**1.Write a program to implement Stack Operations.**

#include<stdio.h>

#define size 5

void push();

void pop();

void peep();

int update();

int view();

int top=-1;

int stack[size];

int i=0;

int main()

{

int choice ;

while(1)

{

printf("\n 1.push 2.pop 3.peep 4.update 5.view\n");

printf("\n Entert opearation want to perform in stack = ");

scanf("%d",&choice);

switch(choice)

{

case 1: push();break;

case 2: pop();break;

case 3: peep();break;

case 4: update();break;

case 5: view(); break;

default :printf("\nInvalid input .");

}

}

}

return 0;

}

void push()

{

int x;

if(top>=size-1)

{

printf(" \nStack overflow.");

}

else

{

printf("\n Enter the no to push in to the stack =");

scanf("%d",&x);

top=top+1;

stack[top]=x;

printf("\npush succesfully.");

}

}

void pop()

{

if(top==-1)

{

printf("\n Stack Under flow.");

}

else

{

int i=stack[top];

printf("\n Your pop elemet is %d",i);

top=top-1;

}

}

void peep()

{

int i;

printf("Enter the index value to peep the elements =");

scanf("%d",&i);

if(i>size)

{

printf("\n Stack is overflow.");

}

else if(top-i+1<0)

{

printf("\nStack is underflow.");

}

else{

printf("\n The peep value is = %d",stack[i]);

}

}

int update()

{

int i;

int old item;

int new item;

print("Enter the index value of the stack =");

scanf("%d",&i);

int old\_no;

if(i>size)

{

printf("\n invalid intex number.");

}

else if(top-i+1<0)

{

printf("\n Invalid index value.");

}

else{

printf("Enter the new item =");

scanf("%d",&new\_item);

// old\_item=stack[top-i+1];

stack[i]=new\_item;

printf("\n update succesfully.");

}

}

int view()

{

printf("\n The stack is =");

for(i=top;i>=0;i--)

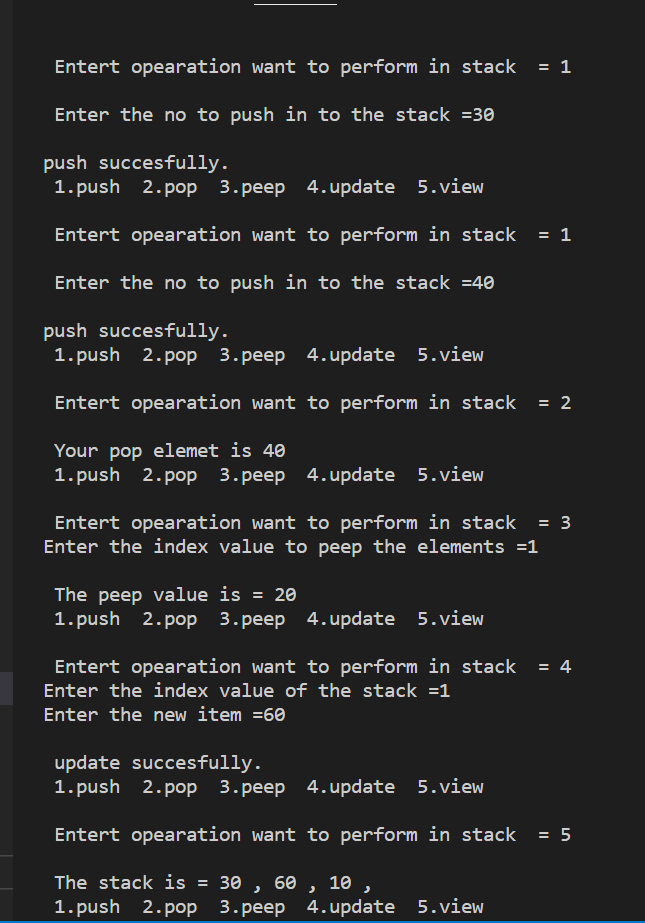
{

printf(" %d ,",stack[i]);

}

return 0;

}



**9.Write program to sort a given list using.**

* **Bubble sort**

#include<stdio.h>

#include<math.h>

void main(){

int n,temp,i,j,a[50];

printf("Enter the number of element you want in an array = \n");

scanf("%d",&n);

printf("Enter the elements of array =");

for(int i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

printf("The sorting arrays is =");

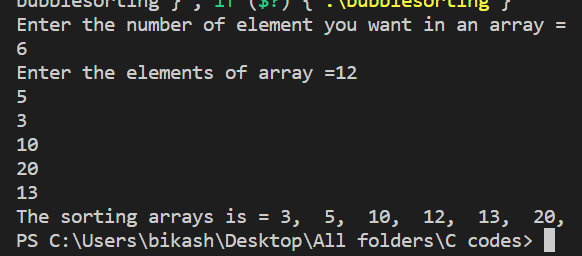
for(int i=0;i<n;i++)

{

printf(" %d, ",a[i]);

}

}



* **Selection sort**

#include<stdio.h>

int main()

{

int a[5]={25,11,23,45,10};

int swap;

int min;

for(int i=0;i<5;i++)

{

int min=i;

for(int j=i+1;j<5;j++){

if(a[min]>a[j])

{

min=j;

}

swap=a[min];

a[min]=a[i];

a[i]=swap;

}

}

printf("Sorted array is =\n");

for(int i=0;i<5;i++)

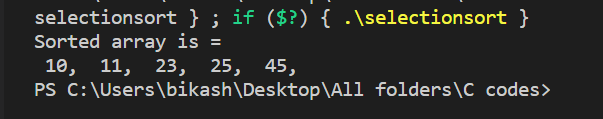
{

printf(" %d, ",a[i]);

}

return 0;

}



● **Insertion sort**

#include <stdio.h>

#include<math.h>

int main()

{

int a[5]={3,23,10,5,50};

int swap;

int i,j;

int current;

for(i=1;i<5;i++)

{

int current=a[i];

j=i-1;

while(current<a[j]&&j>=0)

{

a[j+1]=a[j];

j=j-1;

}

a[j+1]=current;

}

printf("\n Sorted array is =");

for(i=0;i<5;i++)

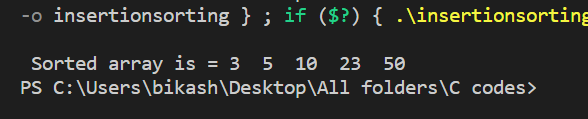
{

printf(" %d ",a[i]);

}

return 0;

}



**● Merge sort**

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

for (i = 0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

int main()

{

int arr[] = { 50, 40, 33, 10, 60, 25 };

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}



* **Quick sort**

#include <stdio.h>

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int array[], int low, int high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int array[], int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

void printArray(int array[], int size) {

for (int i = 0; i < size; ++i) {

printf("%d ", array[i]);

}

printf("\n");

}

int main() {

int data[] = {80, 70, 20, 10, 5, 90, 66};

int n = sizeof(data) / sizeof(data[0]);

printf("Unsorted Array\n");

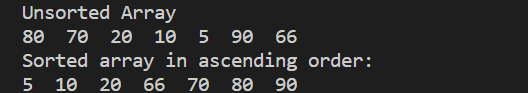
printArray(data, n);

quickSort(data, 0, n - 1);

printf("Sorted array in ascending order: \n");

printArray(data, n);

}



**10. Write program to search an element in a given list using**

**● Linear search**

#include<stdio.h>

int main()

{

int a[50];

int size,i;

int x;

printf("Enter the size of the array = ");

scanf("%d",&size);

printf("Enter the elements of an array =");

for(int i=0;i<size;i++)

{

scanf("%d",&a[i]);

}

int check ;

printf("Enter the elements to check in an array =");

scanf("%d",&check);

for(int i=0;i<size;i++)

{

if(check==a[i])

{

x=1;

}

}

if(x==1)

{

printf("Element is founded,");

}

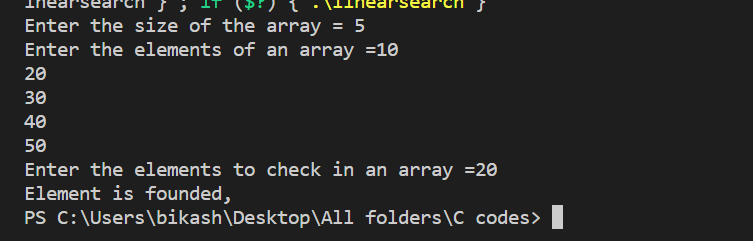
else

{

printf("Element is not found.");

}

}



**● Binary search**

#include <stdio.h>

void main(){

int n,item;

printf("enter the size of array =");

scanf("%d",&n);

int a[n];

printf("Enter the elements to the array =");

for (int i=0;i<n;i++){

scanf("%d",&a[i]);

}

int first =0;

int last=n-1;

printf("The item to searching =");

scanf("%d",&item);

int middle=(first+last )/2;

while (first<=last ){

if(a[middle]==item){

printf("Index no. of the item is =%d",middle);

break;

}

else if(item>a[middle]){

first=middle+1;

}

else

{

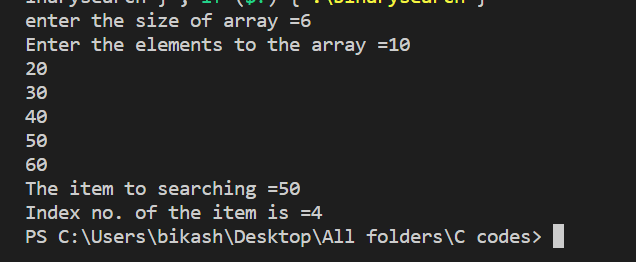
last=middle-1;

}

middle=(first+last)/2;

}

}



**2. Write a program to convert an infix arithmetic expression into postfix notation.**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top=-1;

char temp;

void push(char x)

{

top++;

stack[top] = x;

}

char pop()

{

if(top==-1)

{

return -1;

}else{

temp = stack[top];

top--;

return temp;

}

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

void main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

printf("\n");

e=exp;

scanf("%s",exp);

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

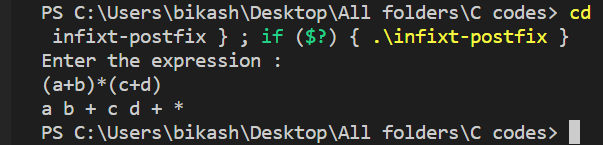
{

printf("%c ",pop());

}

}

**Output:-**



**3. Write a program to evaluate a postfix expression.**

#include<stdio.h>

#include<ctype.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression = ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

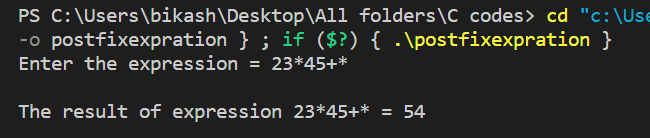
}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}

**Output:-**



**4. Write a program to perform the INSERT and DELETE operations on a simple queue using array.**

#include<stdio.h>

#define size 10

void enqueue();

void dequeue();

void display();

int rear=-1;

int front=-1;

int queue[size];

int main()

{

int ch;

do

{printf("\n1.Insert elements .\n");

printf("2.Remove elements .\n");

printf("3.Display all the elements.\n");

printf("4.Exit from queue.\n");

printf("Choose an operation form above operation from the above :\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

break;

default :

printf("Enter a valid option .\n");

}

}

while(ch<4);

return 0;

}

void enqueue()

{

int add;

if(rear>=size-1)

{

printf("Queue is overflow.");

}

else

{

printf("Enter the elements to add in queue =\n");

scanf("%d",&add);

rear++;

queue[rear]=add;

if(front==-1)

{front++;}

}

}

void dequeue()

{

if(front==-1)

{

printf("Queue is overflow.\n");

return ;

}

else

{

if(front==rear)

{front=rear=-1;}

else{

printf("Successfully Deleted and deleted element is =%d \n",queue[front]);

front++; }

}

}

void display()

{

if(front==-1)

{

printf("Queue is overflow");

}

printf("Elements of queue = ");

for(int i=front ;i<=rear;i++)

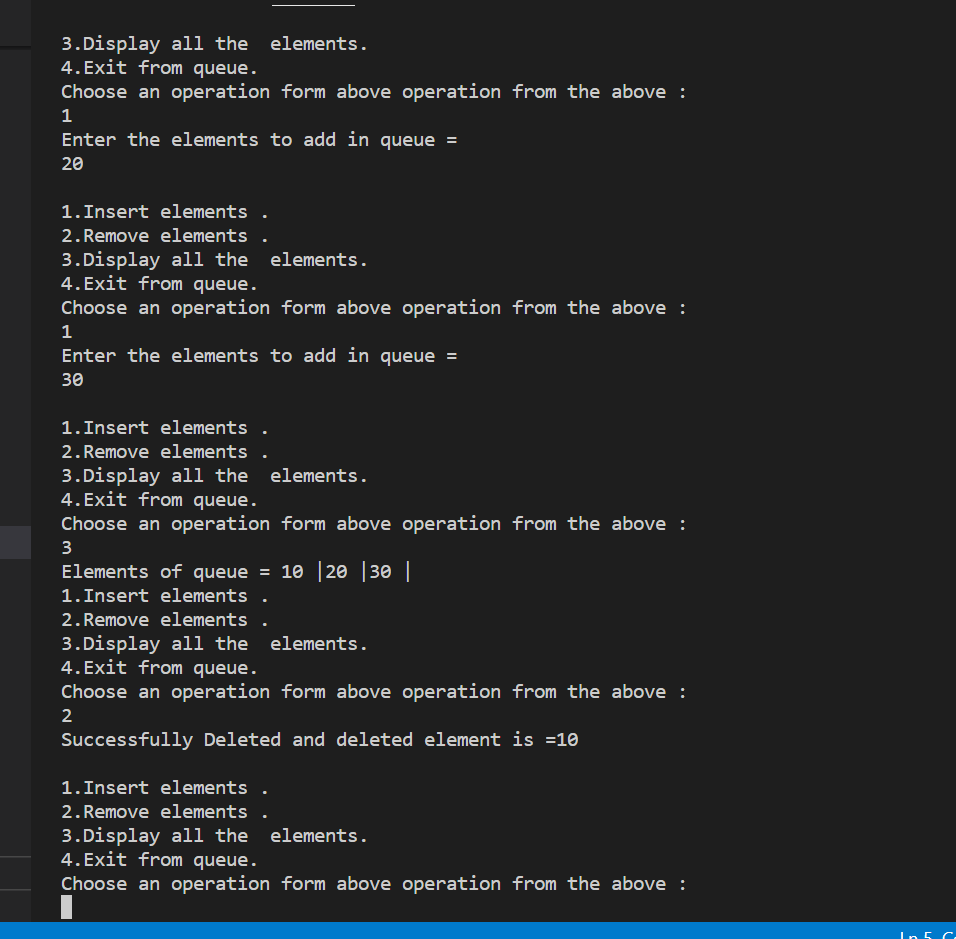
{

printf("%d |",queue[i]);

}

}

**Output:-**



**5. Write a program to perform the INSERT and DELETE operations on a simple queue using linked list.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front;

struct node \*rear;

void insert();

void delete();

void display();

void main ()

{

int choice;

while(choice != 4)

{

printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

printf("\nEnter your choice : ");

scanf("%d",& choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nEnter valid choice??\n");

}

}

}

void insert()

{

struct node \*ptr;

int item;

ptr = (struct node \*) malloc (sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

return;

}

else

{

printf("\nEnter value : ");

scanf("%d",&item);

ptr -> data = item;

if(front == NULL)

{

front = ptr;

rear = ptr;

front -> next = NULL;

rear -> next = NULL;

}

else

{

rear -> next = ptr;

rear = ptr;

rear->next = NULL;

}

}

}

void delete ()

{

struct node \*ptr;

if(front == NULL)

{

printf("\nUNDERFLOW\n");

return;

}

else

{

ptr = front;

front = front -> next;

free(ptr);

printf("%d Succuesfully deleted from linklist.\n",\*ptr);

}

}

void display()

{

struct node \*ptr;

ptr = front;

if(front == NULL)

{

printf("\nEmpty queue\n");

}

else

{ printf("Linklist values are =");

while(ptr != NULL)

{

printf("%d |",ptr -> data);

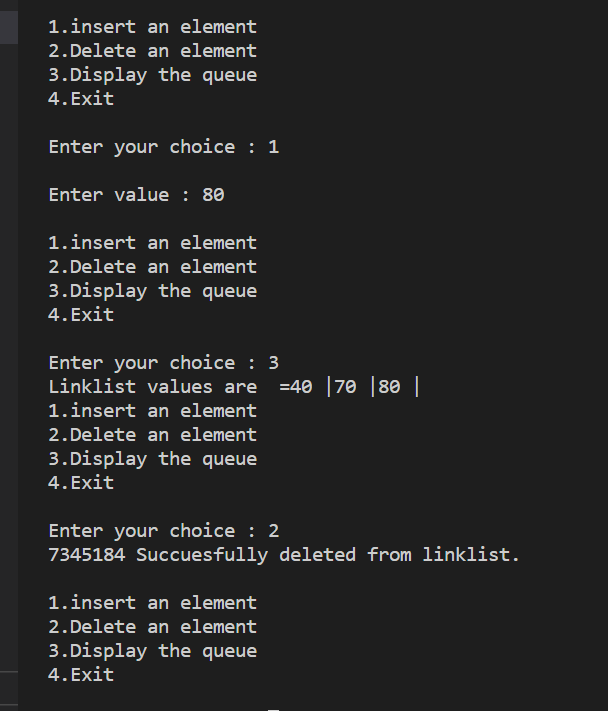
ptr = ptr -> next;

}

}

}

**Output:-**



**6.Write a program to perform the INSERT and DELETE operationson a circular queue.**

#include<stdio.h>

# define MAX 5

int cqueue\_arr[MAX];

int front = -1;

int rear = -1;

void insert(int item)

{

if((front == 0 && rear == MAX-1) || (front == rear+1))

{

printf("Queue Overflow \n");

return;

}

if(front == -1)

{

front = 0;

rear = 0;

}

else

{

if(rear == MAX-1)

rear = 0;

else

rear = rear+1;

}

cqueue\_arr[rear] = item ;

}

void deletion()

{

if(front == -1)

{

printf("Queue Underflow");

return ;

}

printf("Element deleted from queue is : %d\n",cqueue\_arr[front]);

if(front == rear)

{

front = -1;

rear=-1;

}

else

{

if(front == MAX-1)

front = 0;

else

front = front+1;

}

}

void display()

{

int front\_pos = front,rear\_pos = rear;

if(front == -1)

{

printf("Queue is empty");

return;

}

printf("Queue elements :");

if( front\_pos <= rear\_pos )

while(front\_pos <= rear\_pos)

{

printf("%d | ",cqueue\_arr[front\_pos]);

front\_pos++;

}

else

{

while(front\_pos <= MAX-1)

{

printf("%d ",cqueue\_arr[front\_pos]);

front\_pos++;

}

front\_pos = 0;

while(front\_pos <= rear\_pos)

{

printf("%d |",cqueue\_arr[front\_pos]);

front\_pos++;

}

}

printf("\n");

}

int main()

{

int choice,item;

do

{

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1 :

printf("Input the element for insertion in queue : ");

scanf("%d", &item);

insert(item);

break;

case 2 :

deletion();

break;

case 3:

display();

break;

case 4:

break;

default:

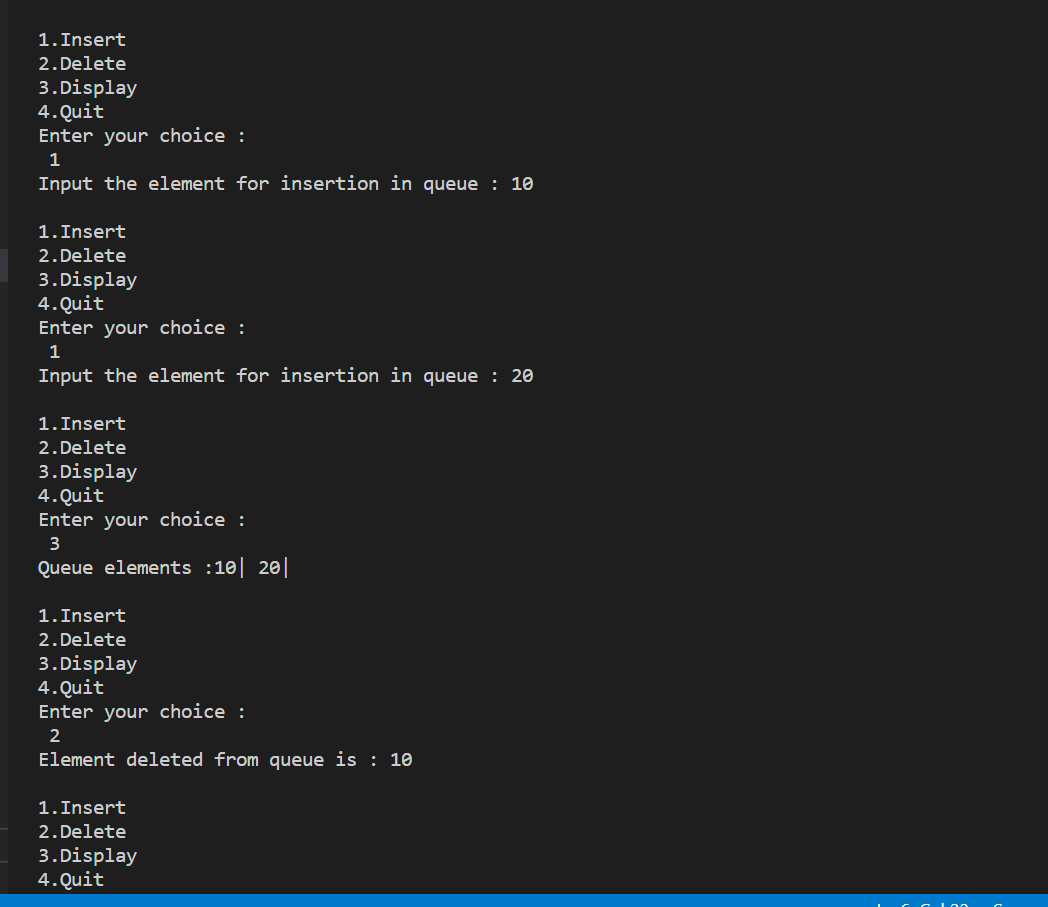
printf("Wrong choicen");

}

}while(choice!=4);

return 0;

}



**7.Write a Program to implement Double ended queue.**

# # include<stdio.h>

# # define Size 5

# 

# int deque\_arr[Size];

# int front = -1;

# int rear = -1;

# 

# void insert\_rear()

# {

# int added\_item;

# if((front == 0 && rear == Size-1) || (front == rear+1))

# { printf("Queue Overflow\n");

# return;}

# if (front == -1) /\* if queue is initially empty \*/

# { front = 0;

# rear = 0;}

# else

# if(rear == Size-1) /\*rear is at last position of queue \*/

# rear = 0;

# else

# rear = rear+1;

# 

# printf("Input the element for adding in queue : ");

# scanf("%d", &added\_item);

# deque\_arr[rear] = added\_item ;

# }

# 

# void insert\_front()

# { int added\_item;

# if((front == 0 && rear == Size-1) || (front == rear+1))

# { printf("Queue Overflow \n");

# return; }

# if (front == -1)/\*If queue is initially empty\*/

# { front = 0;

# rear = 0; }

# else

# if(front== 0)

# front=Size-1;

# else

# front=front-1;

# printf("Input the element for adding in queue : ");

# scanf("%d", &added\_item);

# deque\_arr[front] = added\_item ; }

# 

# void delete\_front()

# { if (front == -1)

# { printf("Queue Underflow\n");

# return ;

# }

# printf("Element deleted from queue is : %d\n",deque\_arr[front]);

# if(front == rear) /\*Queue has only one element \*/

# { front = -1;

# rear=-1;

# }

# else

# if(front == Size-1)

# front = 0;

# else

# front = front+1;

# }

# 

# void delete\_rear()

# {

# if (front == -1)

# {

# printf("Queue Underflow\n");

# return ;

# }

# printf("Element deleted from queue is : %d\n",deque\_arr[rear]);

# if(front == rear) /\*queue has only one element\*/

# {

# front = -1;

# rear=-1;

# }

# else

# if(rear == 0)

# rear=Size-1;

# else

# rear=rear-1; }

# void display\_queue()

# {

# int front\_pos = front,rear\_pos = rear;

# 

# if(front == -1)

# { printf("Queue is empty\n");

# return;

# }

# printf("Queue elements :\n");

# if( front\_pos <= rear\_pos )

# {

# while(front\_pos <= rear\_pos)

# {

# printf("%d ",deque\_arr[front\_pos]);

# front\_pos++;

# }

# }

# else

# {

# while(front\_pos <= Size-1)

# { printf("%d ",deque\_arr[front\_pos]);

# front\_pos++;

# }

# front\_pos = 0;

# while(front\_pos <= rear\_pos)

# {

# printf("%d ",deque\_arr[front\_pos]);

# front\_pos++;

# }

# }

# printf("\n");

# }

# void input\_que()

# { int choice;

# do

# { printf("1.Insert at rear\n");

# printf("2.Delete from front\n");

# printf("3.Delete from rear\n");

# printf("4.Display\n");

# printf("5.Quit\n");

# printf("Enter your choice : ");

# scanf("%d",&choice);

# 

# switch(choice)

# { case 1:

# insert\_rear();

# break;

# case 2:

# delete\_front();

# break;

# case 3:

# delete\_rear();

# break;

# case 4:

# display\_queue();

# break;

# case 5:

# break;

# default:

# printf("Wrong choice\n");

# }

# }while(choice!=5);

# }

# void output\_que()

# { int choice;

# do

# { printf("1.Insert at rear\n");

# printf("2.Insert at front\n");

# printf("3.Delete from front\n");

# printf("4.Display\n");

# printf("5.Quit\n");

# printf("Enter your choice : ");

# scanf("%d",&choice);

# switch(choice)

# {

# case 1:

# insert\_rear();

# break;

# case 2:

# insert\_front();

# break;

# case 3:

# delete\_front();

# break;

# case 4:

# display\_queue();

# break;

# case 5:

# break;

# default:

# printf("Wrong choice\n");

# }

# }while(choice!=5);

# }

# main()

# { int choice;

# printf("1.Input restricted dequeue\n");

# printf("2.Output restricted dequeue\n");

# printf("Enter your choice : ");

# scanf("%d",&choice);

# switch(choice)

# {

# case 1 :

# input\_que();

# break;

# case 2:

# output\_que();

# break;

# default:

# printf("Wrong choice\n");

# }

# }

# 

**8.Write program perform the following operations on a singly linked list**

# Insert an element

# Delete an element

# Find the sum of the element in the list

# Count number of nodes in the linked list

# Search a given element in the linked list

# Reverse the linked list

# Make a copy of the linked list

# Concatenate two linked list

# Merge two linked list

#include <stdio.h>

#include <stdlib.h>

struct node {

int info;

struct node\* link;

};

struct node\* start = NULL;

void createList()

{

if (start == NULL) {

int n;

printf("\nEnter the number of nodes: ");

scanf("%d", &n);

if (n != 0) {

int data;

struct node\* newnode;

struct node\* temp;

newnode = malloc(sizeof(struct node));

start = newnode;

temp = start;

printf("\nEnter number to"

" be inserted : ");

scanf("%d", &data);

start->info = data;

for (int i = 2; i <= n; i++) {

newnode = malloc(sizeof(struct node));

temp->link = newnode;

printf("\nEnter number to"

" be inserted : ");

scanf("%d", &data);

newnode->info = data;

temp = temp->link;

}

}

printf("\nThe list is created\n");

}

else

printf("\nThe list is already created\n");

}

// Function to traverse the linked list

void traverse()

{

struct node\* temp;

// List is empty

if (start == NULL)

printf("\nList is empty\n");

// Else print the LL

else {

temp = start;

while (temp != NULL) {

printf("Data = %d\n", temp->info);

temp = temp->link;

}

}

}

void insertAtFront()

{

int data;

struct node\* temp;

temp = malloc(sizeof(struct node));

printf("\nEnter number to"

" be inserted : ");

scanf("%d", &data);

temp->info = data;

temp->link = start;

start = temp;

}

void insertAtEnd()

{

int data;

struct node \*temp, \*head;

temp = malloc(sizeof(struct node));

// Enter the number

printf("\nEnter number to"

" be inserted : ");

scanf("%d", &data);

// Changes links

temp->link = 0;

temp->info = data;

head = start;

while (head->link != NULL) {

head = head->link;

}

head->link = temp;

}

// Function to insert at any specified

// position in the linked list

void insertAtPosition()

{

struct node \*temp, \*newnode;

int pos, data, i = 1;

newnode = malloc(sizeof(struct node));

// Enter the position and data

printf("\nEnter position and data :");

scanf("%d %d", &pos, &data);

// Change Links

temp = start;

newnode->info = data;

newnode->link = 0;

while (i < pos - 1) {

temp = temp->link;

i++;

}

newnode->link = temp->link;

temp->link = newnode;

}

// Function to delete from the front

// of the linked list

void deleteFirst()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else {

temp = start;

start = start->link;

free(temp);

}

}

// Function to delete from the end

// of the linked list

void deleteEnd()

{

struct node \*temp, \*prevnode;

if (start == NULL)

printf("\nList is Empty\n");

else {

temp = start;

while (temp->link != 0) {

prevnode = temp;

temp = temp->link;

}

free(temp);

prevnode->link = 0;

}

}

void deletePosition()

{

struct node \*temp, \*position;

int i = 1, pos;

if (start == NULL)

printf("\nList is empty\n");

else {

printf("\nEnter index : ");

scanf("%d", &pos);

position = malloc(sizeof(struct node));

temp = start;

// Traverse till position

while (i < pos - 1) {

temp = temp->link;

i++;

}

// Change Links

position = temp->link;

temp->link = position->link;

// Free memory

free(position);

}

}

void maximum()

{

int a[10];

int i;

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

// Otherwise

else {

temp = start;

int max = temp->info;

while (temp != NULL) {

if (max < temp->info)

max = temp->info;

temp = temp->link;

}

printf("\nMaximum number "

"is : %d ",

max);

}

}

void mean()

{

int a[10];

int i;

struct node\* temp;

// If LL is empty

if (start == NULL)

printf("\nList is empty\n");

// Otherwise

else {

temp = start;

// Stores the sum and count of

// element in the LL

int sum = 0, count = 0;

float m;

// Traverse the LL

while (temp != NULL) {

// Update the sum

sum = sum + temp->info;

temp = temp->link;

count++;

}

m = sum / count;

printf("\nMean is %f ", m);

}

}

void sort()

{

struct node\* current = start;

struct node\* index = NULL;

int temp;

if (start == NULL) {

return;

}

else {

while (current != NULL) {

index = current->link;

while (index != NULL) {

if (current->info > index->info) {

temp = current->info;

current->info = index->info;

index->info = temp;

}

index = index->link;

}

current = current->link;

}

}

}

void reverseLL()

{

struct node \*t1, \*t2, \*temp;

t1 = t2 = NULL;

// If LL is empty

if (start == NULL)

printf("List is empty\n");

else {

while (start != NULL) {

t2 = start->link;

start->link = t1;

t1 = start;

start = t2;

}

start = t1;

temp = start;

printf("Reversed linked "

"list is : ");

// Print the LL

while (temp != NULL) {

printf("%d ", temp->info);

temp = temp->link;

}

}

}

// Driver Code

int main()

{

int choice;

while (1) {

printf("\n\t1 To see list\n");

printf("\t2 For insertion at"

" starting\n");

printf("\t3 For insertion at"

" end\n");

printf("\t4 For insertion at "

"any position\n");

printf("\t5 For deletion of "

"first element\n");

printf("\t6 For deletion of "

"last element\n");

printf("\t7 For deletion of "

"element at any position\n");

printf("\t8 To find maximum among"

" the elements\n");

printf("\t9 To find mean of "

"the elements\n");

printf("\t10 To sort element\n");

printf("\t11 To reverse the "

"linked list\n");

printf("\t12 To exit\n");

printf("\nEnter Choice :\n");

scanf("%d", &choice);

switch (choice) {

case 1:

traverse();

break;

case 2:

insertAtFront();

break;

case 3:

insertAtEnd();

break;

case 4:

insertAtPosition();

break;

case 5:

deleteFirst();

break;

case 6:

deleteEnd();

break;

case 7:

deletePosition();

break;

case 8:

maximum();

break;

case 9:

mean();

break;

case 10:

sort();

break;

case 11:

reverseLL();

break;

case 12:

exit(1);

break;

default:

printf("Incorrect Choice\n");

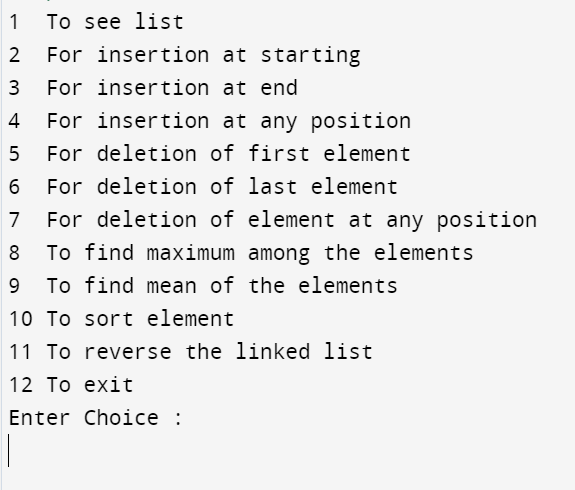
}

}

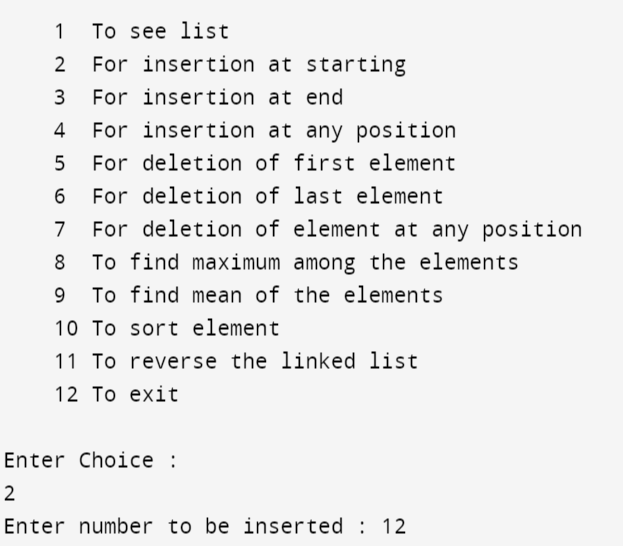
return 0;

}

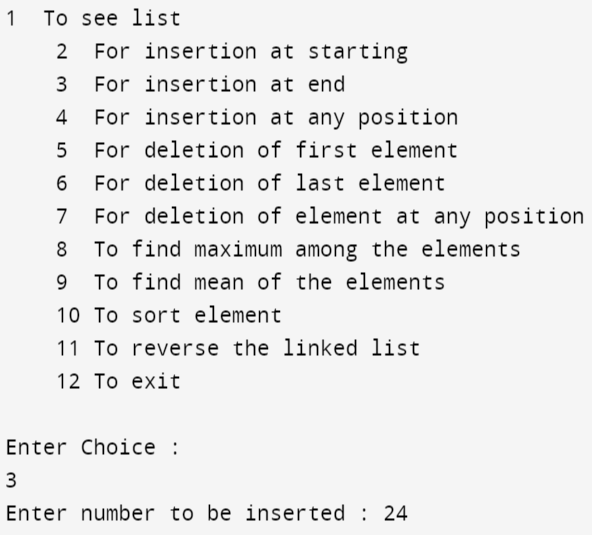
**OUTPUT:**

****

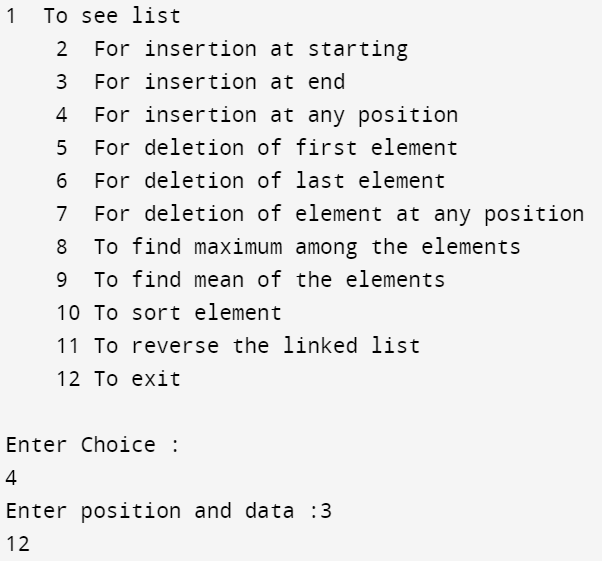
**For Insertion at starting:**

****

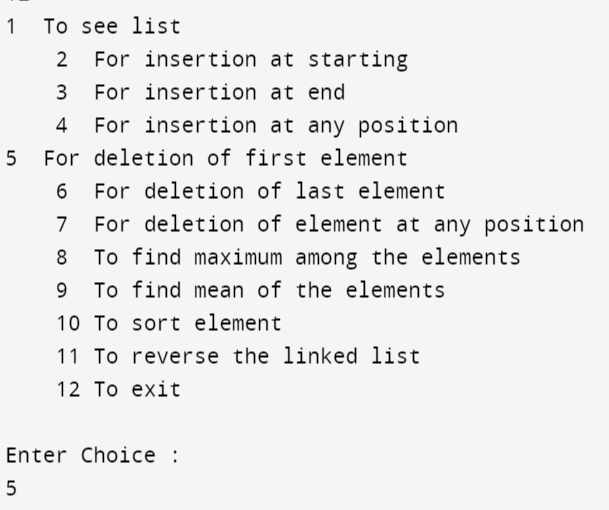
**For Insertion at end:**

****

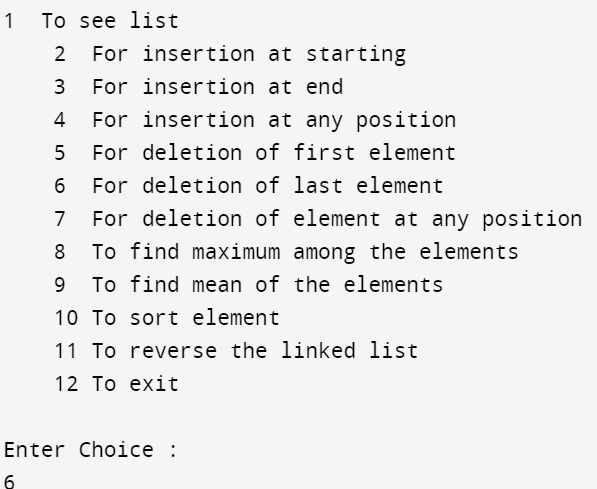
**For Insertion at any position:**

****

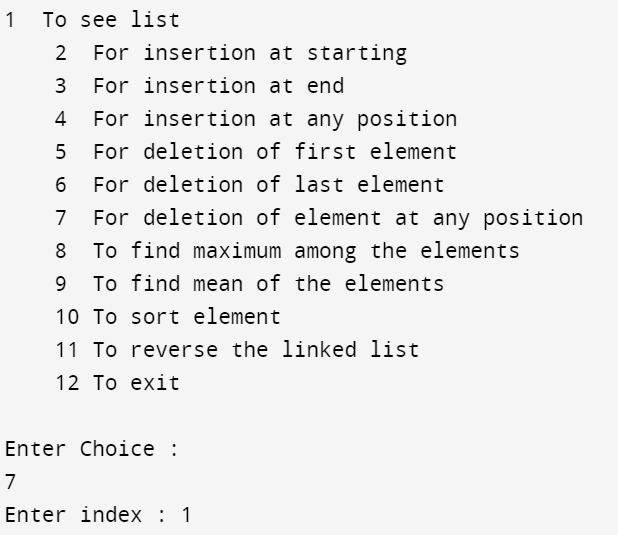
**For Deletion of first element:**

****

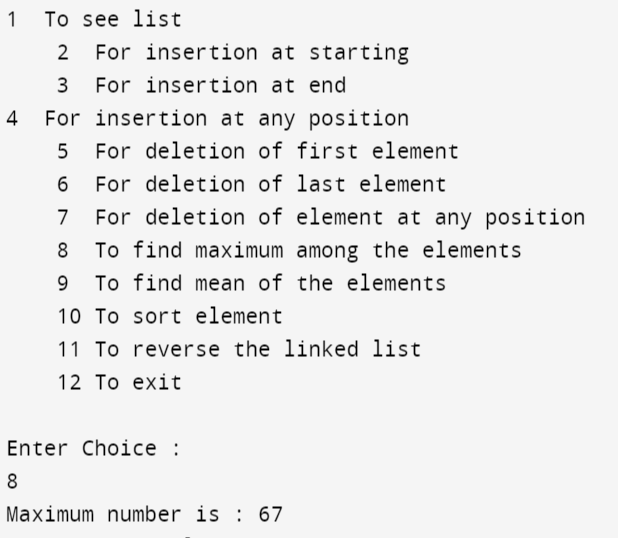
**For Deletion of last element:**

****

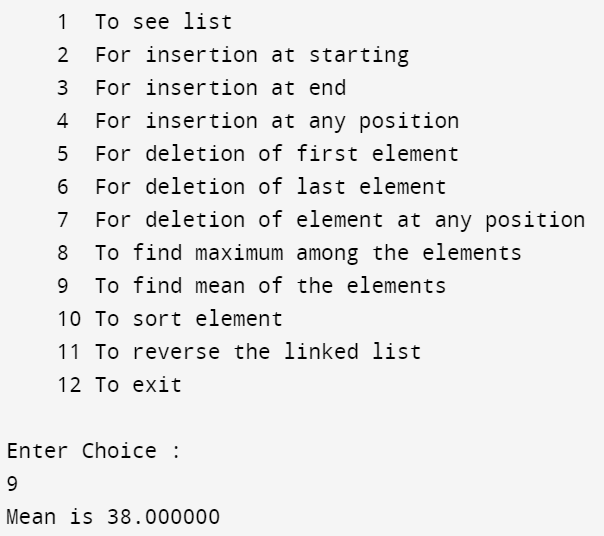
**For Deletion of element any position:**

****

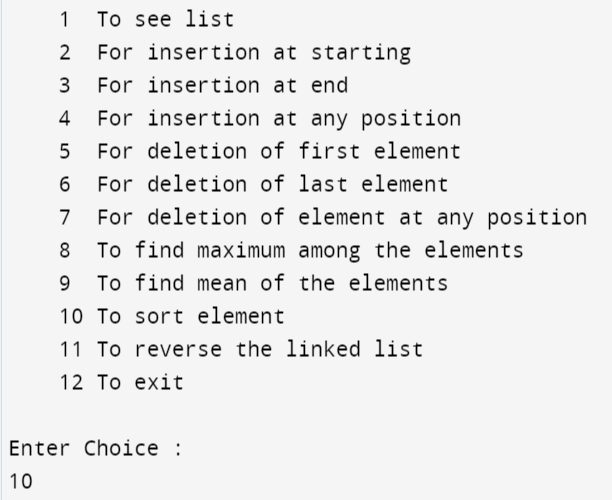
**To find maximum among the elements:**

****

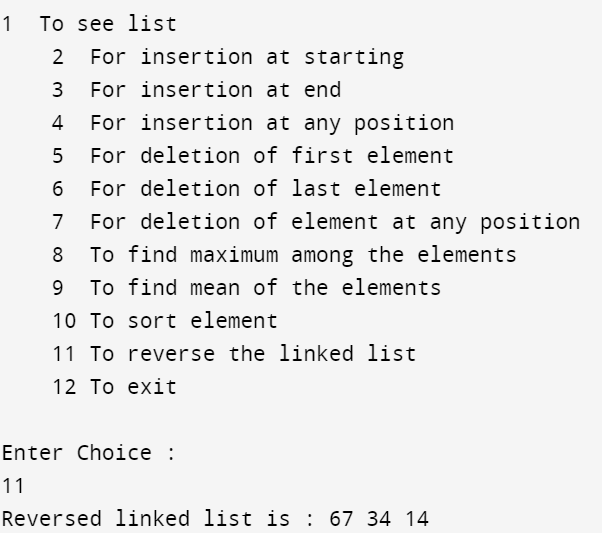
**To find mean of the element:**

****

**To sort element:**

****

**To reverse the linked list:**

****

* **Radix Sort**

#include <stdio.h>

int getMax(int a[], int n) {

int max = a[0];

for(int i = 1; i<n; i++) {

if(a[i] > max)

max = a[i];

}

return max;

}

void countingSort(int a[], int n, int place)

{

int output[n + 1];

int count[10] = {0};

for (int i = 0; i < n; i++)

count[(a[i] / place) % 10]++;

for (int i = 1; i < 10; i++)

count[i] += count[i - 1];

for (int i = n - 1; i >= 0; i--) {

output[count[(a[i] / place) % 10] - 1] = a[i];

count[(a[i] / place) % 10]--;

}

for (int i = 0; i < n; i++)

a[i] = output[i];

}

void radixsort(int a[], int n) {

int max = getMax(a, n);

for (int place = 1; max / place > 0; place \*= 10)

countingSort(a, n, place);

}

void printArray(int a[], int n) {

for (int i = 0; i < n; ++i) {

printf("%d ", a[i]);

}

printf("\n");

}

int main() {

int a[] = {181, 289, 390, 121, 145, 736, 514, 888, 122};

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArray(a,n);

radixsort(a, n);

printf("After applying Radix sort, the array elements are - \n");

printArray(a, n);

}

