**Practical:-1**

**1. Create a Structure with following Data Members: 1. Integer Array 2. Size of the Array Sort the Array using various Sorting algorithms such as**

**(i) Selection Sort**

**(ii) Bubble Sort**

**(iii) Two-way Merge Sort**

**(iv) Quick Sort**

**(v) Heap Sort. And store the sorted Array in a text file.**

**//(i) Selection Sort**

#include<stdio.h>

#include<conio.h>

void main()

{

int i,j,a[10],n;

int temp;

printf("Enter Size: ");

scanf("%d",&n);

printf("Enter Arrays :\n");

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

{

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

}

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

{

for(j=i+1;j<n;j++)

{

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

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("insertion Sort Array is: ");

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

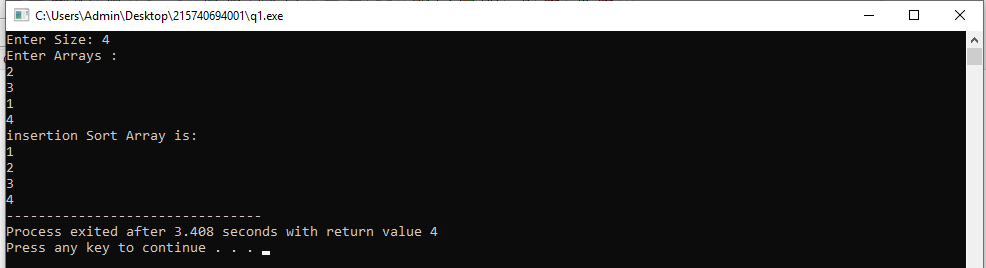
{

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

}

}

**OutPut:-**



**//(ii) Bubble Sort**

#include<stdio.h>

void main()

{

int n,i,j,a[10],temp;

printf("Enter Size:");

scanf("%d",&n);

printf("Enter Arrays:\n");

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

{

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

}

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

{

for(j=i+1;j<n;j++)

{

if(a[j]<a[i])

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("Bubble sort is:");

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

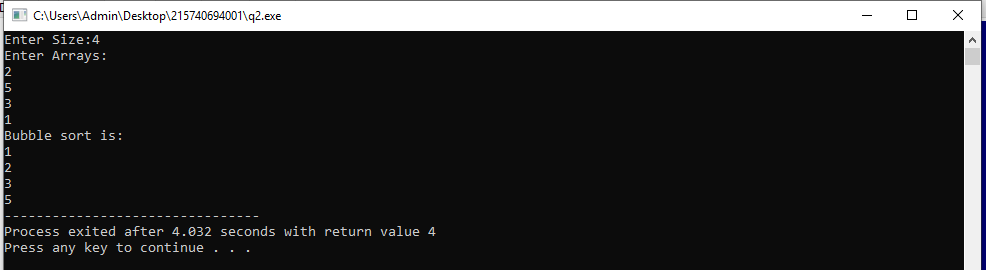
{

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

}

}

**Output:-**

****

**//(iii) Two-way Merge Sort**

#include <stdio.h>

int a[100];

int b[100];

void merging(int low, int mid, int high) {

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++)

a[i] = b[i];

}

void sort(int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merging(low, mid, high);

} else {

return;

}

}

int main() {

int i,max;

printf("Enter array Size\n");

scanf("%d",&max);

printf("Enter array \n");

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

{

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

}

printf("List before sorting\n");

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

{

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

}

sort(0, max);

printf("\nList after sorting\n");

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

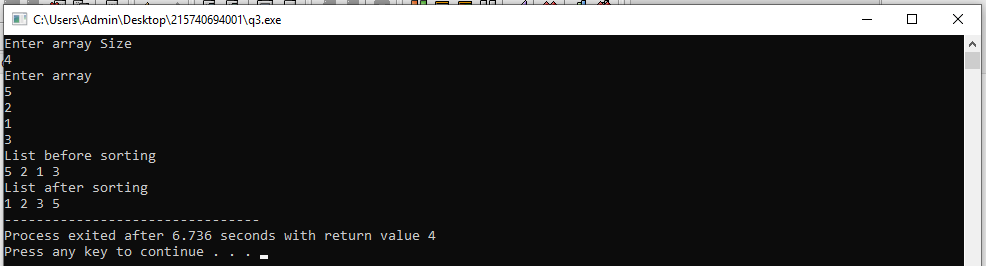
{

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

}

}

**Output:-**

****

**// (iv) Quick Sort**

#include<stdio.h>

#include<conio.h>

void quickSort(int [10],int,int);

void main()

{

int a[20],size,i;

printf("Enter Array size: ");

scanf("%d",&size);

printf("Enter values: \n");

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

{

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

}

quickSort(a,0,size-1);

printf("Array sorting is: ");

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

{

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

}

getch();

}

void quickSort(int a[10],int first,int last)

{

int pivot,i,j,temp;

if(first < last){

pivot = first;

i = first;

j = last;

while(i < j){

while(a[i] <= a[pivot] && i < last)

i++;

while(a[j] > a[pivot])

j--;

if(i <j){

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

temp = a[pivot];

a[pivot] = a[j];

a[j] = temp;

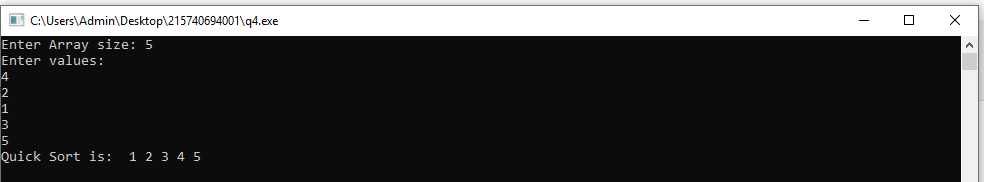
quickSort(a,first,j-1);

quickSort(a,j+1,last);

}

}

**Output:-**



**//(v) Heap Sort**

#include <stdio.h>

void heapify(int a[], int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && a[left] > a[largest])

{

largest = left;

}

if (right < n && a[right] > a[largest])

{

largest = right;

}

if (largest != i)

{

int temp = a[i];

a[i] = a[largest];

a[largest] = temp;

heapify(a, n, largest);

}

}

/\*Function to implement the heap sort\*/

void heapSort(int a[], int n)

{

int i;

for (i = n / 2 - 1; i >= 0; i--)

heapify(a, n, i);

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

int temp = a[0];

a[0] = a[i];

a[i] = temp;

heapify(a, i, 0);

}

}

void printArr(int arr[], int n)

{

int i;

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

{

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

printf(" ");

}

}

void main()

{

int a[] = {48, 10, 23, 43, 28, 26, 1};

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

printf("Before sorting array \n");

printArr(a, n);

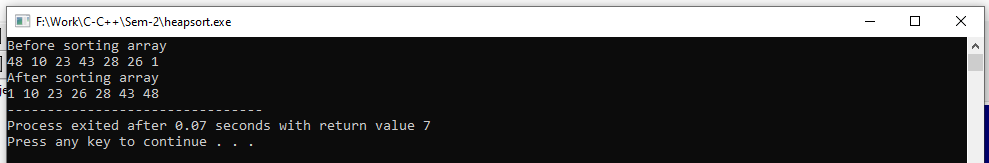
heapSort(a, n);

printf("\nAfter sorting array \n");

printArr(a, n);

}

**Output:-**



**Practical:-2**

**2. Create a Structure with following Data Members:**

**1. Integer Array**

**2. Size of the Array**

**Search an element in Array using Linear (Sequential) Search and Binary Search, and Display result in file. For Sequential Search, assume that input array is Unordered and for Binary Search assume that input array is Ordered and develop programs accordingly.**

#include<stdio.h>

#include<conio.h>

void linear\_search(int \*k,int m,int x)

{

int i,c=0;

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

if (x == k[i])

{

c=i;

}

if(c==0)

{

printf("Element is not present in array\n");

}

else

{

printf("Element is present at index %d\n", c);

}

}

int binary\_search(int \*k,int l,int m, int x)

{

if (m >= l) {

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

if (k[mid] == x)

{

return mid;

}

if (k[mid] > x)

{

return binary\_search(k,l,mid - 1, x);

}

return binary\_search(k, mid + 1,m, x);

}

return -1;

}

void main()

{

int i,j,q,a[10],ch,x;

q=sizeof(a) / sizeof(a[0]);

do

{

printf("Main Menu");

printf("\n1. Input data");

printf("\n2. Linear Search");

printf("\n3. Binary Search");

printf("\n4. Exit");

printf("\nEnter Your Choice ");

scanf("%d",&ch);

if(ch==1)

{

printf("\nEnter data\n");

for(i=0;i<10;i++)

{

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

}

}

else if(ch==2)

{

printf("Enter No for Searching \n");

scanf("%d",&x);

linear\_search(a,q,x);

}

else if(ch==3)

{

int res;

printf("Enter No for Searching \n");

scanf("%d",&x);

res=binary\_search(a,0,q-1,x);

if(res==-1)

{

printf("Element is not present in array\n");

}

else

{

printf("Element is present at index %d \n", res);

}

}

else if(ch==4)

{

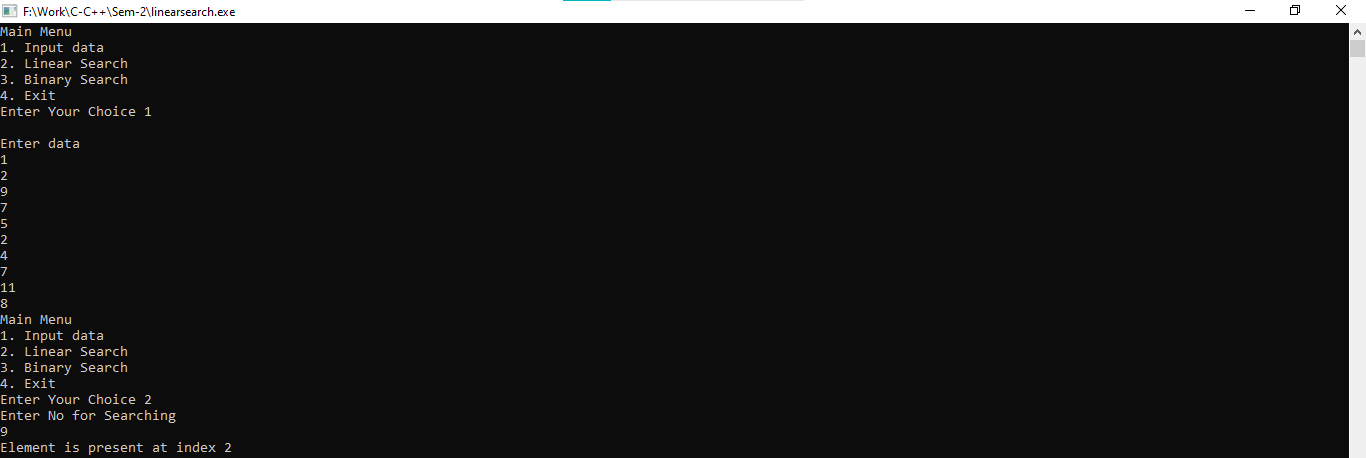
exit(0);

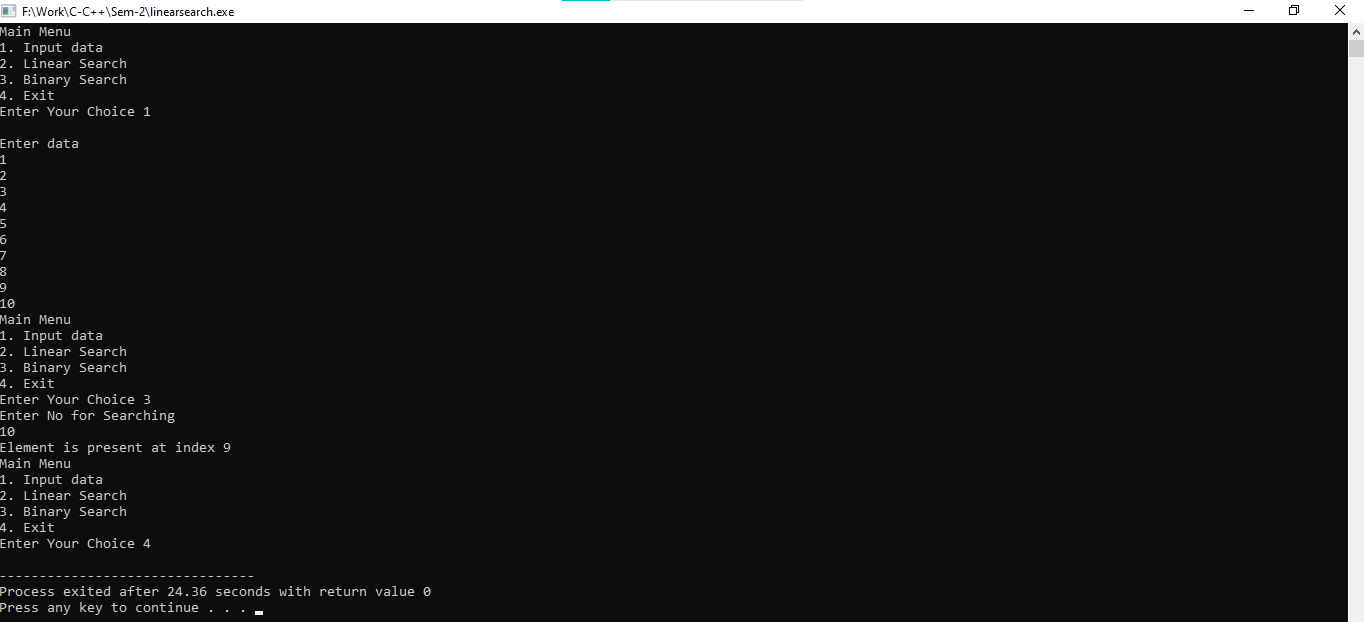
}

}while(1);

}

**Output:-**

****

****

**Practical:-3**

**3. Create a “Stack” data structure with following Data members: 1. Integer Array 2. Stack Pointer (Top of Stack: Is it same as the Size of the Array) Perform the following operations on stack using user-defined functions:**

**1. Push**

**2. Pop**

**3. Isempty**

**4. Isfull**

**5. Peep**

**Create a file which stores all values of Array through Stack. Has it reversed the order of the elements of the Array? Why?**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX\_SIZE 5

int item, c,ch, i;

int arr\_stack[MAX\_SIZE];

int top = 0;

void push() {

if (top == MAX\_SIZE)

printf("\n Stack is Full!");

else {

printf("\nEnter The Value to be pushed : ");

scanf("%d", &item);

printf("\n Position : %d ,Pushed Value : %d ", top, item);

arr\_stack[top++] = item;

}

}

void Isempty()

{

if(top==-1)

{

printf("\nEmpty");

}

else

{

printf("Not Empty");

}

}

void Isfull()

{

if(top==MAX\_SIZE)

{

printf("full\n");

}

else

{

printf("Not full\n");

}

}

int peep()

{

return arr\_stack[top];

}

void pop() {

if (top == 0)

printf("\n Stack is Empty!");

else {

--top;

printf("\n Position : %d , Popped Value : %d ", top, arr\_stack[top]);

}

}

void display() {

printf("\n Stack Size : %d ", top);

for (i = (top - 1); i >= 0; i--)

printf("\n Position : %d , Value : %d ", i, arr\_stack[i]);

}

void main() {

printf("\n Stack Operation");

do {

printf("\n1.Push \n2.Pop \n3.Isempty \n4.ISfull \n5.Peep \n6.Display \nOthers to exit");

printf("\nEnter Your Choice : ");

scanf("%d", &c);

switch (c) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

Isempty();

break;

case 4:

Isfull();

break;

case 5:

printf("%d Top Element of stack",peep());

break;

case 6:

display();

break;

default:

printf("Enter Proper Choice");

break;

}

printf("\n Enter y or Y to exit ");

ch=getch();

if(ch=='y' || ch=='Y')

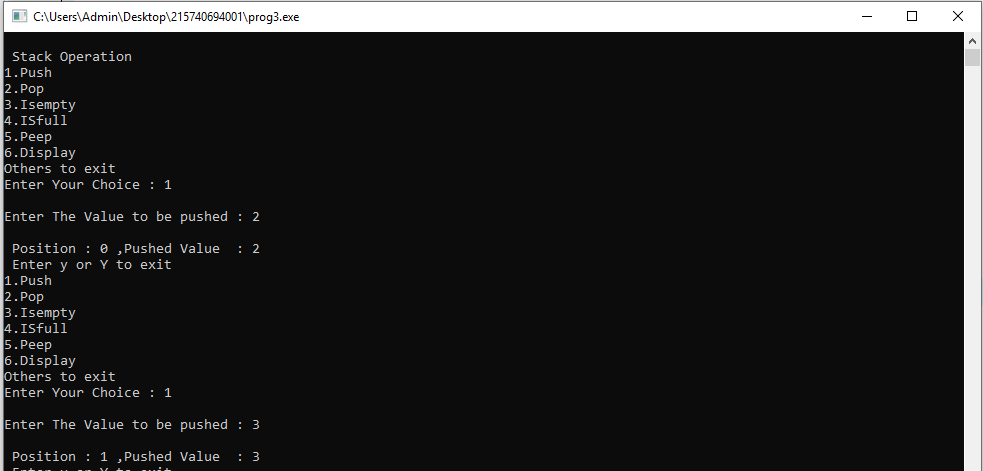
{

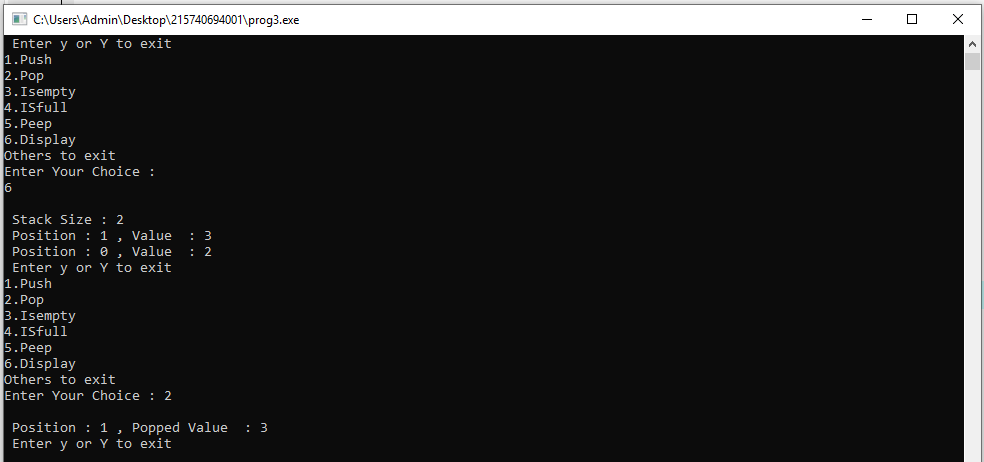
break;

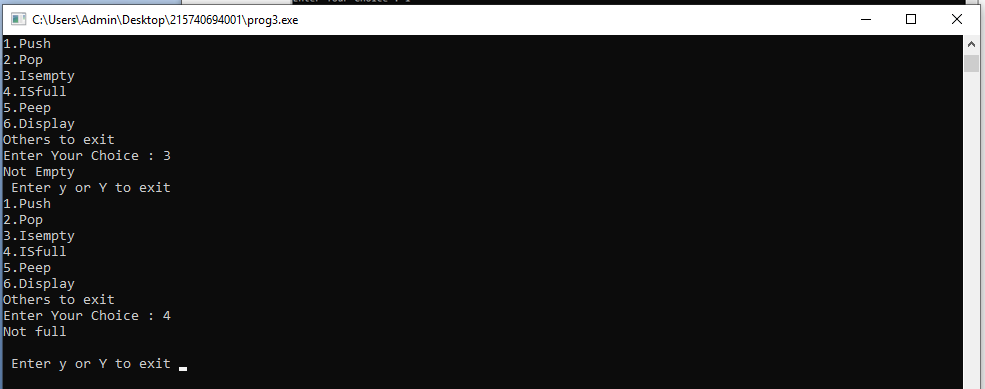
}

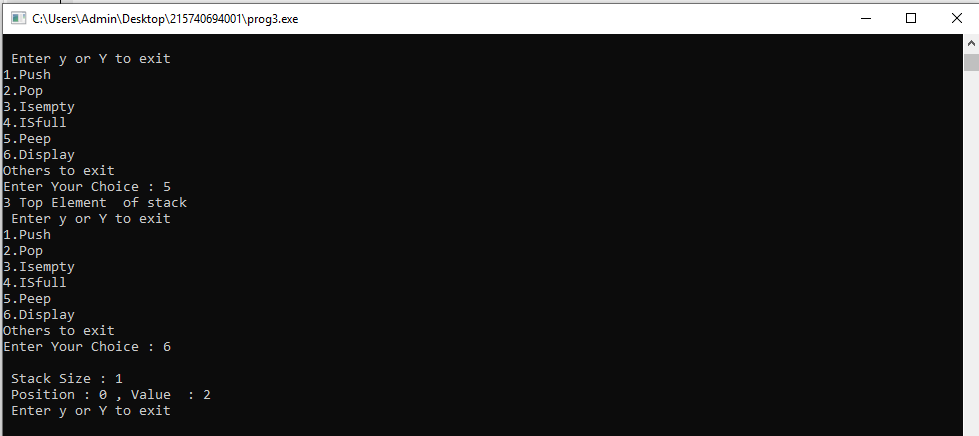
} while("true");

}

**Output:-**







**Practical:-4**

**4. Create a “Linked List” structure with the following data members: 1. A Data 2. A link to the next node Perform the following operations on stack using user-defined functions:**

**1. Insert a value X at the first place**

**2. Insert a value X at the end of the list**

**3. Insert a value X at the place so that it preserves the ordering of the terms in the increasing order.**

**i. Delete an element whose address is given by X**

**ii. Copy a linked linear list**

**Create a file which stores all values of list.**

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_pos();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

{

int choice;

while(1){

printf("\n MENU \n");

printf("\n 1.Create \n");

printf("\n 2.Display \n");

printf("\n 3.Insert at the beginning \n");

printf("\n 4.Insert at the end \n");

printf("\n 5.Insert at specified position \n");

printf("\n 6.Delete from specified position \n");

printf("\n 7.Exit \n");

printf("\n--------------------------------------\n");

printf("Enter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

insert\_begin();

break;

case 4:

insert\_end();

break;

case 5:

insert\_pos();

break;

case 6:

delete\_pos();

break;

case 7:

exit(0);

break;

default:

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

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

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

if(temp==NULL)

{

printf("\n Overflow \n");

exit(0);

}

printf("\nEnter node:\t");

scanf("%d",&temp->info);

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

printf("\n empty:\n");

return;

}

else

{

ptr=start;

printf("\n elements are: ");

while(ptr!=NULL)

{

printf("%d\t",ptr->info );

ptr=ptr->next ;

}

}

}

void insert\_begin()

{

struct node \*temp;

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

if(temp==NULL)

{

printf("\n empty \n");

return;

}

printf("\nEnter node:\t" );

scanf("%d",&temp->info);

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

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

if(temp==NULL)

{

printf("\n empty:\n");

return;

}

printf("\nEnter node:\t" );

scanf("%d",&temp->info );

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

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

if(temp==NULL)

{

printf("\nempty:\n");

return;

}

printf("\nEnter the position for inserted:\t");

scanf("%d",&pos);

printf("\nEnter node:\t");

scanf("%d",&temp->info) ;

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;

if(ptr==NULL)

{

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

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\n Empty:\n");

exit(0);

}

else

{

printf("\nEnter the position of deleted node:\t");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->next ;

printf("\nThe deleted node is:%d\t",ptr->info );

free(ptr);

}

else

{

ptr=start;

for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;

if(ptr==NULL)

{

printf("\nPosition not Found:\n");

return;

}

}

temp->next =ptr->next ;

printf("\nThe deleted node is:%d\t",ptr->info );

free(ptr);

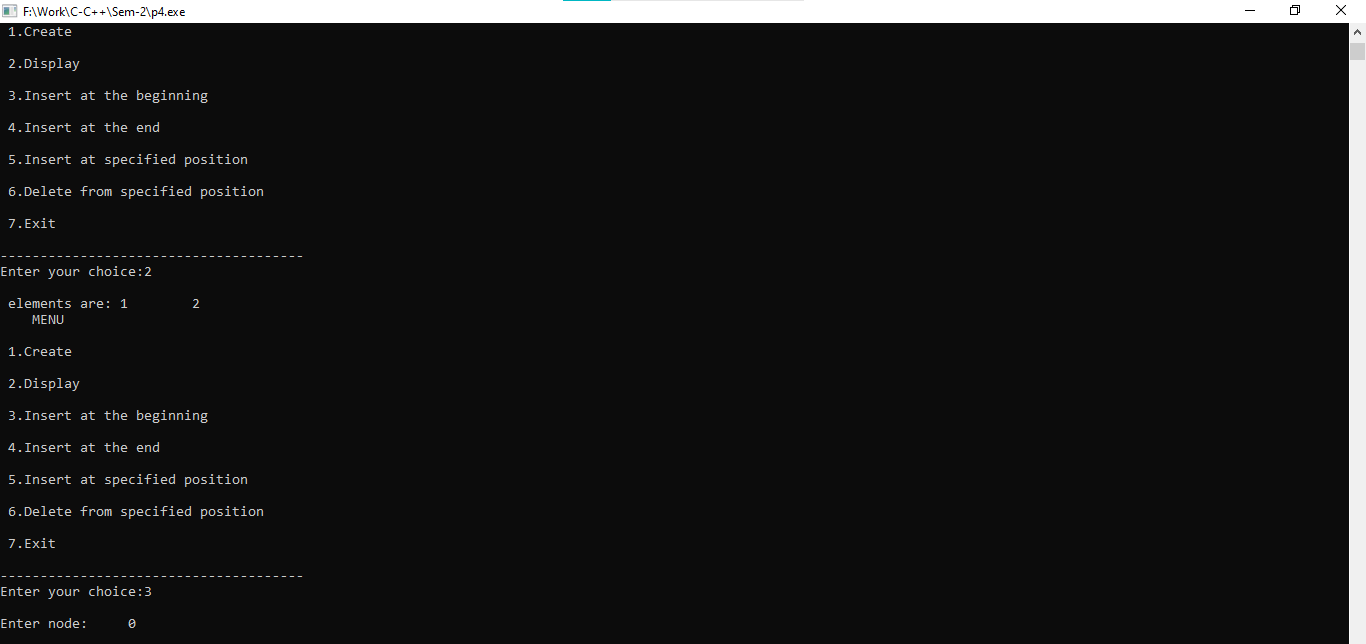
}

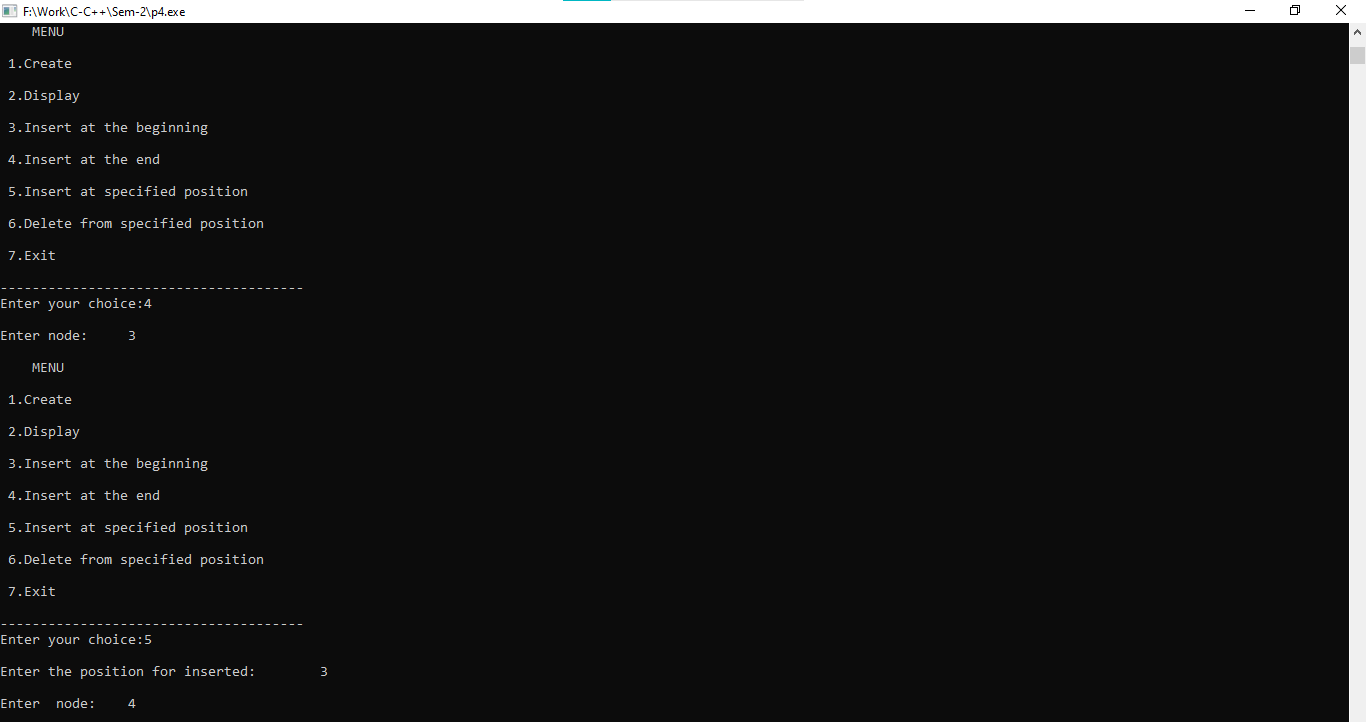
}

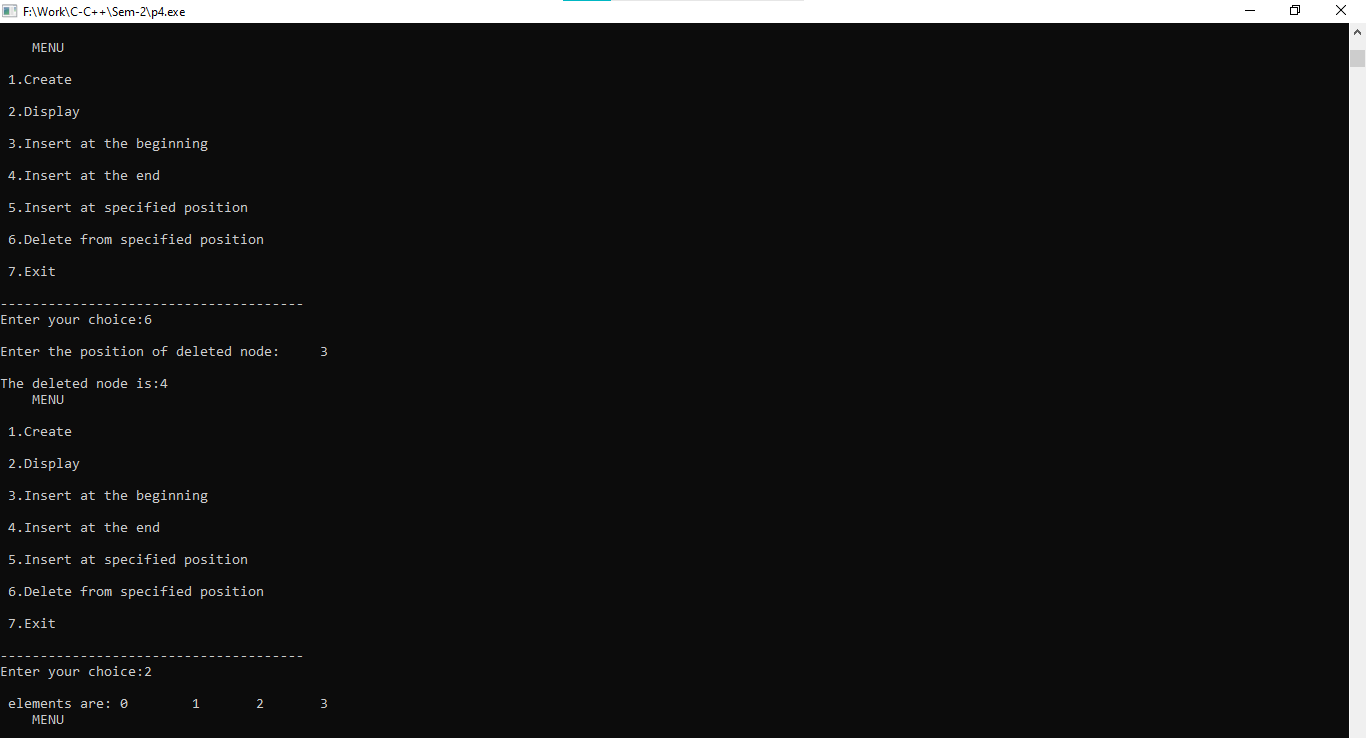
}

**Output:-**

****

****

****

****

**Practical:-5**

**5. Write a program to convert an infix arithmetic expression (parenthesize / unparenthesized) into postfix notation.**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

{

return -1;

}

else

{

return stack[top--];

}

}

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 : ");

scanf("%s",exp);

printf("\n");

e = exp;

printf("infix into postfix notation: ");

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)

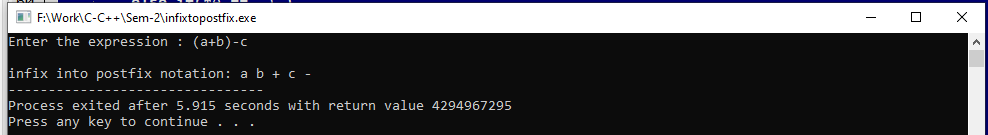
{

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

}

}

**Output:-**



**Practical:-6**

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

#include<stdio.h>

int s[20];

int top = -1;

void push(int x)

{

s[++top] = x;

}

int pop()

{

return s[top--];

}

void 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);

}

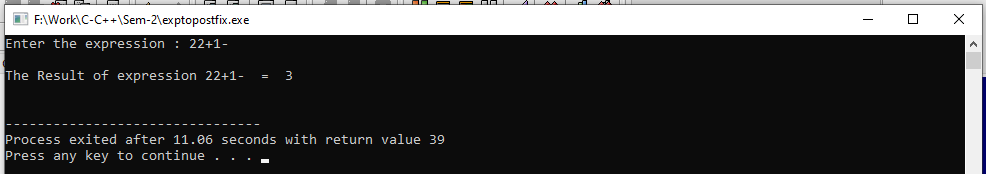
e++;

}

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

}

**Output:-**



**Practical:-7**

**7. Create a structure with the following Data members: 1. Integer Array 2. Size of the Array Search an element in a given list using Binary Search by recursion. And Display result in a file.**

#include<stdio.h>

int binary\_search(int \*k,int l,int m, int x)

{

if (m >= l) {

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

if (k[mid] == x)

{

return mid;

}

if (k[mid] > x)

{

return binary\_search(k,l,mid - 1, x);

}

return binary\_search(k, mid + 1,m, x);

}

return -1;

}

void main()

{

int i,q,a[10],x;

q=sizeof(a) / sizeof(a[0]);

int res;

printf("\nEnter 10 Numbers \n");

for(i=0;i<10;i++)

{

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

}

printf("Enter No for Searching: ");

scanf("%d",&x);

res=binary\_search(a,0,q-1,x);

if(res==-1)

{

printf("Element is not present in array\n");

}

else

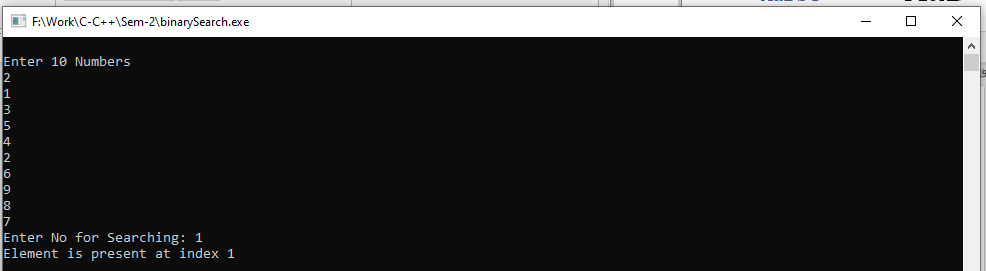
{

printf("Element is present at index %d \n", res);

}

}

**Output:-**



**Practical:-8**

**8. Create a “Queue” structure with following Data members: 1. Integer Array 2. Size of the Array Perform the following operations on Simple queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty Create a file which stores all values of Array.**

#include<stdio.h>

#define SIZE 10

void qinsert();

void qdelete();

void qdisplay();

void isempty();

void isfull();

int queue[SIZE];

int rear = -1;

int front = -1;

void main()

{

int ch;

while(1)

{

printf("\n 1. For insert Opration\n");

printf("\n 2. For delete Opration\n");

printf("\n 3. For Display Opration\n");

printf("\n 4. For Check Empty Or Not Operation\n");

printf("\n 5. For Check Full Or Not Operation\n");

printf("\n 6. For Exit\n");

printf("\n Enter your choice :\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

qinsert();

break;

case 2:

qdelete();

break;

case 3:

qdisplay();

break;

case 4:

isempty();

break;

case 5:

isfull();

break;

case 6:

exit(0);

default:

printf("\n Incorrect choice");

}

}

}

void qinsert()

{

int item;

if(rear == SIZE-1)

{

printf("\n overflow\n");

}

else

{

if(front == -1)

{

front =0;

}

printf("\n Element to be inserted in Queue : ");

scanf("%d",&item);

rear = rear+1;

queue[rear] = item;

}

}

void qdelete()

{

if(front ==-1 || front > rear)

{

printf("\n underflow");

return;

}

else

{

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

front = front+1;

}

}

void qdisplay()

{

int i;

if(front ==-1)

{

printf("\n Queue is empty");

}

else

{

printf("\n queue is : { ");

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

{

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

}

printf("}");

}

}

void isempty()

{

if(rear==-1 && front==-1)

{

printf("\n queue Empty \n");

}

else

{

printf("\n queue Not Empty\n");

}

}

void isfull()

{

if(rear==SIZE-1)

{

printf("\n queue is Full\n");

}

else

{

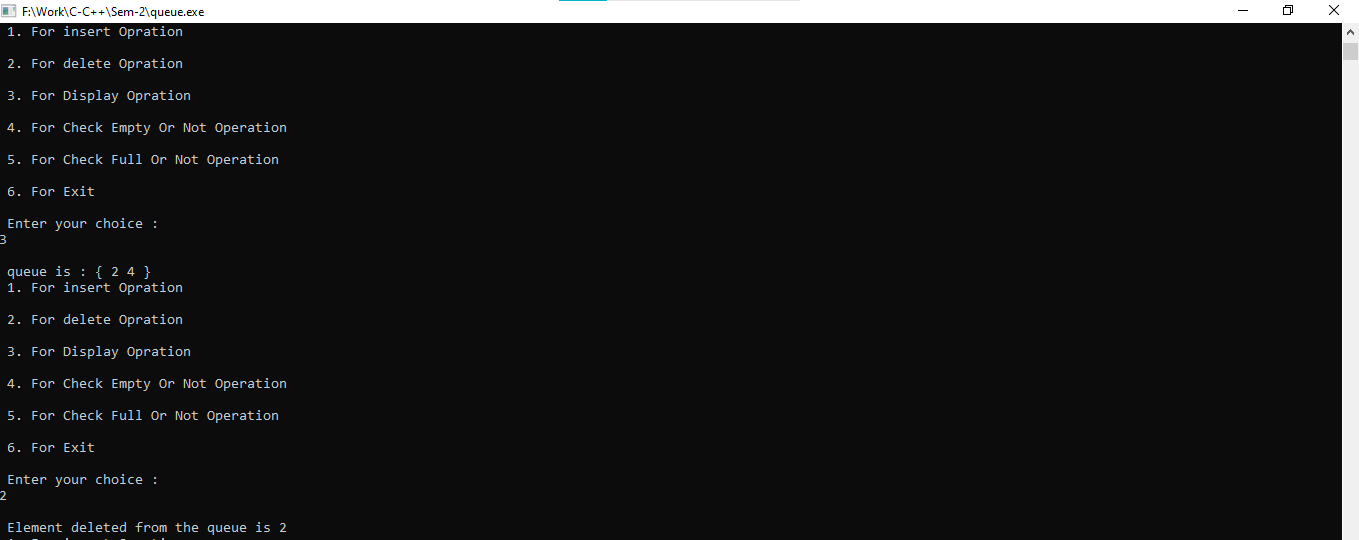
printf("\n queue is Not Full\n");

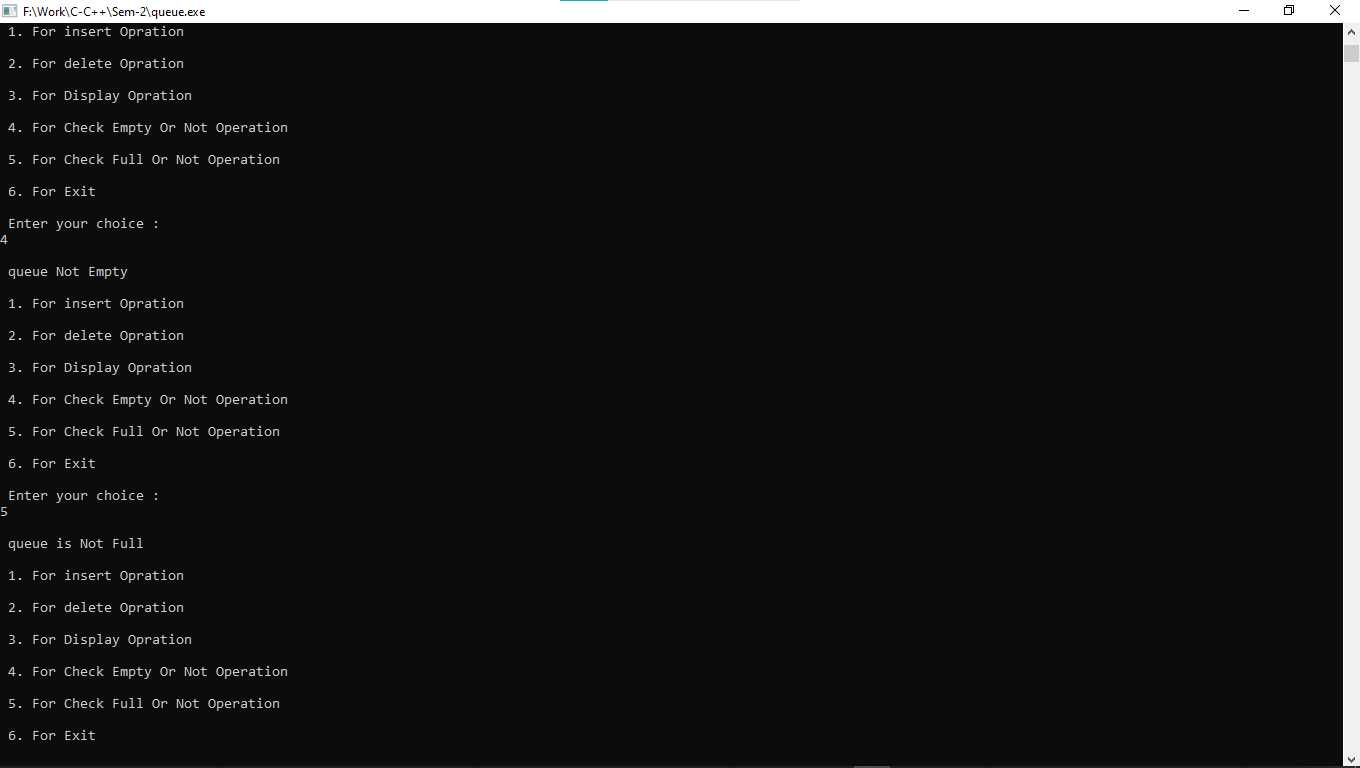
}

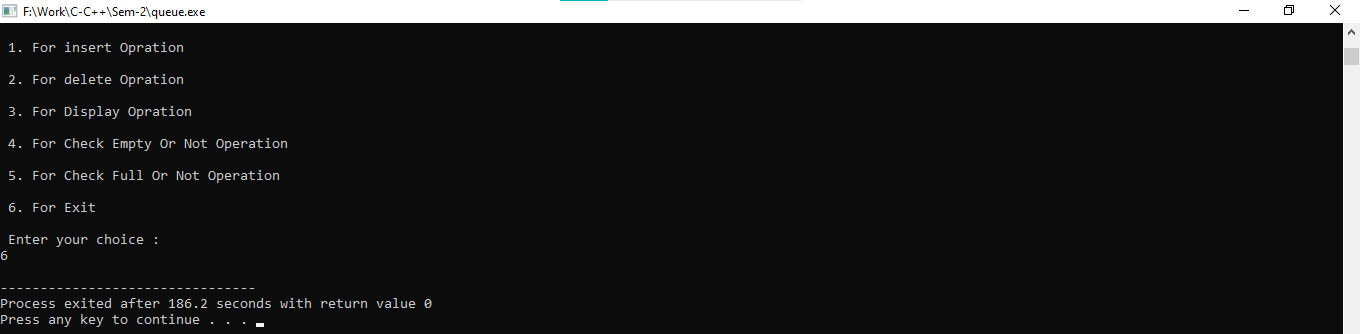
}

**Output:-**









**Practical:-9**

**9. Create a “Queue” user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node Perform the following operations on Simple queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty Create a file which stores all values of 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 isfull();

void isempty();

void main ()

{

int choice;

while(choice != 6)

{

printf("\n1.insert an element\n");

printf("\n2.Delete an element\n");

printf("\n3.Display the queue\n");

printf("\n4.Check queue is full or Not\n");

printf("\n5.Check queue is empty or Not\n");

printf("\n6.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:

isfull();

break;

case 5:

isempty();

break;

case 6:

exit(0);

break;

default:

printf("\nInvalid 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:\n");

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");

}

else

{

ptr = front;

front = front -> next;

free(ptr);

}

}

void display()

{

struct node \*ptr;

ptr = front;

if(front == NULL)

{

printf("\nEmpty queue\n");

}

else

{ printf("\nDisplay Elemets.....\n");

while(ptr != NULL)

{

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

ptr = ptr -> next;

}

}

}

void isfull()

{

struct node \*ptr;

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

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

printf("\nNOT OVERFLOW\n");

}

}

void isempty()

{

if(front == NULL && rear==NULL)

{

printf("\nEmpty queue\n");

}

else

{

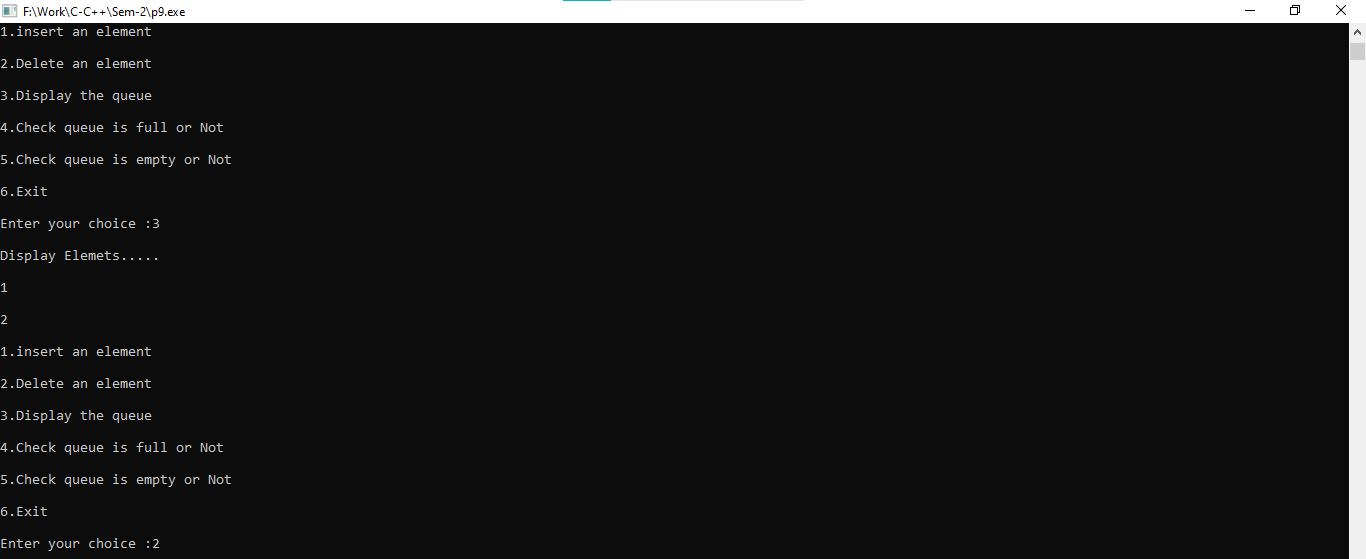
printf("\nqueue Not Empty\n");

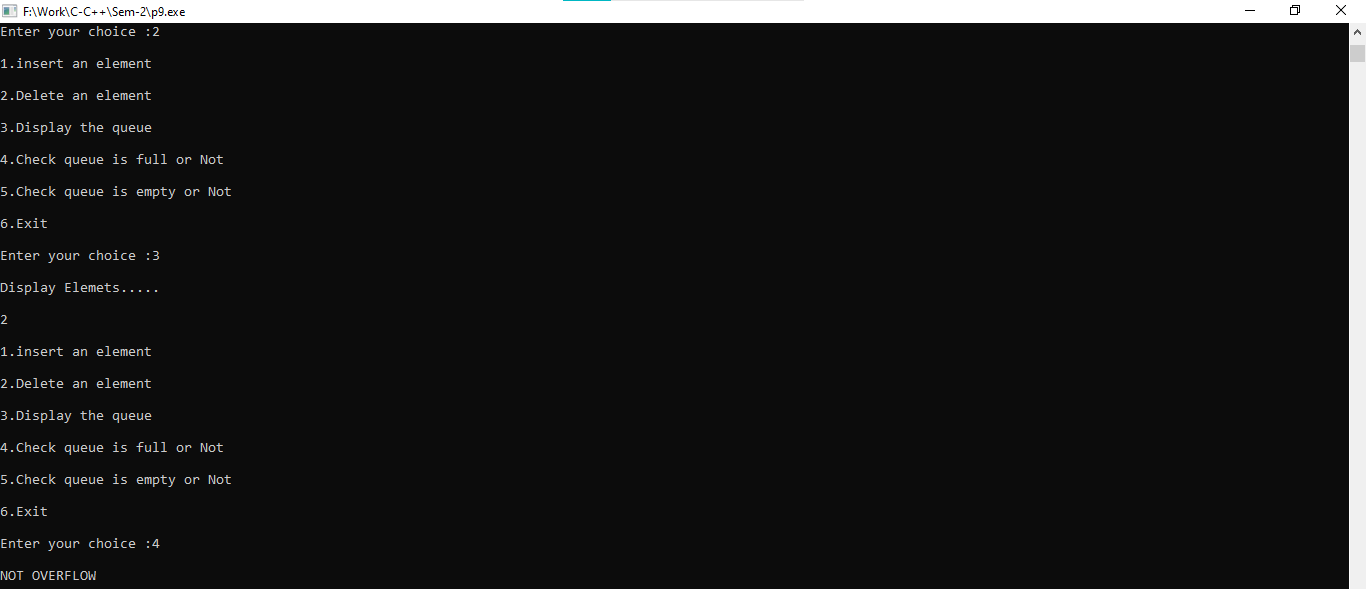
}

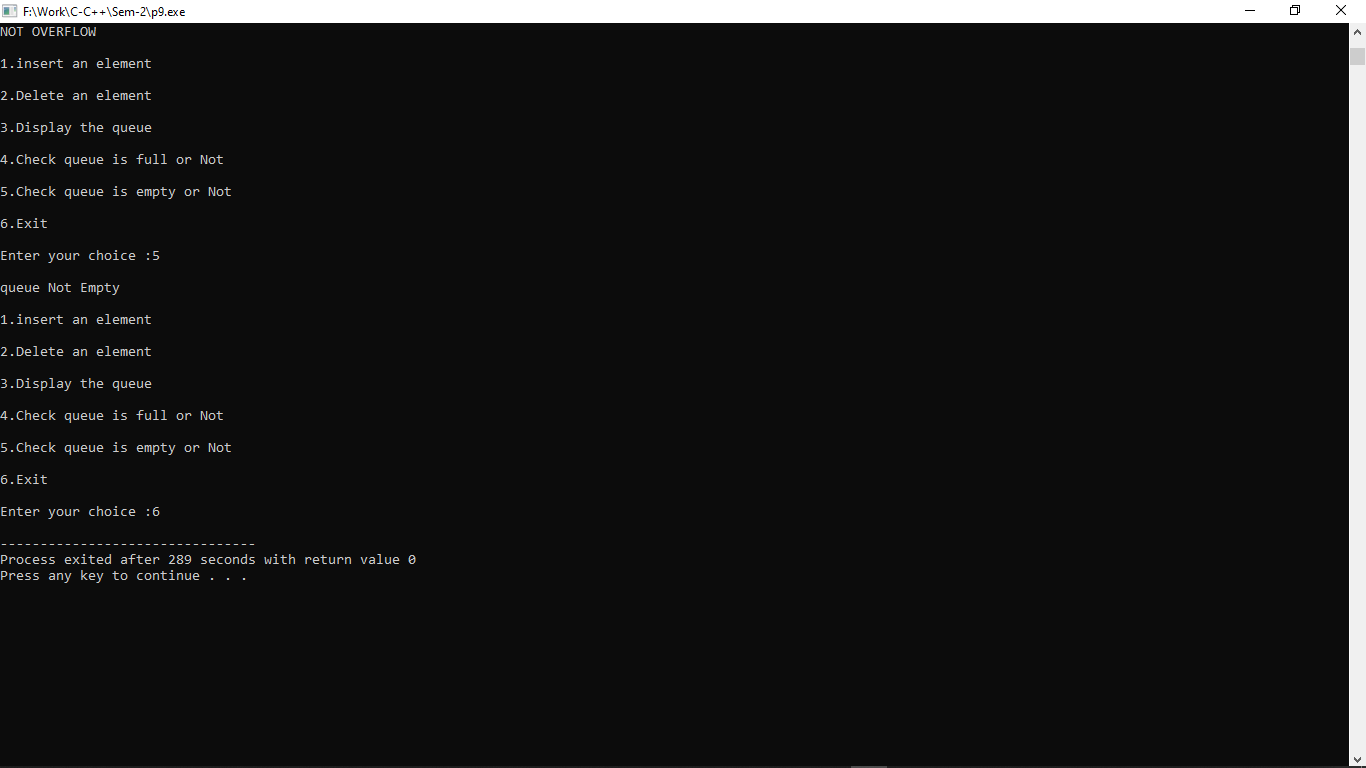
}

**Output:-**









**Practical:-10**

**10. Create a “Circular Queue” structure with following Data members: 1. Integer Array 2. Size of the Array Perform the following operations on Circular queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty. Create a file which stores all values of Array.**

#include<stdio.h>

#define size 5

int front=-1,rear=-1,cq[size],data=0;

void cqinsert();

int cqdelete();

void cqdisplay();

void isempty();

void isfull();

void main()

{

int ch=0;

while(1)

{

printf("\n 1. For insert Opration\n");

printf("\n 2. For delete Opration\n");

printf("\n 3. For Display Opration\n");

printf("\n 4. For Check Empty Or Not Operation\n");

printf("\n 5. For Check Full Or Not Operation\n");

printf("\n 6. For Exit\n");

printf("\n Enter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter Your Data: ");

scanf("%d",&data);

cqinsert(data);

break;

case 2:

data=cqdelete();

printf("Deleted Element is: %d",data);

break;

case 3:

cqdisplay();

break;

case 4:

isempty();

break;

case 5:

isfull();

break;

case 6:

exit(0);

break;

default:

printf("Invalid Choices...");

}

}

}

void cqinsert()

{

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

{

printf("\n Overflow");

}

else

{

if(front==-1)

{

front=0;

rear=0;

}

else if(rear==size-1)

{

rear=0;

}

else

{

rear=rear+1;

}

cq[rear]=data;

}

}

int cqdelete()

{

int v=0;

if(front==-1)

{

printf("\n Overflow\n") ;

}

else

{

v=cq[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

if(front==size-1)

{

front=0;

}

else

{

front=front+1;

}

}

}

return v;

}

void cqdisplay()

{

int i;

if(front==-1)

{

printf("\nEmpty\n");

}

else

{

if(front<=rear)

{

printf("{");

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

{

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

}

printf("}");

}

else

{

printf("{");

for(i=front;i<=size-1;i++)

{

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

}

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

{

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

}

printf("}");

}

}

}

void isempty()

{

if(front==-1 && rear==-1)

{

printf("\nEmpty\n");

}

else

{

printf("\nNot Empty\n");

}

}

void isfull()

{

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

{

printf("\n Overflow");

}

else

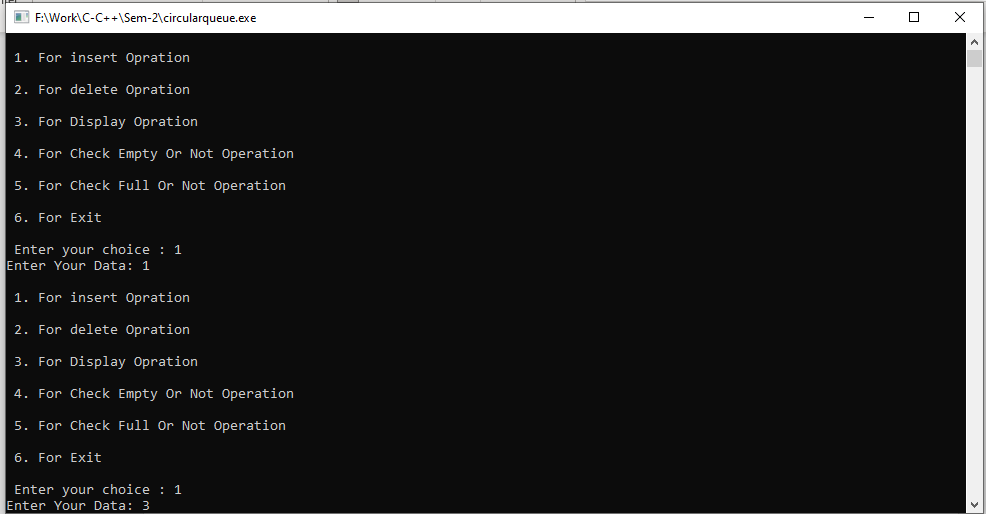
{

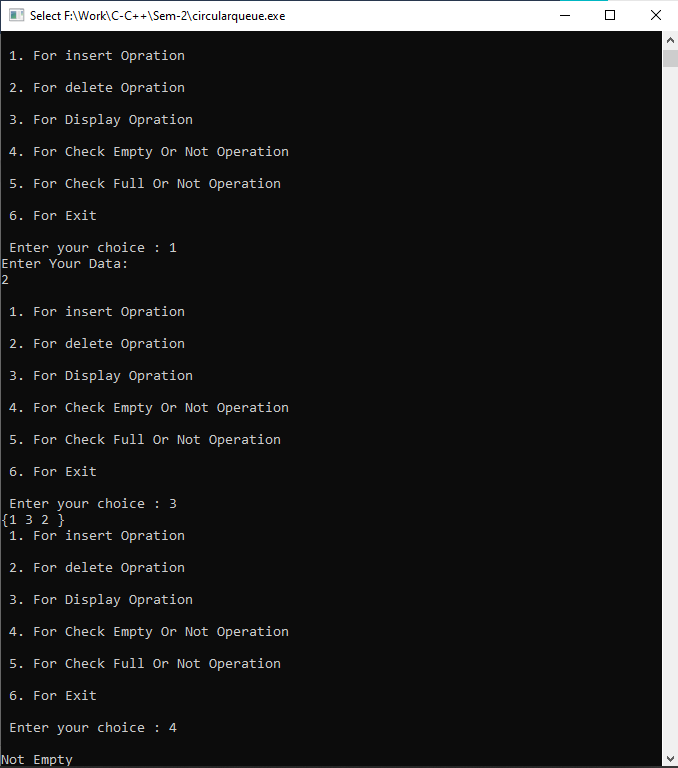
printf("\n Not Overflow");

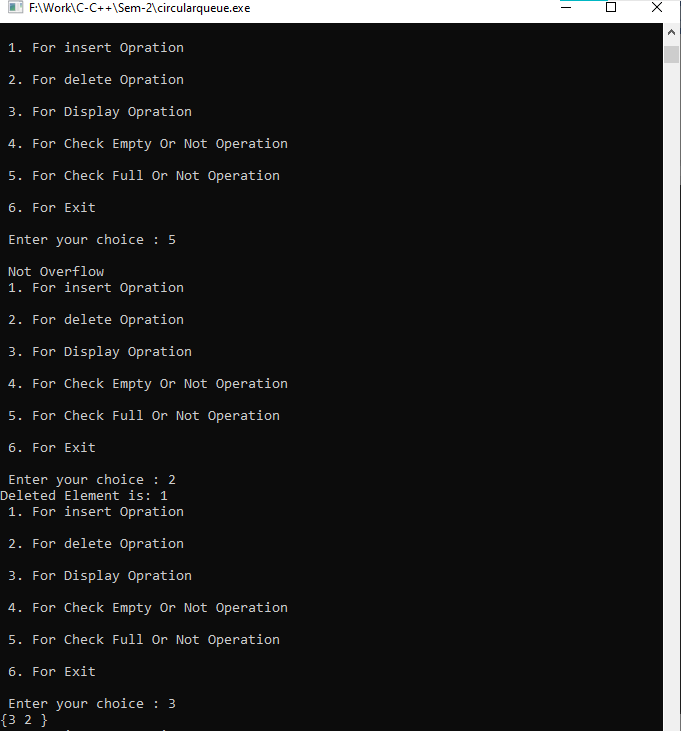
}

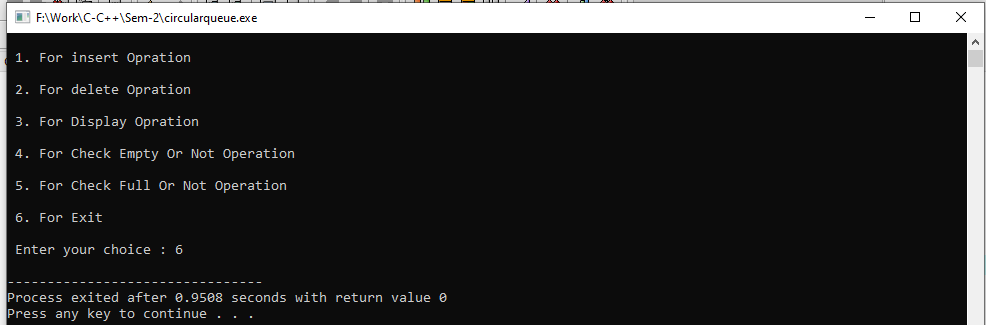
}

**Output:-**









**Practical:-11**

**11. Create a “Circular Queue” user-defined structure with the following data members: 1. A Data 2. A link to the next node Perform the following operations on Circular queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty Create a file which stores all values of list.**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void insert();

void rem();

void display();

void main()

{

do

{

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

printf("2.Remove\n");

printf("3.Display\n");

printf("4.EXIT");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

rem();

break;

}

case 3:

{

display();

break;

}

case 4:

{

break;

}

default:

{

printf ("\nEnter Valid Choice..!");

}

}

}

while(choice!=4);

}

void insert()

{

node \*temp;

temp = (node\*)malloc(sizeof(node));

if(temp==NULL){

printf("\n\tQueue is Full\n");

}

else{

printf("Enter a value to inserte:");

scanf("%d",&i);

temp->data = i;

temp->next=start;

if(start==NULL){

start = temp;

rear = temp;

}

else{

rear->next = temp;

rear = temp;

}

}

}

void rem()

{

node \*temp;

if(start==NULL)

{

printf("\n\t Queue is Empty \n");

}

else

{

temp = start;

start = start->next;

rear->next=start;

printf("\n\t The deleted element is %d",temp->data);

free(temp);

}

}

void display()

{

node \*temp;

if(start!=NULL)

{

temp = start;

while(temp->next->data!=start->data){

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

temp = temp->next;

}

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

}

else

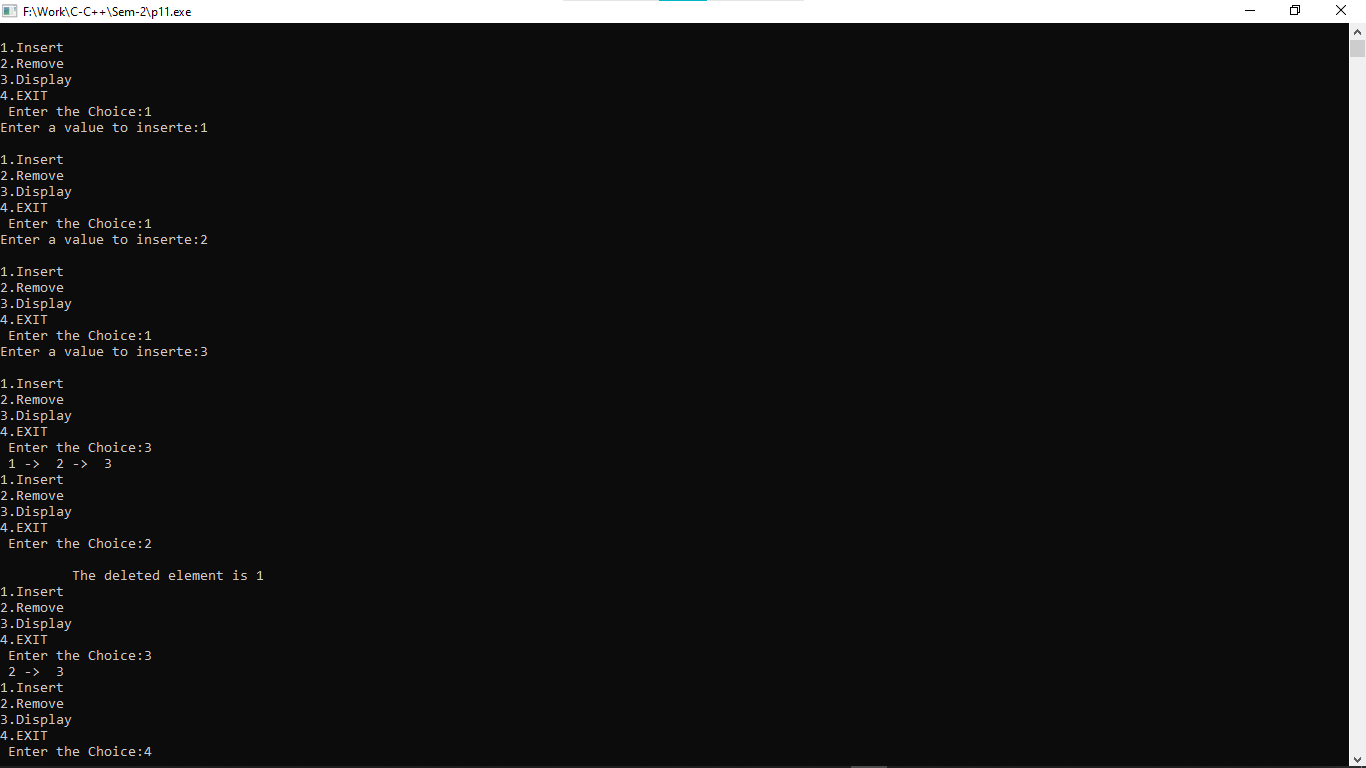
{

printf("\n The Queue is empty");

}

}

**Output:-**



**Practical:-12**

**12. Create a user-defined “Linked List” structure with the following data members:**

**1. A Co-efficient**

**2. An Exponent**

**3. A link to the next node Perform the following operations on Singly list using user-defined functions:**

**1. Create**

**2. Display**

**3. Addition**

**4. Multiplication Create a file which stores all values of list.**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

struct poly

{

int coeff;

int exp;

struct poly \*next;

}\*head1=NULL,\*head2=NULL,\*head3=NULL,\*head4=NULL,\*temp,\*ptr;

void create();

void makenode(int,int);

struct poly \*insertend(struct poly \*);

void display(struct poly \*);

struct poly \*addtwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*subtwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*multwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*dispose(struct poly \*);

int search(struct poly \*,int);

void main()

{

int ch,coefficient,exponent;

int listno;

while(1)

{

printf("\nMenu\n");

printf("1. Create First Polynomial.\n");

printf("2. Display First Polynomial.\n");

printf("3. Create Second Polynomial.\n");

printf("4. Display Second Polynomial.\n");

printf("5. Add Two Polynomials.\n");

printf("6. Display Result of Addition.\n");

printf("7. Subtract Two Polynomials.\n");

printf("8. Display Result of Subtraction.\n");

printf("9. Multiply Two Polynomials.\n");

printf("10. Display Result of Product.\n");

printf("11. Dispose List.\n");

printf("12. Exit\n");

printf("Enter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nGenerating first polynomial:");

printf("\nEnter coefficient: ");

scanf("%d",&coefficient);

printf("\nEnter exponent: ");

scanf("%d",&exponent);

makenode(coefficient,exponent);

head1 = insertend(head1);

break;

case 2:

display(head1);

break;

case 3:

printf("\nGenerating second polynomial:");

printf("\nEnter coefficient: ");

scanf("%d",&coefficient);

printf("\nEnter exponent: ");

scanf("%d",&exponent);

makenode(coefficient,exponent);

head2 = insertend(head2);

break;

case 4:

display(head2);

break;

case 5:

printf("\nDisposing result list.");

head3=dispose(head3);

head3=addtwopoly(head1,head2,head3);

printf("\nAddition successfully done!");

break;

case 6:

display(head3);

break;

case 7:

head3=dispose(head3);

head3=subtwopoly(head1,head2,head3);

printf("\nSubtraction successfully done!");

getch();

break;

case 8:

display(head3);

break;

case 9:

head3=dispose(head3);

head4=dispose(head4);

head4=multwopoly(head1,head2,head3);

break;

case 10:

display(head4);

break;

case 11:

printf("Enter list number to dispose(1 to 4): ");

scanf("%d",&listno);

if(listno==1)

head1=dispose(head1);

else if(listno==2)

head2=dispose(head2);

else if(listno==3)

head3=dispose(head3);

else if(listno==4)

head4=dispose(head4);

else

printf("\nInvalid number specified.");

break;

case 12:

exit(0);

default:

printf("\nInvalid Choice!");

break;

}

}

}

void create()

{

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

if(ptr==NULL)

{

printf("\nMemory Allocation Error!");

exit(1);

}

}

void makenode(int c,int e)

{

create();

ptr->coeff = c;

ptr->exp = e;

ptr->next = NULL;

}

struct poly \*insertend(struct poly \*head)

{

if(head==NULL)

head = ptr;

else

{

temp=head;

while(temp->next != NULL)

temp = temp->next;

temp->next = ptr;

}

return head;

}

void display(struct poly \*head)

{

if(head==NULL)

printf("\nList is empty!");

else

{

temp=head;

while(temp!=NULL)

{

printf("(%d,%d)->",temp->coeff,temp->exp);

temp=temp->next;

}

printf("bb ");

}

getch();

}

struct poly \*addtwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

(5,3)->(6,1) + (7,3)->(9,2) = (12,3)->(6,1)->(9,2)

\*/

struct poly \*temp1,\*temp2,\*temp3;

temp1=h1;

temp2=h2;

while(temp1!=NULL || temp2!=NULL)

{

if(temp1->exp==temp2->exp)

{

makenode(temp1->coeff+temp2->coeff,temp1->exp);

h3=insertend(h3);

}

else

{

makenode(temp1->coeff,temp1->exp);

h3=insertend(h3);

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

}

temp1=temp1->next;

temp2=temp2->next;

}

if(temp1==NULL && temp2!=NULL)

{

while(temp2!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

}

if(temp2==NULL && temp1!=NULL)

{

while(temp1!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp1=temp1->next;

}

}

return h3;

}

struct poly \*subtwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

(5,3)->(6,1) - (7,3)->(9,2) = (-2,3)+(6,1)-(9,2)

\*/

struct poly \*temp1,\*temp2,\*temp3;

temp1=h1;

temp2=h2;

while(temp1!=NULL || temp2!=NULL)

{

if(temp1->exp==temp2->exp)

{

makenode(temp1->coeff-temp2->coeff,temp1->exp);

h3=insertend(h3);

}

else

{

makenode(temp1->coeff,temp1->exp);

h3=insertend(h3);

makenode(-temp2->coeff,temp2->exp);

h3=insertend(h3);

}

temp1=temp1->next;

temp2=temp2->next;

}

if(temp1==NULL && temp2!=NULL)

{

while(temp2!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

}

if(temp2==NULL && temp1!=NULL)

{

while(temp1!=NULL)

{

makenode(-temp2->coeff,temp2->exp);

h3=insertend(h3);

temp1=temp1->next;

}

}

return h3;

}

struct poly \*multwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

h1=(5,3)->(6,1) \* h2=(7,3)->(9,2)

(5,3)->(7,3),(9,2) = (35,6),(45,5)

(6,1)->(7,3),(9,2) = (42,4),(54,3)

h3->(35,6)->(45,5)->(42,4)->(54,3)

(35,6)+(45,5)+(42,4)+(54,3)=Result

\*/

int res=0;

struct poly \*temp1,\*temp2,\*temp3;

printf("\nDisplaying First Polynomial:\n");

display(h1);

printf("\nDisplaying Second Polynomial:\n");

display(h2);

temp1=h1;

while(temp1!=NULL)

{

temp2=h2;

while(temp2!=NULL)

{

makenode(temp1->coeff\*temp2->coeff,temp1->exp+temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

temp1=temp1->next;

}

printf("\nDisplaying Initial Result of Product:\n");

display(h3);

getch();

temp1=h3;

while(temp1!=NULL)

{temp2=temp1->next;

res=0;

while(temp2!=NULL)

{

if(temp1->exp==temp2->exp)

res += temp2->coeff;

temp2=temp2->next;

}

if(search(head4,temp1->exp)==1)

{

makenode(res+temp1->coeff,temp1->exp);

head4=insertend(head4);

}

temp1=temp1->next;

}

return head4;

}

int search(struct poly \*h,int val)

{

struct poly \*tmp;

tmp=h;

while(tmp!=NULL)

{if(tmp->exp==val)

return 0;

tmp=tmp->next;

}

return 1;

}

struct poly \*dispose(struct poly \*list)

{

if(list==NULL)

{

printf("\nList is already empty.");

return list;

}

else

{

temp=list;

while(list!=NULL)

{

free(temp);

list=list->next;

temp=list;

}

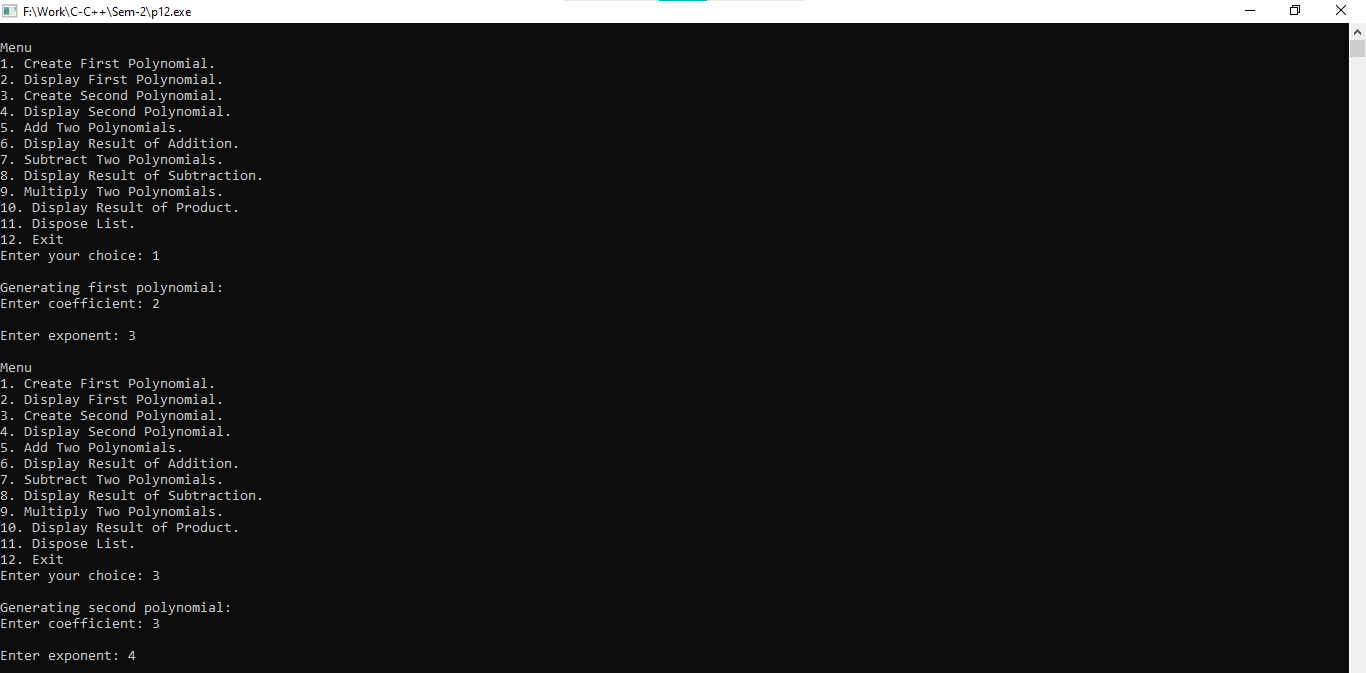
return list;

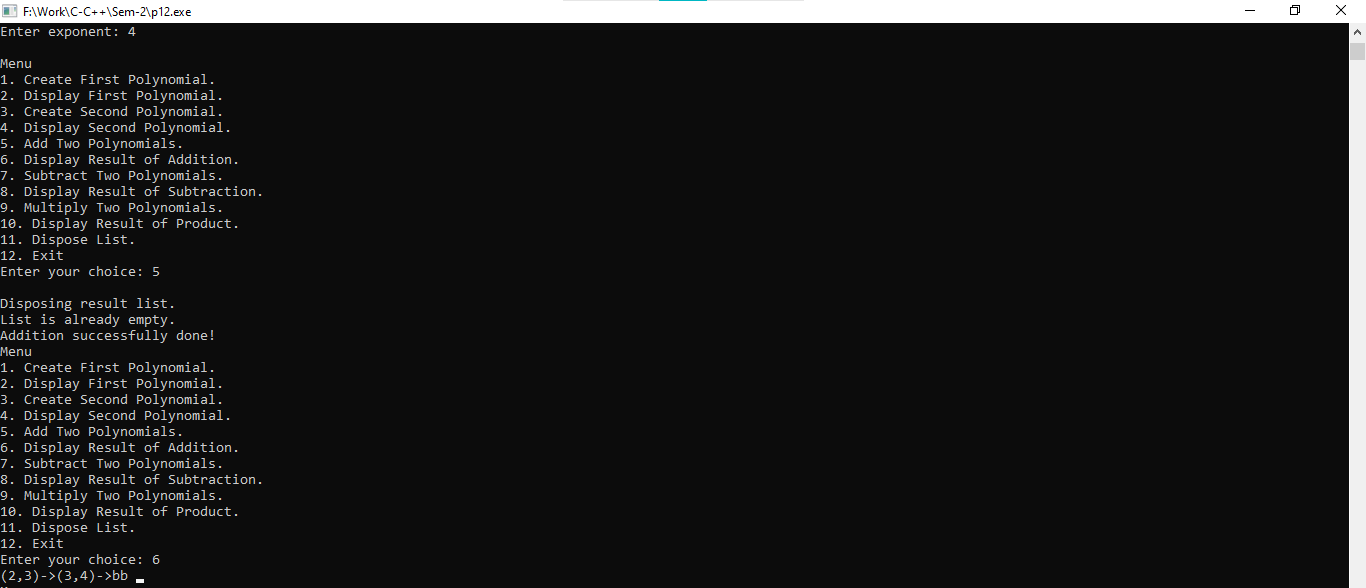
}

}

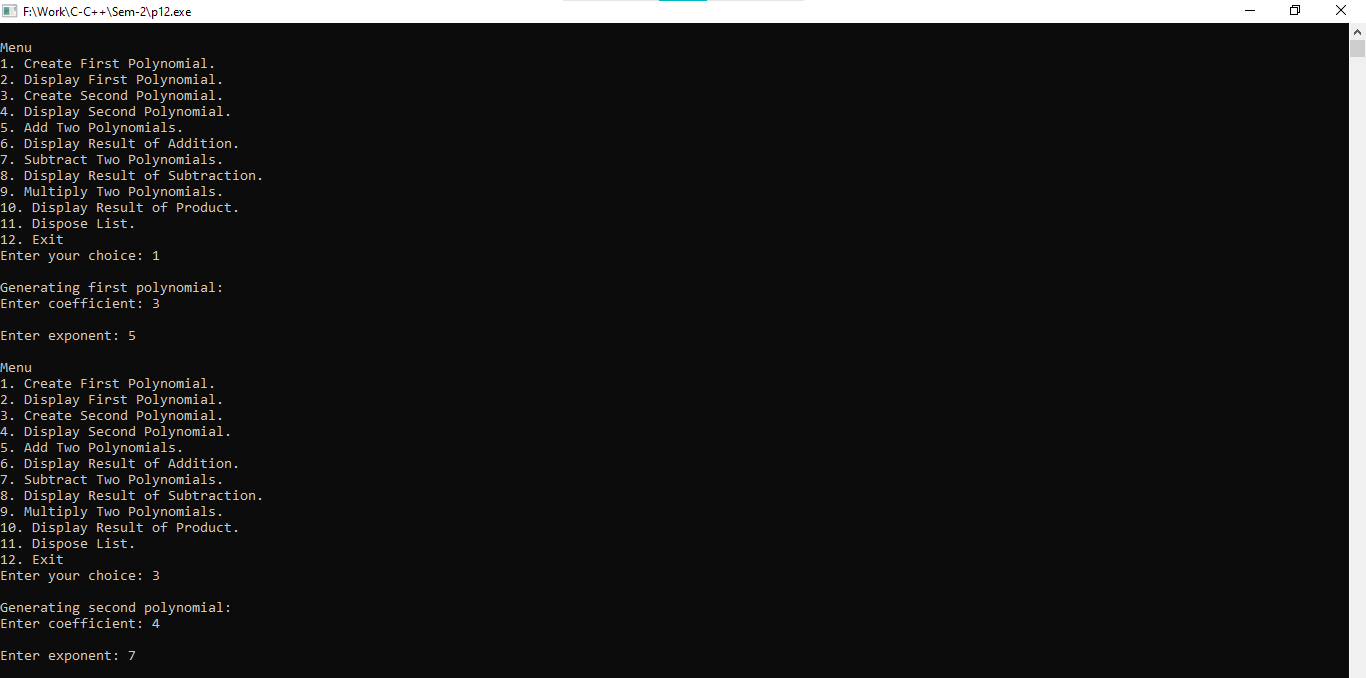
**Output:-**

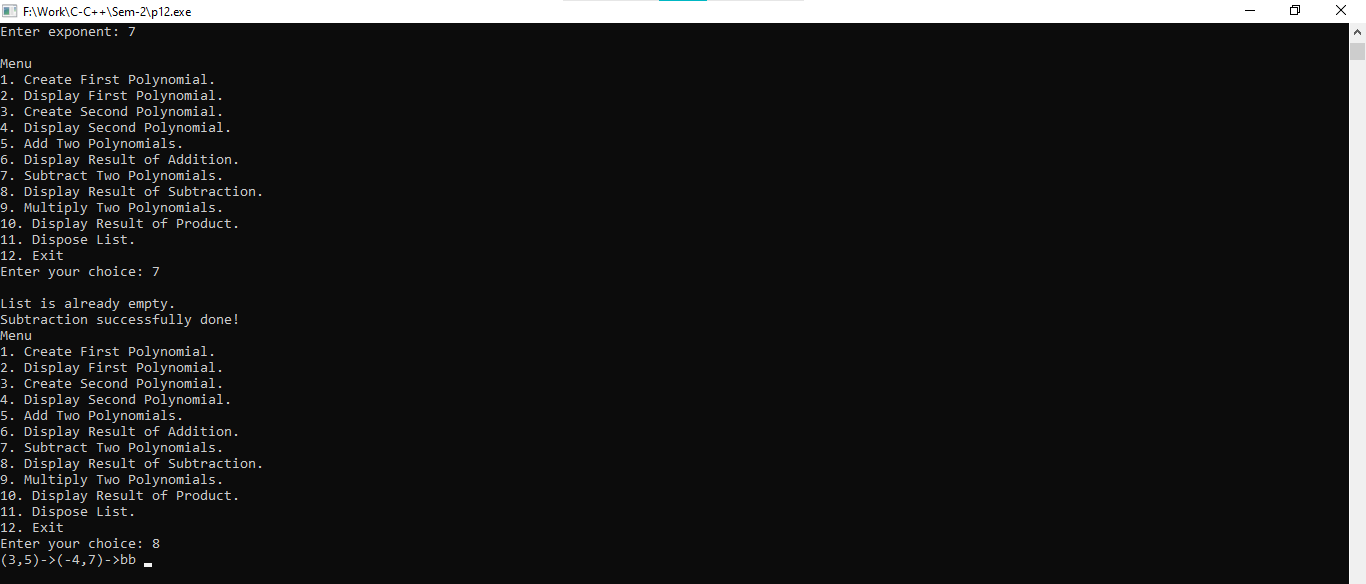
**Addition:**

****

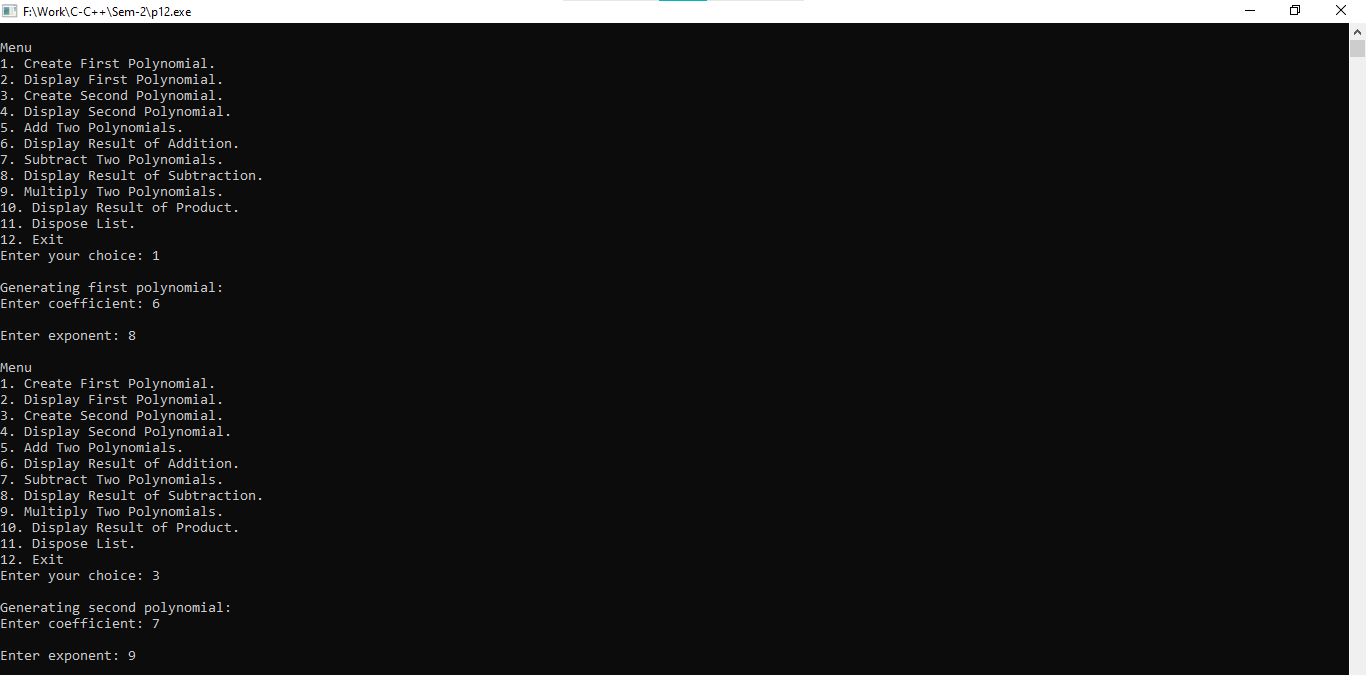
****

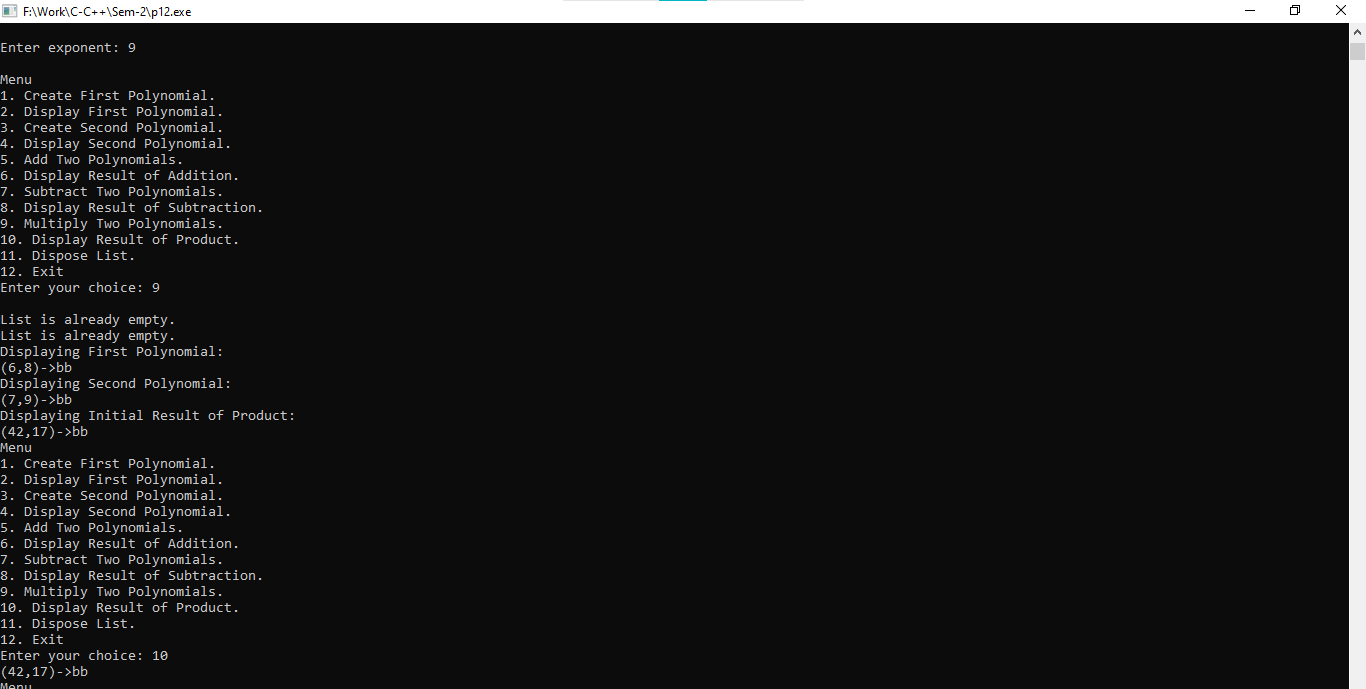
**Substraction:**

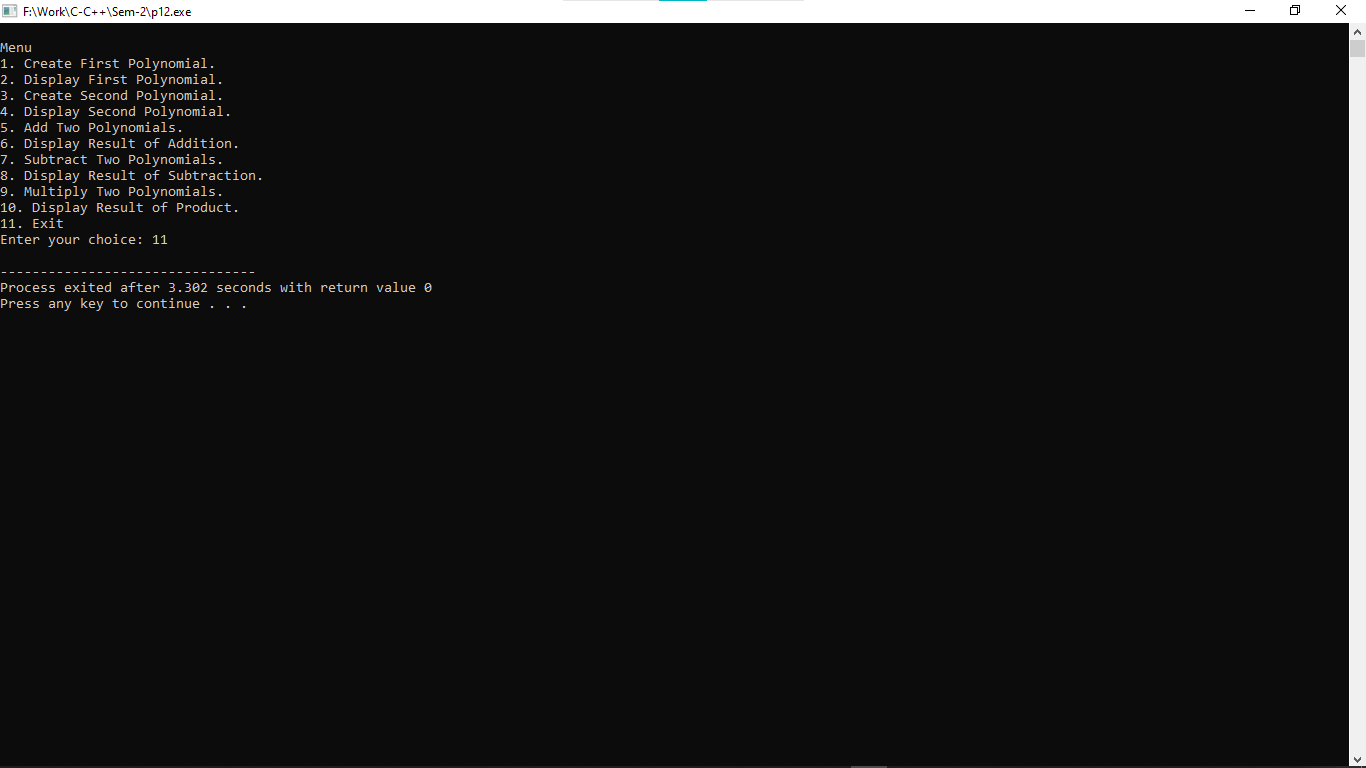
****

****

**Multiplication:**

****

****

****

**Practical:-13**

**13. Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node Perform the following operations on list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

rear->next=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next!=NULL){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next!=NULL){

rear=temp;

temp=temp->next;

}

rear->next=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*prev;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

prev=temp;

temp=temp->next;

if(temp->next->data==i){

prev->next=temp->next;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

rear->next=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

if(count==1){

temp->next=start;

start=temp;

}

else{

trav=start;

while(trav!=NULL){

prev=trav;

trav=trav->next;

if(trav->data==val){

prev->next=temp;

temp->next=trav;

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next!=NULL){

count++;

prev=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

prev->next=temp;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

int count=0,\*rev\_array;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

rev\_array=(int \*)malloc(count\*sizeof(int));

temp=start;

for(i=0;i<count;i++){

\*(rev\_array+i)=temp->data;

temp=temp->next;

}

//printf("%d\n%d\n",sizeof(rev\_array),count);

temp=start;

for(i=count-1;i>=0;i--){

temp->data=\*(rev\_array+i);

rear=temp;

temp=temp->next;

}

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

while(temp!=NULL){

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next!=NULL){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

printf("\n\t--------------------------------");

printf("\n\t 1. Create a list");

printf("\n\t 2. Traverse the whole list");

printf("\n\t 3. Delete first node");

printf("\n\t 4. Delete last node");

printf("\n\t 5. Delete a node before specified data");

printf("\n\t 6. Insert at first position");

printf("\n\t 7. Insert at last position");

printf("\n\t 8. Insert a node before specified data");

printf("\n\t 9. Insert a node at specified position");

printf("\n\t 10. Count");

printf("\n\t 11. Copy");

printf("\n\t 12. Merge two list");

printf("\n\t 13. Reverse");

printf("\n\t 14. Search");

printf("\n\t 15. Sort");

printf("\n\t 16. Save");

printf("\n\t 17. Exit");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

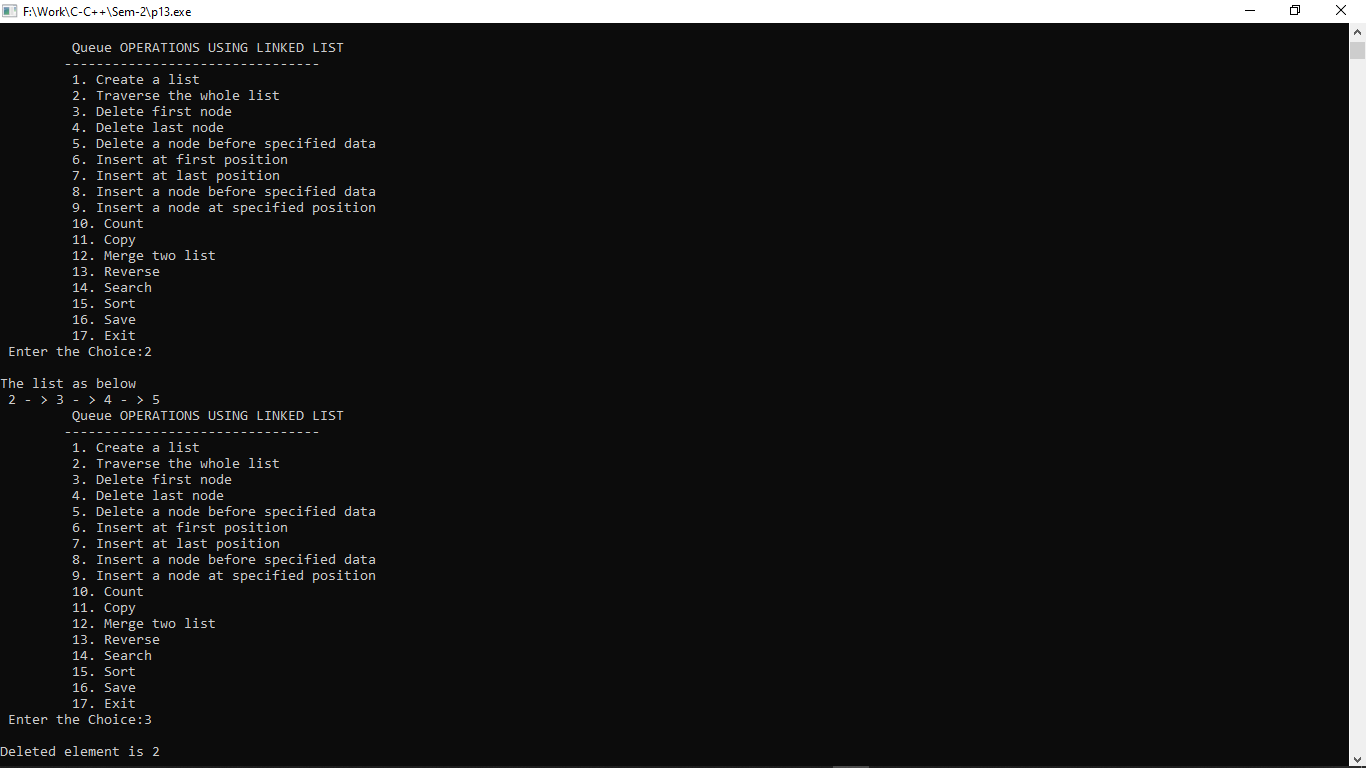
while(choice!=17);

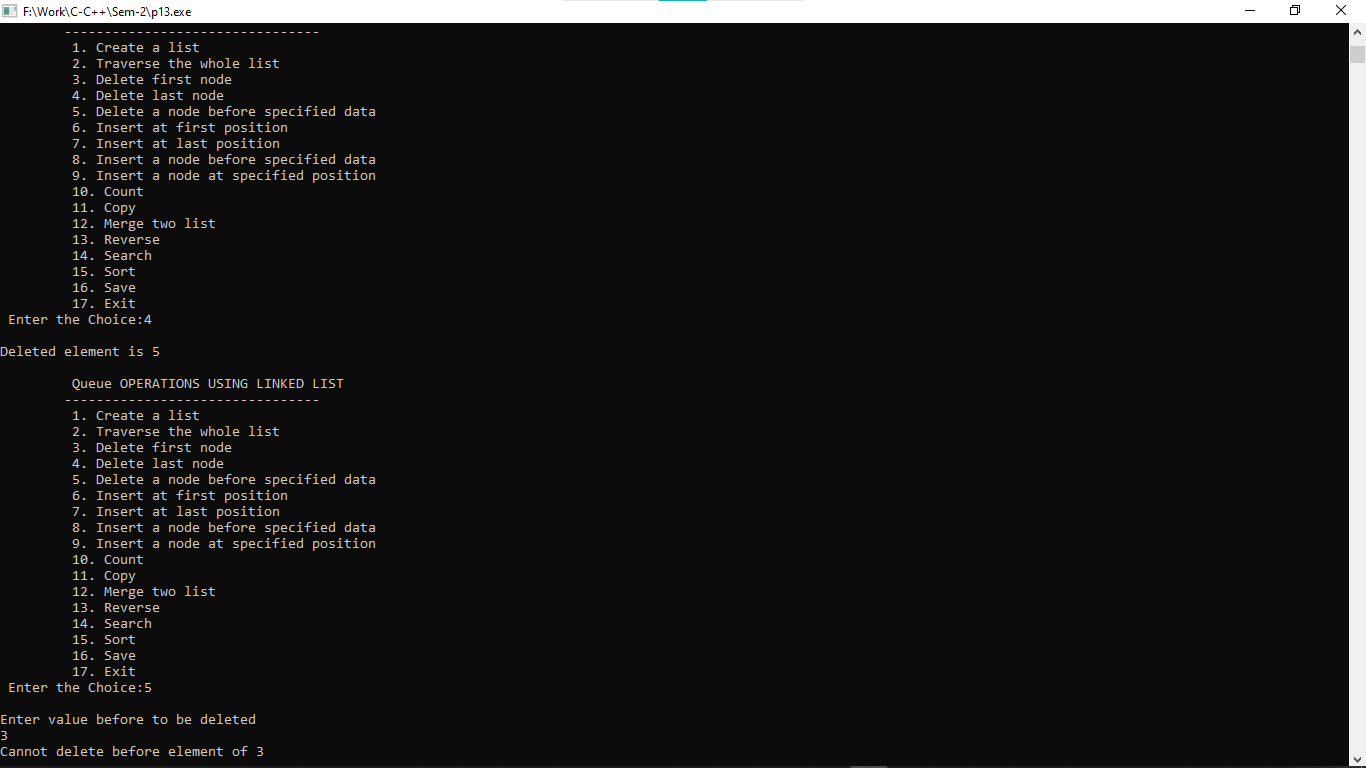
return 0;

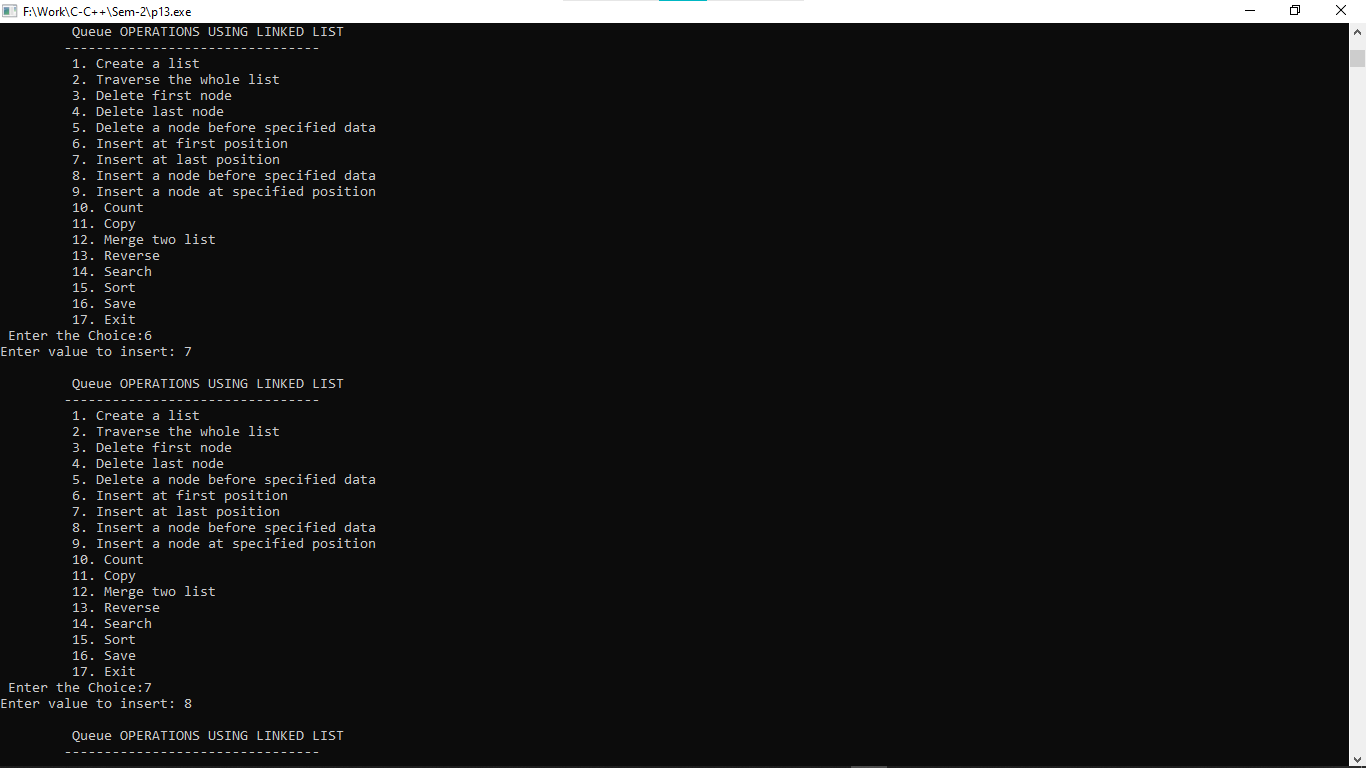
}

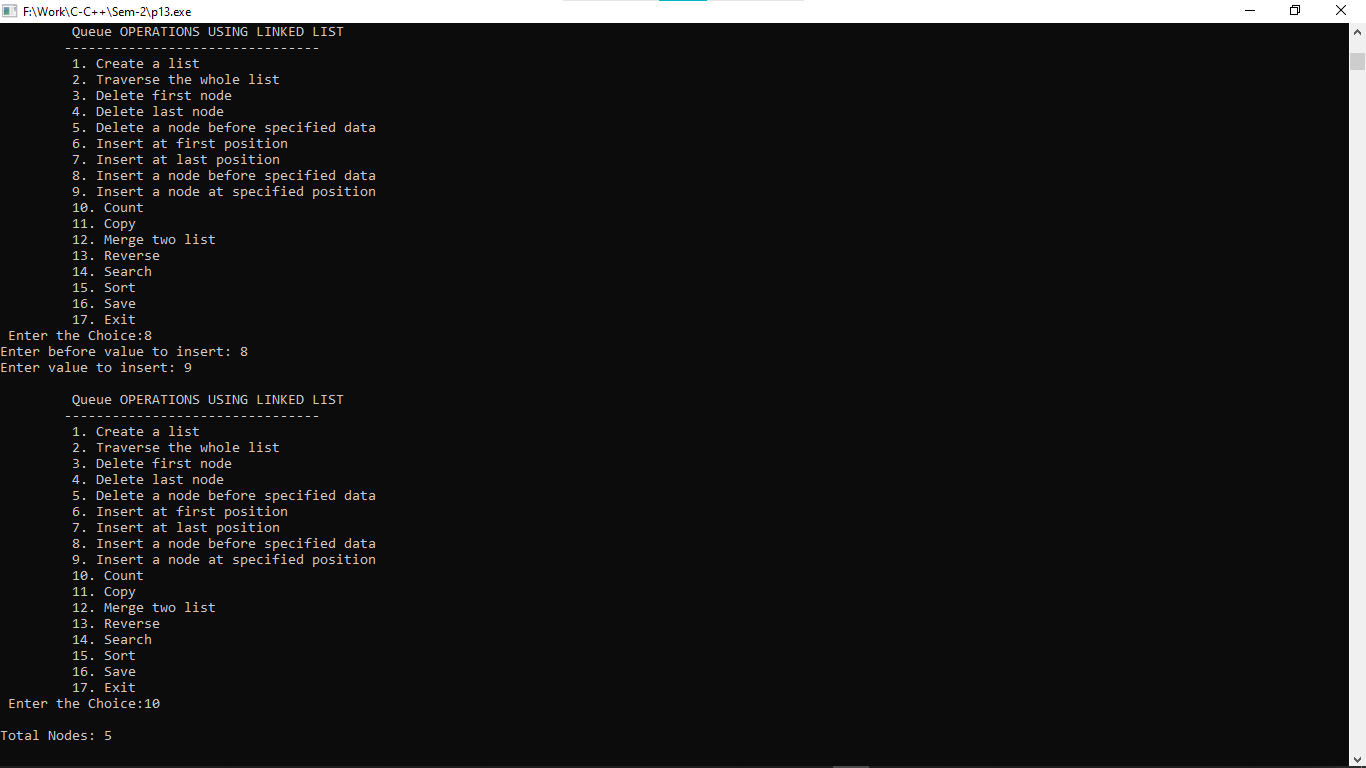
**Output:-**

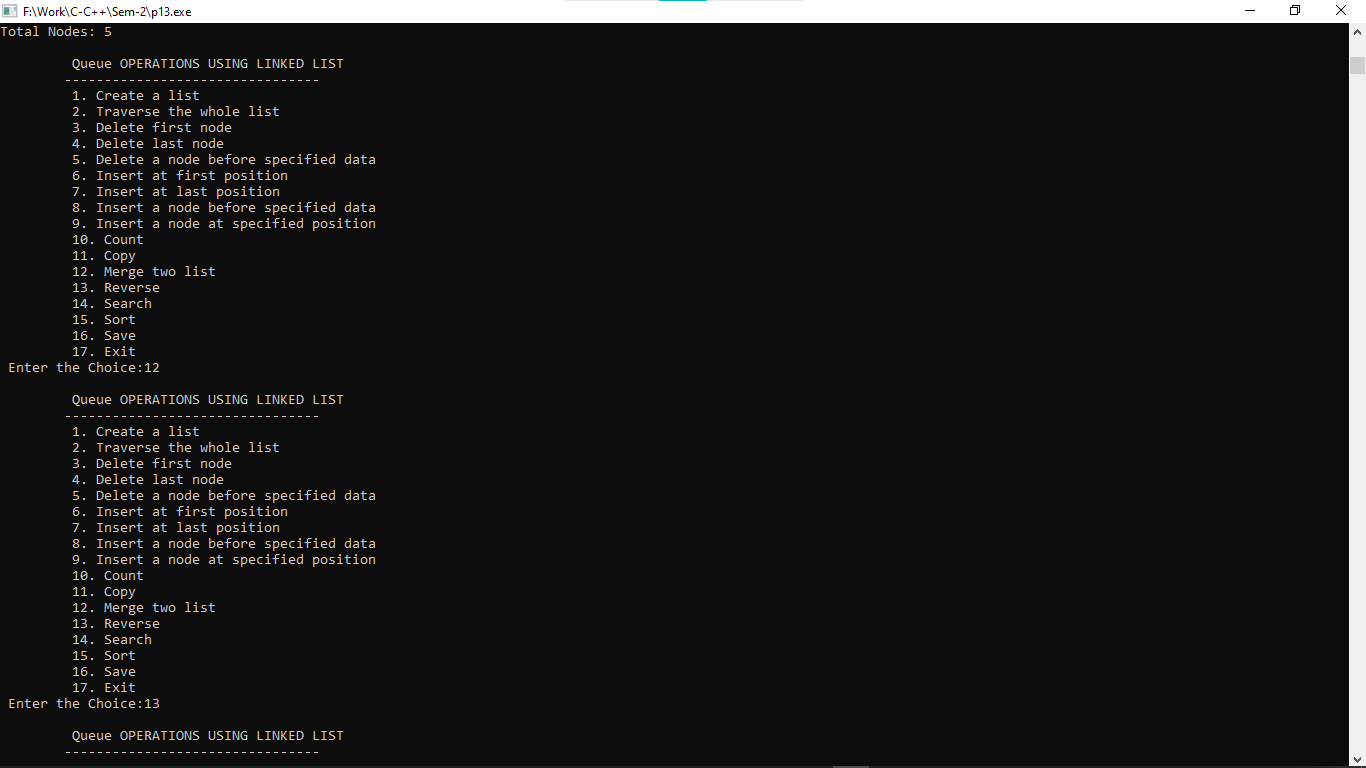


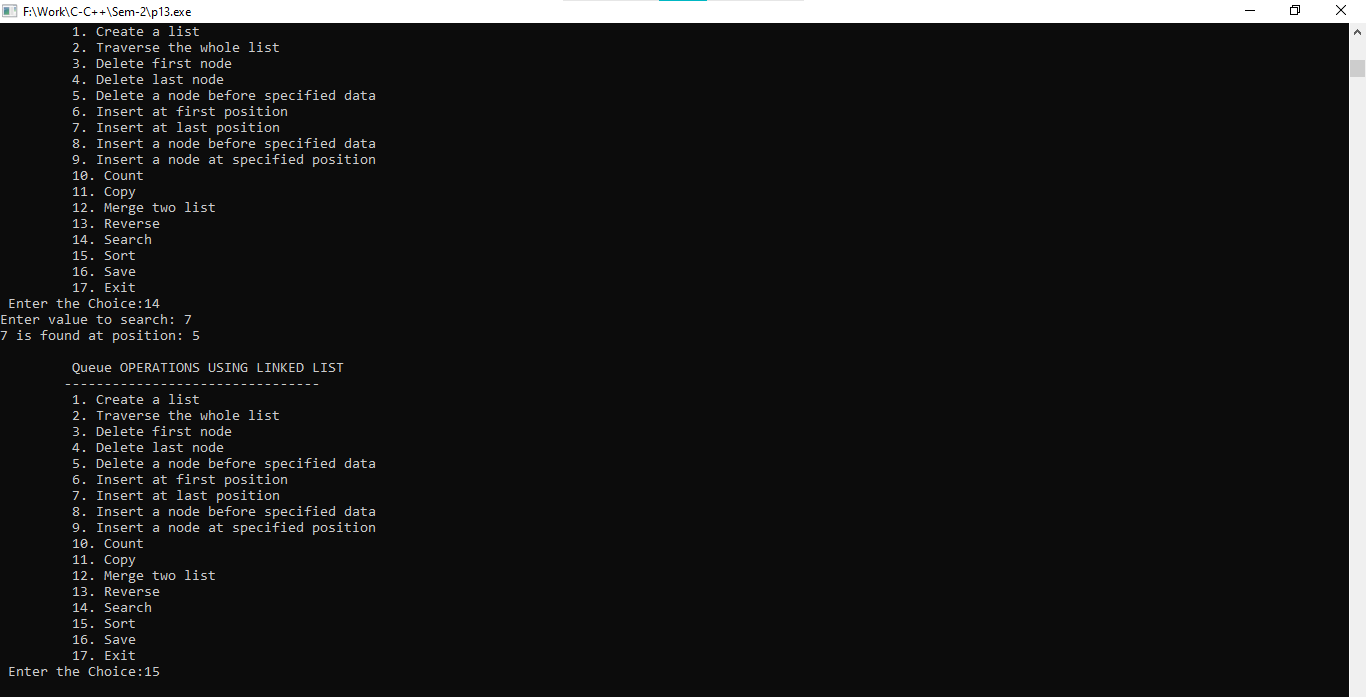
****

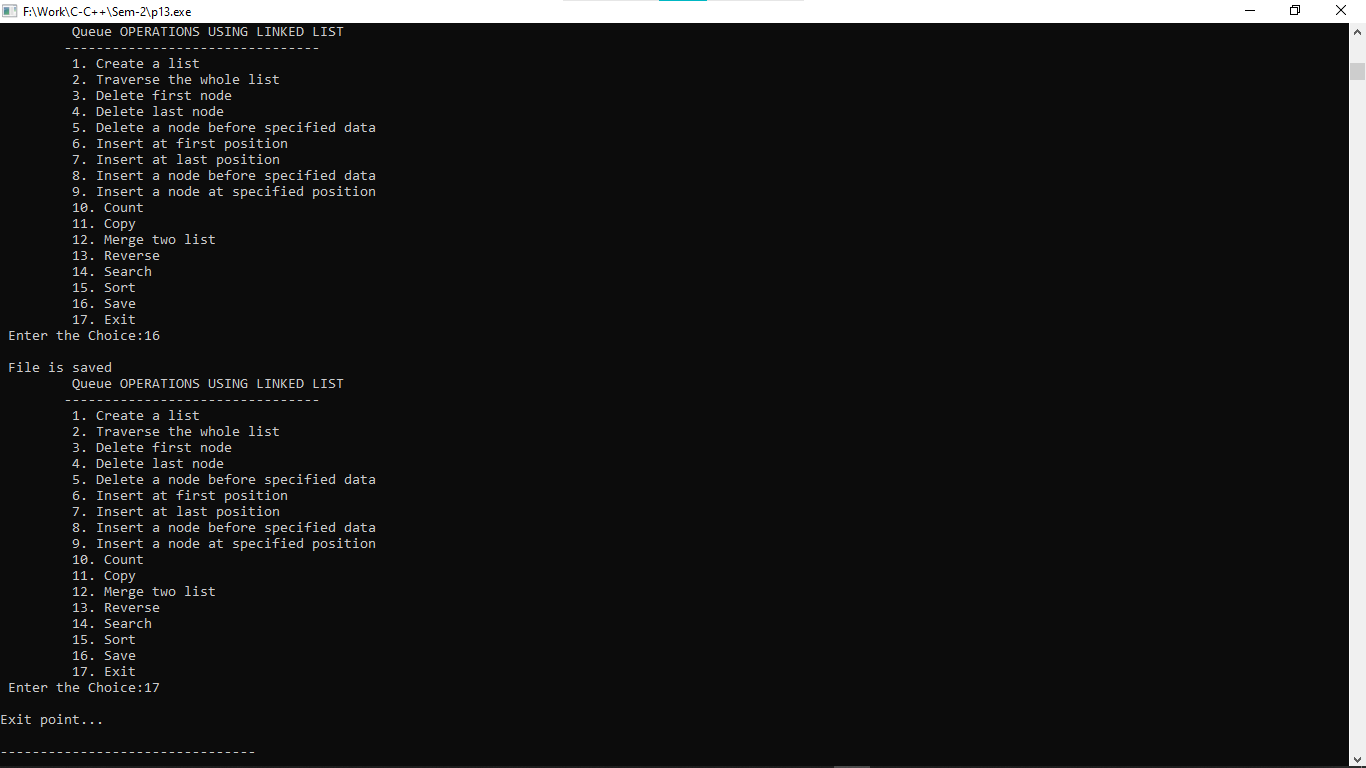
****

****

****

****

****

****

**Practical:-14**

**14. Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node Perform the following operations on Circular list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list.**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

rear->next=temp;

rear=temp;

rear->next=start;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp = start;

while(temp->next->data!=start->data){

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

temp = temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp,\*prev;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next->data!=start->data){

rear=temp;

temp=temp->next;

}

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*prev;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

do{

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

rear->next=start;

free(temp);

}

else{

temp=start;

do{

prev=temp;

temp=temp->next;

if(temp->next->data==i){

prev->next=temp->next;

free(temp);

break;

}

}while(temp->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

start=temp;

rear->next=start;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

rear->next=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

if(count==1){

temp->next=start;

start=temp;

rear->next=start;

}

else{

trav=start;

do{

prev=trav;

trav=trav->next;

if(trav->data==val){

prev->next=temp;

temp->next=trav;

break;

}

}while(trav->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next->data!=start->data){

count++;

prev=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

prev->next=temp;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

int count=0,\*rev\_array;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

rev\_array=(int \*)malloc(count\*sizeof(int));

temp=start;

for(i=0;i<count;i++){

\*(rev\_array+i)=temp->data;

temp=temp->next;

}

//printf("%d\n%d\n",sizeof(rev\_array),count);

temp=start;

for(i=count-1;i>=0;i--){

temp->data=\*(rev\_array+i);

rear=temp;

temp=temp->next;

}

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}while(temp->data!=start->data);

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

temp2=temp->next;

while(temp2->data!=start->data){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}while(temp->data!=start->data);

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next->data!=start->data){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t CircularQueue OPERATIONS USING LINKED LIST");

printf("\n\t--------------------------------");

printf("\n\t 1. Create a list");

printf("\n\t 2. Traverse the whole list");

printf("\n\t 3. Delete first node");

printf("\n\t 4. Delete last node");

printf("\n\t 5. Delete a node before specified data");

printf("\n\t 6. Insert at first position");

printf("\n\t 7. Insert at last position");

printf("\n\t 8. Insert a node before specified data");

printf("\n\t 9. Insert a node at specified position");

printf("\n\t 10. Count");

printf("\n\t 11. Copy");

printf("\n\t 12. Merge two list");

printf("\n\t 13. Reverse");

printf("\n\t 14. Search");

printf("\n\t 15. Sort");

printf("\n\t 16. Save");

printf("\n\t 17. Exit");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

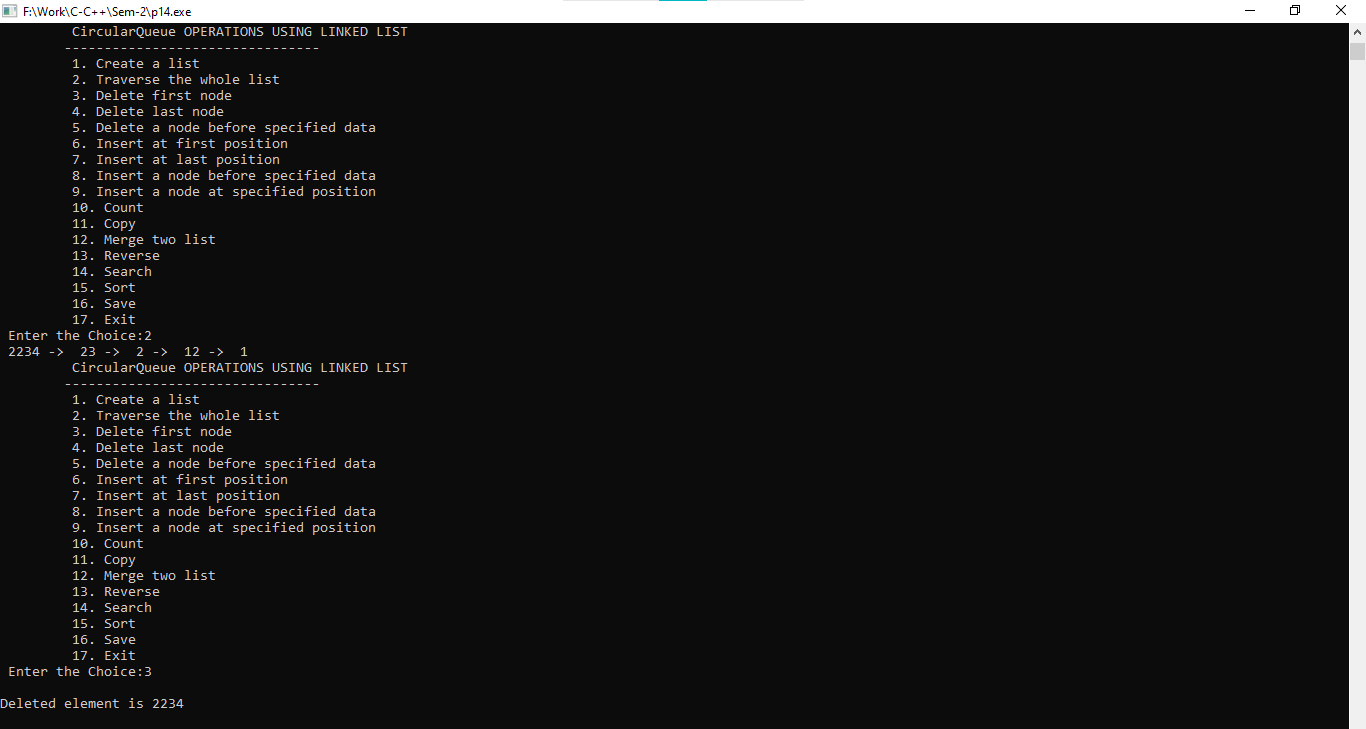
while(choice!=17);

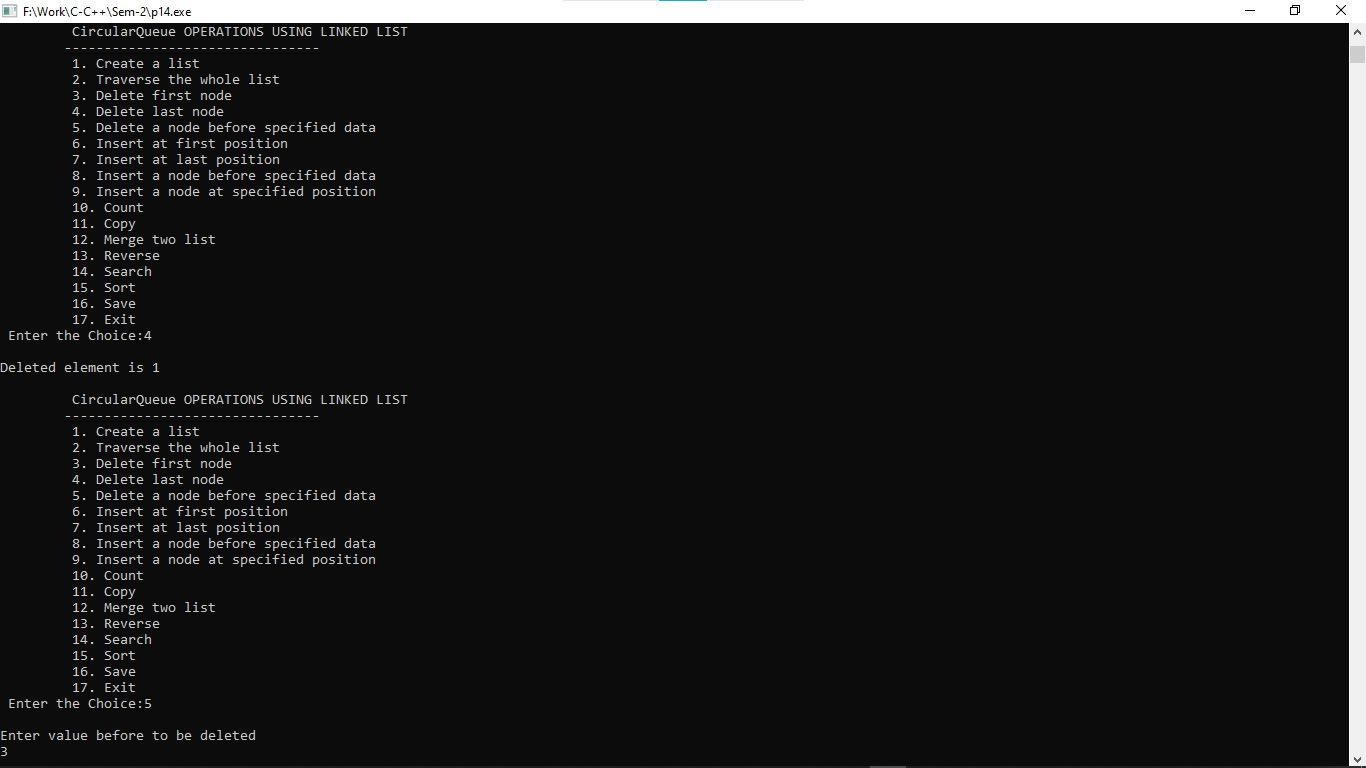
return 0;

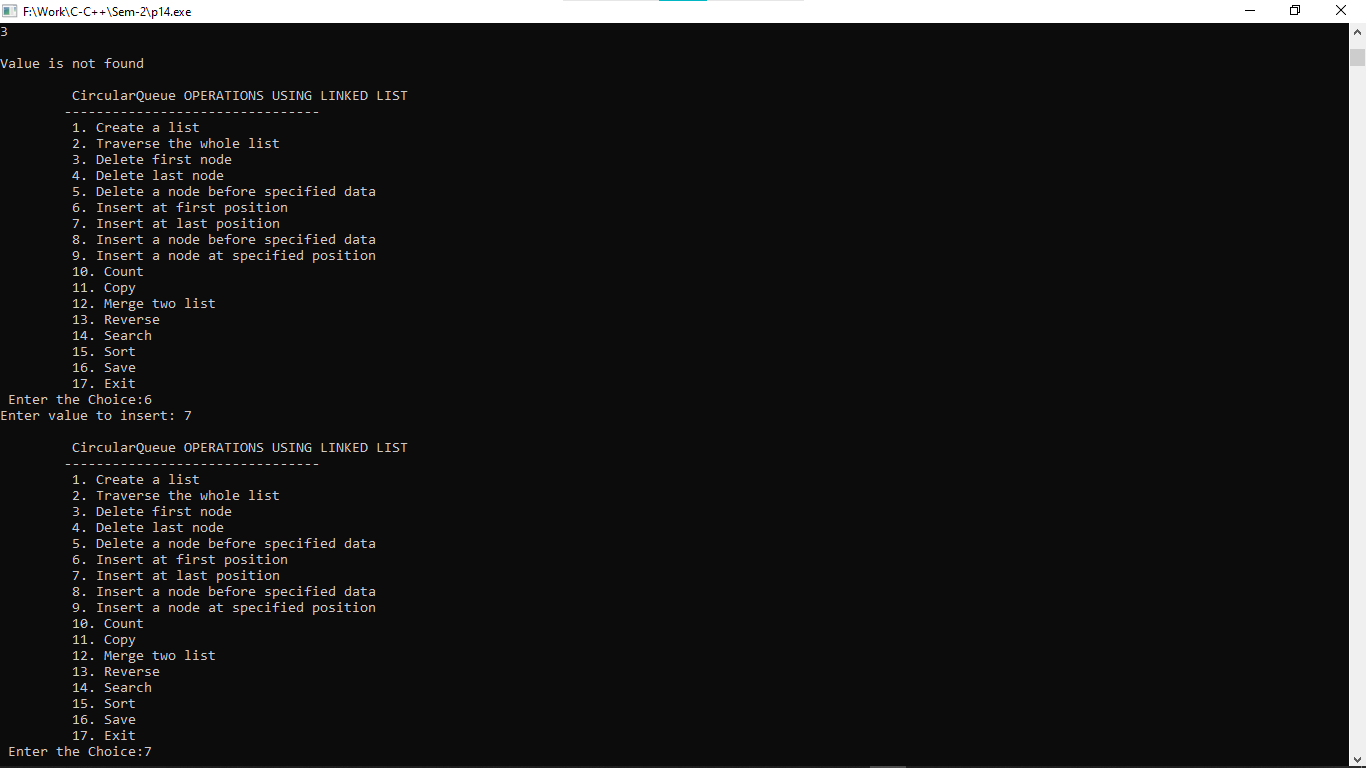
}

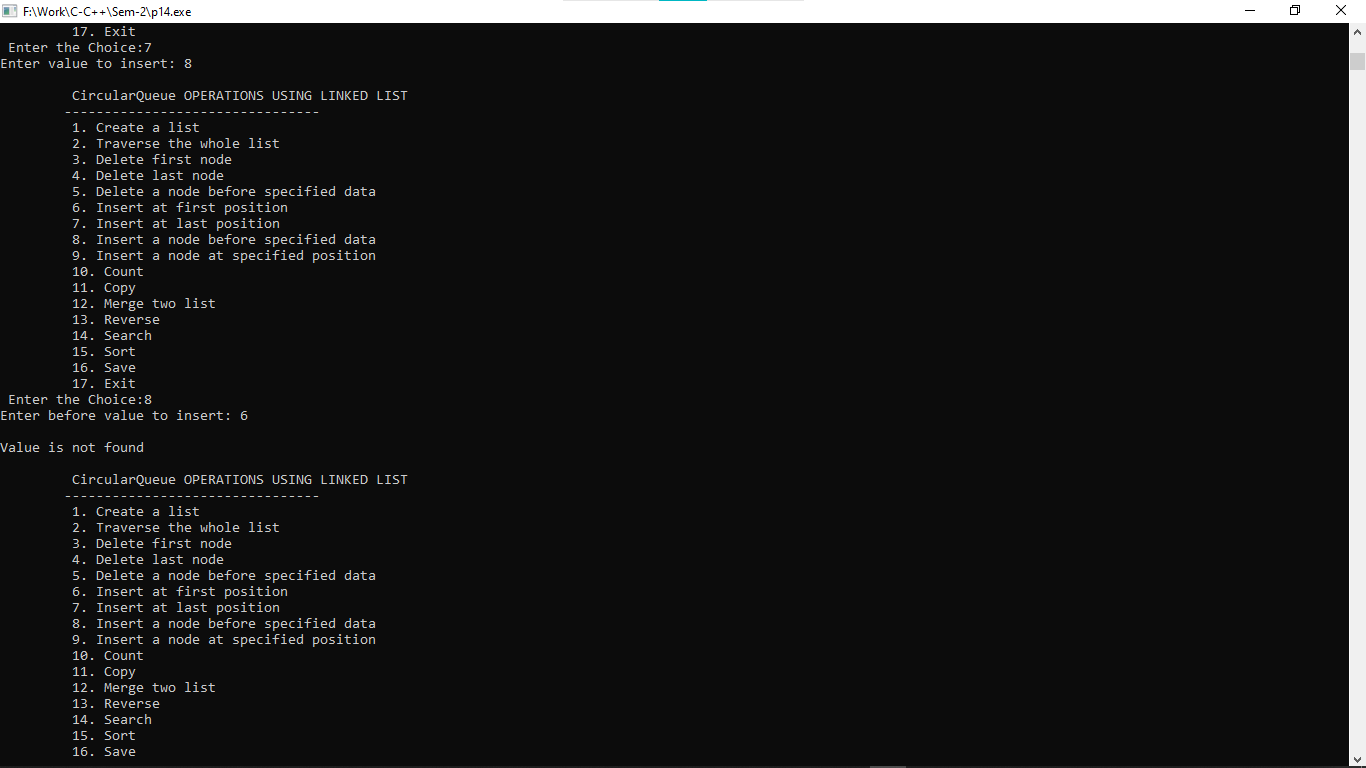
**Output:-**

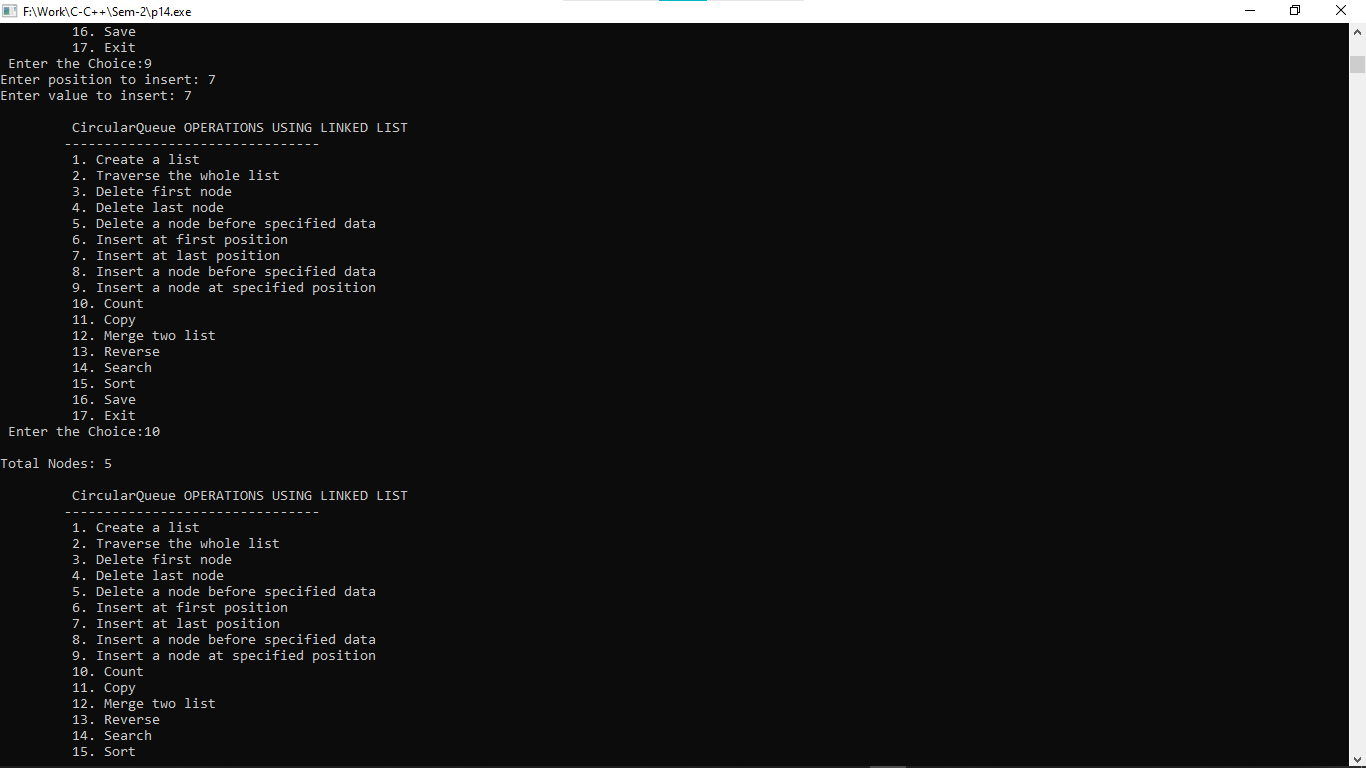
****

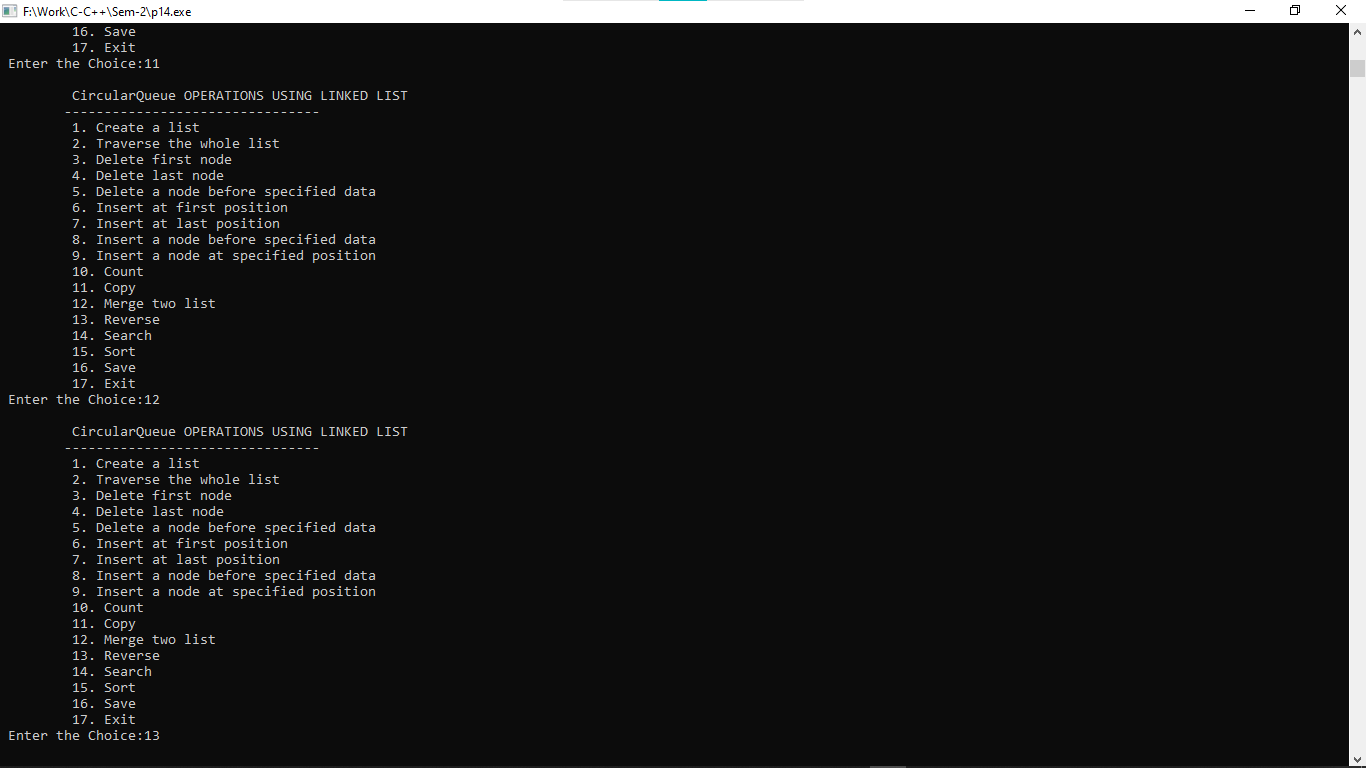
****

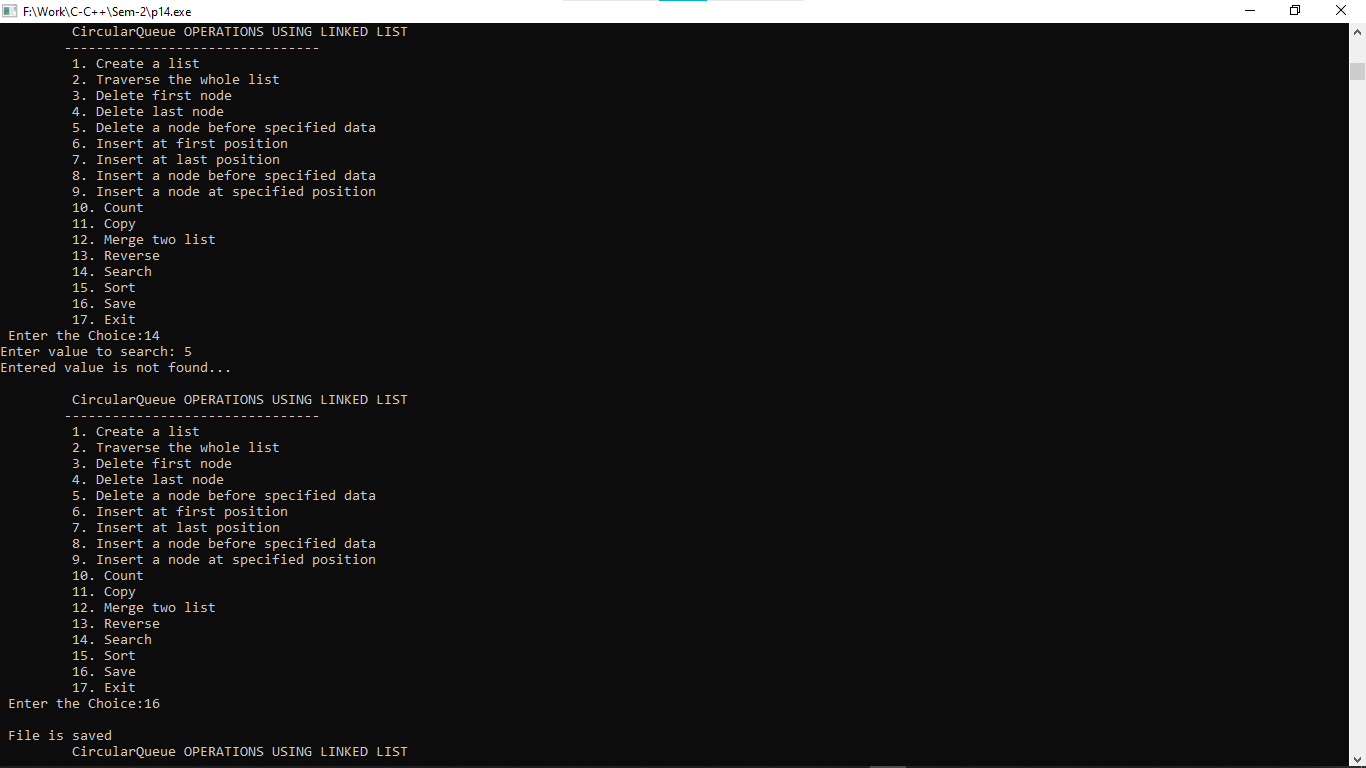
****

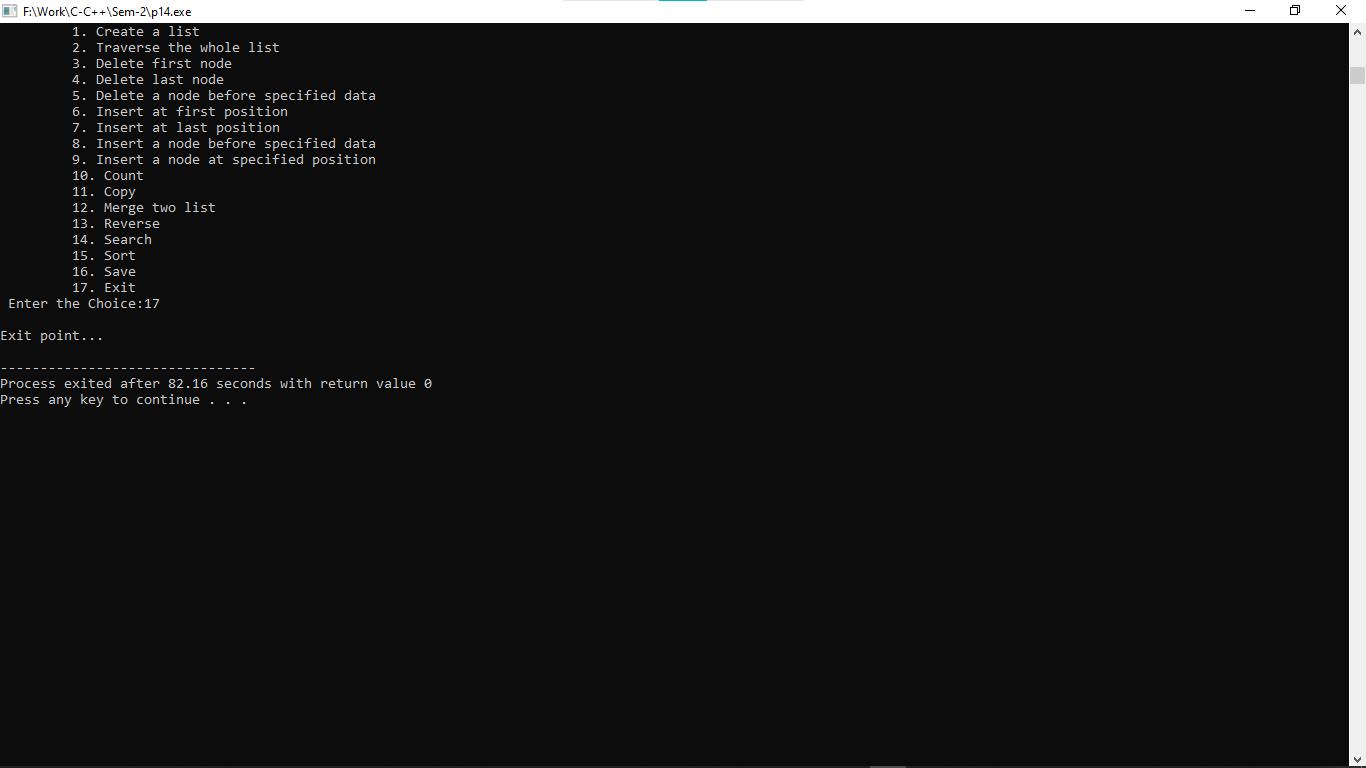
****

****

****

****

****

****

**Practical:-15**

**15. Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node**

**3. A link to the previous node Perform the following operations on the doubly-linked list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort**

#include<stdio.h>

#include<stdlib.h>

struct queue{

struct queue \*prev;

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->prev=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

temp->prev=rear;

rear->next=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next!=NULL){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

start->prev=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next!=NULL){

rear=temp;

temp=temp->next;

}

rear->next=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*pre;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

start->prev=NULL;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

pre=temp;

temp=temp->next;

if(temp->next->data==i){

pre->next=temp->next;

temp->next->prev=pre;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->prev=NULL;

temp->next=start;

start->prev=temp;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

rear->next=temp;

temp->prev=rear;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

temp->prev=NULL;

if(count==1){

temp->next=start;

start->prev=temp;

start=temp;

}

else{

trav=start;

while(trav!=NULL){

pre=trav;

trav=trav->next;

if(trav->data==val){

pre->next=temp;

temp->prev=pre;

temp->next=trav;

trav->prev=temp;

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next!=NULL){

count++;

pre=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

trav->prev=temp;

pre->next=temp;

temp->prev=pre;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=rear;

while(temp->prev!=NULL){

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

temp=temp->prev;

}

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

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

while(temp!=NULL){

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next!=NULL){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

printf("\n\t--------------------------------");

printf("\n\t 1. Create a list");

printf("\n\t 2. Traverse the whole list");

printf("\n\t 3. Delete first node");

printf("\n\t 4. Delete last node");

printf("\n\t 5. Delete a node before specified data");

printf("\n\t 6. Insert at first position");

printf("\n\t 7. Insert at last position");

printf("\n\t 8. Insert a node before specified data");

printf("\n\t 9. Insert a node at specified position");

printf("\n\t 10. Count");

printf("\n\t 11. Copy");

printf("\n\t 12. Merge two list");

printf("\n\t 13. Reverse");

printf("\n\t 14. Search");

printf("\n\t 15. Sort");

printf("\n\t 16. Save");

printf("\n\t 17. Exit");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

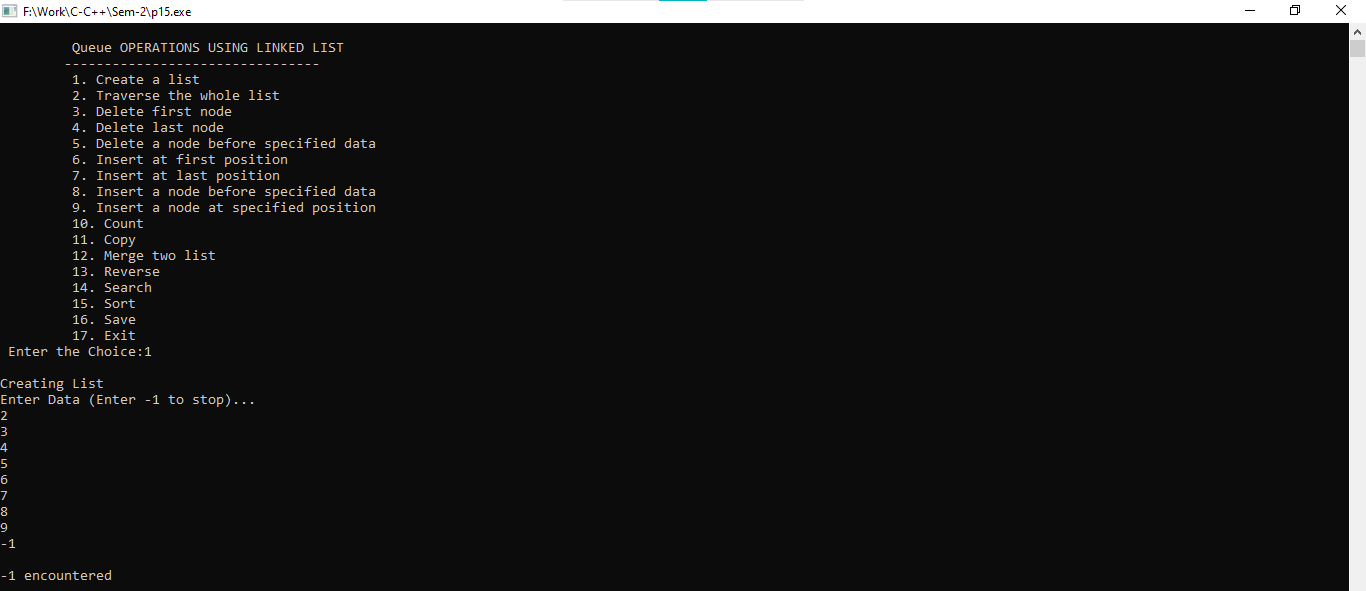
}

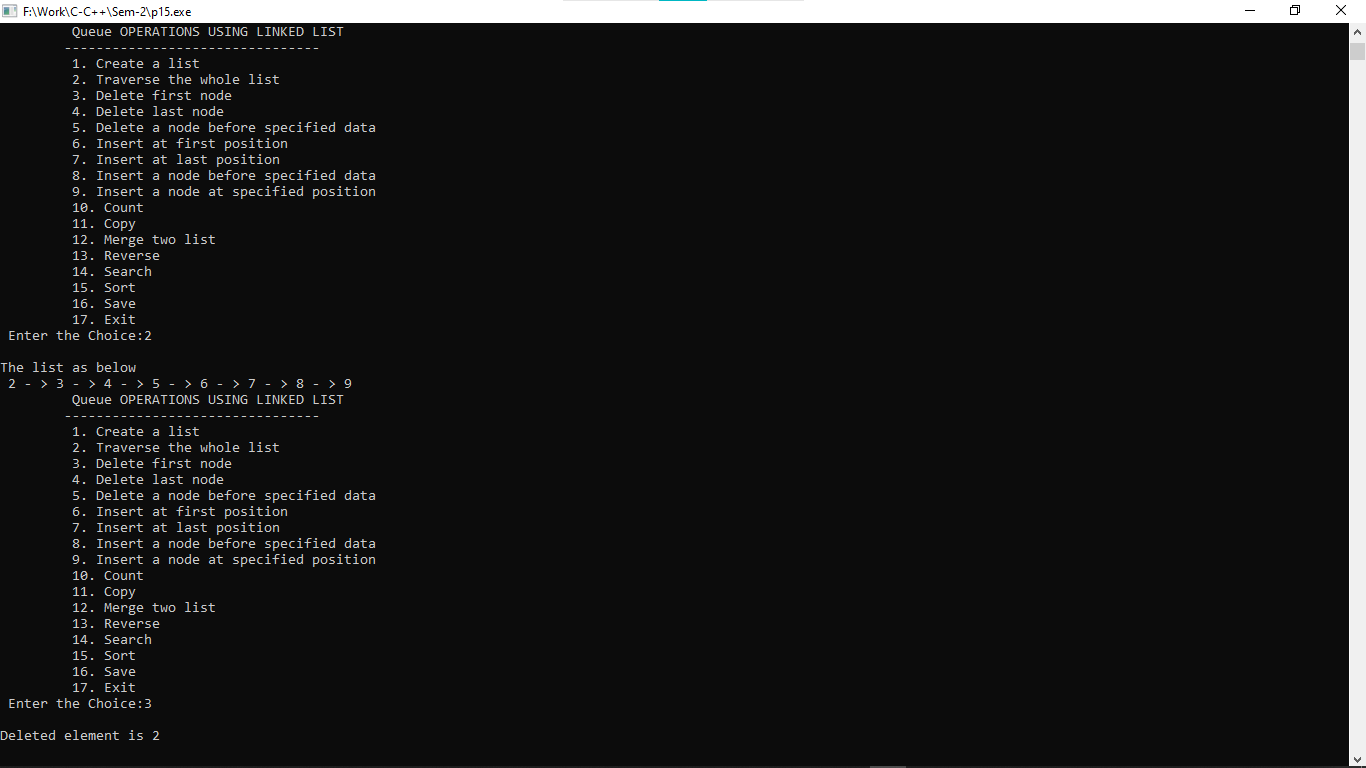
while(choice!=17);

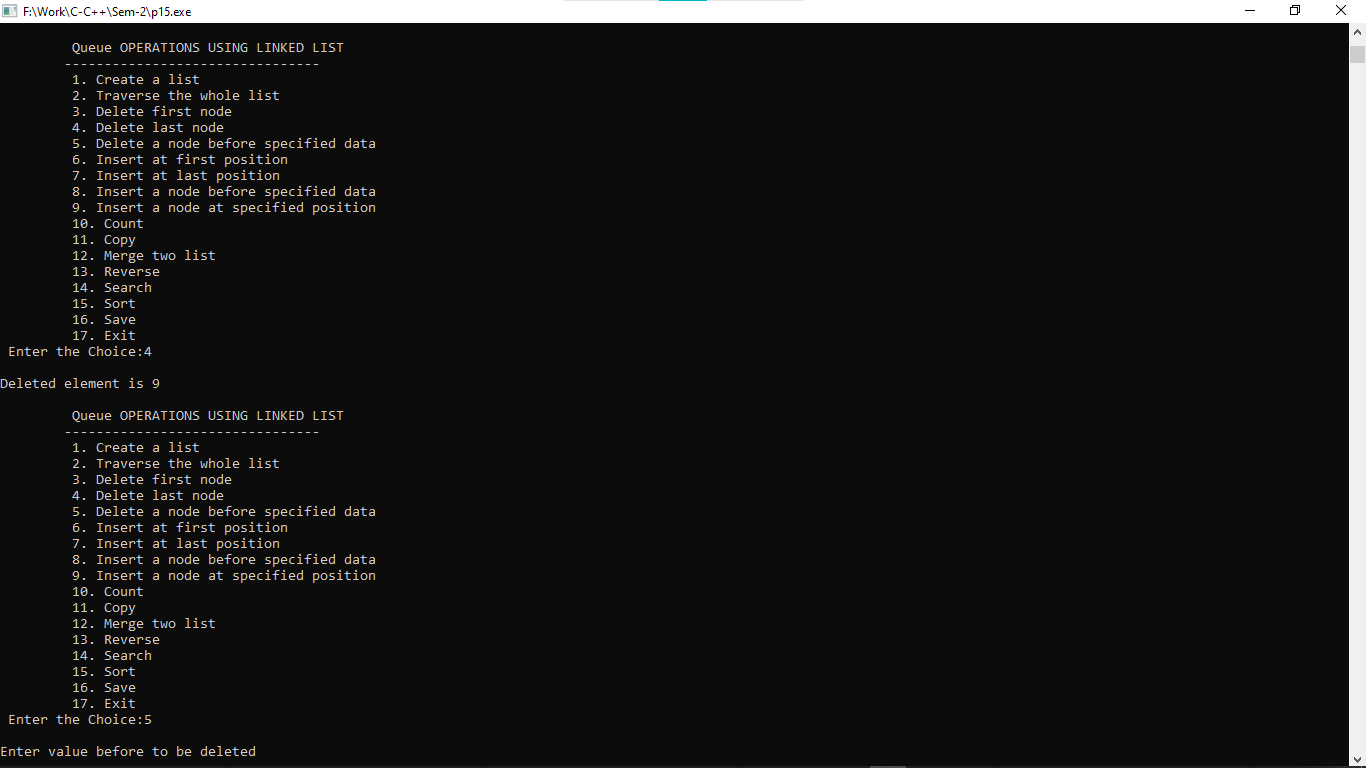
return 0;

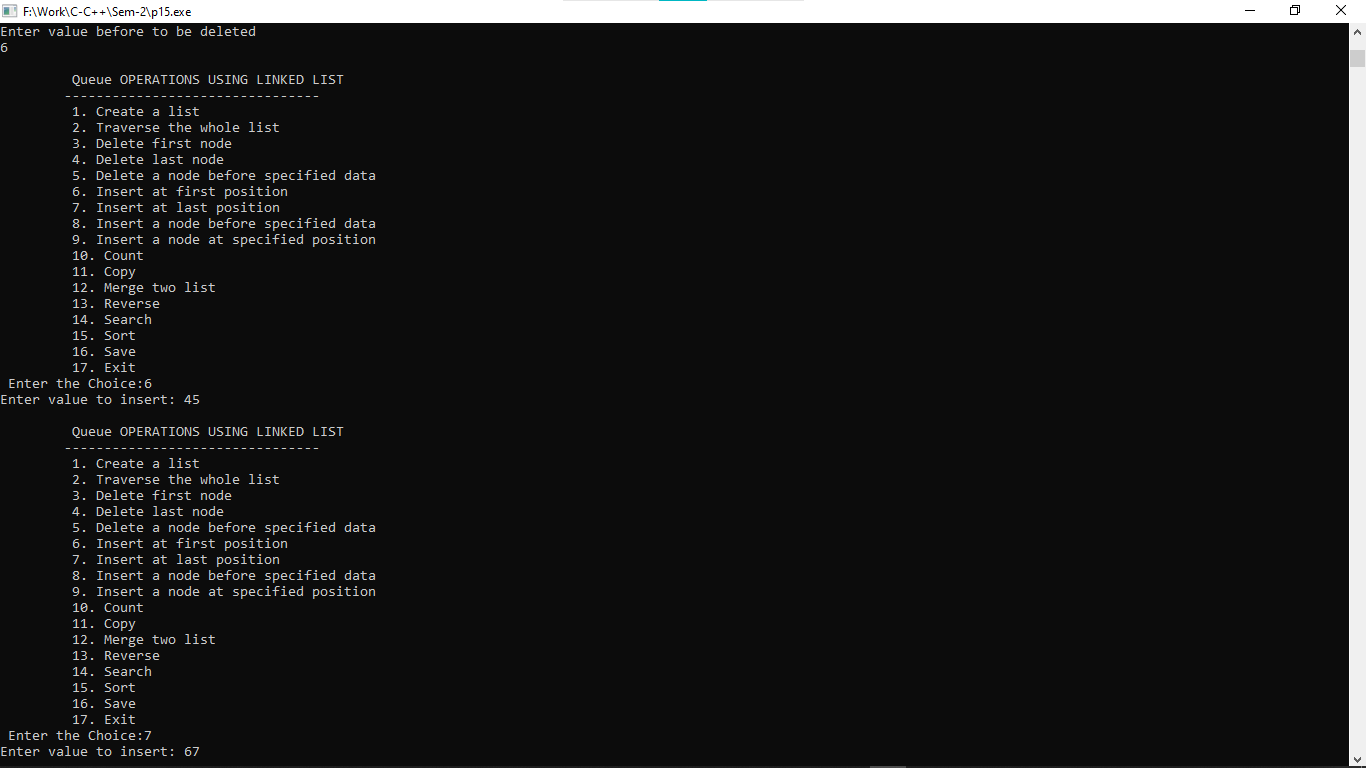
}

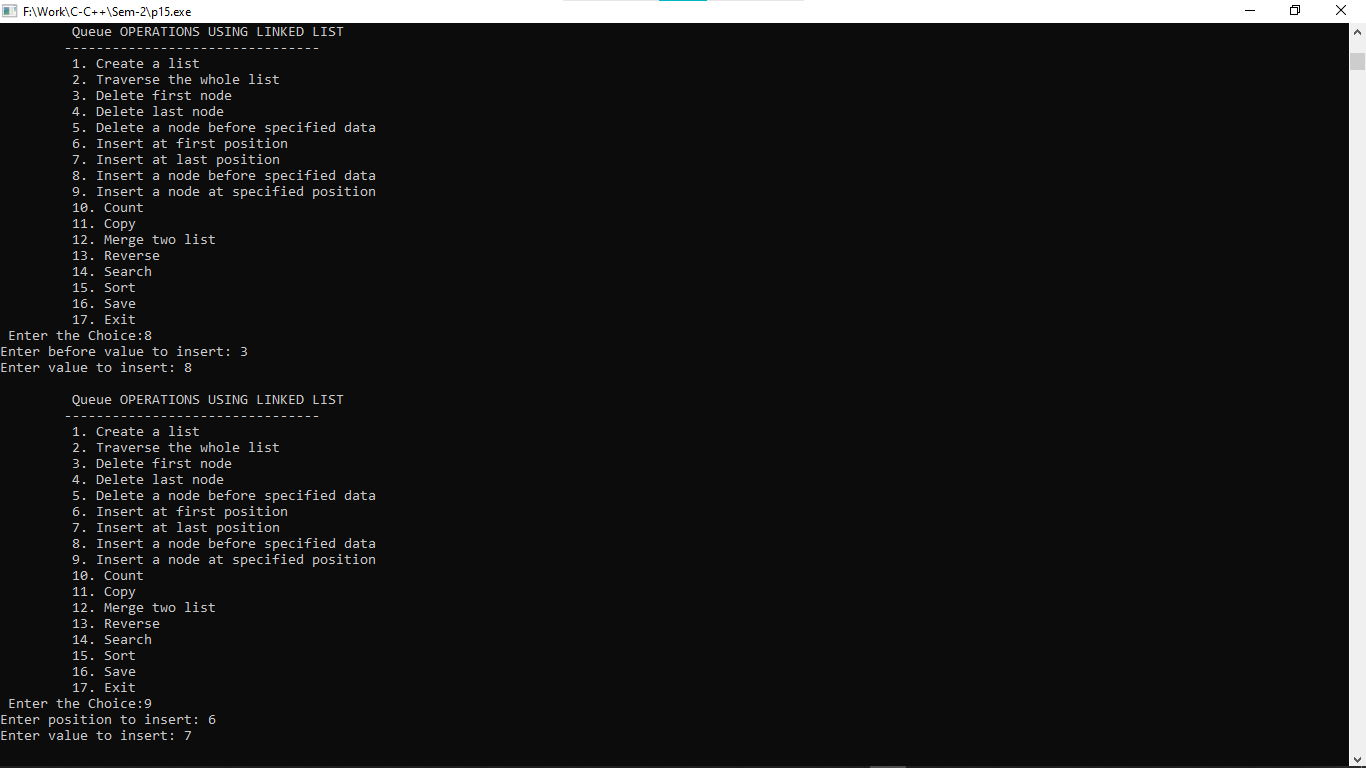
**Output:-**

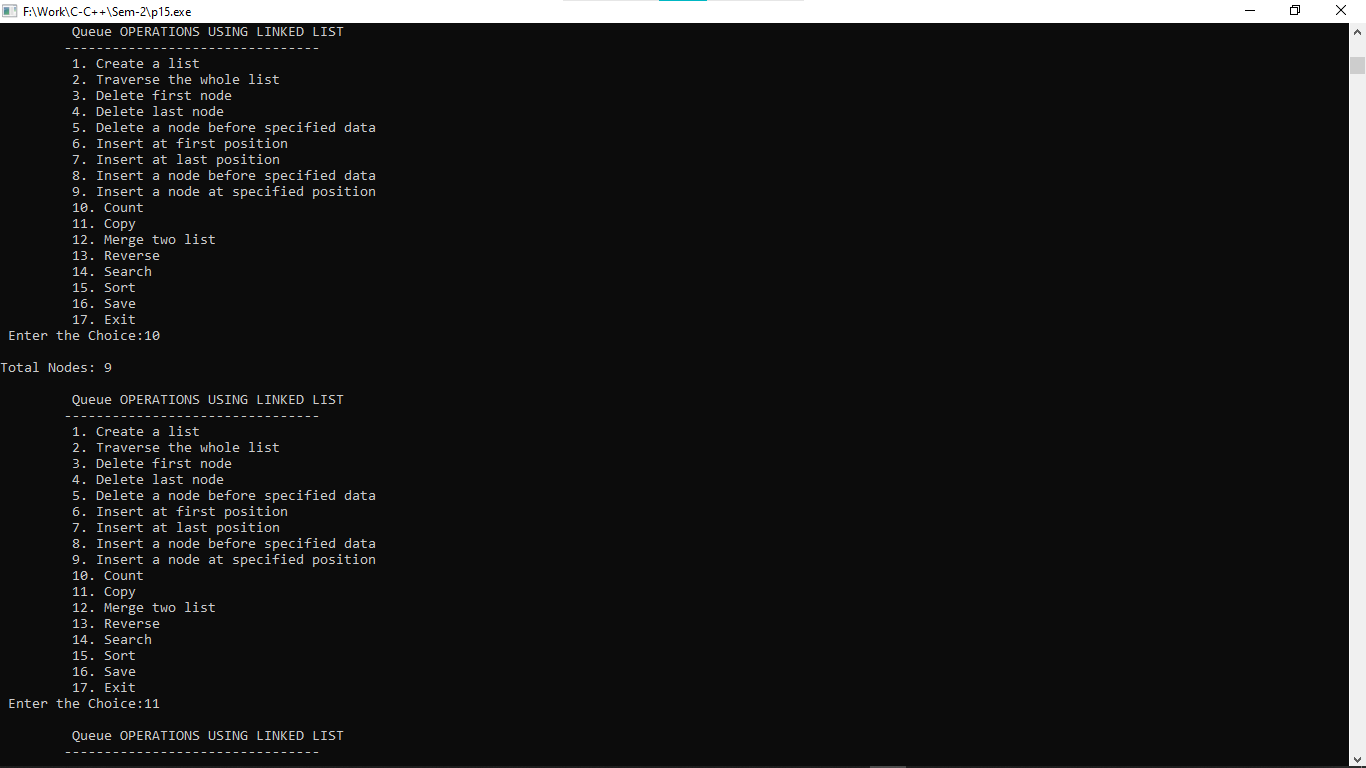
****

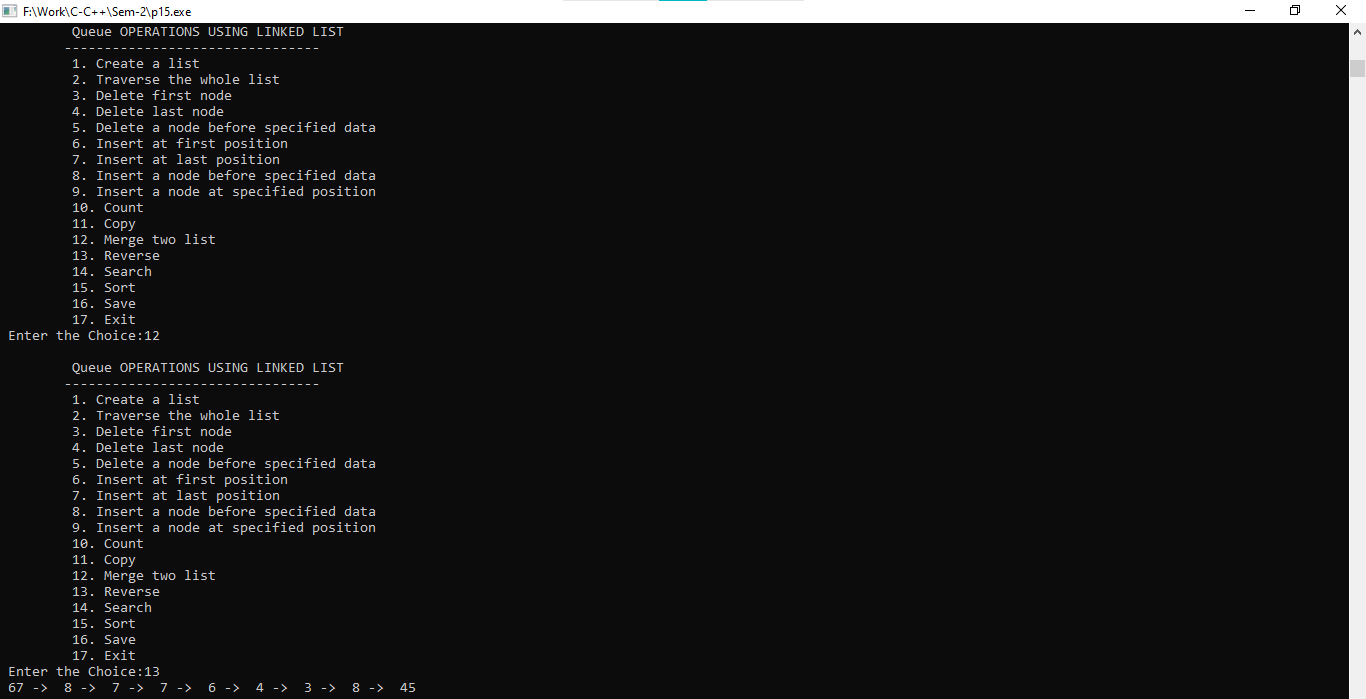
****

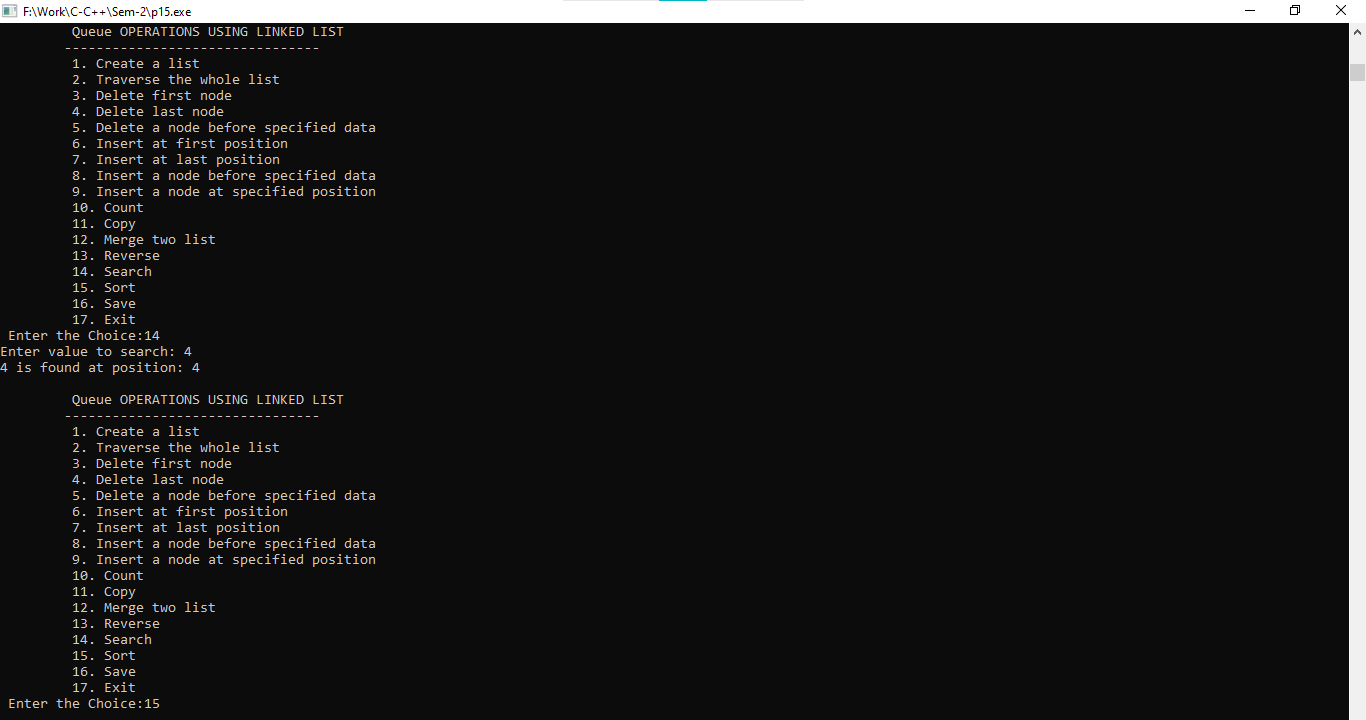
****

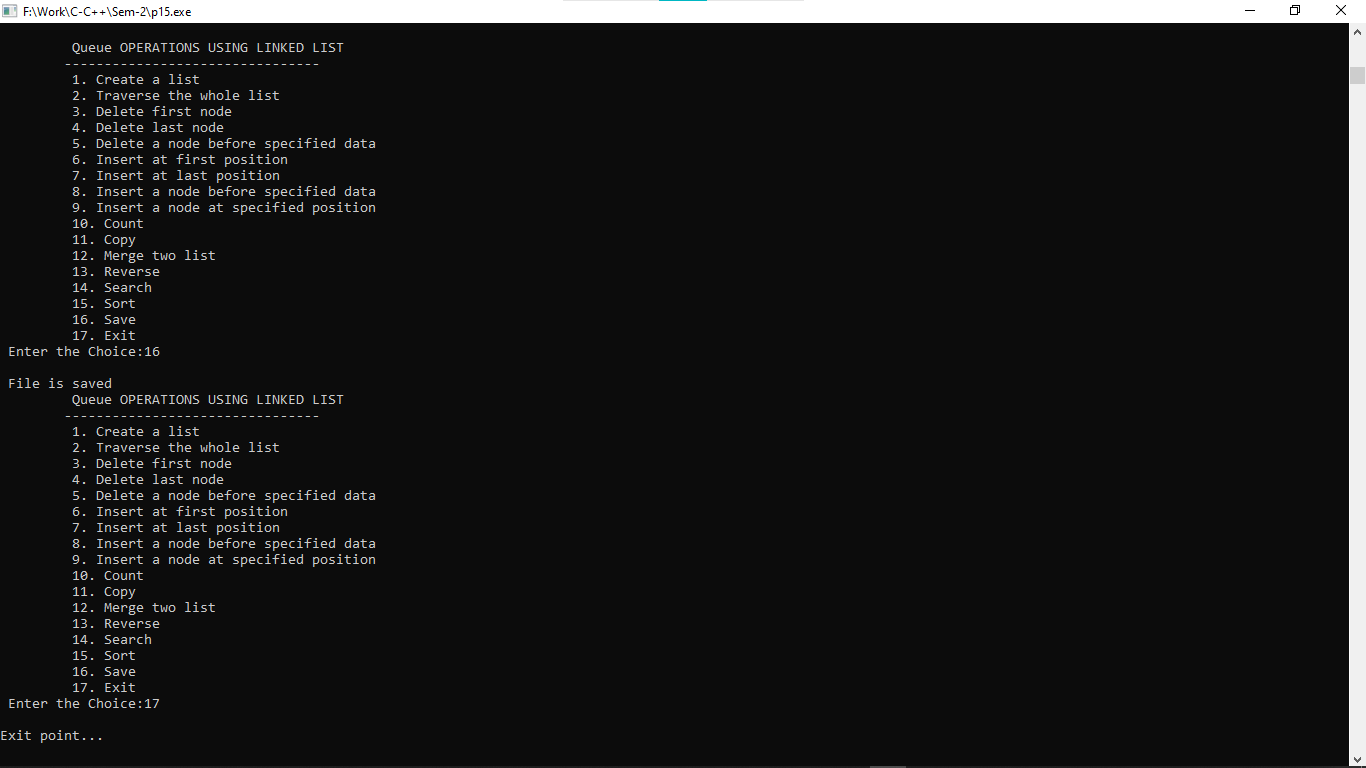
****

****

****

****

****

****

**Practical:-16**

**16. Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node**

**3. A link to the previous node Perform the following operations on doubly-linked Circular list using user defined functions:**

**1. Create a list**

**2. Traverse the whole list**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list.**

#include<stdio.h>

#include<stdlib.h>

struct queue{

struct queue \*prev;

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->prev=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

temp->prev=rear;

rear->next=temp;

temp->next=start;

start->prev=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next->data!=start->data){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

start->prev=rear;

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=rear;

rear=rear->prev;

rear->next=start;

start->prev=rear;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*pre;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

do{

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

start->prev=rear;

rear->next=start;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

pre=temp;

temp=temp->next;

if(temp->next->data==i){

pre->next=temp->next;

temp->next->prev=pre;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->prev=rear;

temp->next=start;

start->prev=temp;

rear->next=temp;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

rear->next=temp;

temp->prev=rear;

start->prev=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

temp->prev=NULL;

if(count==1){

temp->next=start;

start->prev=temp;

rear->next=temp;

temp->prev=rear;

start=temp;

}

else{

trav=start;

do{

pre=trav;

trav=trav->next;

if(trav->data==val){

pre->next=temp;

temp->prev=pre;

temp->next=trav;

trav->prev=temp;

break;

}

}while(trav->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next->data!=start->data){

count++;

pre=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

trav->prev=temp;

pre->next=temp;

temp->prev=pre;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=rear;

while(temp->prev->data!=rear->data){

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

temp=temp->prev;

}

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

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}while(temp->data!=start->data);

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}while(temp->data!=start->data);

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_doublylist.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next->data!=start->data){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t doubly-linked Circular list");

printf("\n\t--------------------------------");

printf("\n\t 1. Create a list");

printf("\n\t 2. Traverse the whole list");

printf("\n\t 3. Delete first node");

printf("\n\t 4. Delete last node");

printf("\n\t 5. Delete a node before specified data");

printf("\n\t 6. Insert at first position");

printf("\n\t 7. Insert at last position");

printf("\n\t 8. Insert a node before specified data");

printf("\n\t 9. Insert a node at specified position");

printf("\n\t 10. Count");

printf("\n\t 11. Copy");

printf("\n\t 12. Merge two list");

printf("\n\t 13. Reverse");

printf("\n\t 14. Search");

printf("\n\t 15. Sort");

printf("\n\t 16. Save");

printf("\n\t 17. Exit");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

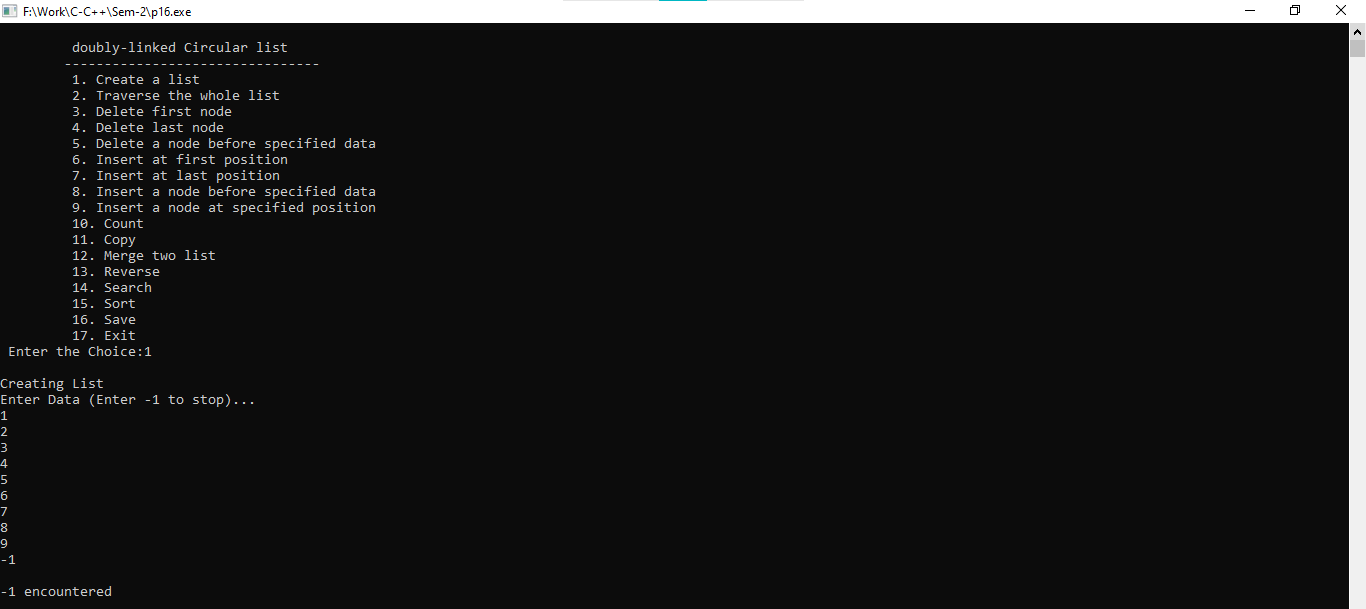
}

while(choice!=17);

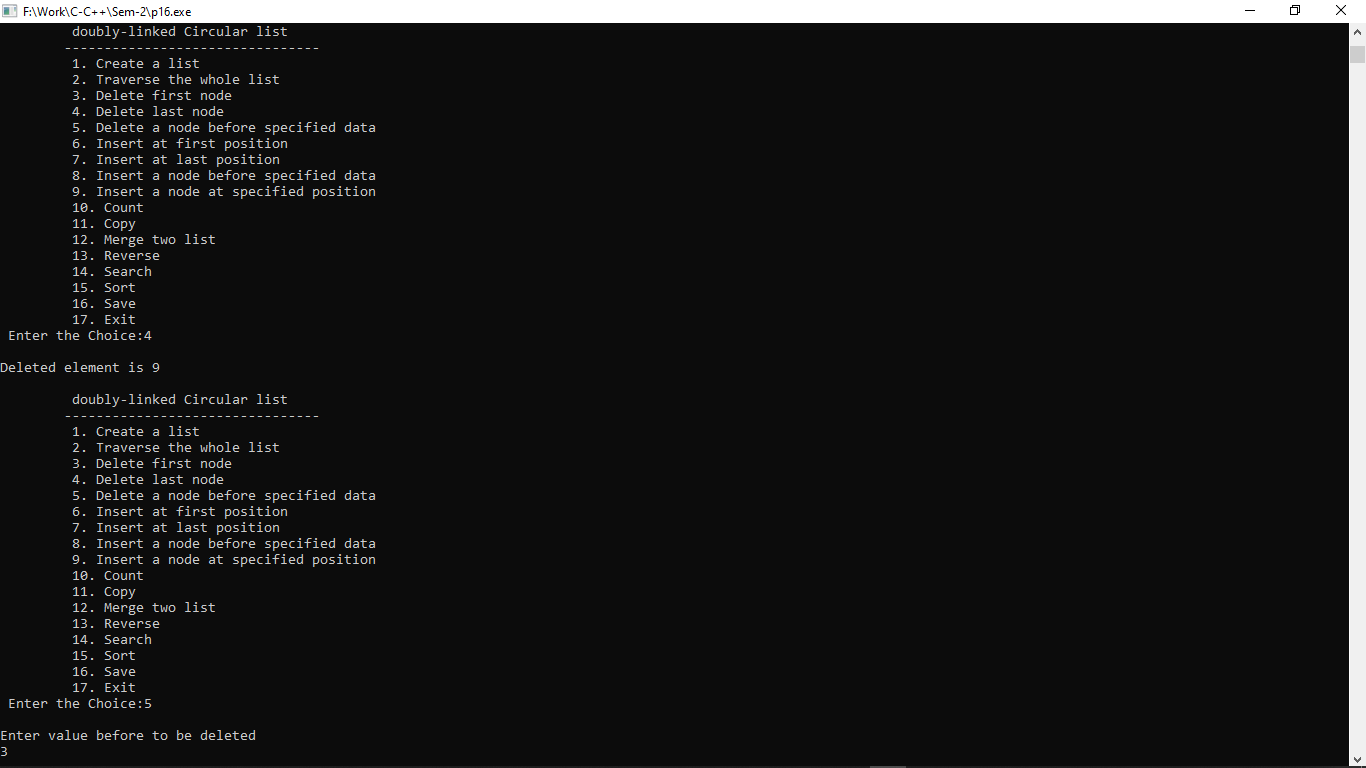
return 0;

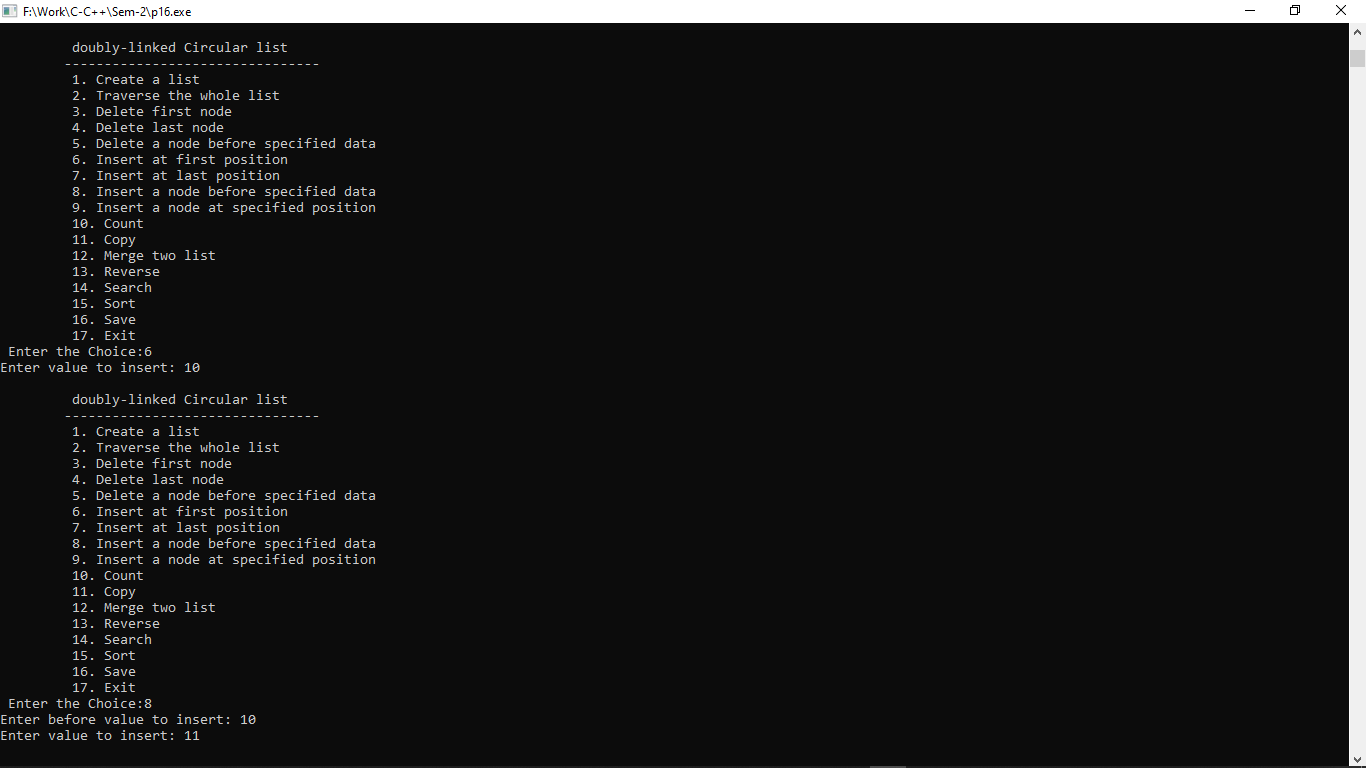
}

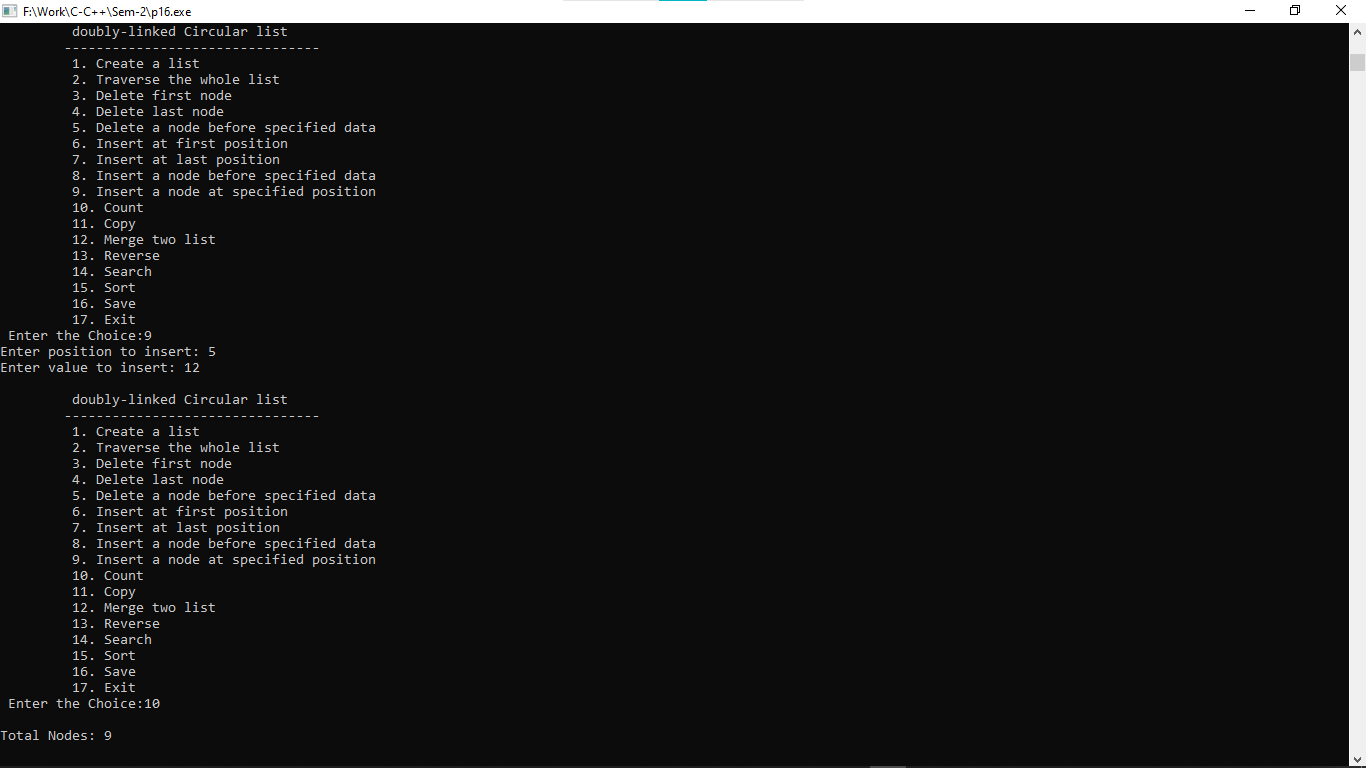
**Output:-**

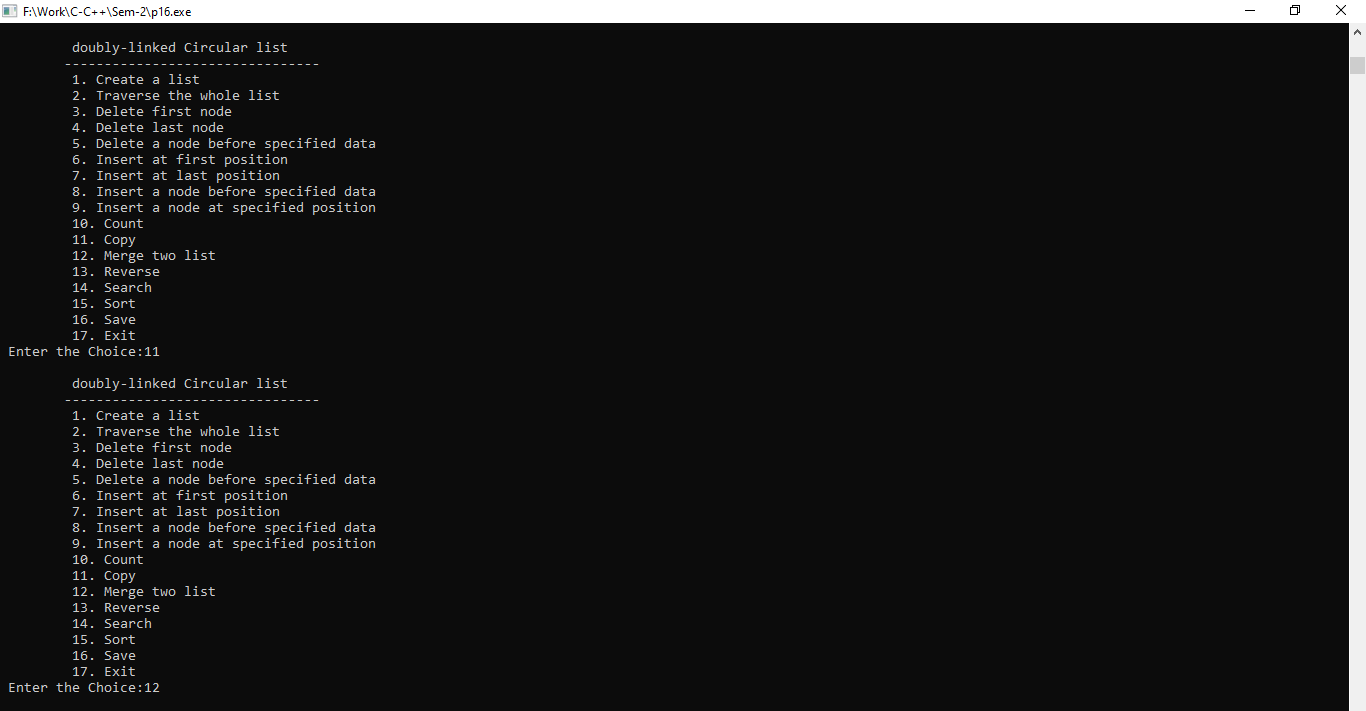
****

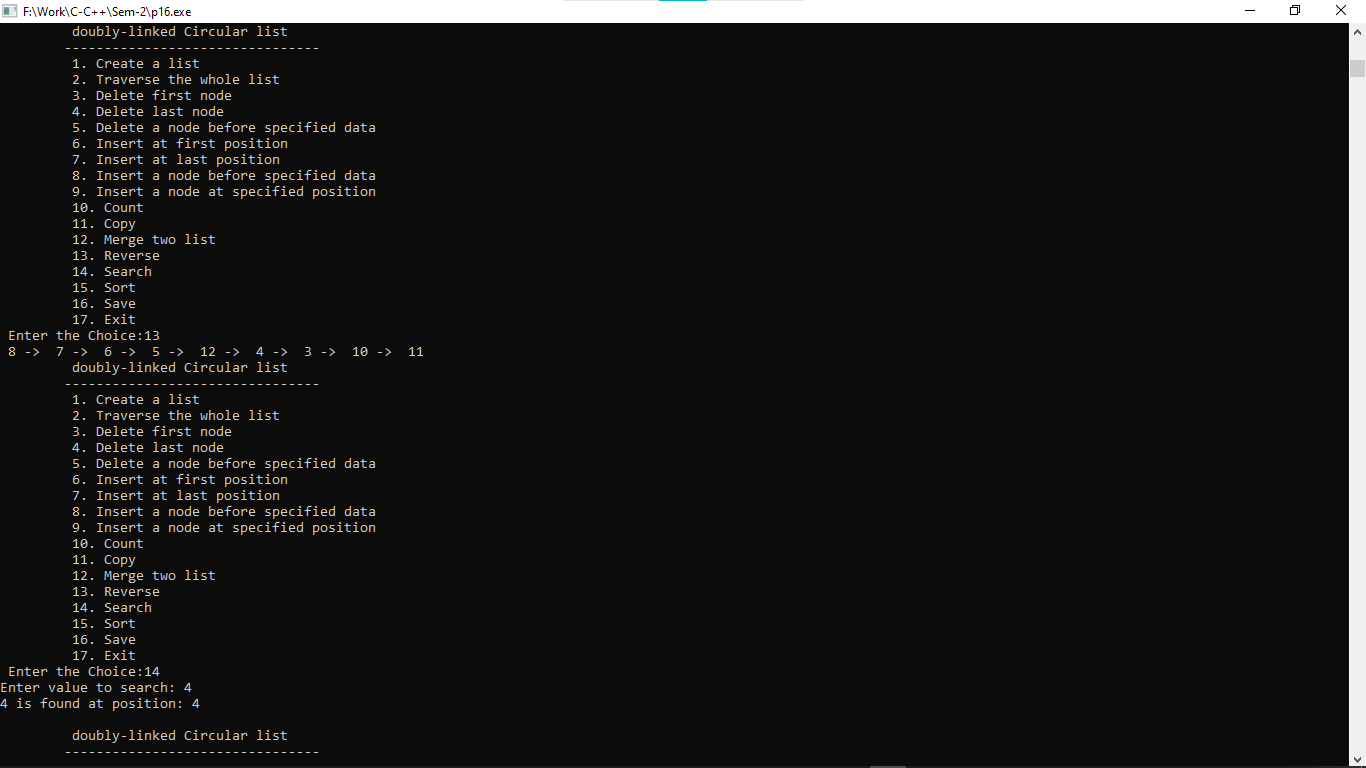
****

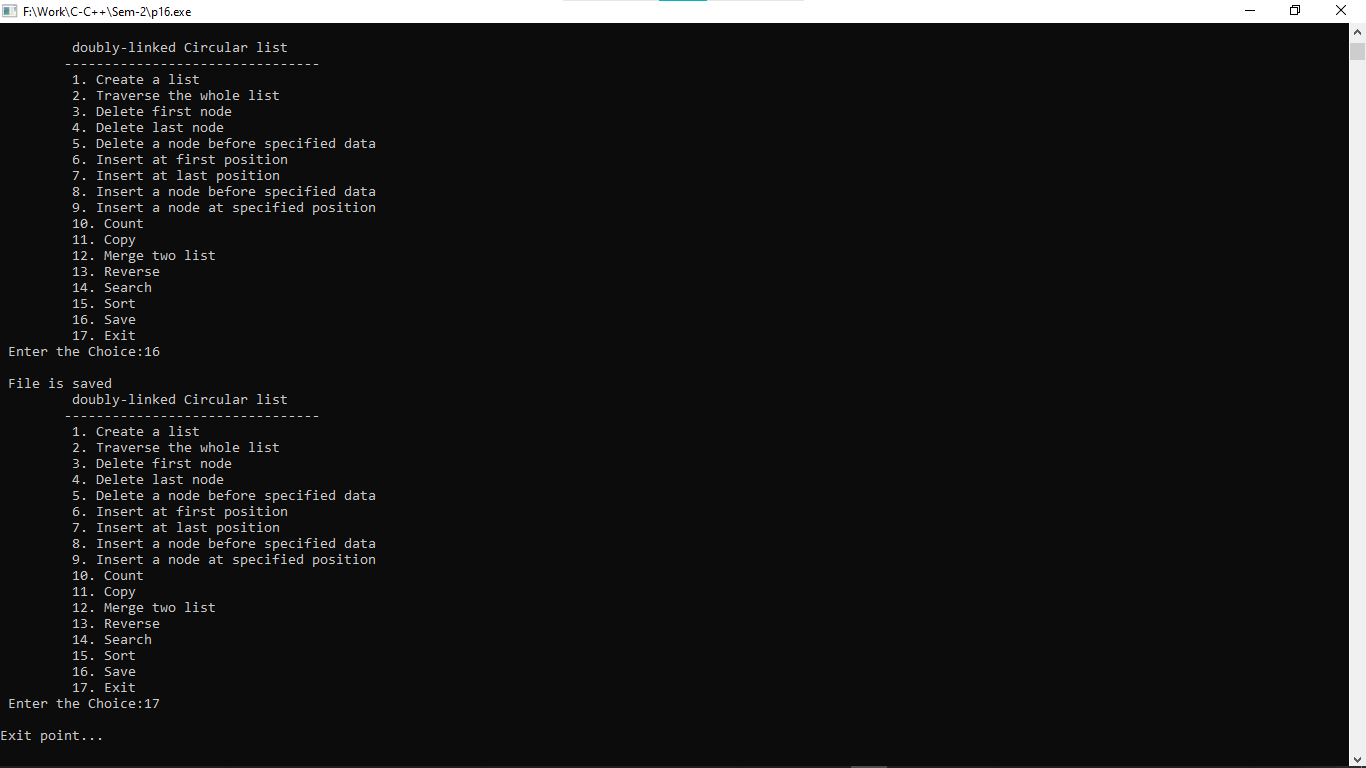
****

****

****

****

****

****

**Practical:-17**

**17. Write a program to represent an undirected graph using the adjacency matrix to implement the graph and perform following operations, with menu driven options for following tasks:**

**1. Create graph**

**2. Insert an edge**

**3. Print Adjacency Matrix**

**4. List all vertices that are adjacent to a specified vertex.**

**5. Print out vertices using depth first search**

**6. Print out vertices using breadth first search**

**7. Exit program**

#include <stdio.h>

#include <stdlib.h>

int dir\_graph();

int undir\_graph();

int read\_graph(int adj\_mat[50][50], int n );

void main()

{

int option;

do

{

printf("Adjacency Matrix method \n ");

printf("\n 1. Directed Graph ");

printf("\n 2. Un-Directed Graph ");

printf("\n 3. Exit ");

printf("\n\n Select a proper option : ");

scanf("%d", &option);

switch(option)

{

case 1 : dir\_graph();

break;

case 2 : undir\_graph();

break;

case 3 : exit(0);

} // switch

}while(1);

}

int dir\_graph()

{

int adj\_mat[50][50];

int n;

int in\_deg, out\_deg, i, j;

printf("\n How Many Vertices : ");

scanf("%d", &n);

read\_graph(adj\_mat, n);

printf("\n Vertex \t In\_Degree \t Out\_Degree \t Total\_Degree ");

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

{

in\_deg = out\_deg = 0;

for ( j = 1 ; j <= n ; j++ )

{

if ( adj\_mat[j][i] == 1 )

in\_deg++;

}

for ( j = 1 ; j <= n ; j++ )

if (adj\_mat[i][j] == 1 )

out\_deg++;

printf("\n\n %5d\t\t\t%d\t\t%d\t\t%d\n\n",i,in\_deg,out\_deg,in\_deg+out\_deg);

}

return;

}

int undir\_graph()

{

int adj\_mat[50][50];

int deg, i, j, n;

printf("\n How Many Vertices : ");

scanf("%d", &n);

read\_graph(adj\_mat, n);

printf("\n Vertex \t Degree ");

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

{

deg = 0;

for ( j = 1 ; j <= n ; j++ )

if ( adj\_mat[i][j] == 1)

deg++;

printf("\n\n %5d \t\t %d\n\n", i, deg);

}

return;

}

int read\_graph ( int adj\_mat[50][50], int n )

{

int i, j;

char reply;

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

{

for ( j = 1 ; j <= n ; j++ )

{

if ( i == j )

{

adj\_mat[i][j] = 0;

continue;

}

printf("\n Vertices %d & %d are Adjacent (Y/N) :",i,j);

scanf("%c", &reply);

if ( reply == 'y' || reply == 'Y' )

adj\_mat[i][j] = 1;

else

adj\_mat[i][j] = 0;

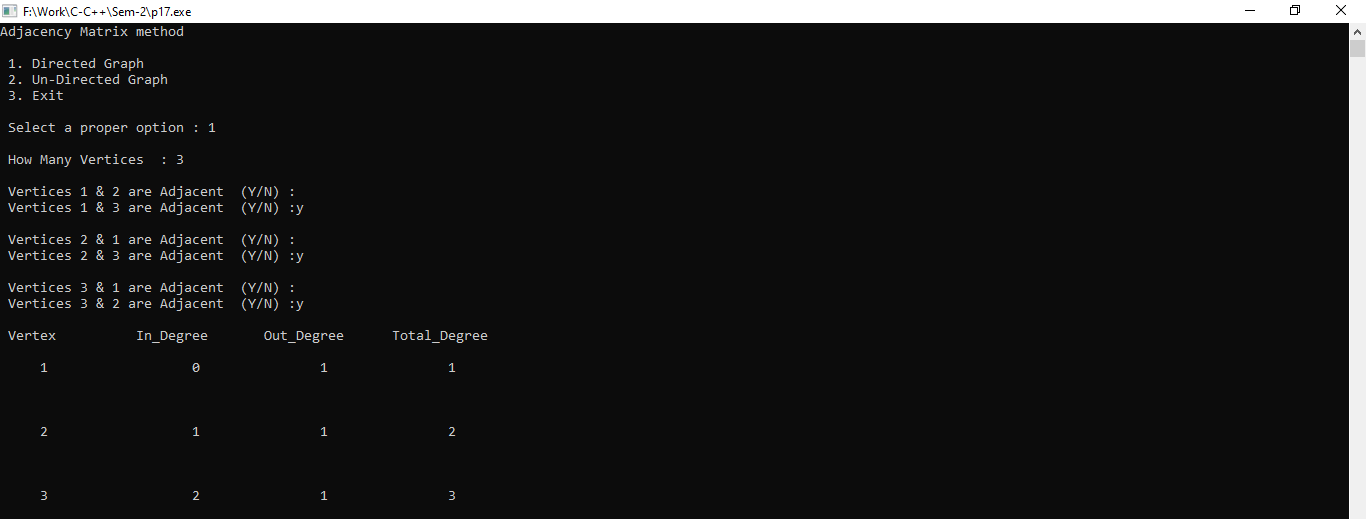
}

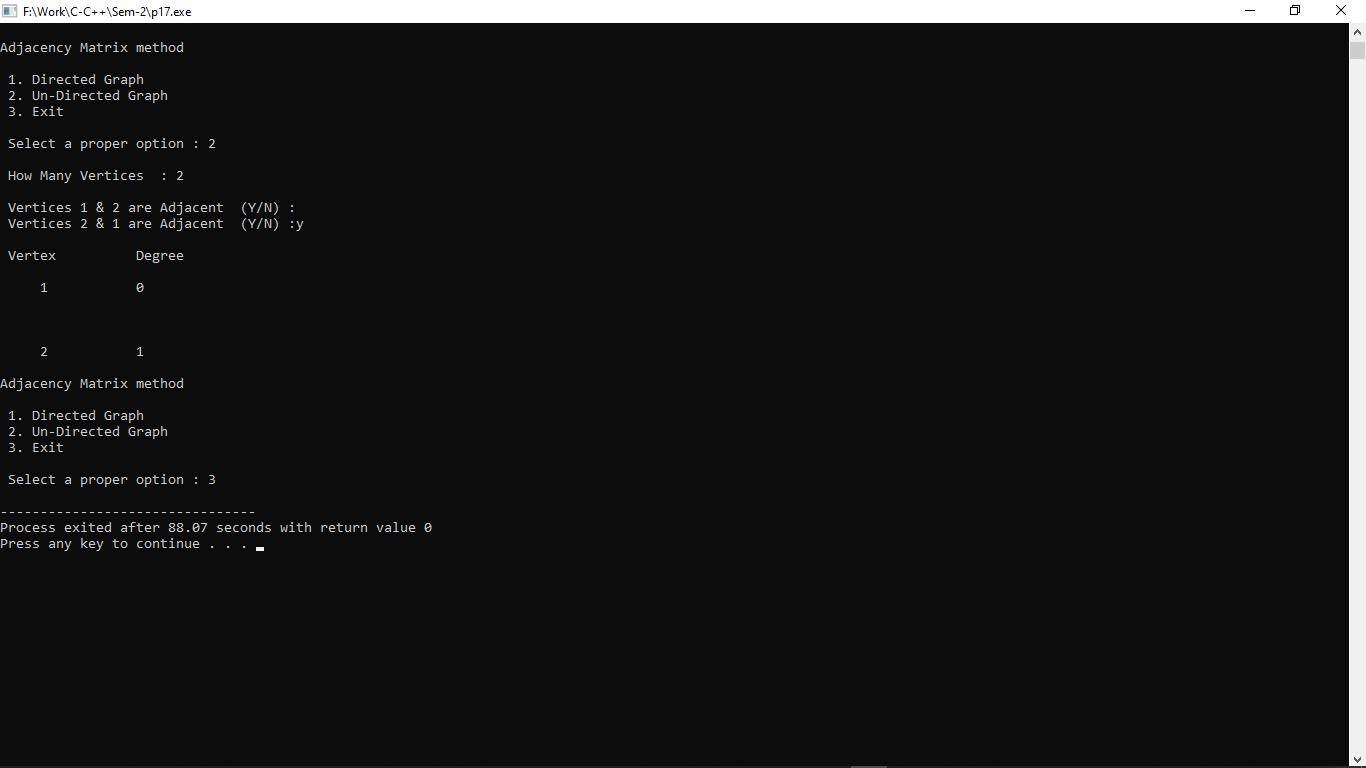
}

return;

}

**Output:-**

****

****

**Practical:-18**

**18. Create a user-defined structure with the following data members:**

**1. A Data 2. A link to the Left child**

**3. A link to the Right child Perform the following operations on Binary Search Tree using recursion:**

**1. Create 2. Traverse (Inorder, Preorder, Postorder)**

**3. Insert**

**4. Delete**

**5. Search**

**6. Create a file which stores all values of traversal.**

#include<stdio.h>

#include<malloc.h>

#include<conio.h>

struct tree {

int info;

struct tree \*left;

struct tree \*right;

};

struct tree \*insert(struct tree \*,int);

void inorder(struct tree \*);

void postorder(struct tree \*);

void preorder(struct tree \*);

struct tree \*delet(struct tree \*,int);

struct tree \*search(struct tree \*);

int main(void) {

struct tree \*root;

int choice, item,item\_no;

root = NULL;

do {

do {

printf("\n1. Insert in Binary Tree ");

printf("\n2. Delete from Binary Tree ");

printf("\n3. Inorder traversal of Binary tree");

printf("\n4. Postorder traversal of Binary tree");

printf("\n5. Preorder traversal of Binary tree");

printf("\n6. Search and replace ");

printf("\n7. Exit ");

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

scanf(" %d",&choice);

if(choice<1 || choice>7)

printf("\n Invalid choice - try again");

}

while (choice<1 || choice>7);

switch(choice) {

case 1:

printf("\n Enter new element: ");

scanf("%d", &item);

root= insert(root,item);

printf("\n root is %d",root->info);

printf("\n\n Inorder traversal of binary tree is : ");

inorder(root);

printf("\n");

break;

case 2:

printf("\n Enter the element to be deleted : ");

scanf(" %d",&item\_no);

root=delet(root,item\_no);

inorder(root);

printf("\n");

break;

case 3:

printf("\n Inorder traversal of binary tree is : ");

inorder(root);

printf("\n");

break;

case 4:

printf("\n Postorder traversal of binary tree is : ");

postorder(root);

printf("\n");

break;

case 5:

printf("\n Preorder traversal of binary tree is : ");

preorder(root);

printf("\n");

break;

case 6:

printf("\n Search and replace operation in binary tree \n");

root=search(root);

printf("\n");

break;

default:

printf("\n exit ");

}

}

while(choice !=7);

return(0);

}

struct tree \*insert(struct tree \*root, int x) {

if(!root) {

root=(struct tree\*)malloc(sizeof(struct tree));

root->info = x;

root->left = NULL;

root->right = NULL;

return(root);

}

if(root->info > x)

root->left = insert(root->left,x); else {

if(root->info < x)

root->right = insert(root->right,x);

}

return(root);

}

void inorder(struct tree \*root) {

if(root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

return;

}

void postorder(struct tree \*root) {

if(root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

return;

}

void preorder(struct tree \*root) {

if(root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

return;

}

struct tree \*delet(struct tree \*ptr,int x) {

struct tree \*p1,\*p2;

if(!ptr) {

printf("\n Node not found ");

return(ptr);

} else {

if(ptr->info < x) {

ptr->right = delet(ptr->right,x);

} else if (ptr->info >x) {

ptr->left=delet(ptr->left,x);

return ptr;

} else

{

if(ptr->info == x)

{

if(ptr->left == ptr->right)

{

free(ptr);

return(NULL);

} else if(ptr->left==NULL)

{

p1=ptr->right;

free(ptr);

return p1;

} else if(ptr->right==NULL)

{

p1=ptr->left;

free(ptr);

return p1;

} else {

p1=ptr->right;

p2=ptr->right;

while(p1->left != NULL)

p1=p1->left;

p1->left=ptr->left;

free(ptr);

return p2;

}

}

}

}

return(ptr);

}

struct tree \*search(struct tree \*root) {

int no,i,ino;

struct tree \*ptr;

ptr=root;

printf("\n Enter the element to be searched :");

scanf(" %d",&no);

fflush(stdin);

while(ptr) {

if(no>ptr->info)

ptr=ptr->right; else if(no<ptr->info)

ptr=ptr->left; else

break;

}

if(ptr) {

printf("\n Element %d which was searched is found and is = %d",no,ptr->info);

printf("\n\n Do you want replace it, press 1 for yes : ");

scanf(" %d",&i);

if(i==1) {

printf("\n Enter new element :");

scanf(" %d",&ino);

ptr->info=ino;

} else

printf("\n\t It's okay");

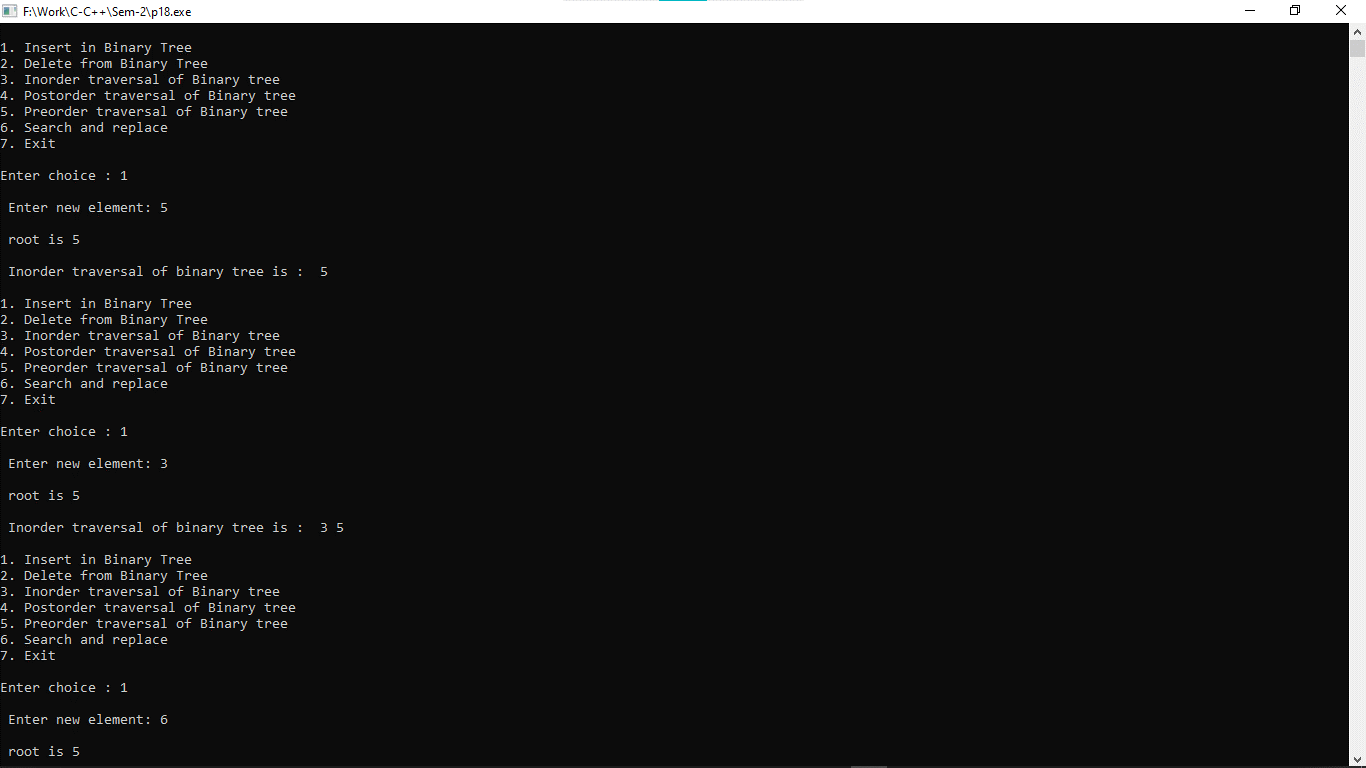
} else

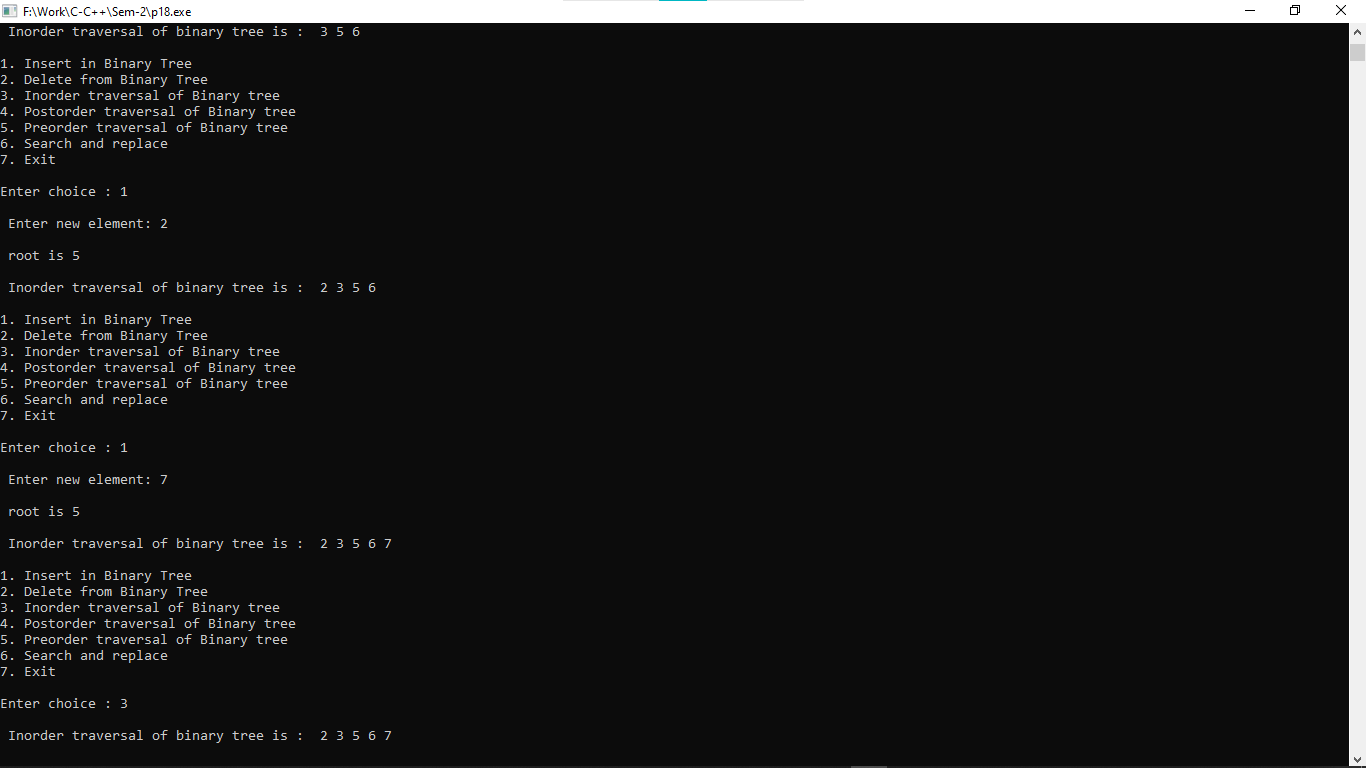
printf("\n Element %d does not exist in the binary tree",no);

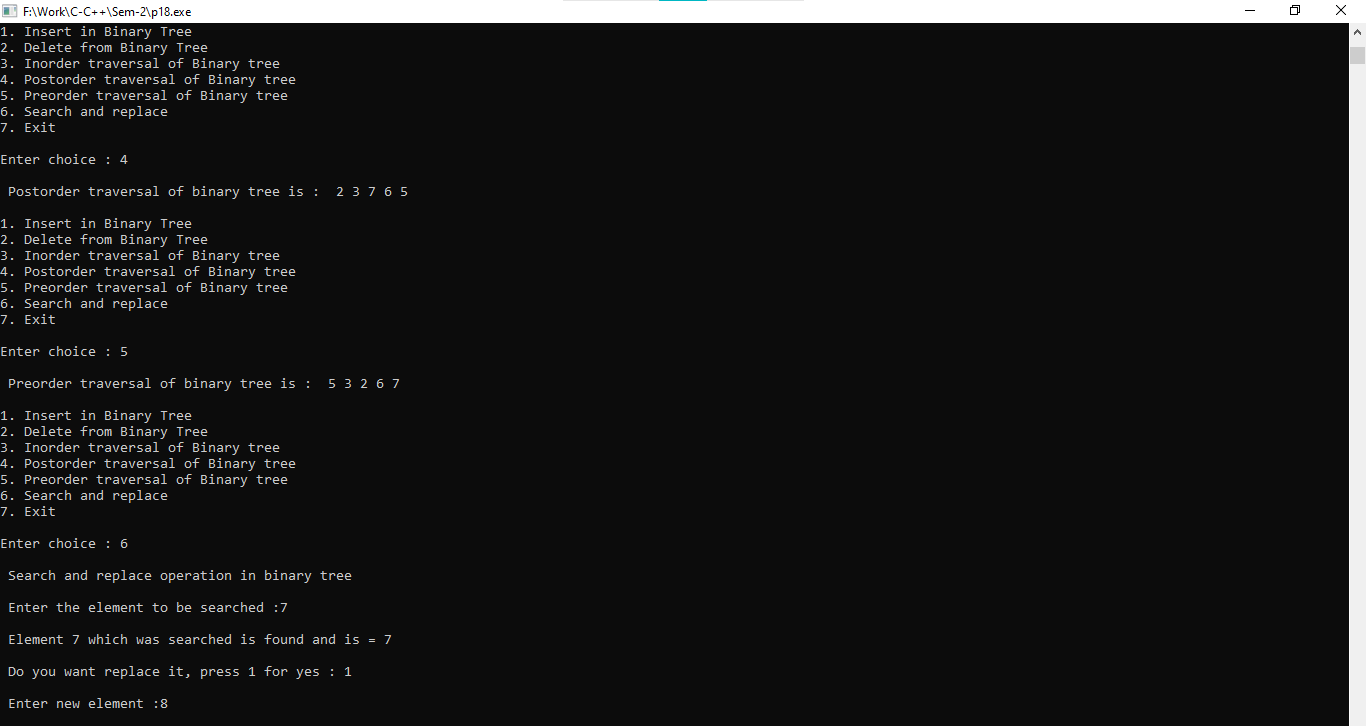
return(root);

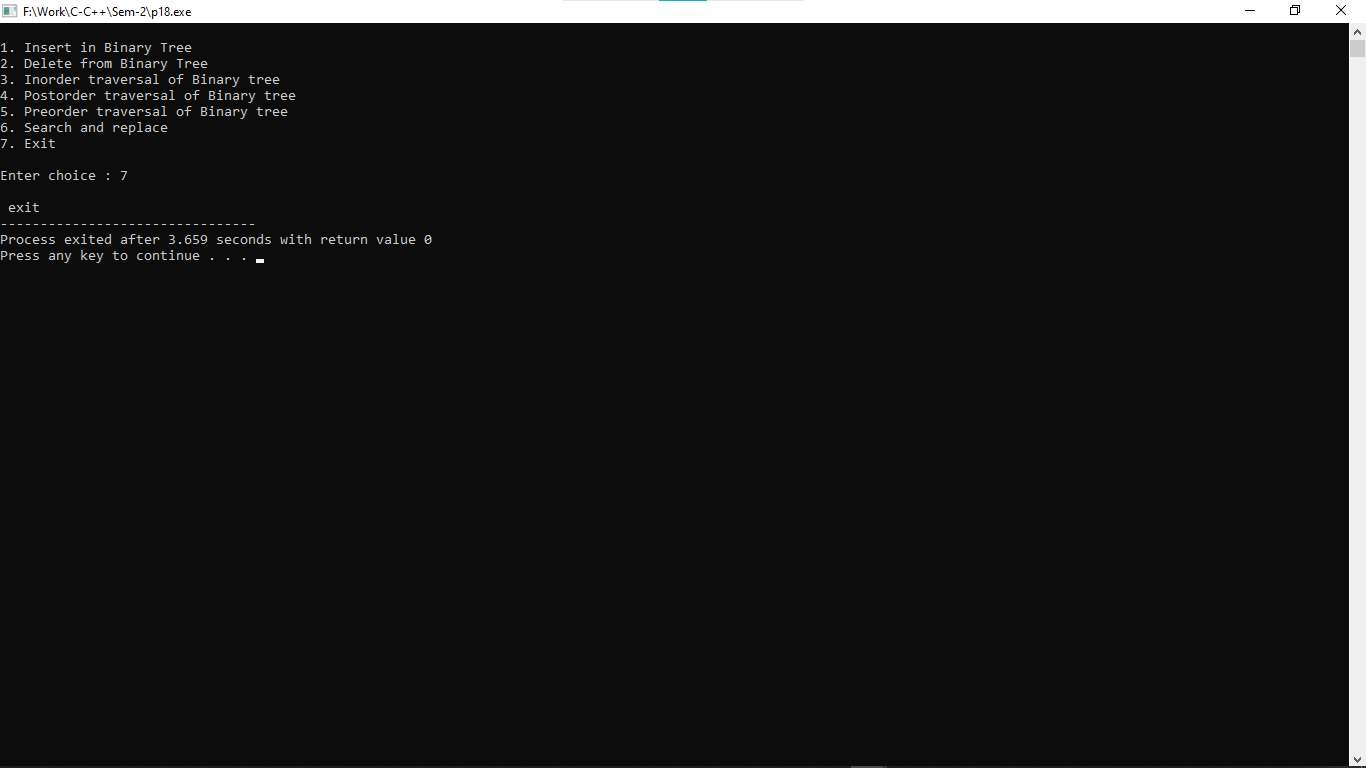
}

**Output:-**



****

****

****