Lab6:-

WAP to Implement Singly Linked List with following operations
a) a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

Program:-

```
#include<stdio.h>
#include<stdlib.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_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 delete_front(NODE first)
{
NODE temp;
if(first==NULL)
```

```
{
printf("List is empty!Can't delete an item'\n");
return first;
}
temp=first;
temp=temp->link;
printf("The Item deleted at front-end is = %d\n",first->info);
free(first);
return temp;
}
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_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;
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)
{
```

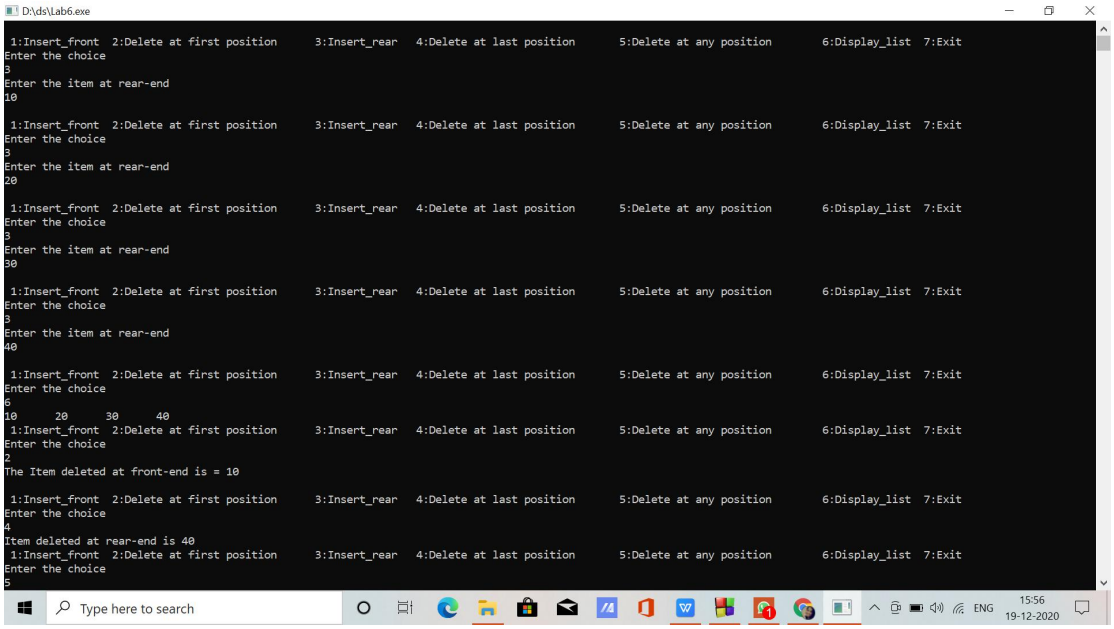
```
printf("invalid position\n");
return first;
}
printf("item deleted at given position is %d\n",cur->info);
prev->link=cur->link;
freenode(cur);
return first;
}



void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("List empty cannot display items\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\t",temp->info);
  }
}


int main()
{
int item,choice,pos;
NODE first=NULL;
for(;;)
```

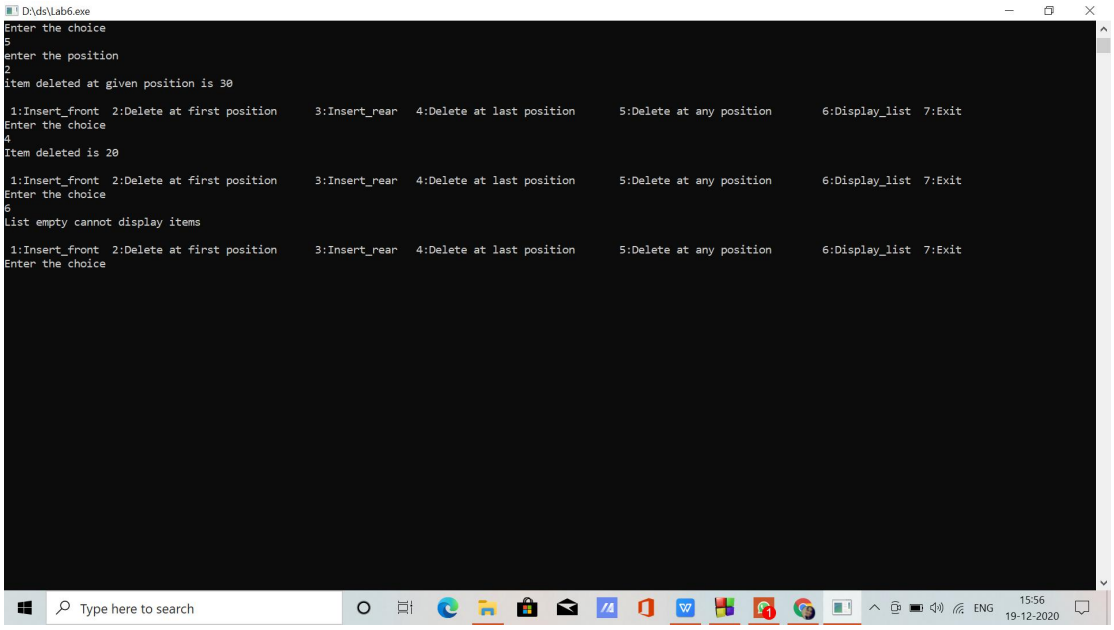
```
{
printf("\n 1:Insert_front\t 2:Delete at first position\t 3:Insert_rear\t 4:Delete at last position\t 5:Delete at any position\t 6:Display_list\t 7:Exit\n");
printf("Enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("Enter the item at front-end\n");
         scanf("%d",&item);
         first=insert_front(first,item);
         break;
  case 2:first=delete_front(first);
         break;
  case 3:printf("Enter the item at rear-end\n");
         scanf("%d",&item);
         first=insert_rear(first,item);
         break;
  case 4:first=delete_rear(first);
         break;
  case 5:printf("enter the position\n");
         scanf("%d",&pos);
         first=delete_pos(pos,first);
         break;
  case 6:display(first);
         break;
 default:exit(0);
         break;
 }
```

}

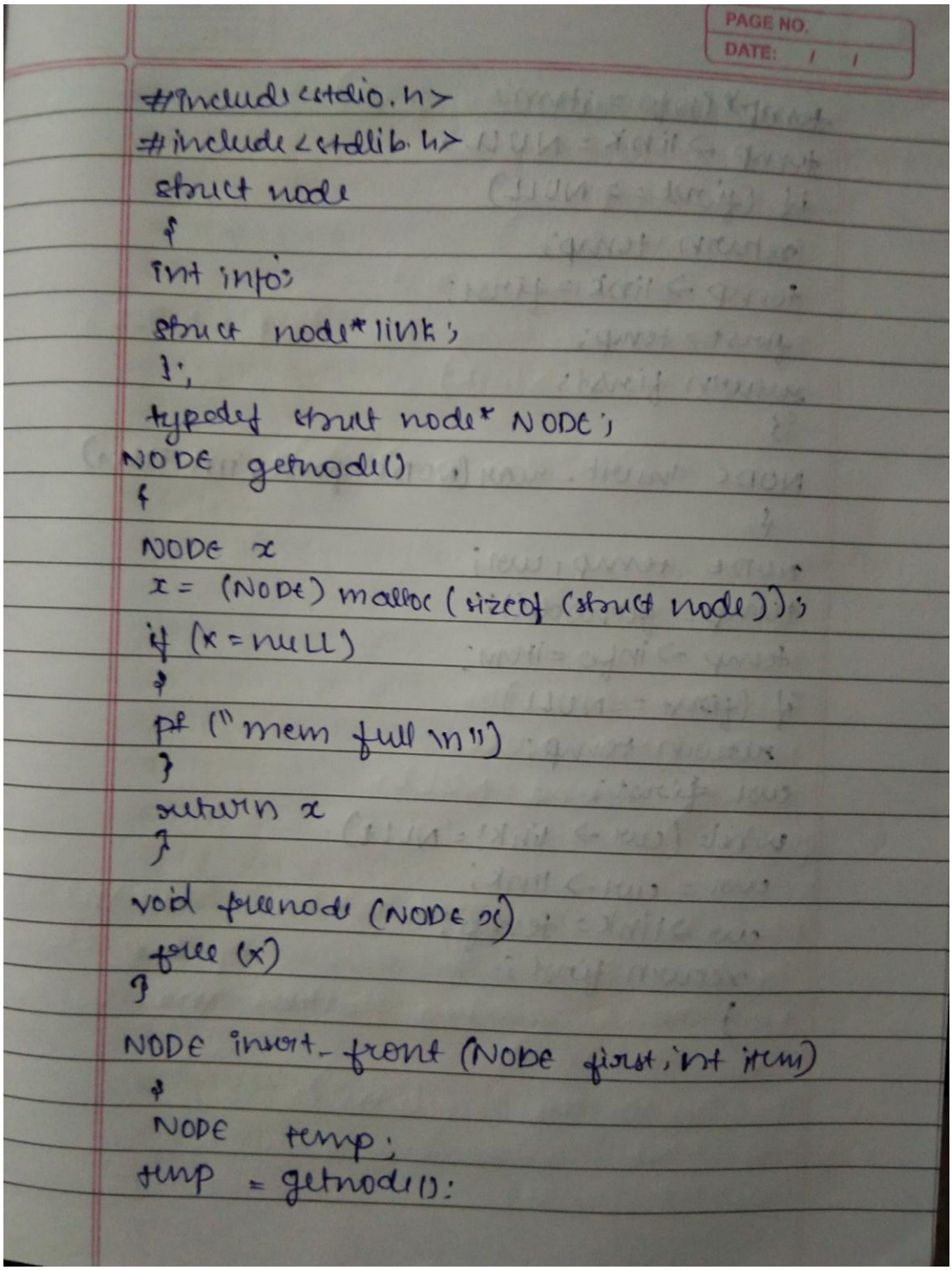
}

Output Screenshot:-





Written pictures:-



```
#include<stdio.h>
#include<stdlib.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)
{
pf ("mem full \n")
}
return x
}
void freenode (NODE x)
free (x)
}
NODE insert_front (NODE first, int item)
{
NODE temp;
temp = getnode();
```

```
void display(NODE first)
{
NODE temp;
if (first == NULL)
pf ("list is empty \n");
for (temp = first; temp != NULL;
temp = temp -> link)
{
pf ("%d \n", temp -> info);
}
}
int main()
{
int item, choice, pos;
NODE first = NULL;
NODE second = NULL;
for (;;)
{
pf (" \n1. Insert front \n2 Insert rear \n3 deletion
of first element \n4: deletion of last element
\n deletion @ specified position \n6 'display
\n7 exit \n");
pf ("enter the choice \n");
sf ("%d ", &choice);
switch (choice)
```

```
{
case 1: pf ("enter the item @ front-end \n
sf ("%d", &item);
first = insert_front (first, item);
break;
case 2: pf ("enter the item @ rear-end \n
sf ("%d", &item);
first = insert_rear (first, item);
break.
case 3: first = delete_front (first)
break.
case 4: first = delete_rear (first);
break;
case 5:
pf ("enter the position to delete \n");
scanf ("%d", &pos);
first = delete_pos (pos, first);
break;
case 6: display (first);
break;
default: exit (0);
break;
}
}
}
```

```
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;
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)
{
pf ("list is empty \n");
return first;
}
temp = first
temp = temp -> link;
pf ("item deleted = %d \n", first -> info);
free (first);
return temp;
}
NODE delete_rear (NODE first)
{
NODE cur, prev;
if (first == NULL)
{
pf ("list is empty \n");
return first;
}
if (first -> link == NULL)
{
pf ("item deleted is %d \n", first -> info);
return NULL;
}
```