**LINKED LIST**

1. Singly Linked List :

Code:

#include<stdio.h>

#include<stdlib.h>

void insertB();

void insertE();

void display();

void sorting();

void deleteB();

void deleteE();

void deleteP();

void reverse();

void insertBeforeP();

void insertAfterP();

void counting();

void search();

int counter=0;

// A linked list node

struct Node

{

int data;

struct Node \*next;

}\*temp, \*list = NULL, \*s, \*r, \*q, \*p;

//menu driven program for insertion deletion and to display linked list

int main()

{

int ch;

do

{

printf("\n Enter Choice :-\n");

printf("\n 1. Insert at the beginning.\n 2. Insert at the end.\n 3. Insert before a particular node.\n 4. Insert after a particular node.\n 5. Delete at the beginning.\n 6. Delete at the end.\n 7. Delete particular element. \n 8. Display.\n 9. Exit.\n 10. Sort.\n 11. Reversal.\n 12. Search\n 13. Count.\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

insertB();

break;

case 2:

insertE();

break;

case 3:

insertBeforeP();

break;

case 4:

insertAfterP();

break;

case 5:

deleteB();

break;

case 6:

deleteE();

break;

case 7:

deleteP();

break;

case 10:

sorting();

break;

case 8:

display();

break;

case 9:

printf("\n Bye!!!\n");

exit(0);

break;

case 11:

reverse();

break;

case 12:

search();

break;

case 13:

counting();

break;

default:

printf("\n Invalid Choice\n");

break;

}

}while(ch!=9);

}

void insertB()

{

int data1;

if(list==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = NULL;

list = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));

printf("\n Enter an element to insert.\n");

scanf("%d",&data1);

p->data = data1;

p->next = list;

list = p; //setting new element's next to point to previous element.

}

}

void insertE()

{

int data1;

if(list==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = NULL;

list = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

p->next = NULL; //NULL because it'll be the last element of the linked list.

q = list;

while(q->next != NULL)

{

q = q->next; //To check pointer q has reached last location of the linked list.

}

q->next = p; //setting last element to p which will be like adding a new element at the last location.

}

}

void insertBeforeP()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to insert before an element.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

r = list;

while(q->data != data1 && q!=NULL)

{

r = q; //storing before data matches pointer in r.

q = q->next;

}

if(q->data == data1) //if data found

{

if(q == list)

{

insertB(); //if it's first element

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

p->next = q; //setting newly added node->next to searched node

r->next = p; //setting before node->next to newly added node p

}

}

else

{

printf("\n Element not found.\n");

}

}

}

void insertAfterP()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to insert before an element.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

r = list;

while(q->data != data1 && q!=NULL)

{

q = q->next;

r = q; // storing after matching element found node to r

}

if(q->data == data1) //if data found

{

if(q->next == NULL)

{

insertE(); //if it's list element

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

r = q->next;

p->next = r; //setting newly added node->next to after searched node

q->next = p; //setting before node->next to newly added node p

}

}

else

{

printf("\n Element not found.\n");

}

}

}

void deleteB()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{

q = list;

list = list->next; //since list is pointing at first location and we're storing it in q and pointing out list to next element. Free q will delocate memeory.

if(q->next == NULL)

{

list = NULL;

}

free(q);

printf("\n Deleted first element.\n");

}

}

void deleteE()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{

q = list;

while(q->next!=NULL) //while to point q pointer at the very last location of linked list, we're running it till end.

{

r = q; //r will store previous pointed out location of linked list

q=q->next; //to increment q

}

r->next=NULL; //setting previous location's next pointer to NULL since it's very last element so it should point to NULL.

if(q->next == NULL)

{

list = NULL;

}

free(q); // delocating memory of q

printf("\n Deleted last element.\n");

}

}

void deleteP()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{ int data1; //temporary data

q = list; // q to point first location

printf("\n Enter Data to delete.\n");

scanf("%d",&data1); //taking value from the user to search that particular element to delete.

while(q->data!=data1 && q->next!=NULL) //to check given input is present in the linked list or not.

{

r = q; //r will store previous pointed out location of linked list

q=q->next; //to increment q

}

if(q == list)

{

deleteB(); //if q is first element

}

else if(q->next == NULL)

{

deleteE(); //if q is last element

}

else if(q->data == data1) //if given value matches

{

r->next = q->next; //previous next will point to q's next

free(q); //delocating memery of q

printf("\n Deleted element :- %d\n",data1);

}

else

{

printf("\n Element not found.\n"); //if given data doesn't matches

}

}

}

void sorting()

{ int xo=0;

q = list; //first location

r = q->next; //second location

if(list == NULL) //to check whether it is empty or not

{

printf("\n Linked List is empty.\n");

}

else{

while(q!=NULL) //running a while till it reaches its very last element

{

r = q->next; //r to point out next of q

while(r!=NULL) //running till r the end of the linked list

{

if(q->data>r->data) //checking whether q's data is smaller or not

{

xo = q->data; //xo storing into temporary variable for swappp

q->data = r->data; //swapping the data of q and r for swappp

r->data = xo; //back to r for swappp

}

r = r->next; //r to point out it's next\*

}

q = q->next; //q to point out it's next\*

}

}

}

void display()

{

if(list==NULL)

{

printf("\n Linked List is empty.\n");

}

else

{

printf(" Elements :-");

q=list;

counter = 0;

while(q != NULL)

{

printf("%d\t",q->data);

q = q->next;

counter++; // to count total number of elements in the linked list

}

printf(" Total count of elements = %d",counter);

}

}

void counting()

{

if(list==NULL)

{

printf("\n Linked List is empty.\n");

}

else

{

q=list;

counter = 0;

while(q != NULL)

{

q = q->next;

counter++; // to count total number of elements in the linked list

}

printf("\n Total count of elements = %d\n",counter);

}

}

void search()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to search.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

while(q->data != data1 && q!=NULL)

{

if(q->next == NULL)

{

break;

}

q = q->next;

}

if(q->data == data1)

{

printf("\n Element found.\n");

}

else

{

printf("\n Element not found\n");

}

}

}

void reverse()

{

if(list == NULL)

{

printf("\n Linked list is empty. \n");

}

else

{

q = s = list; // q and s setting these to list which is very first element of the linked list.

temp = NULL; // setting temp pointer to NULL.

r = q->next; // r to second element

while(r!=NULL) // loop till r isn't NULL

{

temp = q; // storing q first element pointer to temp

q = r; // q to store next location

r = q->next; // while we set r to q's next element pointer which is 3rd element

q->next = temp; // setting q to very first element back again.

}

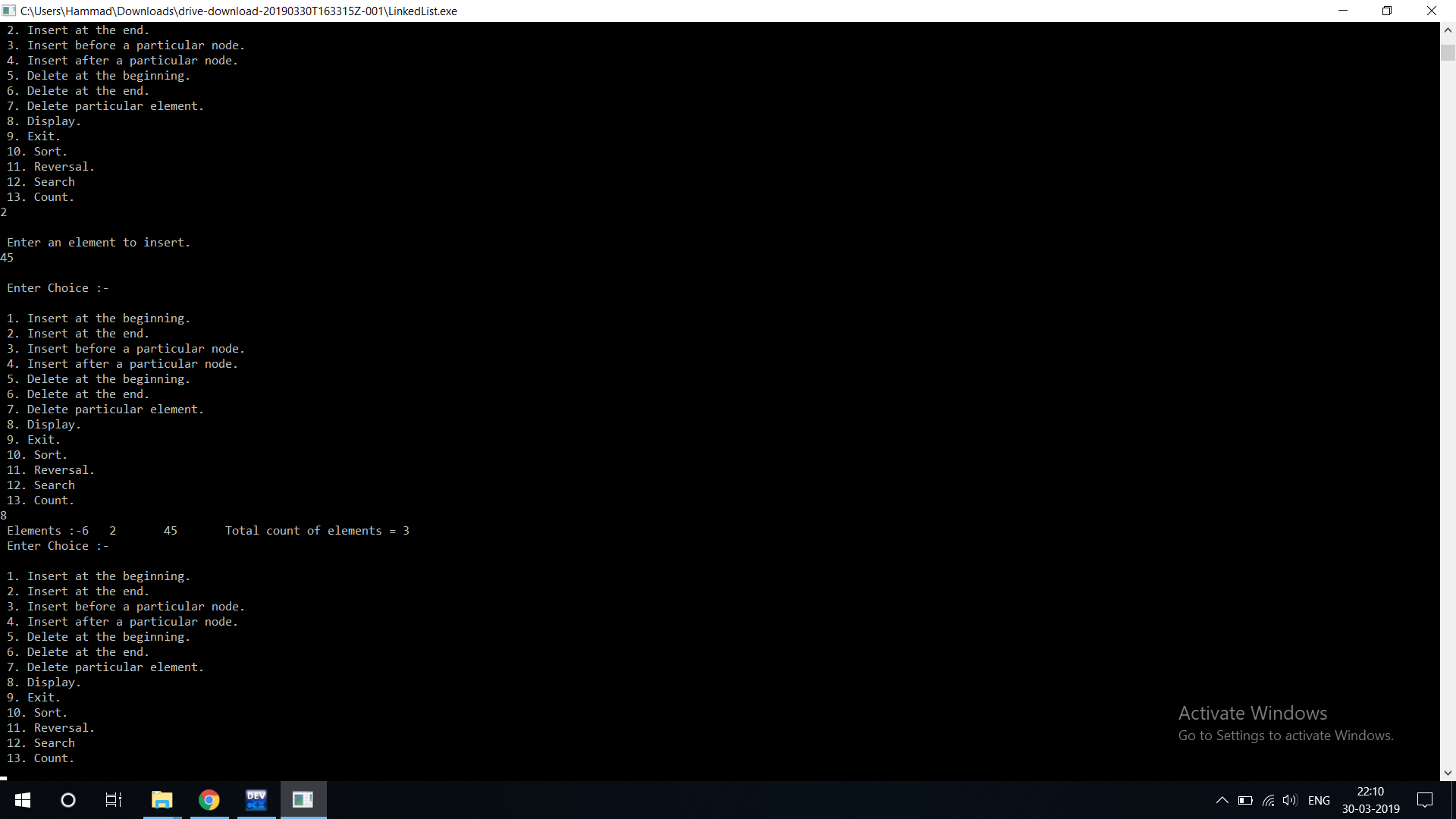
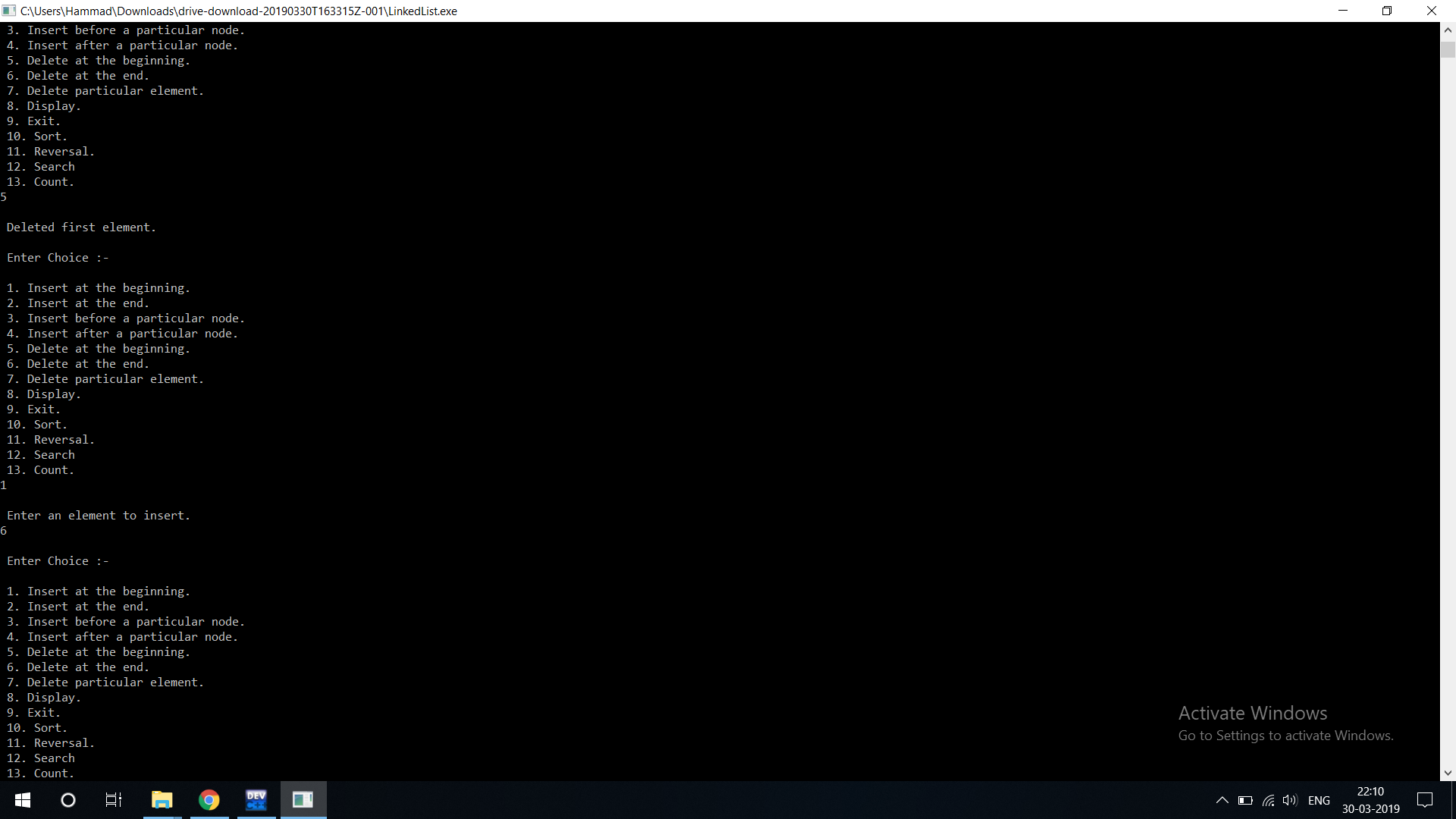
list = q; // very important to set list to very first location because its now reversed to list to should point to very first element of linked list

s->next = NULL; // s->next is NULL because very last element linked list should be NULL that's why we are storing NULL into s->next 'cause s is already pointing at last element since it's reversed.

}

}

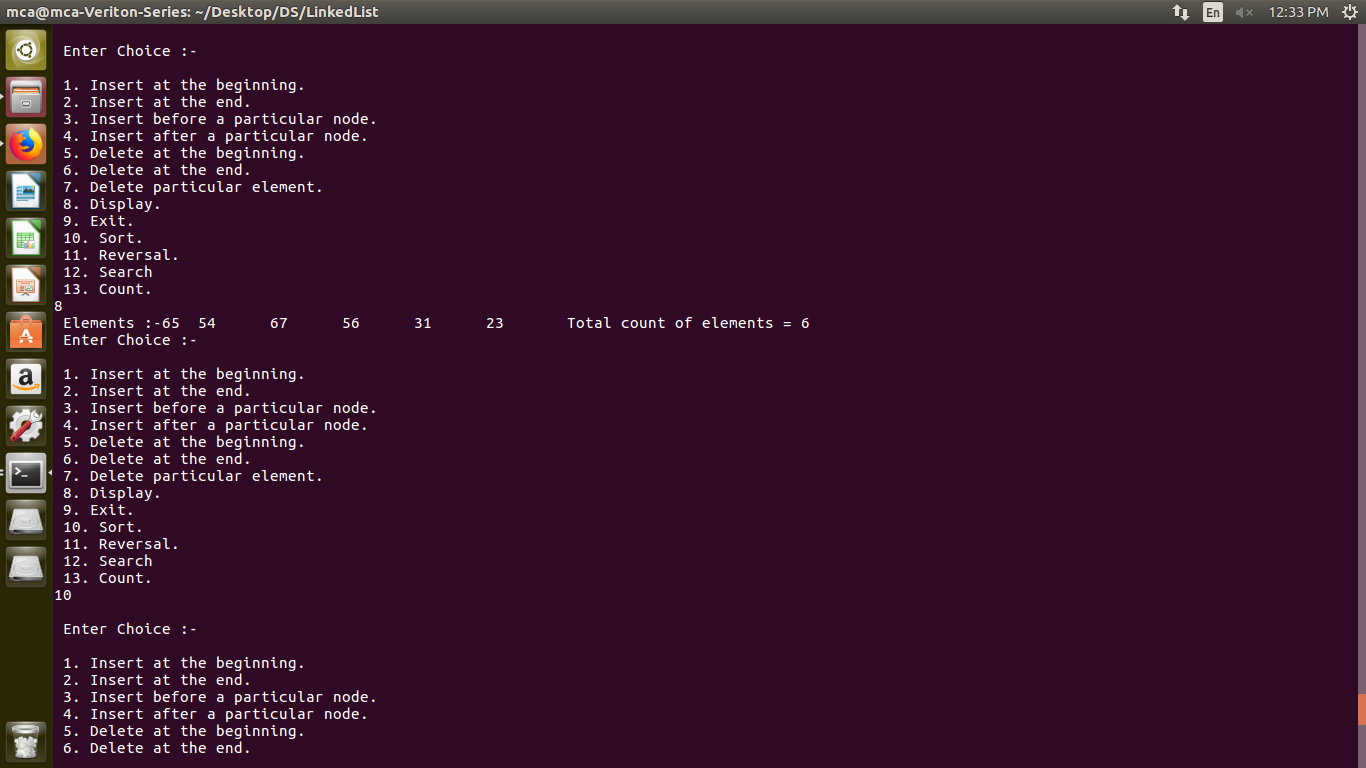
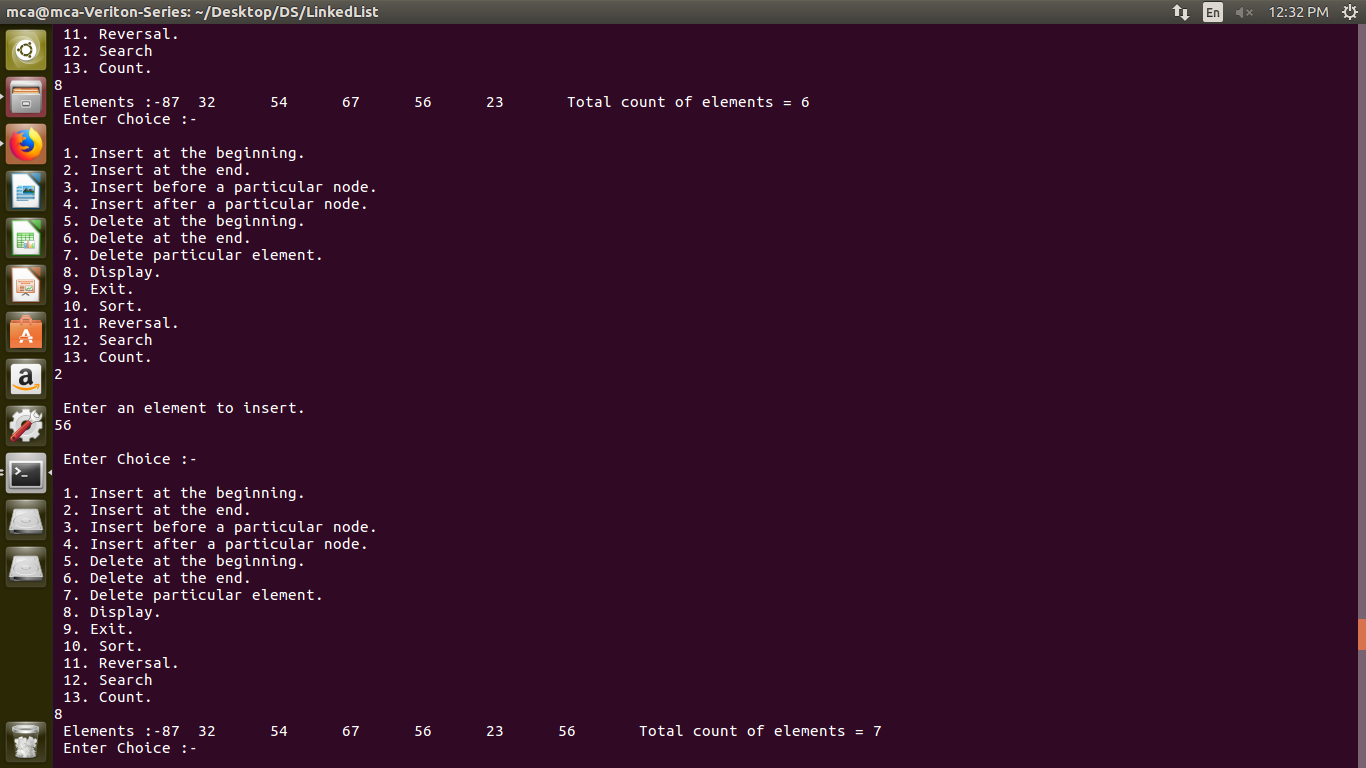
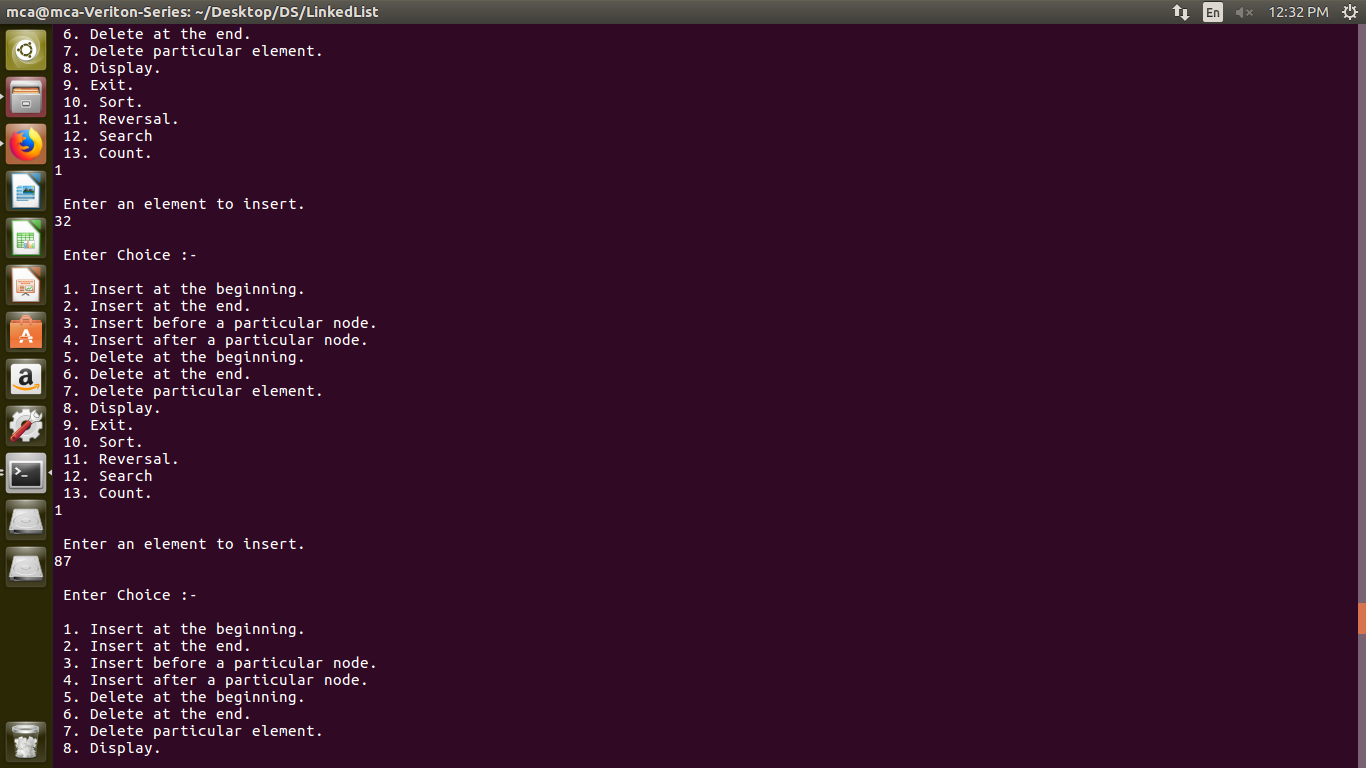
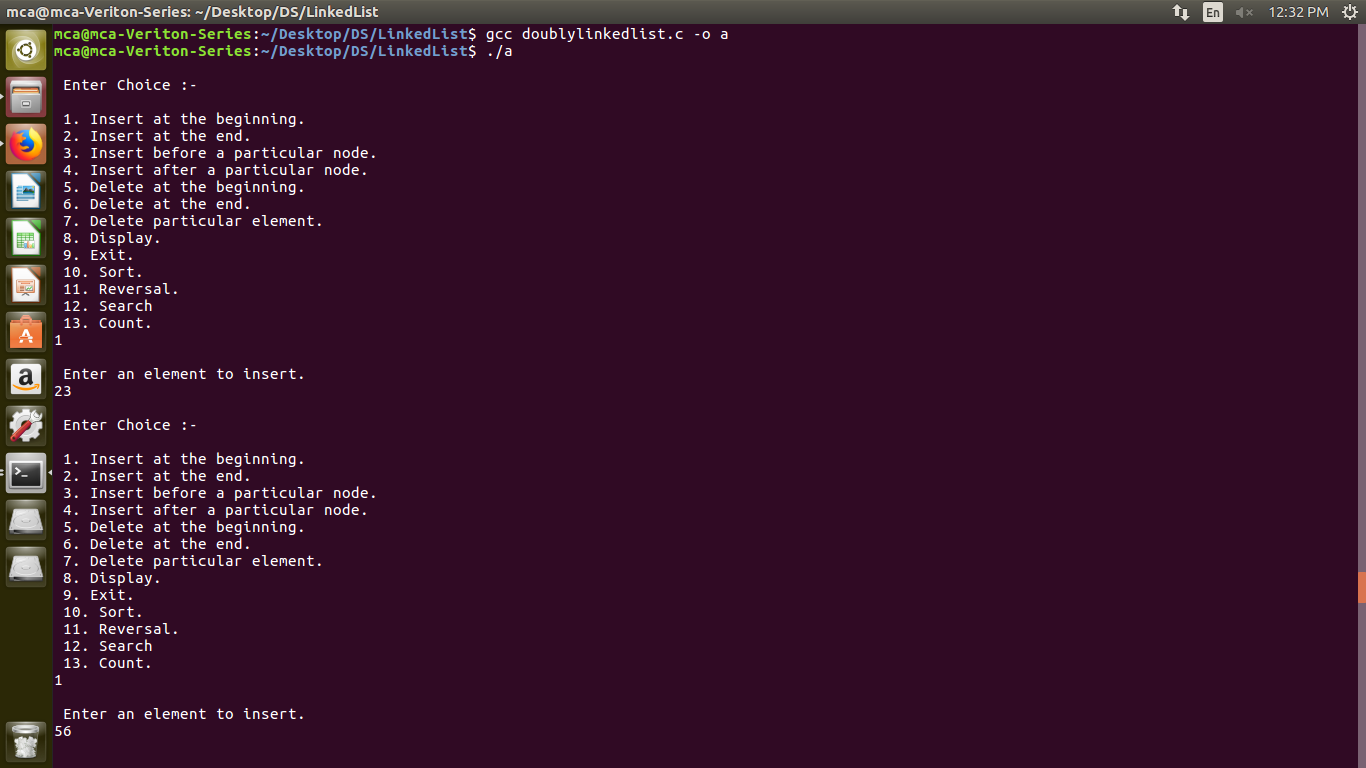
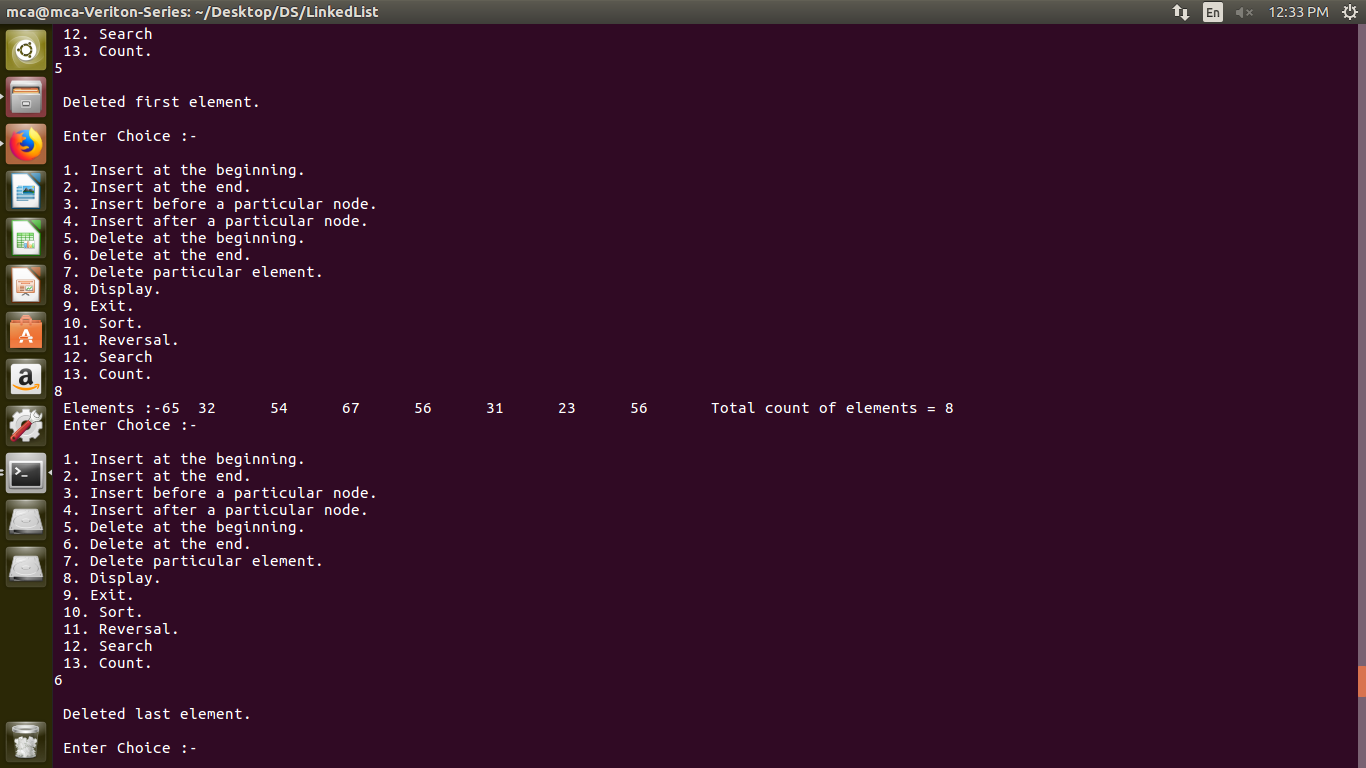
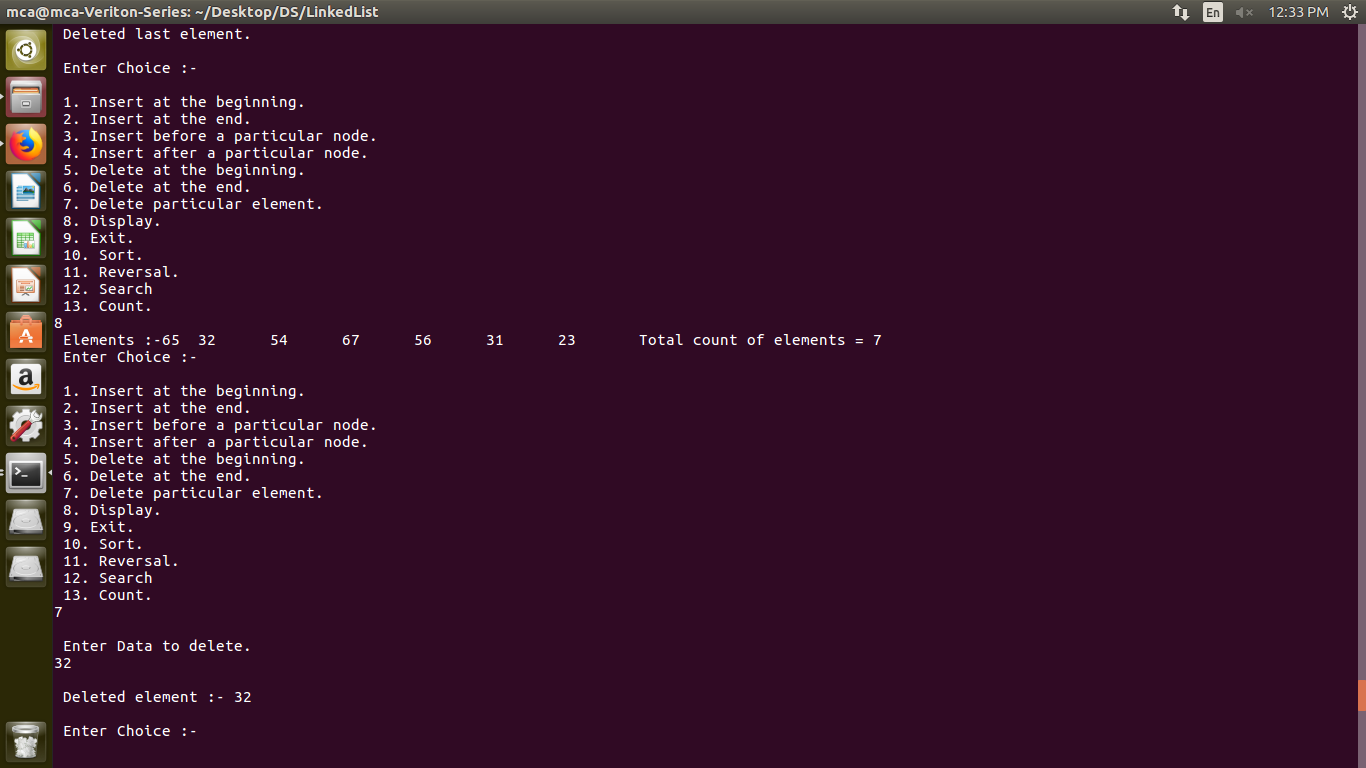
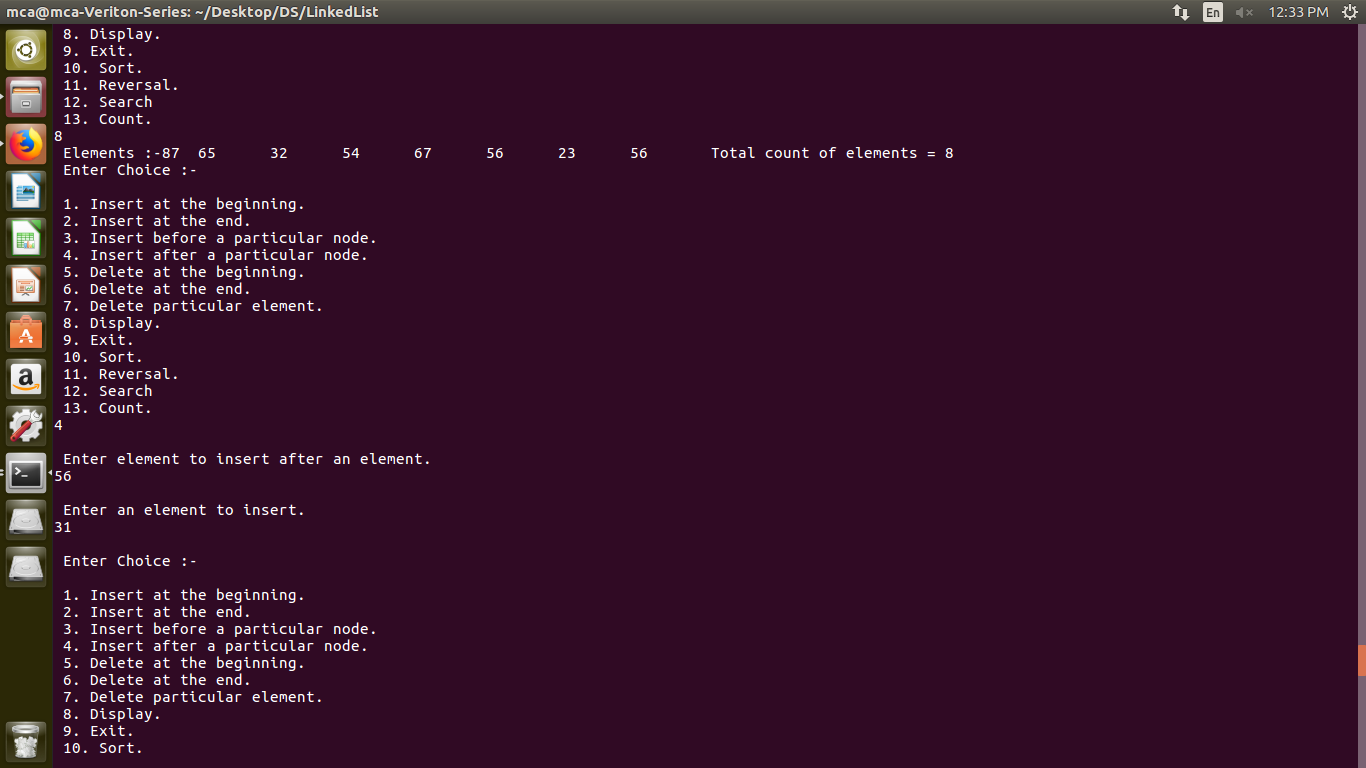
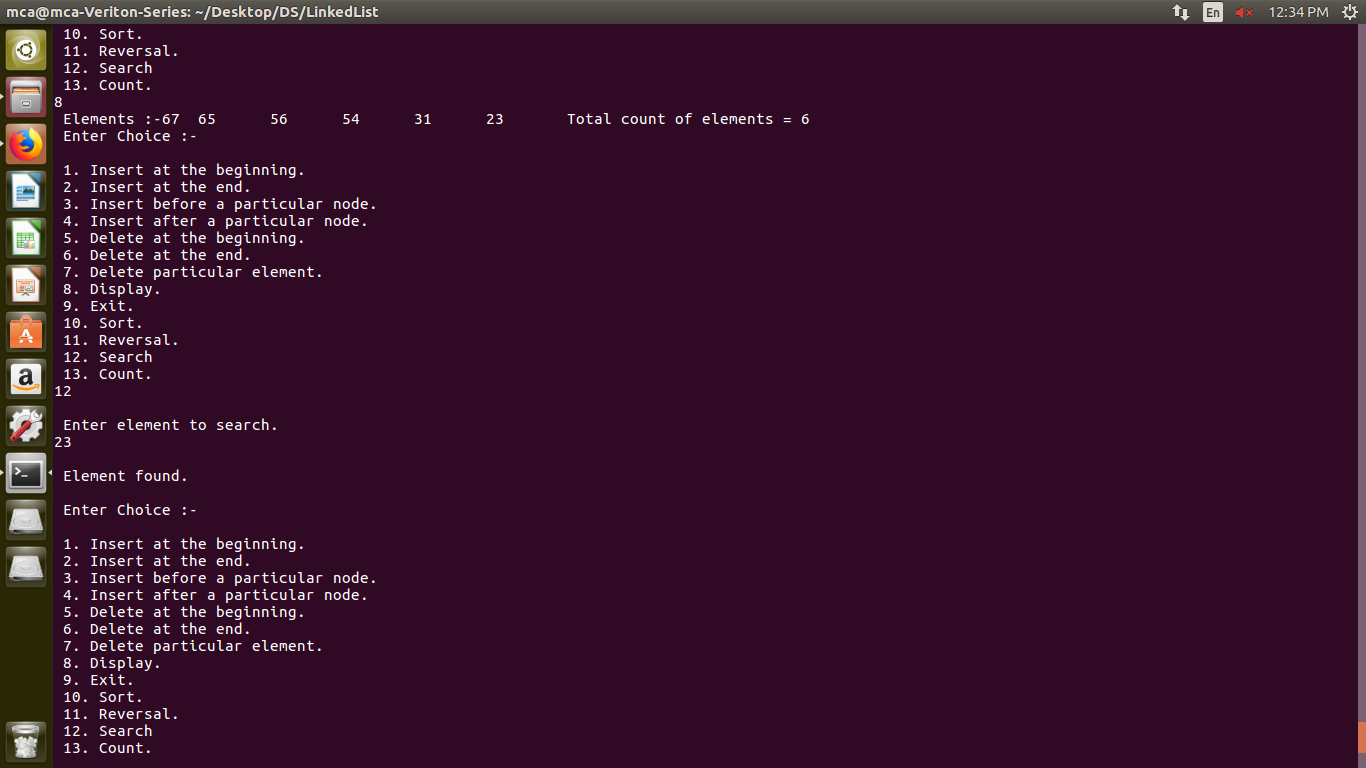
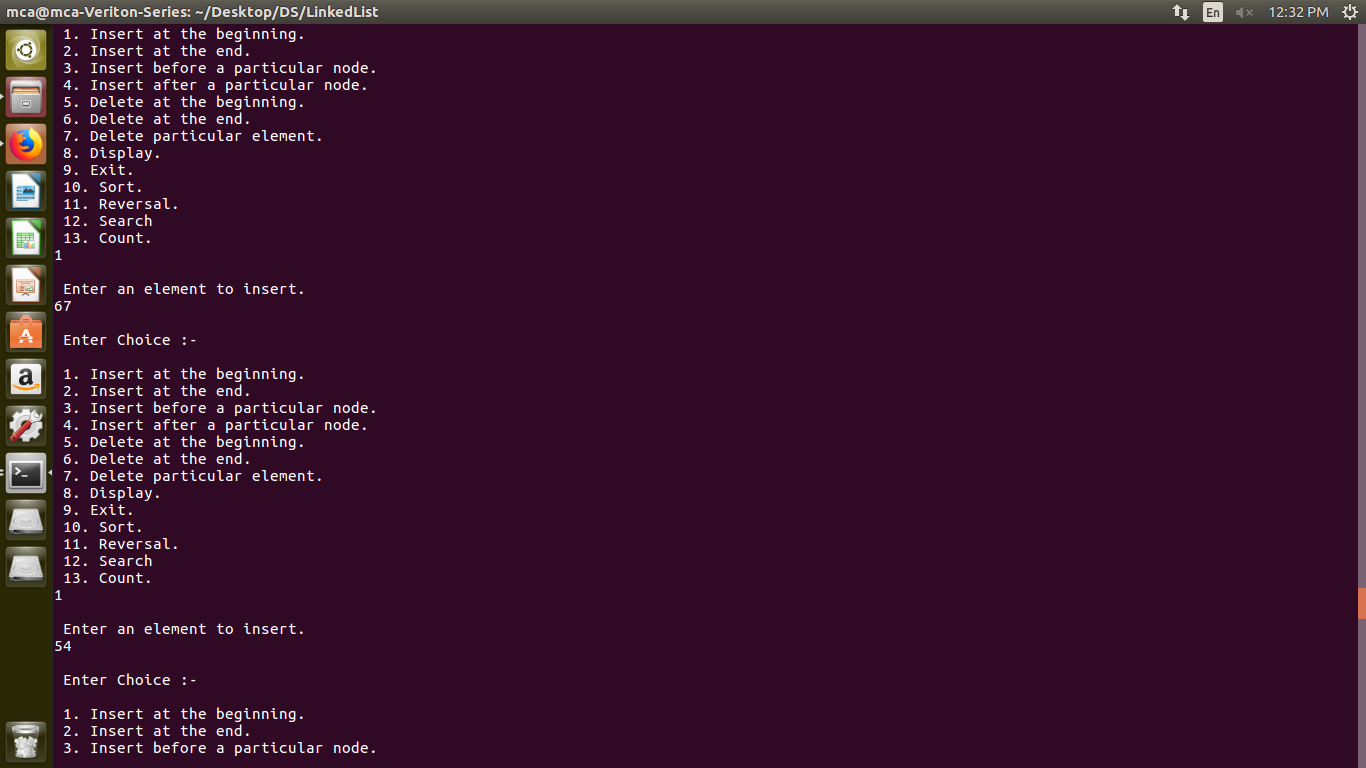
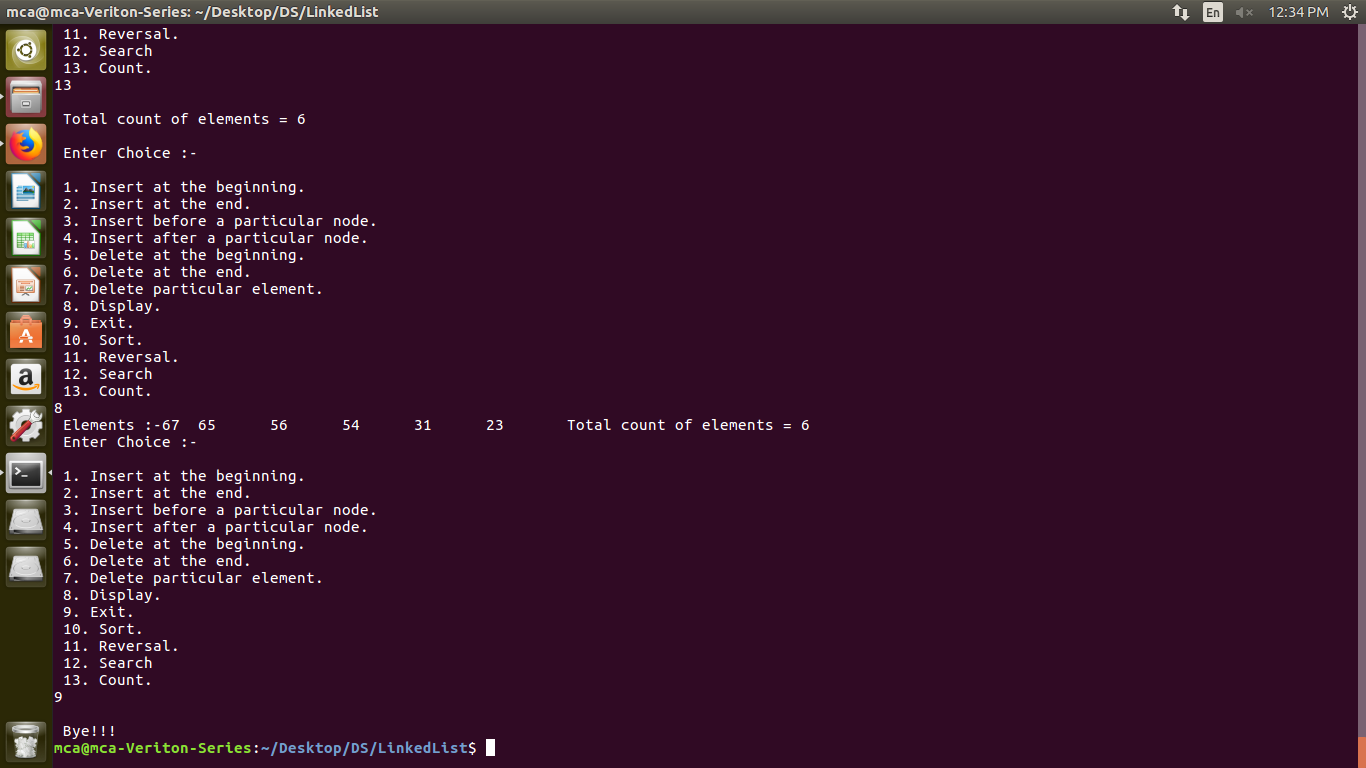
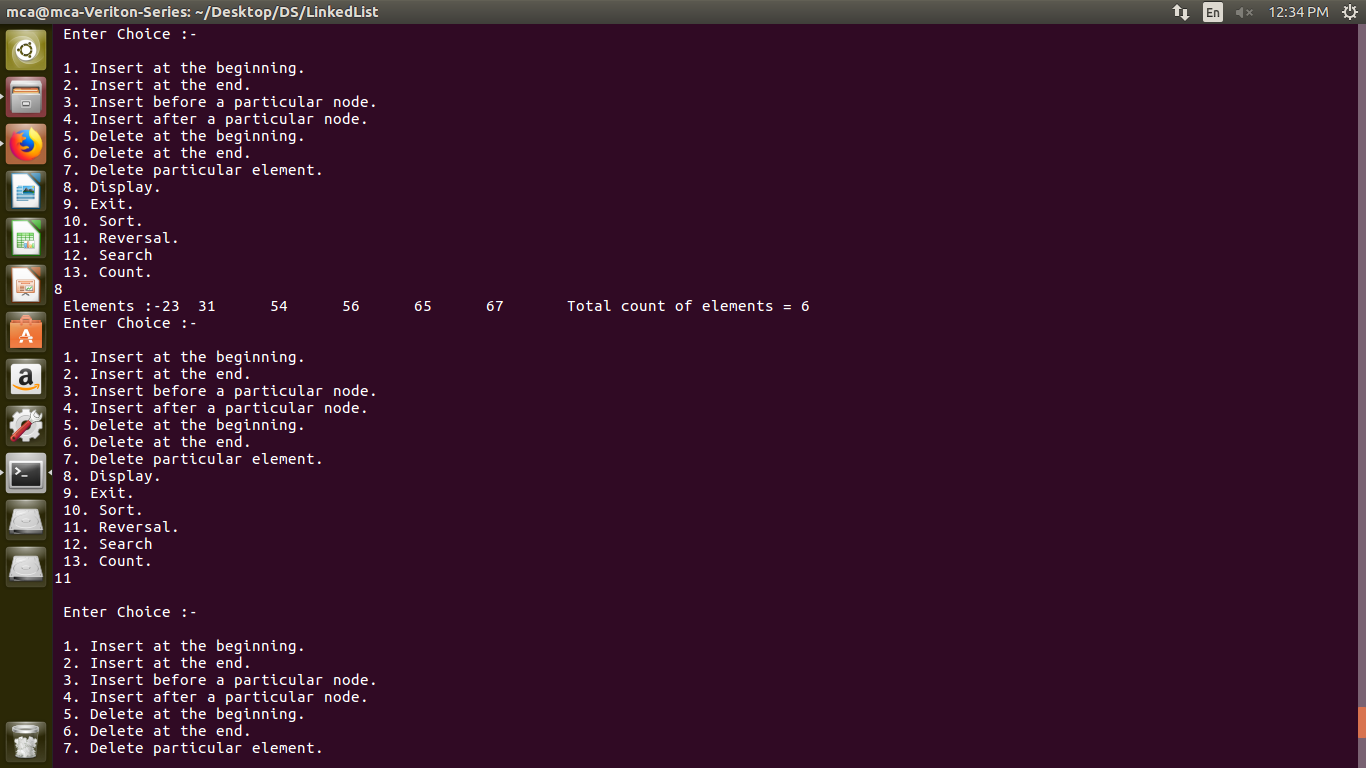
Screenshot:



1. Doubly Linked List :

Code :

#include<stdio.h>  
#include<stdlib.h>  
  
void insertB();  
void insertE();  
void display();  
void sorting();  
void deleteB();  
void deleteE();  
void deleteP();  
void reverse();  
void insertBeforeP();  
void insertAfterP();  
void search();  
void counting();  
int counter=0;  
  
// A linked list node   
struct Node   
{  
 struct Node \*lptr;  
 int data;  
 struct Node \*rptr;  
}\*temp, \*list = NULL, \*s, \*r, \*q, \*p;  
  
//menu driven program for insertion deletion and to display linked list  
int main()  
{  
 int ch;  
   
 do  
 {  
 printf("\n Enter Choice :-\n");  
 printf("\n 1. Insert at the beginning.\n 2. Insert at the end.\n 3. Insert before a particular node.\n 4. Insert after a particular node.\n 5. Delete at the beginning.\n 6. Delete at the end.\n 7. Delete particular element. \n 8. Display.\n 9. Exit.\n 10. Sort.\n 11. Reversal.\n 12. Search\n 13. Count.\n");  
  
 scanf("%d",&ch);  
 switch(ch)  
 {  
 case 1:  
 insertB();  
 break;  
 case 2:  
 insertE();  
 break;  
 case 3:  
 insertBeforeP();  
 break;  
 case 4:  
 insertAfterP();  
 break;  
 case 5:  
 deleteB();  
 break;  
 case 6:  
 deleteE();  
 break;  
 case 7:  
 deleteP();  
 break;  
 case 10:  
 sorting();  
 break;  
 case 8:  
 display();  
 break;  
 case 9:   
 printf("\n Bye!!!\n");  
 exit(0);  
 break;  
 case 11:  
 reverse();  
 break;  
 case 12:  
 search();  
 break;  
 case 13:  
 counting();  
 break;  
 default:  
 printf("\n Invalid Choice\n");  
 break;  
 }  
 }while(ch!=9);  
}  
  
void insertB()  
{  
 int data1;  
 if(list==NULL)  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.  
 printf("\n Enter an element to insert.\n");//input.  
 scanf("%d",&data1);  
 p->lptr = NULL; // both pointers will be NULL  
 p->data = data1; //setting very first element->next to null FORMAT of linked list LPTR - DATA - RPTR  
 p->rptr = NULL;  
 list = p; //list will be the very first element of the linked list.  
  
 }  
 else  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));  
 printf("\n Enter an element to insert.\n");  
 scanf("%d",&data1);  
 p->data = data1;  
 p->lptr = NULL; //left pointer will be NULL 'cause it'll be the very first element  
 p->rptr = list; //right pointer should point at next element since we are adding element at the beginning that's why p's right pointer will be list.  
 list->lptr = p; //and list's left pointer should point at it's previous element which is very first element p.  
 list = p; //setting new element's next to point to previous element.  
 }  
}  
  
void insertE()  
{  
 int data1;  
 if(list==NULL)  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.  
 printf("\n Enter an element to insert.\n");//input.  
 scanf("%d",&data1);  
 p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.  
 p->lptr = NULL;  
 p->rptr = NULL;  
 list = p;  
  
 }  
 else  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.  
 printf("\n Enter an element to insert.\n");//input.  
 scanf("%d",&data1);  
 p->data = data1;  
 p->rptr = NULL; //NULL because it'll be the last element of the linked list.  
 q = list;  
 while(q->rptr != NULL)  
 {  
 q = q->rptr; //To check pointer q has reached last location of the linked list.  
 }  
 p->lptr = q; //since p will be the last element that's why p->lptr is q.  
 q->rptr = p; //setting last element to p which will be like adding a new element at the last location.  
  
 }  
}  
  
void insertBeforeP()  
{  
 int data1;  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //if it's empty.  
 }  
 else  
 {  
   
 printf("\n Enter element to insert before an element.\n");  
 scanf("%d",&data1); // to search the particular node.  
 q = list;  
 r = list;  
 while(q->data != data1 && q!=NULL)  
 {  
 r = q; //storing before data matches pointer in r.  
 q = q->rptr;  
 }  
 if(q->data == data1) //if data found  
 {  
 if(q == list)  
 {  
 insertB(); //if it's first element  
 }  
 else  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.  
 printf("\n Enter an element to insert.\n");//input.  
 scanf("%d",&data1);  
 p->rptr = q;  
 p->data = data1;  
 p->lptr = r; //setting newly added node->next to searched node  
 r->rptr = p; //setting before node->next to newly added node p  
 q->lptr = p;  
 }  
 }  
 else  
 {  
 printf("\n Element not found.\n");  
 }  
 }  
  
}  
  
void insertAfterP()  
{  
 int data1;  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //if it's empty.  
 }  
 else  
 {  
 printf("\n Enter element to insert after an element.\n");  
 scanf("%d",&data1); // to search the particular node.  
 q = list;  
 r = list;  
 while(q->data != data1 && q!=NULL)  
 {  
 q = q->rptr;  
 }  
 r = q->rptr;  
 if(q->data == data1) //if data found  
 {  
 if(q->rptr == NULL)  
 {  
 insertE(); //if it's list element  
 }  
 else  
 {  
 p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.  
 printf("\n Enter an element to insert.\n");//input.  
 scanf("%d",&data1);  
 p->lptr = q;  
 p->data = data1;  
 p->rptr = r; //setting newly added node->next to after searched node  
 q->rptr = p; //setting before node->next to newly added node p  
 r->lptr = p; //r is after searched element which should point to it's previous element p  
 }  
 }  
 else  
 {  
 printf("\n Element not found.\n");  
 }  
 }  
  
}  
  
void deleteB()  
{  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //to check whether it is empty or not  
 }  
 else  
 {  
 q = list;  
 list = list->rptr; //since list is pointing at first location and we're storing it in q and pointing out list to next element. Free q will delocate memeory.  
 list->lptr = NULL;  
 if(q->rptr == NULL && q->lptr == NULL)  
 {  
 list = NULL;  
 }  
 free(q);  
 printf("\n Deleted first element.\n");  
 }  
}  
  
void deleteE()  
{  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //to check whether it is empty or not  
 }  
 else  
 {  
 q = list;  
 while(q->rptr!=NULL) //while to point q pointer at the very last location of linked list, we're running it till end.  
 {  
 r = q; //r will store previous pointed out location of linked list  
 q = q->rptr; //to increment q  
 }  
 r->rptr=NULL; //setting previous location's next pointer to NULL since it's very last element so it should point to NULL.  
 if(q->lptr == NULL && q->rptr == NULL)  
 {  
 list = NULL;  
 }  
 free(q); // delocating memory of q  
 printf("\n Deleted last element.\n");  
 }  
}  
  
void deleteP()  
{  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //to check whether it is empty or not  
  
 }  
 else  
 { int data1; //temporary data  
 q = list; // q to point first location  
 printf("\n Enter Data to delete.\n");  
 scanf("%d",&data1); //taking value from the user to search that particular element to delete.  
  
 while(q->data!=data1 && q->rptr!=NULL) //to check given input is present in the linked list or not.  
 {   
 r = q; //r will store previous pointed out location of linked list  
 q=q->rptr; //to increment q  
 }  
 if(q->rptr == NULL && q->lptr == NULL)  
 {  
 list = NULL;  
 }  
 else if(q == list)  
 {  
 deleteB(); //if q is first element  
 }  
 else if(q->rptr == NULL)  
 {  
 deleteE(); //if q is last element  
 }  
 else if(q->data == data1) //if given value matches  
 {  
 r->rptr = q->rptr; //previous next will point to q's next  
 free(q); //delocating memery of q  
 printf("\n Deleted element :- %d\n",data1);  
 }  
 else  
 {  
 printf("\n Element not found.\n"); //if given data doesn't matches  
 }  
 }  
}  
  
  
void sorting()  
{ int xo=0;  
 q = list; //first location  
 r = q->rptr; //second location  
 if(list == NULL) //to check whether it is empty or not  
 {  
 printf("\n Linked List is empty.\n");  
 }  
 else{  
 while(q!=NULL) //running a while till it reaches its very last element  
 {  
 r = q->rptr; //r to point out next of q  
 while(r!=NULL) //running till r the end of the linked list  
 {  
 if(q->data>r->data) //checking whether q's data is smaller or not  
 {  
 xo = q->data; //xo storing into temporary variable for swappp  
 q->data = r->data; //swapping the data of q and r for swappp  
 r->data = xo; //back to r for swappp  
 }  
 r = r->rptr; //r to point out it's next\*  
 }  
 q = q->rptr; //q to point out it's next\*  
 }  
 }  
}  
  
void display()  
{  
 if(list==NULL)  
 {  
 printf("\n Linked List is empty.\n");  
 }  
 else  
 {  
 printf(" Elements :-");  
 q=list;  
 counter = 0;  
 while(q != NULL)  
 {   
 printf("%d\t",q->data);  
 q = q->rptr;  
 counter++; // to count total number of elements in the linked list  
 }  
 printf(" Total count of elements = %d",counter);  
 }  
}  
  
void counting()  
{  
 if(list==NULL)  
 {  
 printf("\n Linked List is empty.\n");  
 }  
 else  
 {  
 q=list;  
 counter = 0;  
 while(q != NULL)  
 {  
 q = q->rptr;  
 counter++; // to count total number of elements in the linked list  
 }  
 printf("\n Total count of elements = %d\n",counter);  
 }  
}  
  
void search()  
{  
 int data1;  
 if(list == NULL)  
 {  
 printf("\n Linked List is empty.\n"); //if it's empty.  
 }  
 else  
 {  
 printf("\n Enter element to search.\n");  
 scanf("%d",&data1); // to search the particular node.  
 q = list;  
 r = list;  
 while(q->data != data1 && q!=NULL)  
 {  
 q = q->rptr;  
 }  
 if(q->data == data1)  
 {  
 printf("\n Element found.\n");  
 }  
 else  
 {  
 printf("\n Element not found\n");  
 }  
 }  
}  
  
void reverse()  
{  
 if(list == NULL)  
 {  
 printf("\n Linked list is empty. \n");  
 }  
 else  
 {  
 q = s = list; // q and s setting these to list which is very first element of the linked list.  
 temp = NULL; // setting temp pointer to NULL.  
 r = q->rptr; // r to point at 2nd element of the linked list  
 while(r!=NULL) // loop till r isn't NULL  
 {  
 temp = q; // storing q first element pointer to temp  
 q = r; // q to store r which is next element  
 r = q->rptr; // r to store next location   
 q->rptr = temp; // setting q's next pointer to temp  
 temp->lptr = q; // temp's previous pointer to point at q  
 }  
 list = q; // very important to set list to very first location because its now reversed to list to should point to very first element of linked list  
 s->rptr=NULL; // last element's right pointer should point at NULL  
 list->lptr = NULL; // and first element's left pointer should be NULL  
 }  
}

Screenshot:

3. Circular Linked List :-

Code:

#include<stdio.h>

#include<stdlib.h>

void insertB();

void insertE();

void display();

void sorting();

void deleteB();

void deleteE();

void deleteP();

void reverse();

void insertBeforeP();

void insertAfterP();

void counting();

void search();

int counter=0;

// A linked list node

struct Node

{

int data;

struct Node \*next;

}\*temp, \*list = NULL, \*s, \*r, \*q, \*p;

//menu driven program for insertion deletion and to display linked list

int main()

{

int ch;

do

{

printf("\n Enter Choice :-\n");

printf("\n 1. Insert at the beginning.\n 2. Insert at the end.\n 3. Insert before a particular node.\n 4. Insert after a particular node.\n 5. Delete at the beginning.\n 6. Delete at the end.\n 7. Delete particular element. \n 8. Display.\n 9. Exit.\n 10. Sort.\n 11. Reversal.\n 12. Search\n 13. Count.\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

insertB();

break;

case 2:

insertE();

break;

case 3:

insertBeforeP();

break;

case 4:

insertAfterP();

break;

case 5:

deleteB();

break;

case 6:

deleteE();

break;

case 7:

deleteP();

break;

case 10:

sorting();

break;

case 8:

display();

break;

case 9:

printf("\n Bye!!!\n");

exit(0);

break;

case 11:

reverse();

break;

case 12:

search();

break;

case 13:

counting();

break;

default:

printf("\n Invalid Choice\n");

break;

}

}while(ch!=9);

}

void insertB()

{

int data1;

if(list==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = p;

list = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));

printf("\n Enter an element to insert.\n");

scanf("%d",&data1);

p->data = data1;

q=list;

while(q->next!=list)

{

q=q->next;

}

q->next = p;

p->next = list;

list = p; //setting new element's next to point to previous element.

}

}

void insertE()

{

int data1;

if(list==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = p;

list = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

q = list;

while(q->next != list)

{

q = q->next; //To check pointer q has reached last location of the linked list.

}

q->next = p; //setting last element to p which will be like adding a new element at the last location.

p->next = list;

}

}

void insertBeforeP()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to insert before an element.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

r = list;

while(q->data != data1 && q->next != list)

{

r = q; //storing before data matches pointer in r.

q = q->next;

}

if(q->data == data1) //if data found

{

if(q == list)

{

insertB(); //if it's first element

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

p->next = q; //setting newly added node->next to searched node

r->next = p; //setting before node->next to newly added node p

}

}

else

{

printf("\n Element not found.\n");

}

}

}

void insertAfterP()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to insert before an element.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

r = list;

while(q->data != data1 && q->next != list)

{

q = q->next;

r = q; // storing after matching element found node to r

}

if(q->data == data1) //if data found

{

if(q->next == list)

{

insertE(); //if it's list element

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

r = q->next;

p->next = r; //setting newly added node->next to after searched node

q->next = p; //setting before node->next to newly added node p

}

}

else

{

printf("\n Element not found.\n");

}

}

}

void deleteB()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{

q = list;

list = list->next; //since list is pointing at first location and we're storing it in q and pointing out list to next element. Free q will delocate memeory.

if(q->next == q)

{

list = NULL;

}

else

{

temp = r = list;

while(r->next != list)

{ temp = r;

r = r->next;

}

temp->next = list;

}

free(q);

printf("\n Deleted first element.\n");

}

}

void deleteE()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{

q = list;

while(q->next != list) //while to point q pointer at the very last location of linked list, we're running it till end.

{

r = q; //r will store previous pointed out location of linked list

q=q->next; //to increment q

}

r->next=list; //setting previous location's next pointer to NULL since it's very last element so it should point to NULL.

if(q->next == q)

{

list = NULL;

}

free(q); // delocating memory of q

printf("\n Deleted last element.\n");

}

}

void deleteP()

{

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //to check whether it is empty or not

}

else

{

int data1; //temporary data

q = list; // q to point first location

printf("\n Enter Data to delete.\n");

scanf("%d",&data1); //taking value from the user to search that particular element to delete.

while(q->data!=data1 && q->next != list) //to check given input is present in the linked list or not.

{

r = q; //r will store previous pointed out location of linked list

q=q->next; //to increment q

}

if(q == list)

{

deleteB(); //if q is first element

}

else if(q->next == list)

{

deleteE(); //if q is last element

}

else if(q->data == data1) //if given value matches

{

r->next = q->next; //previous next will point to q's next

free(q); //delocating memery of q

printf("\n Deleted element :- %d\n",data1);

}

else

{

printf("\n Element not found.\n"); //if given data doesn't matches

}

}

}

void sorting()

{

int xo=0;

q = s = list; //first location

r = q->next; //second location

if(list == NULL) //to check whether it is empty or not

{

printf("\n Linked List is empty.\n");

}

else{

list=list->next;

while(q->next!=list) //running a while till it reaches its very last element

{

r = q->next; //r to point out next of q

while(r->next!=list) //running till r the end of the linked list

{

if(q->data>r->data) //checking whether q's data is smaller or not

{

xo = q->data; //xo storing into temporary variable for swappp

q->data = r->data; //swapping the data of q and r for swappp

r->data = xo; //back to r for swappp

}

r = r->next; //r to point out it's next\*

}

q = q->next; //q to point out it's next\*

}

}

list = s;

}

void display()

{

if(list==NULL)

{

printf("\n Linked List is empty.\n");

}

else

{

printf(" Elements :-\n");

r=q=list;

counter = 0;

do

{

printf("%d\t",q->data);

q = q->next;

counter++; // to count total number of elements in the linked list

}while(q!=list);

printf(" Total count of elements = %d",counter);

}

}

void counting()

{

if(list==NULL)

{

printf("\n Linked List is empty.\n");

}

else

{

q=list;

counter = 0;

while(q->next != list)

{

q = q->next;

counter++; // to count total number of elements in the linked list

}

printf("\n Total count of elements = %d\n",counter+1);

}

}

void search()

{

int data1;

if(list == NULL)

{

printf("\n Linked List is empty.\n"); //if it's empty.

}

else

{

printf("\n Enter element to search.\n");

scanf("%d",&data1); // to search the particular node.

q = list;

do{

q = q->next;

}while(q->data != data1 && q->next!=list);

if(q->data == data1)

{

printf("\n Element found.\n");

}

else

{

printf("\n Element not found\n");

}

}

}

void reverse()

{

if(list == NULL)

{

printf("\n Linked list is empty. \n");

}

else

{

q = s = list; // q and s setting these to list which is very first element of the linked list.

temp = NULL; // setting temp pointer to NULL.

r = q->next; // r to second element

while(r!=NULL) // loop till r isn't NULL

{

temp = q; // storing q first element pointer to temp

q = r; // q to store next location

r = q->next; // while we set r to q's next element pointer which is 3rd element

q->next = temp; // setting q to very first element back again.

}

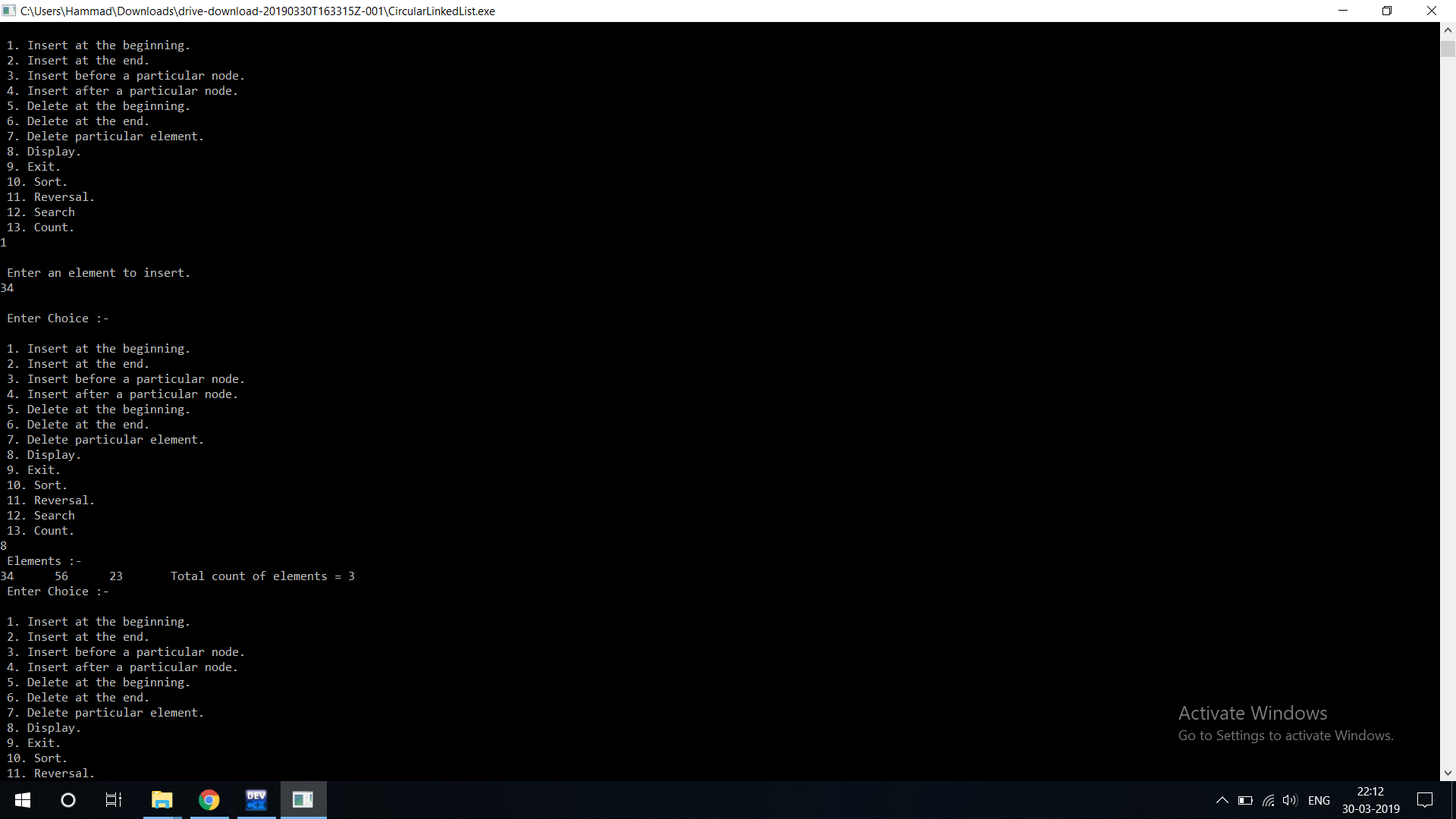
list = q; // very important to set list to very first location because its now reversed to list to should point to very first element of linked list

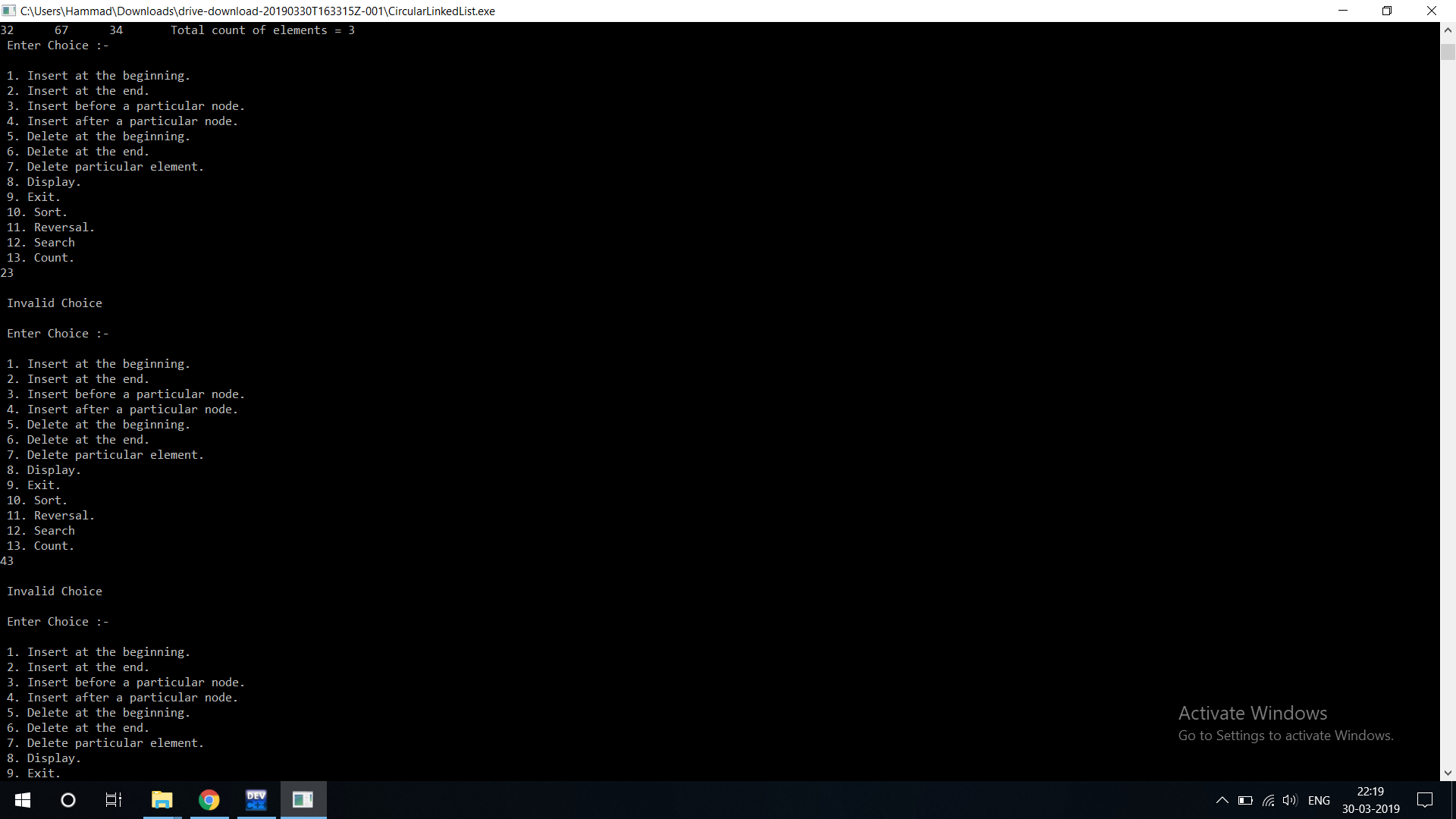
s->next = NULL; // s->next is NULL because very last element linked list should be NULL that's why we are storing NULL into s->next 'cause s is already pointing at last element since it's reversed.

}

}

Screenshot:





4) Polynomial Addition using Linked List

Code :

#include<stdio.h>

#include<malloc.h>

#include<conio.h>

struct link{

int coeff;

int pow;

struct link \*next;

};

struct link \*poly1=NULL,\*poly2=NULL,\*poly=NULL;

void create(struct link \*node)

{

char ch;

do

{

printf("\n enter coeff:");

scanf("%d",&node->coeff);

printf("\n enter power:");

scanf("%d",&node->pow);

node->next=(struct link\*)malloc(sizeof(struct link));

node=node->next;

node->next=NULL;

printf("\n continue(y/n):");

ch=getch();

}

while(ch=='y' || ch=='Y');

}

void show(struct link \*node)

{

while(node->next!=NULL)

{

printf("%dx^%d",node->coeff,node->pow);

node=node->next;

if(node->next!=NULL)

printf("+");

}

}

void polyadd(struct link \*poly1,struct link \*poly2,struct link \*poly)

{

while(poly1->next && poly2->next)

{

if(poly1->pow>poly2->pow)

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff;

poly1=poly1->next;

}

else if(poly1->pow<poly2->pow)

{

poly->pow=poly2->pow;

poly->coeff=poly2->coeff;

poly2=poly2->next;

}

else

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff+poly2->coeff;

poly1=poly1->next;

poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link));

poly=poly->next;

poly->next=NULL;

}

while(poly1->next || poly2->next)

{

if(poly1->next)

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff;

poly1=poly1->next;

}

if(poly2->next)

{

poly->pow=poly2->pow;

poly->coeff=poly2->coeff;

poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link));

poly=poly->next;

poly->next=NULL;

}

}

main()

{

char ch;

do{

poly1=(struct link \*)malloc(sizeof(struct link));

poly2=(struct link \*)malloc(sizeof(struct link));

poly=(struct link \*)malloc(sizeof(struct link));

printf("\nenter 1st number:");

create(poly1);

printf("\nenter 2nd number:");

create(poly2);

printf("\n1st Number:");

show(poly1);

printf("\n2nd Number:");

show(poly2);

polyadd(poly1,poly2,poly);

printf("\nAdded polynomial:");

show(poly);

ch=getch();

}

while(ch=='y' || ch=='Y');

}

Screenshot :

