void main(){
NODE first=NULL, a, b;
int item, ch, n,
for(;;){
printf("Enter your choice.\n 1. insert front
\n2 insert rear \n 3 delete front \n4
delete rear \n5 display \n6 sort \n7
search reverse \n8 concat);
scanf("%d",&ch);
switch(ch){
case 1: printf("Enter item to be inserted");
        scanf("%d", &item);
        first=insert_front(first, item);
        break;
case 2: printf("Enter item to be inserted");
        scanf("%d",&item);
        first = insert_rear (first, item);
        break;

case 3: first = delete-front( first);
        break;
case 4: first = delete_rear(first);

case 5: display( first);
        break;
case 6: first = sort(first);
        break;
case 7: first = reverse (first);
        break;
case 8: ~~printf ("Enter 1st string");~~
        ~~scan~~
        printf ("Enter no. of nodes in 1st string");
        scanf ("%d", &n);
       for (i=0;i<n;i++){
        printf(" Enter the item");
        scanf ("%d",&item);
      a= insert_rear(a, item); }
     printf ("Enter no. of nodes in 2nd string");
     scanf("%d",&n);
     for (i=0; i<n;i++){
      printf ("Enter item");
      scanf ("%d",&item);

      b= insert_rear(b, item);

       ~~b~~ }
     a = concat(a,b);
      display(a);
     break;

   }}}
```

```
struct node {
 int item;
 struct node *link;};
typedef struct node * NODE;
NODE getnode ( ){
  NODE x;
  x =(NODE) malloc (sizeof (struct node));
  if (x==NULL)
  { printf(" memory full");
    exit(0);
  }
 return x;
}

void freenode (NODE x){
 free(x);
}

                                        pos
 NODE ~~insert~~ front (NODE front, int ~~item~~){
  NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL)
  {  return temp;}
  temp -> link = first ;t
   first = temp;
   return first;
   }
```

```
NODE insert_rear( NODE first, int item){
 NODE curr, temp;
 temp = getnode();
 temp→ info = item;
 temp → link = NULL;
 if (first == NULL)
   return temp;
 curr = first;
 while (curr→link != NULL)
     curr = curr → link;
     curr → link = temp;
 return first;
}

NODE delete_front(NODE first) {
 NODE temp;
 if (first == NULL) {
 printf(" list is empty");
  return first; }
temp = first;
temp = temp → link;
printf(" item deleted at front end");
 free (first);
return temp;
}

NODE delete_rear ( NODE first) {
 NODE curr, prev;
 if (first == NULL) {
  printf("list is empty");
 return first; }
 if (first -> link == NULL)
 { printf("item deleted is %d", first -> info);
   free(first);
   return NULL;
 }
 prev = NULL;
 curr = first;
 while(curr -> link != NULL)
  { prev = curr;
    curr = curr -> link;
  }
 printf("item deleted is %d", curr -> info);
 free (curr);
 prev -> link = NULL;
 return first;
}

void display(NODE first){
  NODE temp;
  if (first == NULL)
  printf("list is empty");
  for (temp=first; temp != NULL; temp =
     temp -> link){
      printf("%d \n", temp->info);
  }.
```

```
NODE sort(NODE first){
 NODE curr, temp;
  if (first == NULL)
    return NULL;
 curr = first;
 while(curr != NULL){
  if (temp -> info < curr -> info){
      int info = curr -> info;
      curr -> info = temp -> [illegible] info;}
      ~~temp info = i~~
      temp = temp -> link;
      }
      curr = curr -> link; }
    return first;
}

NODE reverse (NODE first){
  NODE curr, temp;
  curr = NULL;
  while(first != NULL)
  { temp = first;
     first = first -> link;
     temp -> link = curr;
     curr = temp;
   }
 return curr;
 }

NODE concat (NODE first, NODE second){
```

```
{NODE cur;
 if (first ==NULL)
  return second;
 if(second = = NULL)
 return first;
cur = first;
~~cur → f~~
 while(cur != NULL)
  cur = cur → link;
 cur → link = second;
return first; }
```