Lab 7 Program: 25/11/20

1BM19CS0
RAHIL

WAP to Implement LL with following operations.
a) Sorting b) Reversing c) Concatenation.

code:

```
#include<stdio.h>
struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
NODE getnode ()
{
    NODE x;
    x = (NODE) malloc (sizeof (struct node));
    if (x==NULL)
    {
        printf("Memory full \n");
        exit (0);
    }
    return x;
}
void freenode (NODE x)
{
    free (x);
}
NODE insert_front (NODE first, int item)
{
    NODE temp;
    temp = getnode ();
    temp -> info = item;
    temp -> link = NULL;
    if (first == NULL) return temp;
```

Scanned with CamScanner

1BM19CS090
RAHIL

```
    temp -> link = first;
    first = temp;
    return first;
}
NODE   insert_rear (NODE   first, int   item)
{
    NODE temp , cur;
    temp = getnode ();
     temp -> info = item;
     temp -> link = NULL;
     if (first == NULL)
            return temp;
      cur = first;
      while (cur -> link != NULL)
          cur = cur -> link;
          cur -> link = temp;
          return first;
}
NODE     delete_front (NODE    first)
{
     NODE  temp;
     if (first == NULL)
     {
          printf(" List is Empty . Cannot Delete !! \n");
          return first;
     }
     temp = first;
     temp = temp -> link;
     print (" Item Deleted at front-end = %d \n", first -> info);
     free (first);
}    return temp;
```

```
NODE dalete_rear (NODE first)
{
    NODE cur, prev;
    if (first == NULL)
    {
        print ("List is Empty . Cannot Delete !! \n");
        return first ;
    }
    if ( first -> link == NULL)
    {
        printf (" Item deleted at Rear - end : %d \n", first -> info
        free ( first );
        return NULL ;
    }
    prev = NULL;
    cur = first;
    while ( cur -> link ! = NULL)
    {
        prev = cur;
        cur = cur -> link;
    }
    printf (" Item Deleted at Rear - end = %d \n", cur -> info);
    free (cur) ;
    prev -> link = NULL;
    return first ;
}
void display (NODE first)
{
    NODE temp;
    if ( first == NULL)
        printf (" Empty List . Cannot Display Items . \n");
    for (temp = first; temp != NULL; temp = temp -> link)
        printf ("%d \n", temp -> info );
}
```

Scanned with CamScanner

1BM19CS090.
RAHIL

```
NODE sort ( NODE first)
{
  NODE temp1;
  NODE temp2;
  for (temp1 = first; temp1 != NULL ; temp1 = temp1 -> link)
  {
     for (temp2 = temp1 -> link ; temp2 != NULL ; temp2 = temp2
                                                  -> link)
     {
        first = temp1 -> info;
        temp1 -> info = temp2 -> info;
        temp2 -> info = first ;
     }
  }
}
NODE reverse (NODE first)
{
   NODE cur , temp ;
   cur = NULL;
   while (first != NULL)
   {
      temp = first ;
      first = first -> link ;
      temp -> link = cur ;
      cur = temp;
   }
   return cur;
}
NODE concat (NODE first, NODE sec)
{
  NODE cur;
  if (first == NULL) return sec;
```

Scanned with CamScanner

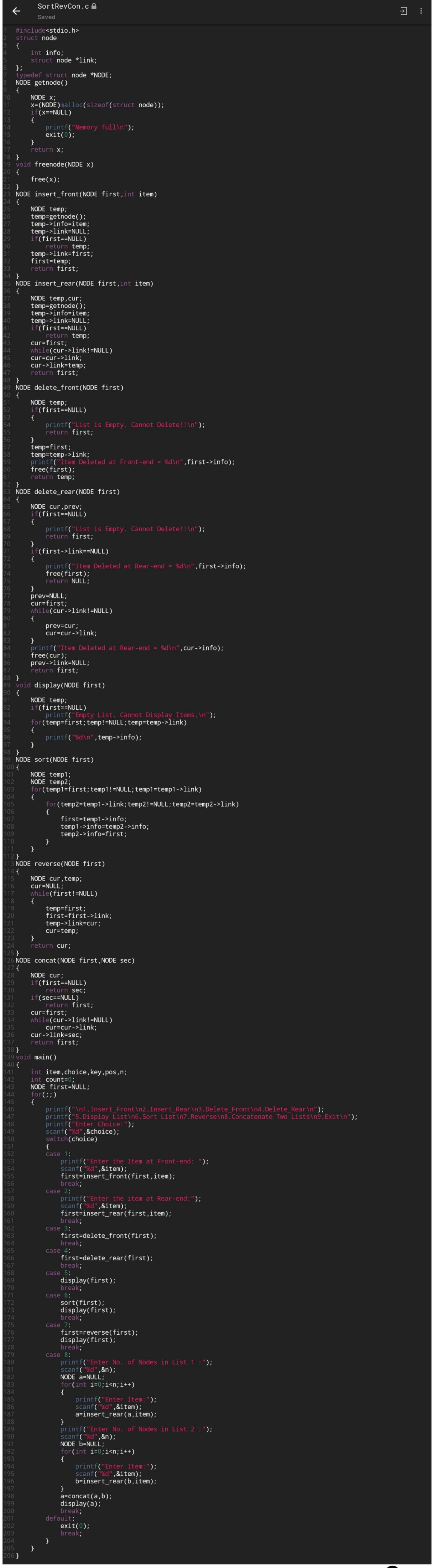
```
if (sec == NULL)
        return first;
  cur = first;
  while (cur -> link != NULL)
        cur = cur -> link;
  cur -> link = sec;
  return first;
}
void main()
{
   int item, choice, key, pos, n;
   int count = 0;
   NODE first = NULL;
   for (;;)
   {
       printf("\n1. Insert_front \n2. Insert_Rear \n3. Delete _Front
       \n4. Delete_Rear \n 5. Display List \n6. Sort List \n7. Reverse
       \n 8. Concatenate \n 9. Exit"); printf("Enter Choice: ");
       scanf("%d", &choice);
       Switch (choice)
       {
           case 1:
               printf("Enter the Item at front-end: ");
               scanf("%d", &item);
               first = insert_front (first, item);
               break;
           case 2;
               printf("Enter the Item at Rear - end: ");
               scanf("%d", &item);
               first = insert_rear (first, item);
               break;
```

1BM19CS090
RAHIL

```
case 3:
    first = delete_front ( first ) ;
    break ;
Case 4;
     first = delete _ rear ( first );
     break;

 case 5:
     display (first );
      break;

 Case 6:
      Sort ( first );
      display ( first );
       break;

 case 7:
      first = reverse ( first );
       display (first);
        break;

 Case 8 :
     printf (" Enter No. of Nodes in List 1:" );
     scanf (" %d", &n);
     NODE a = NULL;
      for ( int i =0; i<n ; i++)
      {
             printf (" Enter Item :" );
             Scanf (" %d ", &item );
             a = insert_ rear (a, item );
       }
    printf(" Enter No. of Nodes in list 2:" );
    Scanf (" %d, &n);
```

Scanned with CamScanner

```
                NODE b = NULL;
                  for (int i=0 ; i <n ; i++)
                  {
                      printf ("Enter Item :" );
                        scanf (" %d ", &item );
                        b= insert_rear (b, item);
                    }
                     a = concat (a,b);
                     display (a);
                     break;
            default:
                     exit (0);
                      break;
            }

        }

}
```

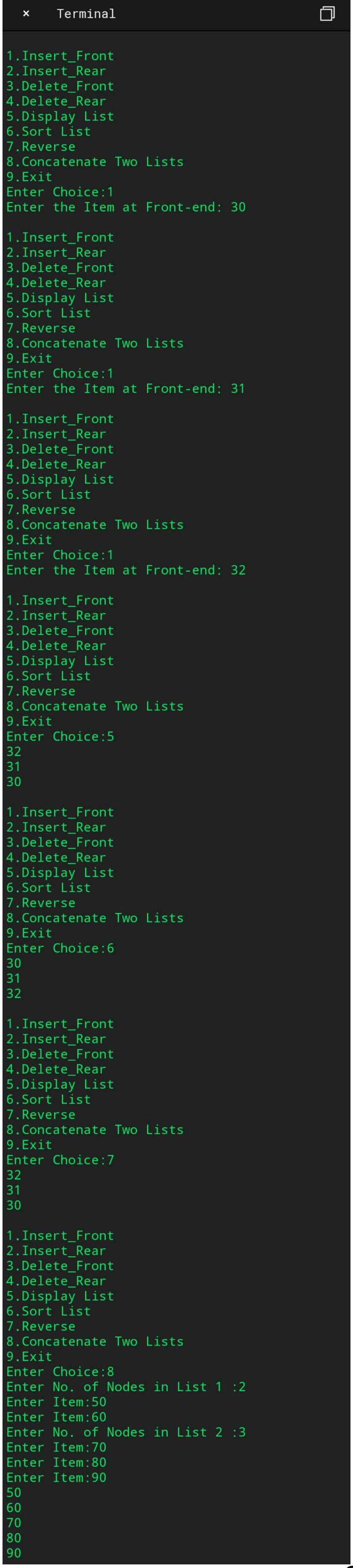


SortRevCon.c

Saved

```c
#include<stdio.h>
struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==NULL)
    {
        printf("Memory full\n");
        exit(0);
    }
    return x;
}
void freenode(NODE x)
{
    free(x);
}
NODE insert_front(NODE first,int item)
{
    NODE temp;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL)
        return temp;
    temp->link=first;
    first=temp;
    return first;
}
NODE insert_rear(NODE first,int item)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL)
        return temp;
    cur=first;
    while(cur->link!=NULL)
    cur=cur->link;
    cur->link=temp;
    return first;
}
NODE delete_front(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("List is Empty. Cannot Delete!!\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("Item Deleted at Front-end = %d\n",first->info);
    free(first);
    return temp;
}
NODE delete_rear(NODE first)
{
    NODE cur,prev;
    if(first==NULL)
    {
        printf("List is Empty. Cannot Delete!!\n");
        return first;
    }
    if(first->link==NULL)
    {
        printf("Item Deleted at Rear-end = %d\n",first->info);
        free(first);
        return NULL;
    }
    prev=NULL;
    cur=first;
    while(cur->link!=NULL)
    {
        prev=cur;
        cur=cur->link;
    }
    printf("Item Deleted at Rear-end = %d\n",cur->info);
    free(cur);
    prev->link=NULL;
    return first;
}
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
        printf("Empty List. Cannot Display Items.\n");
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\n",temp->info);
    }
}
NODE sort(NODE first)
{
    NODE temp1;
    NODE temp2;
    for(temp1=first;temp1!=NULL;temp1=temp1->link)
    {
        for(temp2=temp1->link;temp2!=NULL;temp2=temp2->link)
        {
            first=temp1->info;
            temp1->info=temp2->info;
            temp2->info=first;
        }
    }
}
NODE reverse(NODE first)
{
    NODE cur,temp;
    cur=NULL;
    while(first!=NULL)
    {
        temp=first;
        first=first->link;
        temp->link=cur;
        cur=temp;
    }
    return cur;
}
NODE concat(NODE first,NODE sec)
{
    NODE cur;
    if(first==NULL)
        return sec;
    if(sec==NULL)
        return first;
    cur=first;
    while(cur->link!=NULL)
        cur=cur->link;
    cur->link=sec;
    return first;
}
void main()
{
    int item,choice,key,pos,n;
    int count=0;
    NODE first=NULL;
    for(;;)
    {
        printf("\n1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
        printf("5.Display List\n6.Sort List\n7.Reverse\n8.Concatenate Two Lists\n9.Exit\n");
        printf("Enter Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            printf("Enter the Item at Front-end: ");
            scanf("%d",&item);
            first=insert_front(first,item);
            break;
        case 2:
            printf("Enter the item at Rear-end:");
            scanf("%d",&item);
            first=insert_rear(first,item);
            break;
        case 3:
            first=delete_front(first);
            break;
        case 4:
            first=delete_rear(first);
            break;
        case 5:
            display(first);
            break;
        case 6:
            sort(first);
            display(first);
            break;
        case 7:
            first=reverse(first);
            display(first);
            break;
        case 8:
            printf("Enter No. of Nodes in List 1 :");
            scanf("%d",&n);
            NODE a=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter Item:");
                scanf("%d",&item);
                a=insert_rear(a,item);
            }
            printf("Enter No. of Nodes in List 2 :");
            scanf("%d",&n);
            NODE b=NULL;
            for(int i=0;i<n;i++)
            {
                printf("Enter Item:");
                scanf("%d",&item);
                b=insert_rear(b,item);
            }
            a=concat(a,b);
            display(a);
            break;
        default:
            exit(0);
            break;
        }
    }
}
```

Scanned with CamScanner



```
1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 31

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:1
Enter the Item at Front-end: 32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:5
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:6
30
31
32

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:7
32
31
30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Display List
6.Sort List
7.Reverse
8.Concatenate Two Lists
9.Exit
Enter Choice:8
Enter No. of Nodes in List 1 :2
Enter Item:50
Enter Item:60
Enter No. of Nodes in List 2 :3
Enter Item:70
Enter Item:80
Enter Item:90
50
60
70
80
90
```

Scanned with CamScanner

Code :

```
#include <stdio.h>
Struct  node
{
    int  info;
    struct  node  *link;
};
typedef  struct  node  *NODE;
NODE  getnode ()
{
  NODE  x;
  x = (NODE) malloc (sizeof (struct  node));
  if (x == NULL)
  {
     printf (" Mem  full \n");
     exit (0);
  }
  return  x;
}
void  freenode (NODE  x)
{
   free (x);
}
NODE  insert_rear (NODE  first, int  item)
{
    NODE  temp, cur;
```

Scanned with CamScanner

IBM19CS090
RAHIL

```
    temp = getnode();
    temp-> info = item;
    temp -> link = NULL;
    if (first == NULL)
    return temp;
    cur = first;
    while (cur -> link != NULL)
        cur = cur -> link;
    cur -> link = temp;
    return first;
}
NODE   delete_rear (NODE first)
{
  NODE   cur, prev;
  if ( first == NULL)
  {
     printf ("List is Empty Cannot Delete \n");
     return first;
  }
  if  (first -> link == NULL)
  {
      printf ("Item deleted is %d \n", first -> info);
      free (first);
      return NULL;
  }
  prev = NULL;
  cur = first;
  while ( cur -> link != NULL)
  {
      prev = cur;
      cur = cur -> link;
  }
  printf (" item Deleted at rear-end is %d", cur -> info);
  free (cur);
  prev -> link = NULL;
```

Scanned with CamScanner

IBM19CS RAHUL

```
    return first;
}
NODE insert_pos (int item, int pos, NODE first)
{
    NODE temp, cur, prev;
    int count;
    temp = getnode ();
    temp ->info = item;
    temp --> link = NULL;
    if (first == NULL && pos ==1)
    {
        return temp;
    }
    if (first == NULL)
    {   printf ("Invalid ~~Expr~~ Position\n");
        return first;
    }
    if (pos ==1)
    {
       temp -> link = first;
       first = temp;
       return temp;
    }
    count = 1;
    prev = NULL;
    cur = first;
    while (cur != NULL && count != pos)
    {
       prev = cur;
       cur = cur -> link;
       count ++;
    }
```

Scanned with CamScanner

IBM19CS090
RAHIL

```
    if ( count == pos)
    {
        prev -> link = temp;
        temp -> link = cur;
        return first;
    }
    printf ("Invalid position\n");
    return first;
}
NODE delete_pos (int pos, NODE first)
{
    NODE cur;
    NODE prev;
    int count, flag = 0;
    if (first == NULL || pos <0)
    {
        printf ("Invalid position \n");
        return NULL)
    }
    if (pos == 1)
    {
        cur = first;
        first = first -> link;
        freenode (cur);
        return first;
    }
    prev = NULL;
    cur = first;
    count =1;
    while (cur != NULL)
    {
        if (count == pos) { flag =1; break;}
        count ++;
        prev = cur;
```

Scanned with CamScanner

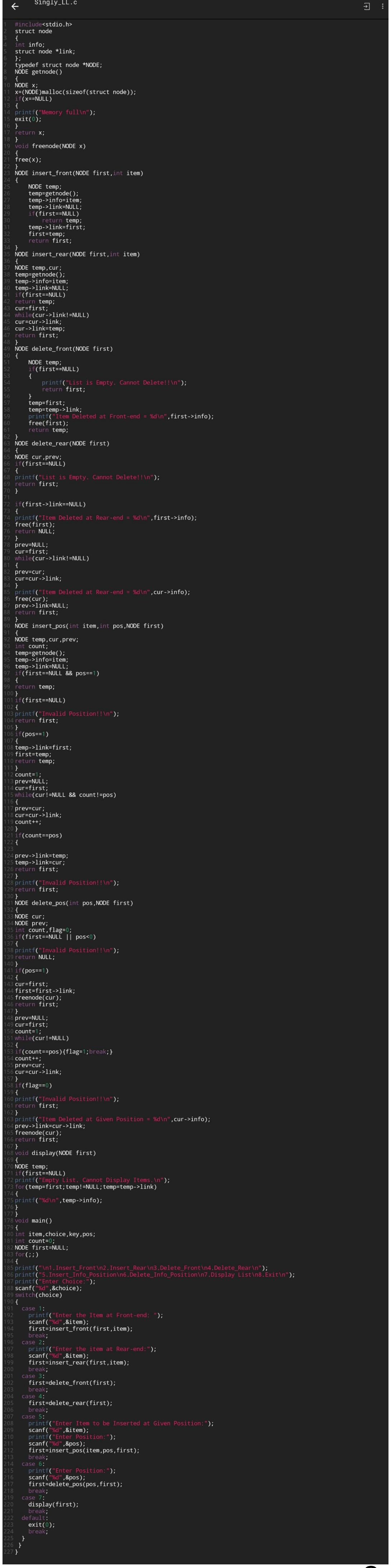
1BM19CS09
RAHIL

```
        curr = curr -> link;
      }
      if (flag ==0)
      {
         printf ("Invalid position \n");
         return first;
      }
      printf ("Item deleted at given position is %d \n", curr ->
      prev -> link = curr -> link;
      freenode (curr);
      return first.
  }
  void display (NODE first)
  {
      NODE temp;
      if (first == NULL)
      print ("list empty cannot display items\n");
      for (temp = first; temp != NULL ; temp = temp -> link)
      {
         printf (" %d \n" , temp -> info );
      }
 }
 void main ()
 {
    int item , choice , key , pos;
    int count =0;
    NODE first = NULL;
    for(;;)
    {
      printf ("\n 1. Insert_rear \n 2. Delete_rear \n 3. Insert_info_position
              \n 4. Delete_info_position \n 5. Display_list \n 6. Exit \n");
      printf ("Enter choice: \n");
```

1BM19CS090
RAHIL

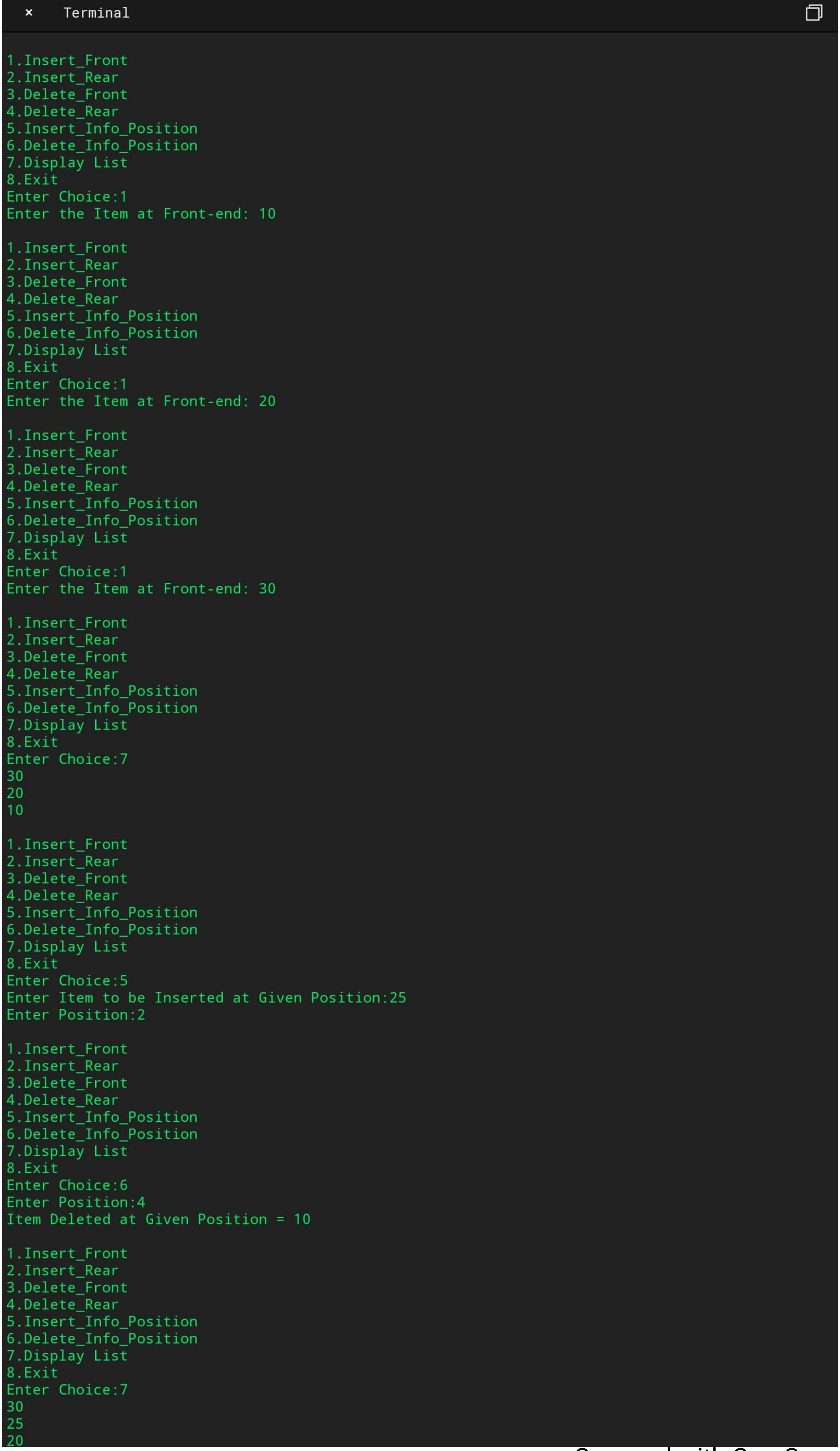
```
scanf ("%d", &choice);
switch (choice);
{
 case 1:
     printf (" Enter item at rear-end \n");
     scanf (" %d", ~~&choice~~ item);
      ~~fairst~~ first = insert_rear (first, item);
      break;
  case 2:
      first = delete_rear (first);
       break;
  case 3:
       printf (" Enter item to be inserted at given position \n");
        scanf ("%d", & item);
         printf (" Enter position");
          scanf(" %d", &pos);
           first = insert_pos (item, pos, first);
            break;
   case 4:   printf(" Enter the position \n");
              scanf (" %d", &pos);
               first = delete_pos (pos, first);
                break;
     case 5 ;   display (first);
                 break ;
       default : exit (0);
                 break;
   }
 }
}
```

Scanned with CamScanner


Singly_LL.c

```c
#include<stdio.h>
struct node
{
int info;
struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("Memory full\n");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first,int item)
{
    NODE temp;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL)
        return temp;
    temp->link=first;
    first=temp;
    return first;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
cur=first;
while(cur->link!=NULL)
cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_front(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("List is Empty. Cannot Delete!!\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("Item Deleted at Front-end = %d\n",first->info);
    free(first);
    return temp;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
{
printf("List is Empty. Cannot Delete!!\n");
return first;
}

if(first->link==NULL)
{
printf("Item Deleted at Rear-end = %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("Item Deleted at Rear-end = %d\n",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE insert_pos(int item,int pos,NODE first)
{
NODE temp,cur,prev;
int count;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL && pos==1)
{
return temp;
}
if(first==NULL)
{
printf("Invalid Position!!\n");
return first;
}
if(pos==1)
{
temp->link=first;
first=temp;
return temp;
}
count=1;
prev=NULL;
cur=first;
while(cur!=NULL && count!=pos)
{
prev=cur;
cur=cur->link;
count++;
}
if(count==pos)
{

prev->link=temp;
temp->link=cur;
return first;
}
printf("Invalid Position!!\n");
return first;
}
NODE delete_pos(int pos,NODE first)
{
NODE cur;
NODE prev;
int count,flag=0;
if(first==NULL || pos<0)
{
printf("Invalid Position!!\n");
return NULL;
}
if(pos==1)
{
cur=first;
first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
count=1;
while(cur!=NULL)
{
if(count==pos){flag=1;break;}
count++;
prev=cur;
cur=cur->link;
}
if(flag==0)
{
printf("Invalid Position!!\n");
return first;
}
printf("Item Deleted at Given Position = %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}
void display(NODE first)
{
NODE temp;
if(first==NULL)
printf("Empty List. Cannot Display Items.\n");
for(temp=first;temp!=NULL;temp=temp->link)
{
printf("%d\n",temp->info);
}
}
void main()
{
int item,choice,key,pos;
int count=0;
NODE first=NULL;
for(;;)
{
printf("\n1.Insert_Front\n2.Insert_Rear\n3.Delete_Front\n4.Delete_Rear\n");
printf("5.Insert_Info_Position\n6.Delete_Info_Position\n7.Display List\n8.Exit\n");
printf("Enter Choice:");
scanf("%d",&choice);
switch(choice)
{
  case 1:
    printf("Enter the Item at Front-end: ");
    scanf("%d",&item);
    first=insert_front(first,item);
    break;
  case 2:
    printf("Enter the item at Rear-end:");
    scanf("%d",&item);
    first=insert_rear(first,item);
    break;
  case 3:
    first=delete_front(first);
    break;
  case 4:
    first=delete_rear(first);
    break;
  case 5:
    printf("Enter Item to be Inserted at Given Position:");
    scanf("%d",&item);
    printf("Enter Position:");
    scanf("%d",&pos);
    first=insert_pos(item,pos,first);
    break;
  case 6:
    printf("Enter Position:");
    scanf("%d",&pos);
    first=delete_pos(pos,first);
    break;
  case 7:
    display(first);
    break;
  default:
    exit(0);
    break;
  }
 }
}
```

Scanned with CamScanner



```
× Terminal

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 20

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:1
Enter the Item at Front-end: 30

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
20
10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:5
Enter Item to be Inserted at Given Position:25
Enter Position:2

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:6
Enter Position:4
Item Deleted at Given Position = 10

1.Insert_Front
2.Insert_Rear
3.Delete_Front
4.Delete_Rear
5.Insert_Info_Position
6.Delete_Info_Position
7.Display List
8.Exit
Enter Choice:7
30
25
20
```

Scanned with CamScanner