1.MATRIX MULTIPLICATON

**PROGRAM**

include<stdio.h>

void main()

{

int m,n,p,q,c,d,k,sum=0;

int first[10][10],second[10][10],multiply[10][10];

printf("enter the number of rows and cloumns of first\n");

scanf("%d%d",&m,&n);

printf("enter the elements of first matrix \n");

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

{

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

{

scanf("%d",&first[c][d]);

}

}

printf("enter the number of rows and columns of second matrix\n");

scanf("%d%d",&p,&q);

if(n!=p)

{

printf("matrices can't be multipled with each other\n");

}

else

{

printf("enter the elements of second matrix\n");

for(c=0;c<p;c++)

{

for(d=0;d<q;d++)

{

scanf("%d",&second[c][d]);

}

}

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

{

for(d=0;d<q;d++)

{

for(k=0;k<p;k++)

{

sum=sum+first[c][k]\*second[k][d];

}

multiply[c][d]=sum;

sum=0;

}

}

printf("product of matrices\n");

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

{

for(d=0;d<q;d++)

{

printf("%d\t",multiply[c][d]);

printf("\n");

}

}

}

}

**OUTPUT**

Enter number of rows and columns of first matrix 2 2

Enter elements of first matrix 1 1 1 1

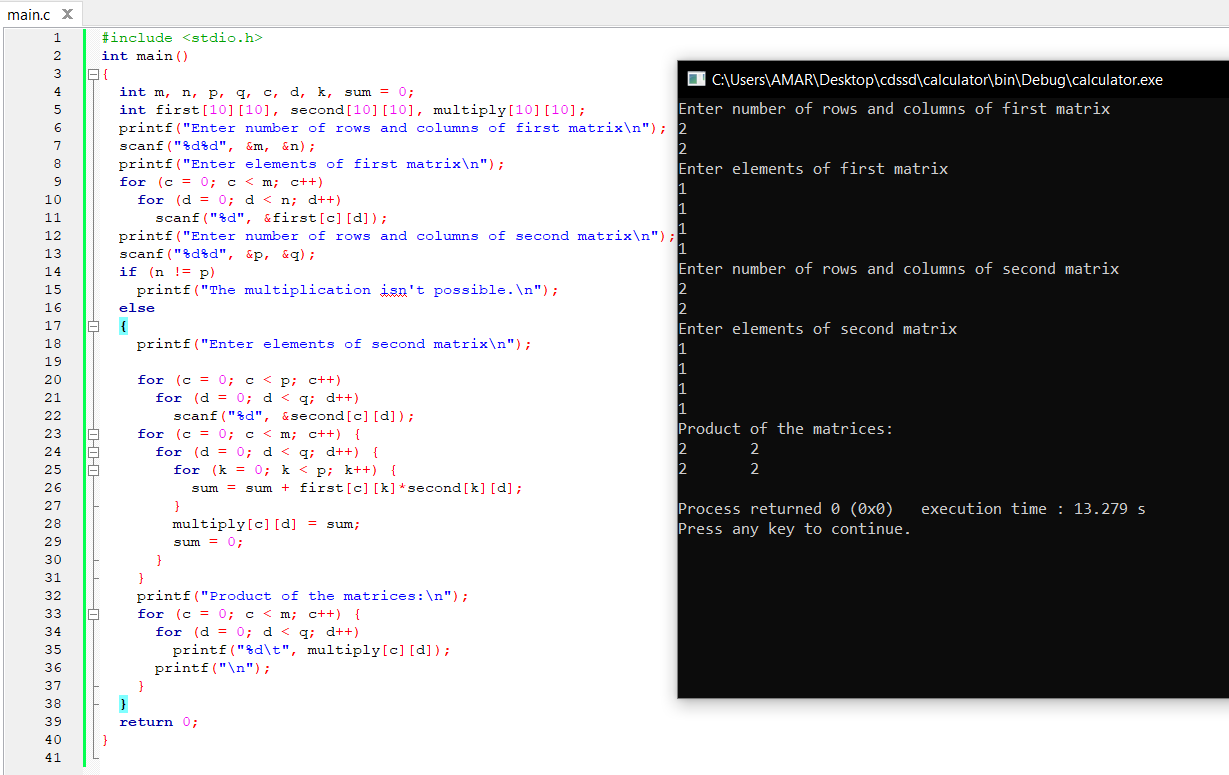
Enter number of rows and columns of second matrix 2 2

Enter elements of second matrix 1 1 1 1

Product of the matrix:

2 2

2 2



**2.ODD OR EVEN NUMBER FROM A GIVEN SET OF NUMBERS**

**PROGRAM**

#include<stdio.h>

Main()

{

Int num;

Printf(“enter the numbers:”);

Scanf(“%d”,&num);

If(num%2==0)

Printf(“Even number is %d “,num);

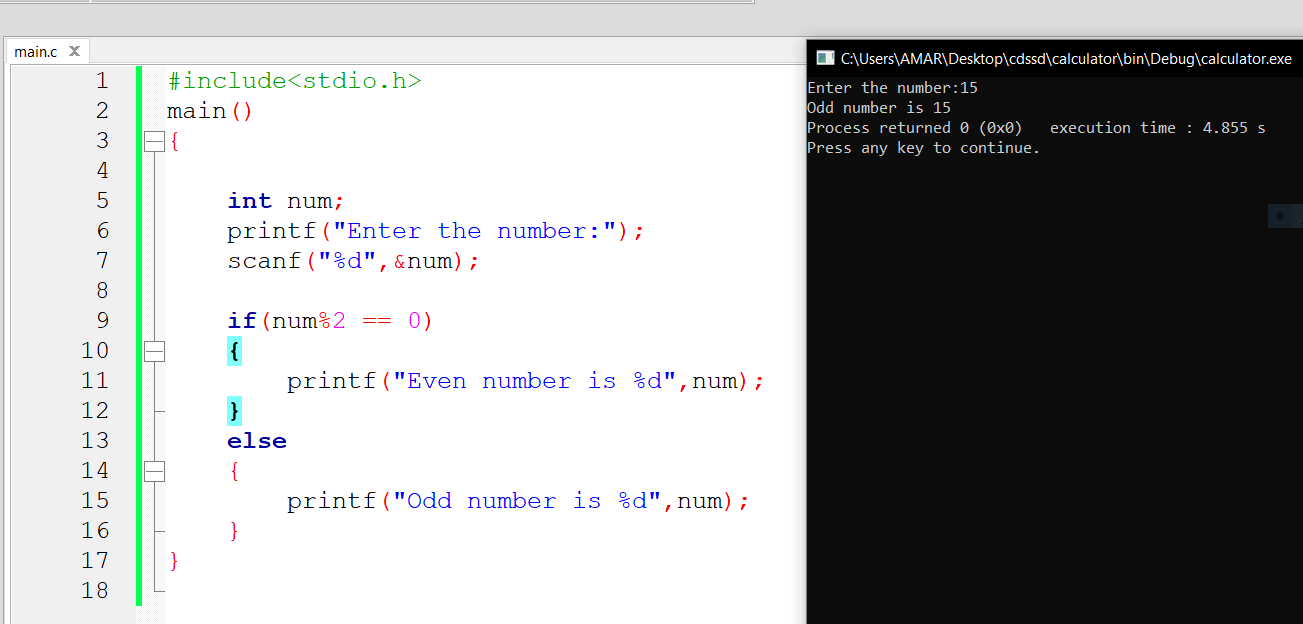
Else

Printf(“Odd number is %d”,num);

}

**OUTPUT**

Enter the number: 15

Odd number is 15

**3.FACTORIAL OF A GIVEN NUMBER WITHOUT USING RECURSION**

**PROGRAM**

#include<stdio.h>

Void main

{

Int I,n,fact=1;

Printf(“Enter the value to find factorial”);

Scanf(“%d”,&n);

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

{

Fact\*=I;

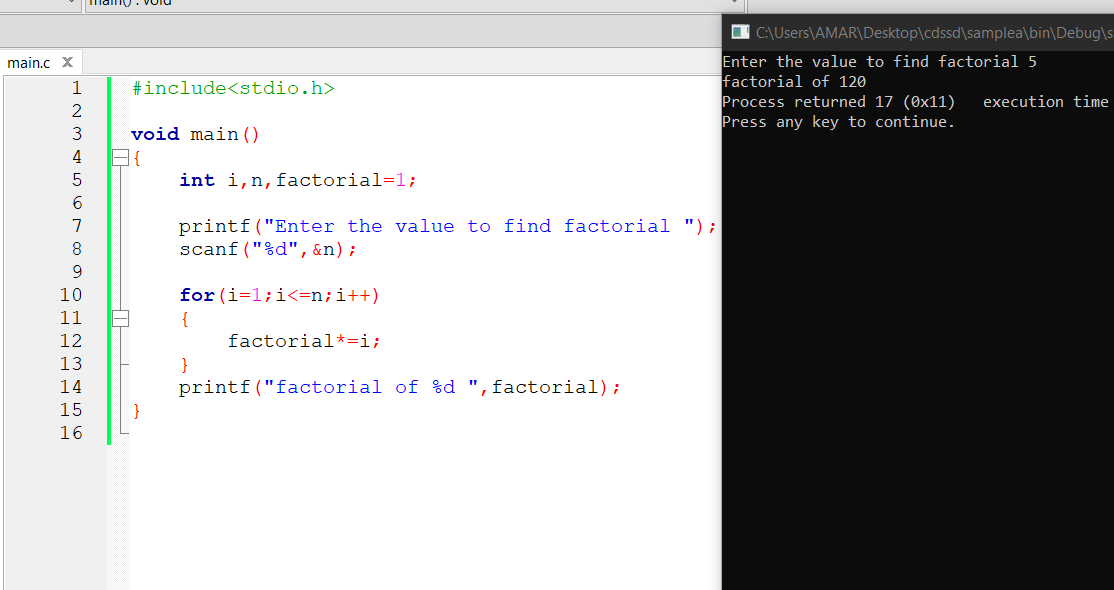
}

Printf(“factorial of %d is %d”,n,fact);

}

**OUTPUT**

Enter the value to find factorial 5

Factorial of 5 is 120 

**4.FIBONACCI SERIES WITHOUT USING RECCURSION**

**PROGRAM**

#include<stdio.h>

int main()

{

int n1=0,n2=1,n3,i,number;

printf("Enter the number of elements:");

scanf("%d",&number);

printf("\n%d %d",n1,n2);

for(i=2;i<number;++i)

{

n3=n1+n2;

printf(" %d",n3);

n1=n2;

n2=n3;

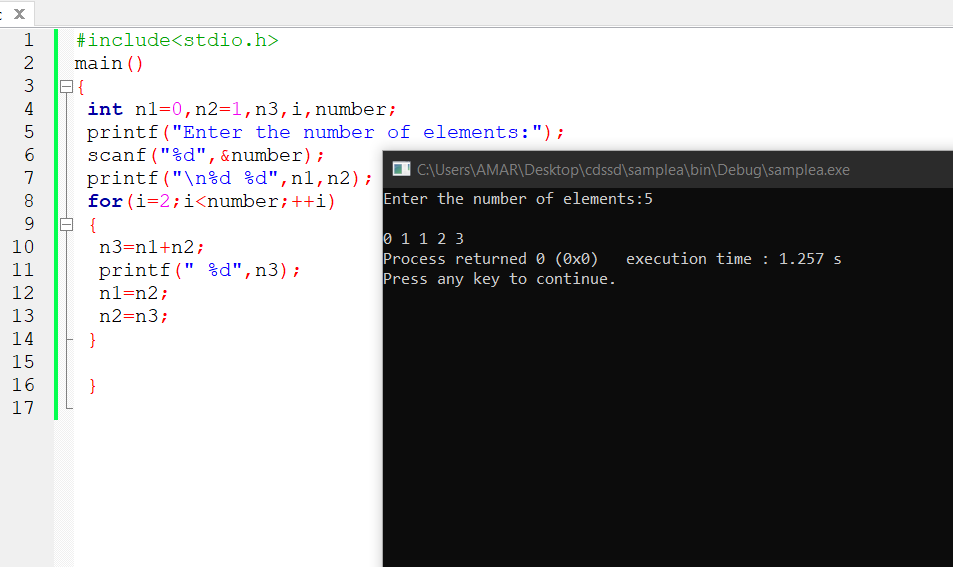
}

return 0;

}

**OUTPUT**

Enter the number of elements:5

0 1 1 2 3

**5.FACTORIAL OF A GIVEN NUMBER USING RECCURSION**

**PROGRAM**

#include<stdio.h>

Main()

{

Int n,res;

Printf(“Enter n value”);

Scanf(“%d”,&n);

Res=fact(n);

Printf(“Result %d “,res);

}

Int fact (int n)

{

Int res;

If(n==0)

Res=1;

Else

Res=n\*fact(n-1);

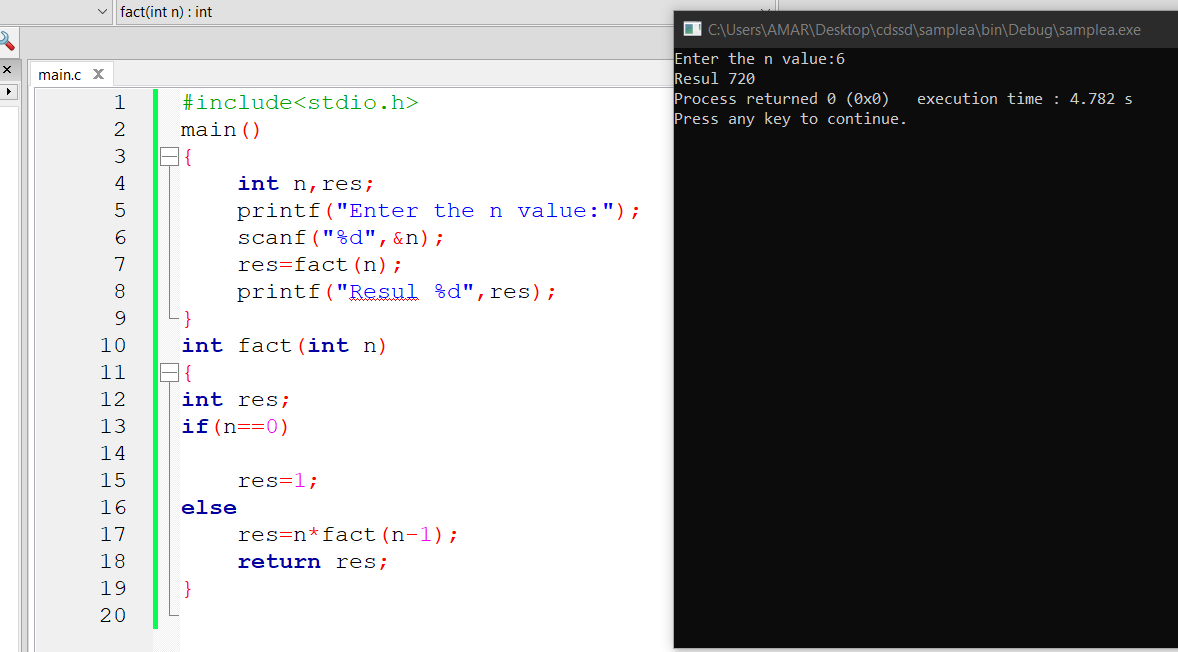
Return res;

}

**OUTPUT**

Enter the n value : 6

Result 720



**6.FIBONACCI SERIES USING RECCURSION**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int fibonacci(int);

int main(){

int n, i;

printf("Enter the number of element you want in series :\n");

scanf("%d",&n);

printf("fibonacci series is : \n");

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

printf("%d ",fibonacci(i));

}

}

int fibonacci(int i){

if(i==0) return 0;

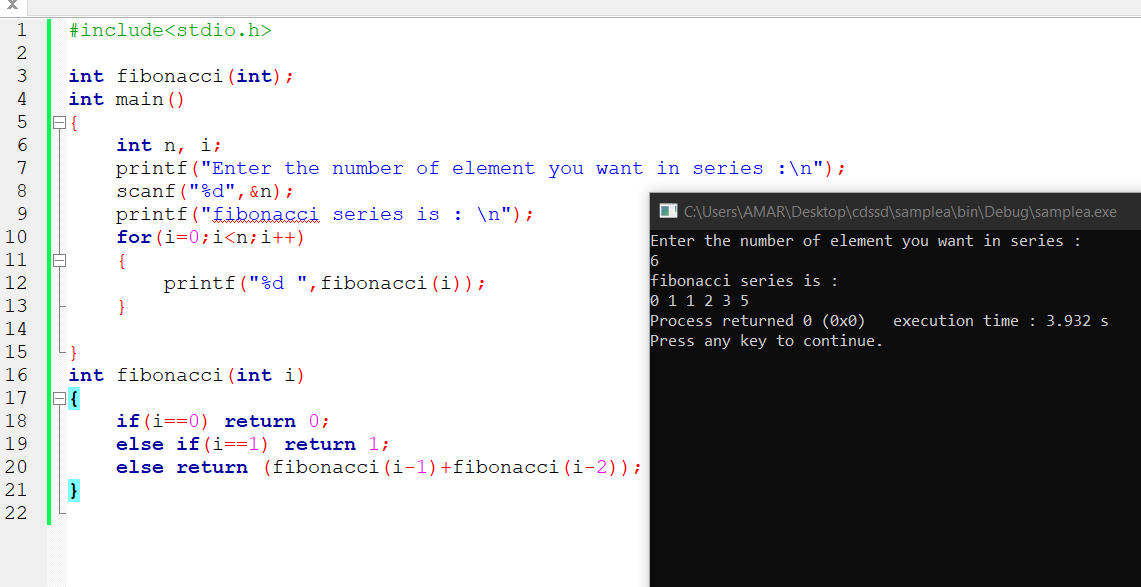
else if(i==1) return 1;

else return (fibonacci(i-1)+fibonacci(i-2));

}

**OUTPUT**

Enter the number of elements you want in series : 6

Fibonacci series is : 0 1 1 2 3 5  


**7. IMPLEMENT ARRAY OPERATIONS**

**PROGRAM**

#include<stdio.h>

void display();

void insert();

void delet();

int a[10],n,i;

void main()

{

printf("Enter the value of n ");

scanf("%d",&n);

display();

insert();

delet();

}

void display()

{

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

{

printf("element %d:",i);

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

}

printf("The elements are\n");

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

{

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

}

}

void insert()

{

int pos,data;

printf("Enter the data you want to insert ");

scanf("%d",&data);

printf("Enter the position ");

scanf("%d",&pos);

for(i=n;i>=pos-1;i--)

{

a[i+1]=a[i];

}

a[pos]=data;

n++;

printf("The elements after inserting\n");

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

{

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

}

}

void delet()

{

int pos;

printf("Enter the position of the data which you want to delete: ");

scanf("%d",&pos);

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

{

a[i]=a[i+1];

}

n--;

printf("The elements after deleting\n");

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

{

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

}

}

**OUTPUT**

Enter the value of n 3

Element 0:1

Element 1:2

Element 2:3

The elements are

Element 0=1

Element 1=2

Element 2=3

Enter the data you want to insert 4

Enter the position 3

The element after inserting

Element 0=1

Element 1=2

Element 2=3

Element 3-4

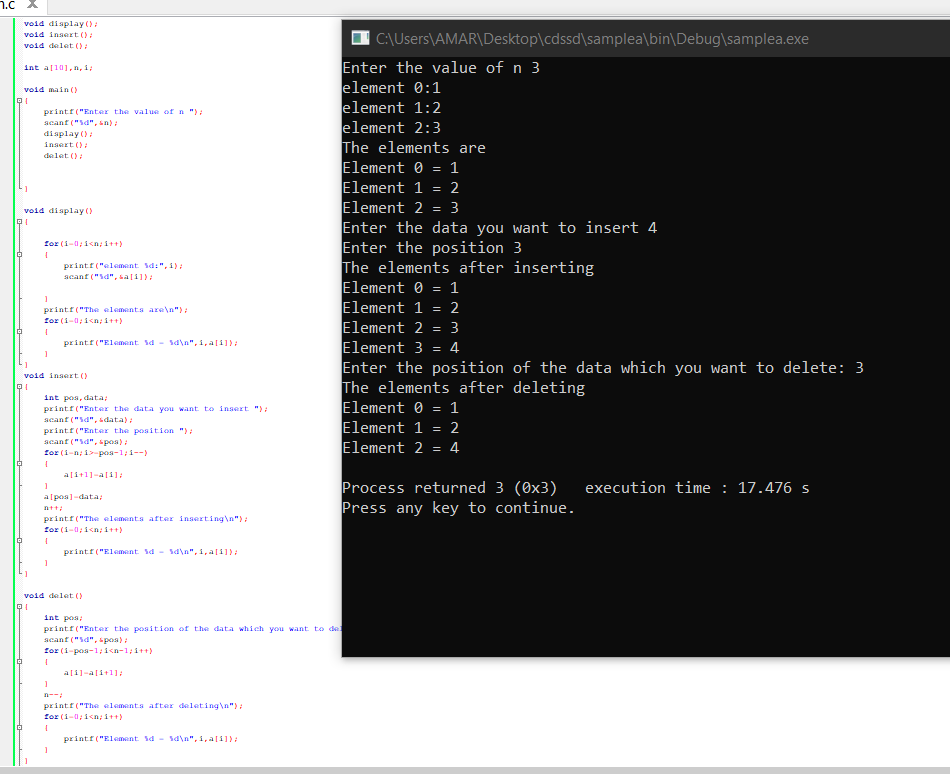
Enter the position of the data which you want to delete 3

The elements after deleting

Element 0=1

Element 1=2

Element 2=4



**8.IMPLEMENT LINKED LIST OPERATIONS**

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void insertAtBeginning(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void insertAfter(struct Node\* prev\_node, int new\_data) {

if (prev\_node == NULL) {

printf("the given previous node cannot be NULL");

return;}

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = prev\_node->next;

prev\_node->next = new\_node;

}

void insertAtEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* last = \*head\_ref;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;}

while (last->next != NULL) last = last->next;

last->next = new\_node;

return;}

void deleteNode(struct Node\*\* head\_ref, int key) {

struct Node \*temp = \*head\_ref, \*prev;

if (temp != NULL && temp->data == key) {

\*head\_ref = temp->next;

free(temp);

return;}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

int searchNode(struct Node\*\* head\_ref, int key) {

struct Node\* current = \*head\_ref;

while (current != NULL) {

if (current->data == key) return 1;

current = current->next;

}

return 0;}

void sortLinkedList(struct Node\*\* head\_ref) {

struct Node \*current = \*head\_ref, \*index = NULL;

int temp;

if (head\_ref == NULL) {

return;

} else {

while (current != NULL) {

index = current->next;

while (index != NULL) {

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

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

}

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtBeginning(&head, 2);

insertAtBeginning(&head, 3);

insertAtEnd(&head, 4);

insertAfter(head->next, 5);

printf("Linked list after inserting an elements:");

printList(head);

printf("\nAfter deleting an element: ");

deleteNode(&head, 3);

printList(head);

sortLinkedList(&head);

printf("\nSorted List: ");

printList(head);

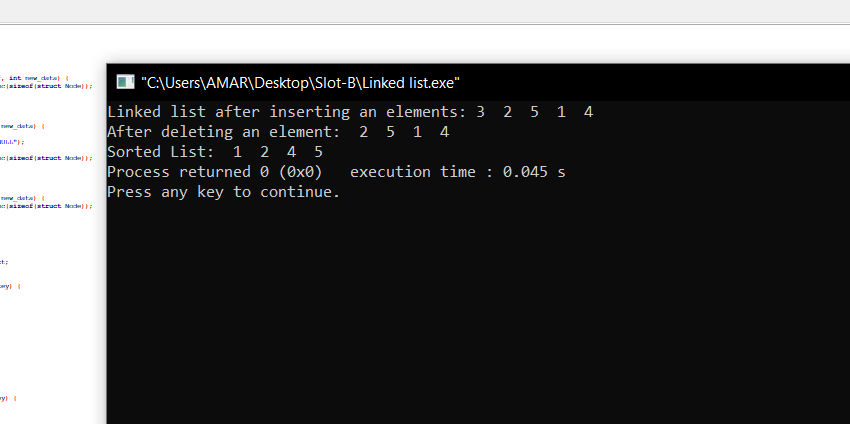
}

**OUTPUT**

Linked list after inserting an elements : 3 2 5 1 4

After deleting an element : 2 5 1 4

Sorted list: 1 2 4 5



**9.LINEAR SEARCH**

**PROGRAM**

#include<stdio.h>

main()

{

int num;

int search,i,find=0;

printf("Enter the number of elements:");

scanf("%d",&num);

int array[num];

printf("Enter the elements\n");

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

{

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

}

printf("Enter the element to search");

scanf("%d",&search);

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

{

if(search==array[i]);

{

printf("%d is present at location %d\n",search,i+1);

break;

}

}

if(i==num)

printf("Not found");

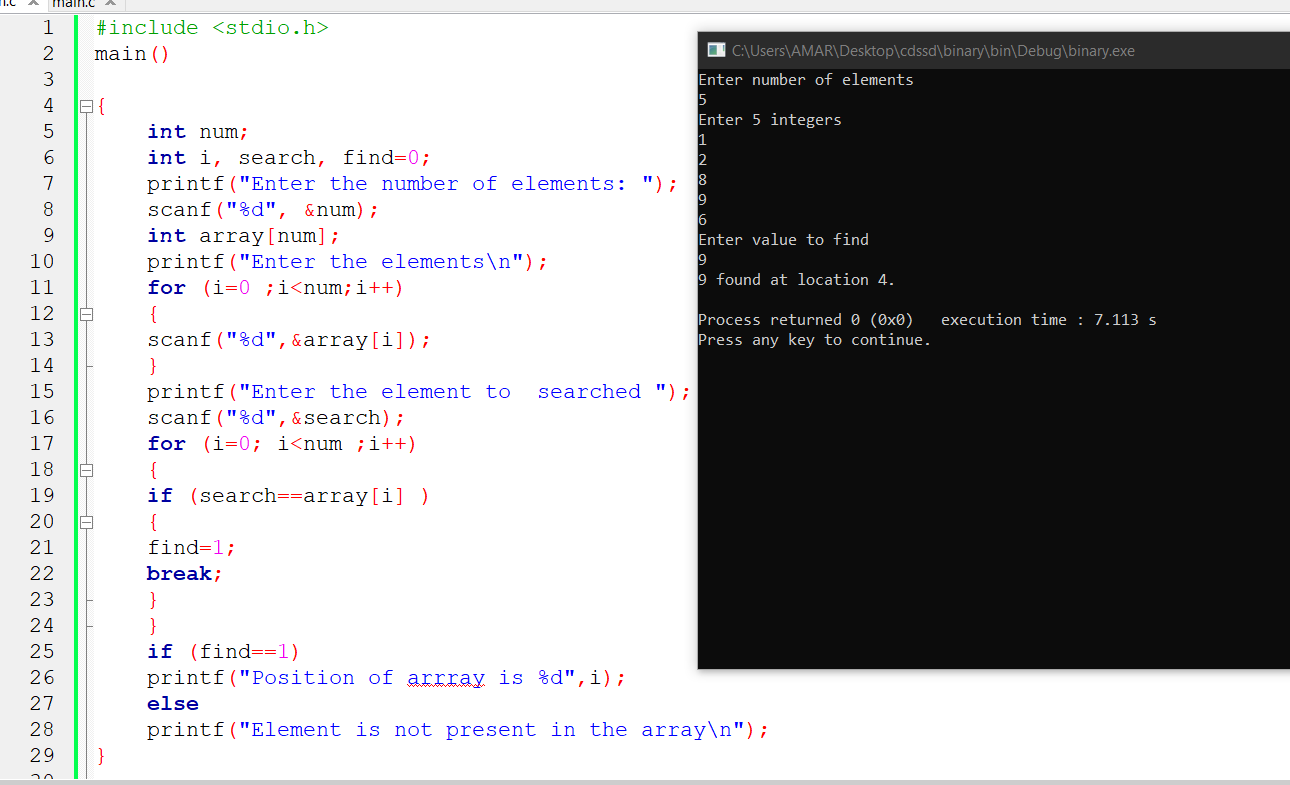
}

OUTPUT  
Enter number of elements 5

Enter 5 integers 1 2 8 9 6

Enter value to find 9

9 found at location 4



**10.BINARY SEARCH**

**PROGRAM**

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

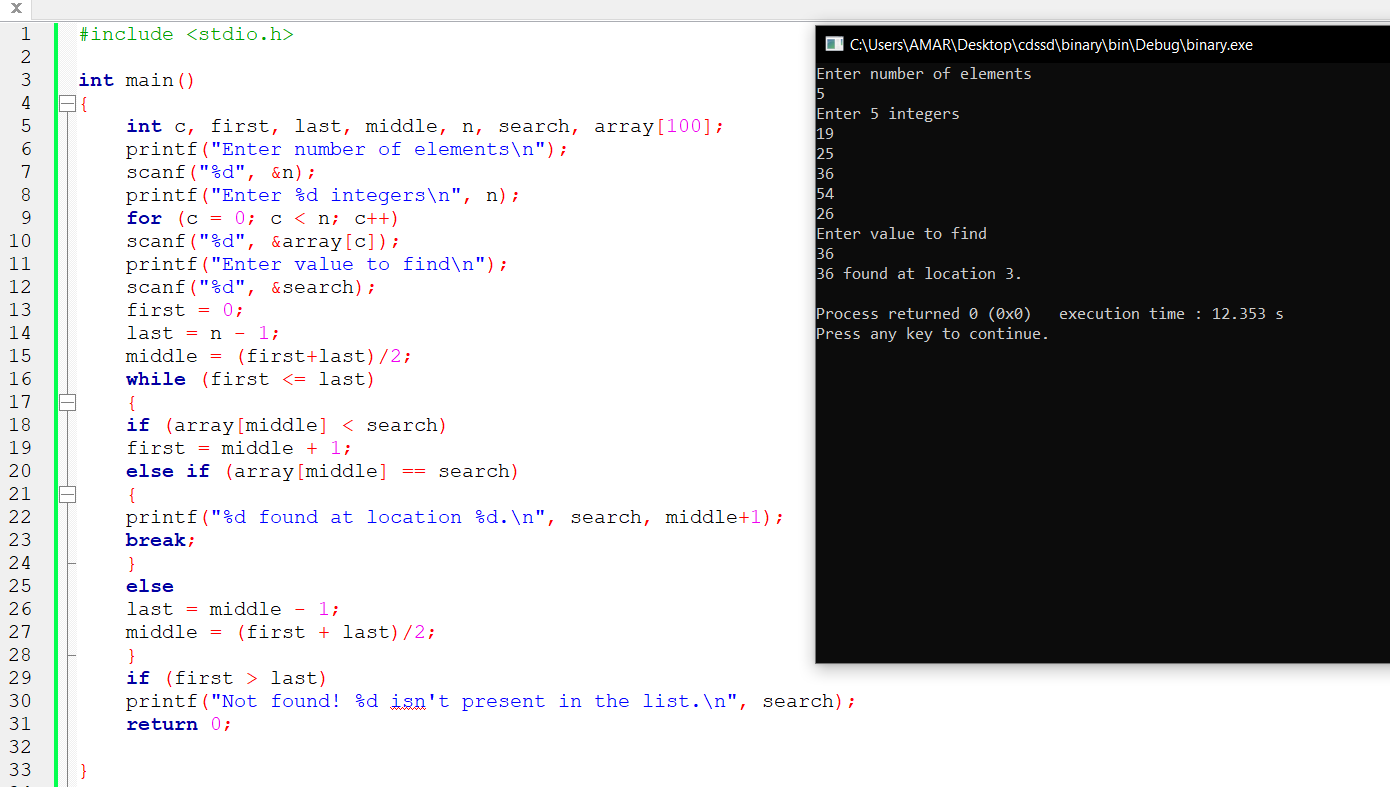
**OUTPUT**

Enter number of elements 5

Enter 5 integers 19 25 36 54 26

Enter value to find 36

36 found at location 3



**11.IMPLEMENTATION OF STACK OPERATIONS**

**PROGRAM**

#include<stdio.h>

#define n 5

int stack[n];

int top=-1;

void push()

{

int x;

printf(“Enter data “);

scanf(“%d”,&x);

if(top==n-1)

{

printf(“overflow”);

}

else

{

top++;

stack[top]=x;

}

}

void pop()

{

int item;

if(top==-1)

{

printf(“underflow”);

}

else

{

item=stack[top];

top--;

printf(“%d poped item\n”,item);

}

}

void peek()

{

if(top==-1)

{

printf(“no data”);

}

else

{

printf(“%d is at top\n”,stack[top]);

}

}

void display()

{

int I;

printf(“Items in the stack are – “);

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

{

printf(“%d “,stack[i]);

}

}

main()

{

char ch;

do

{

printf(“Enter your choice\n1.push\n2.pop\n3.peek\n4.display\n”);

scanf(“%d”,&ch);

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

display();

break;

default:

printf(“invalid”);

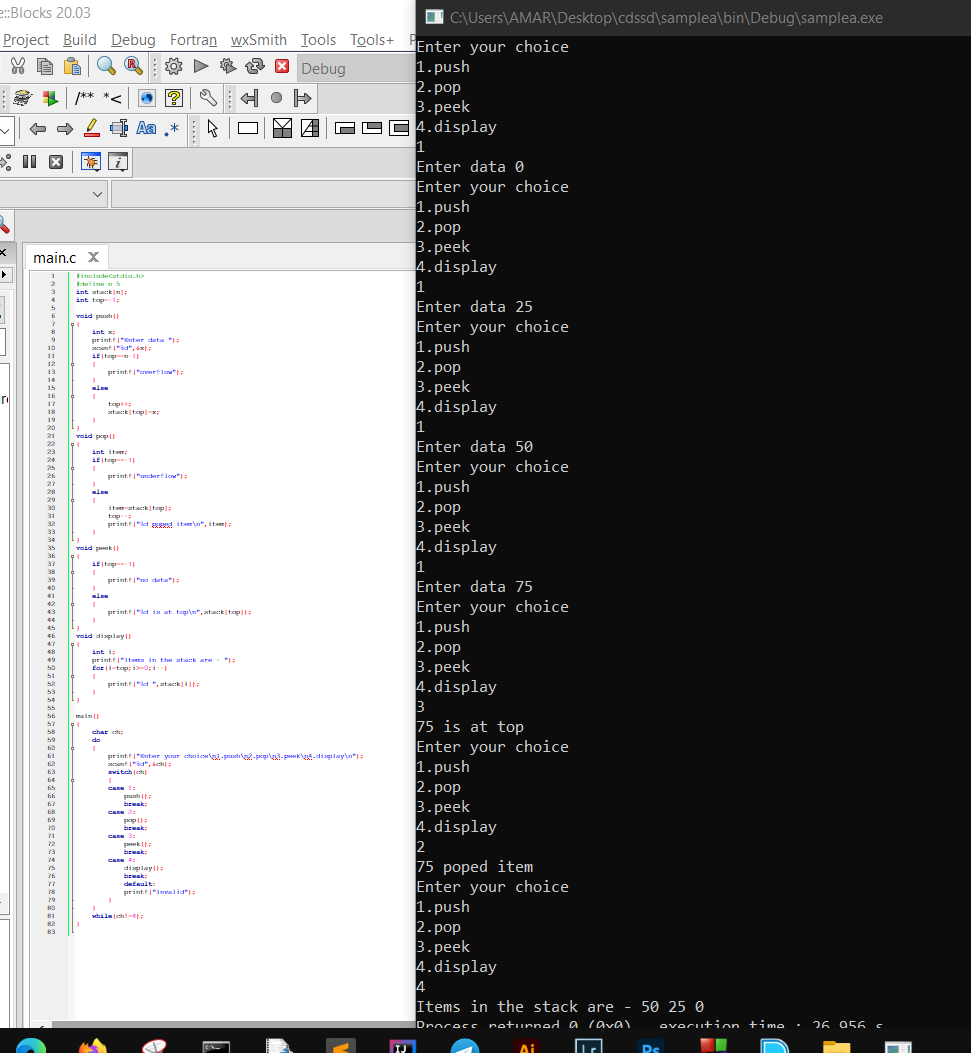
}

}

while(ch!=4);

}

**OUTPUT**



**12.IMPLEMENTATION OF QUEUE OPERATIONS**

**PROGRAM**

#include<stdio.h>

#define n 5

int queue[n];

int front=0;

int rear=0;

void enqueue()

{

if(n==rear)

{

printf("queue is full");

}

else

{

int x;

printf("Enter the data ");

scanf("%d",&x);

queue[rear]=x;

rear++;

}

}

void dequeue()

{

if(front==rear)

{

printf("queue is empty");

}

else

{

printf("deleted element is %d",queue[front]);

}

}

void display()

{

if(front==rear)

{

printf("empty");

}

else

{

int i;

printf("\nData is:");

for(i=front+1;i<rear;i++)

{

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

}

}

}

main()

{

printf("Enter the data into the queue\n");

enqueue();

enqueue();

enqueue();

dequeue();

display();

printf("In the queue");

}

**OUTPUT**

Enter the data into the queue

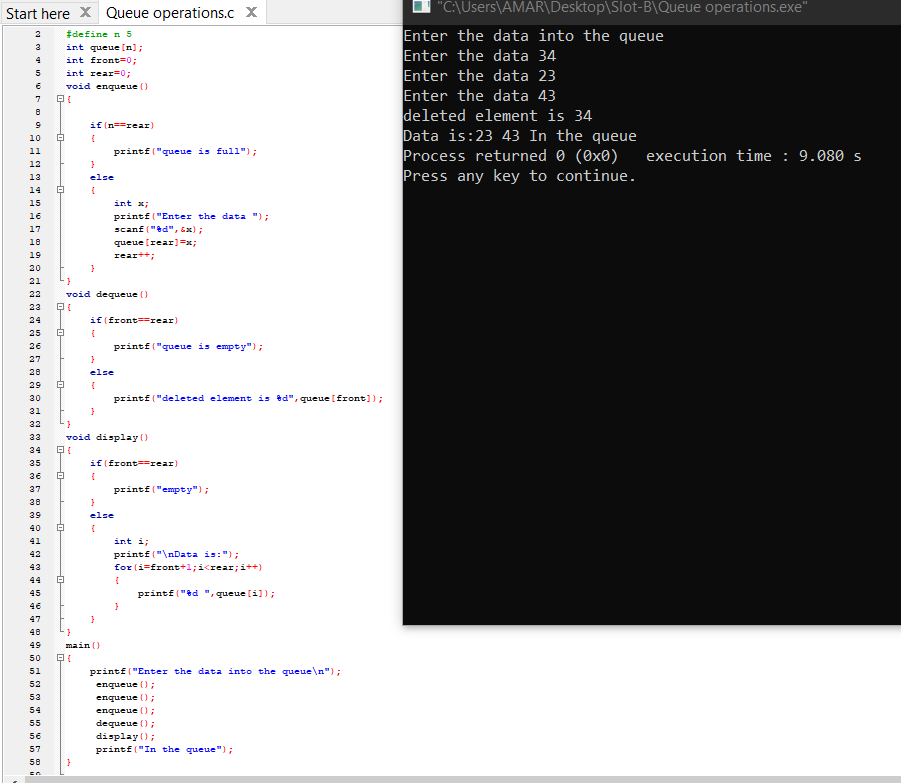
Enter the data 34

Enter the data 23

Enter the data 43

Deleted element is 34

Data is : 23 43 in the queue



**13.TREE TRAVERSAL**

**PROGRAM**

#include <stdio.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

struct node\* newNode(int data)

{

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

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void printPostorder(struct node\* node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

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

}

void printInorder(struct node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

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

printInorder(node->right);

}

void printPreorder(struct node\* node)

{

if (node == NULL)

return;

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

printPreorder(node->left);

printPreorder(node->right);

}

void main()

{

struct node\* root = newNode(5);

root->left = newNode(3);

root->right = newNode(7);

root->right->left = newNode(6);

root->right->right = newNode(8);

root->left->left = newNode(2);

root->left->right = newNode(4);

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

printPreorder(root);

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

printInorder(root);

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

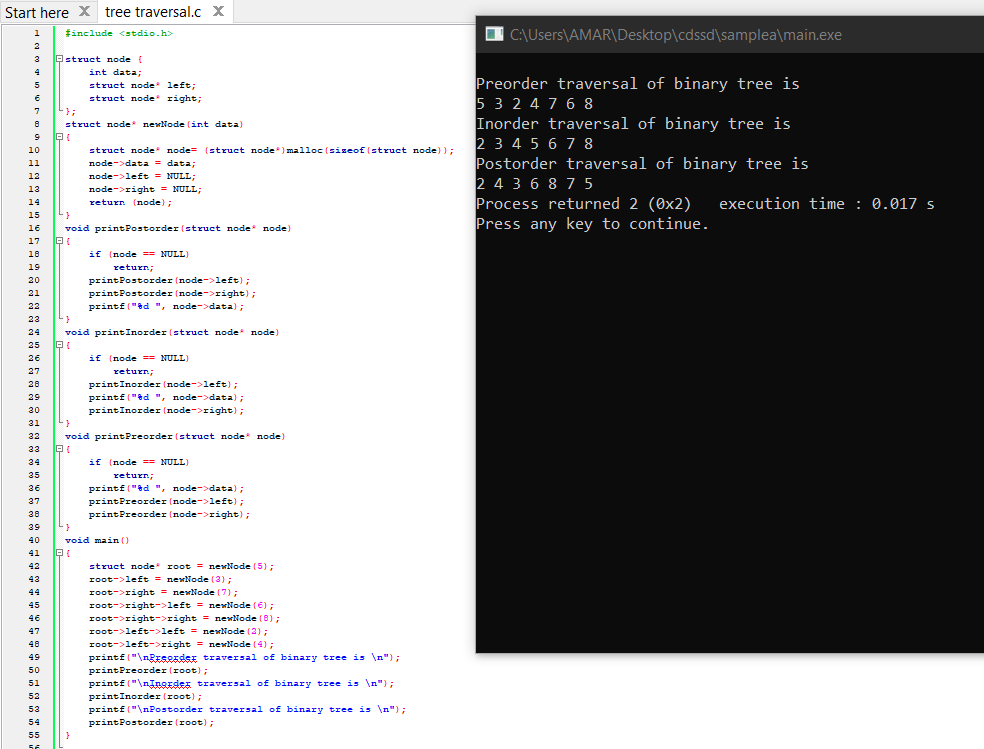
printPostorder(root);

}

**OUTPUT**

Preorder traversal of binary tree is 5 3 2 4 7 6 8

Inorder traversal of binary tree is 2 3 4 5 6 7 8

Postorder traversal of binary tree is 2 4 3 6 8 7 5

**14.HASH TABLE COLLISION RESOLUTION**

**PROGRAM**

#include<stdio.h>

struct pair

{

int value;

int key;

};

void display(struct pair[],int size);

void main()

{

int size,i,temp;

printf("Enter the size of table ");

scanf("%d",&size);

struct pair hash\_table[size];

printf("Enter the elements\n");

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

{

scanf("%d",&temp);

hash\_table[temp%size].value=temp;

hash\_table[temp%size].key=temp%size;

}

printf("\n");

display(hash\_table,size);

int a=0;

}

void display(struct pair ar[],int size)

{

int i;

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

{

printf("%d\t%d\n",ar[i].value,ar[i].key);

}

}

**OUTPUT**

Enter the size of table 10

Enter the elements 5 8 4 7 9 6 2 11 33 50

50 0

11 1

2 2

33 3

4 4

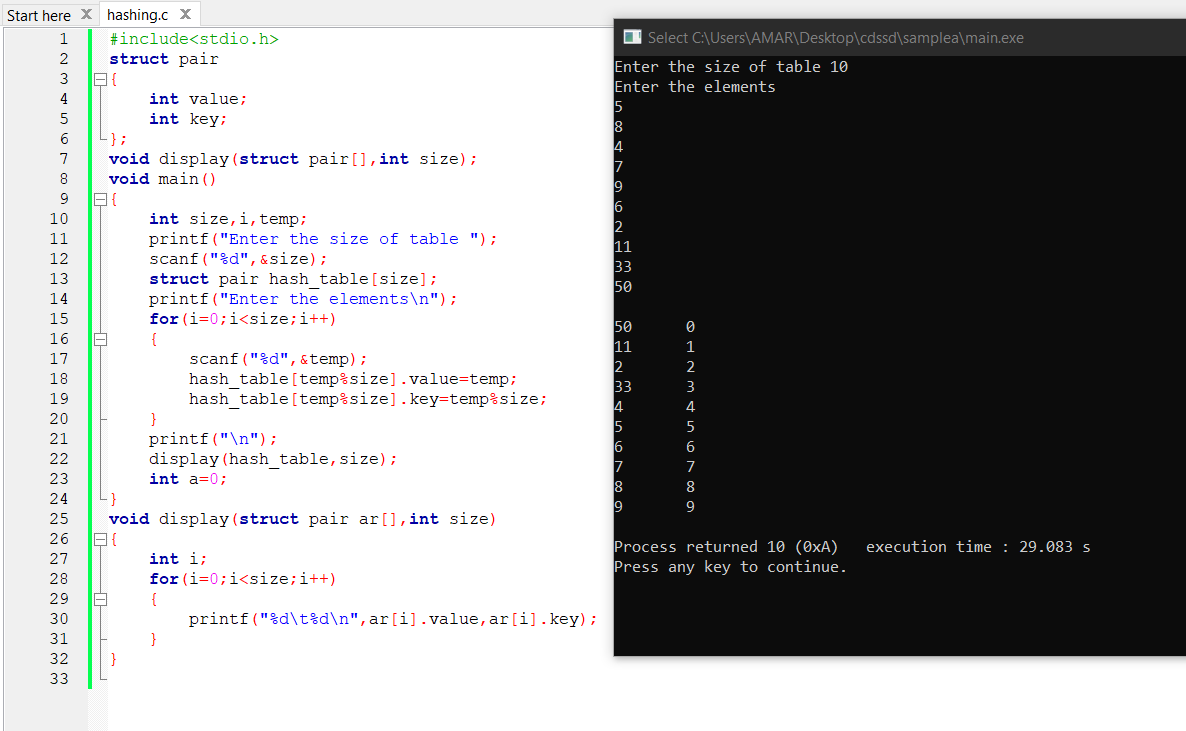
5 5

6 6

7 7

8 8

9 9



**15.INSRTION SORT**

**PROGRAM**

#include<stdio.h>

main()

{

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

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

scanf("%d",&n);

printf("Enter the element in array\n");

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

{

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

}

printf("Before sorting\n");

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

{

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

}

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

{

temp=a[i];

j=i-1;

while(j>=0 && a[j]>temp)

{

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

j=j-1;

}

a[j+1]=temp;

}

printf("\nAfter sorting\n");

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

{

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

}

}

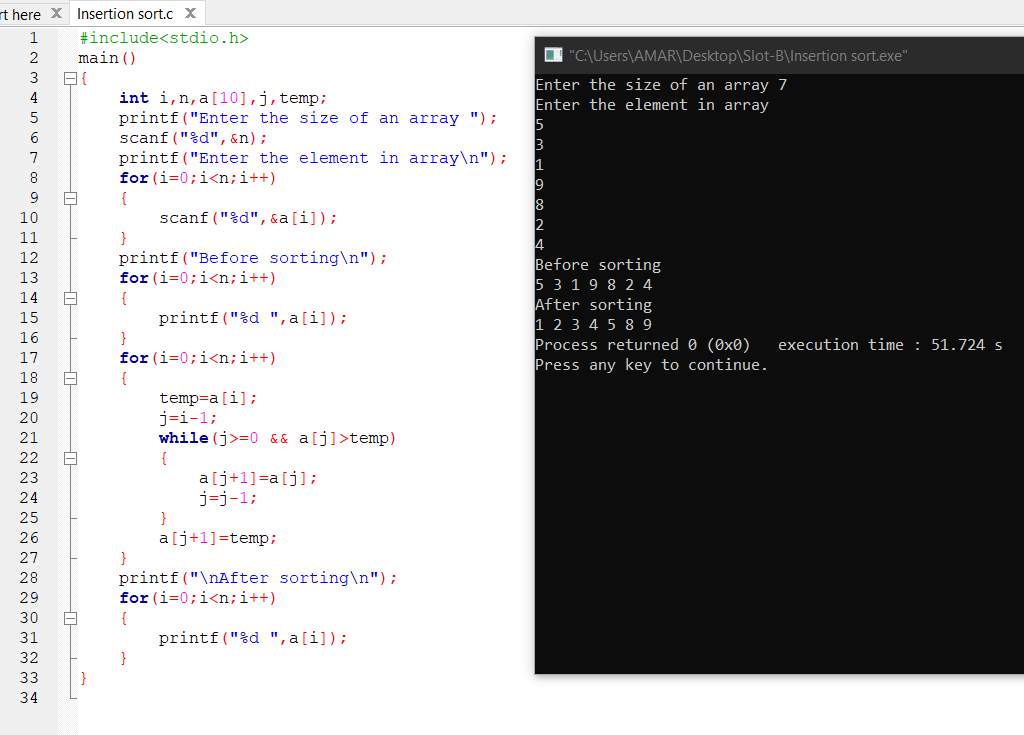
**OUTPUT**

Enter the size of an array 7

Enter the element in array 5 3 1 9 8 2 4

Before soting 5 3 1 9 8 2 4

After sorting 1 2 3 4 5 8 9



**16.QUICK SORT**

**PROGRAM**

#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

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

i++;

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

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[25];

printf("Enter Elements ");

scanf("%d",&count);

printf("Enter %d elements\n", count);

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

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

quicksort(number,0,count-1);

printf("The Sorted Order is: ");

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

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

return 0;

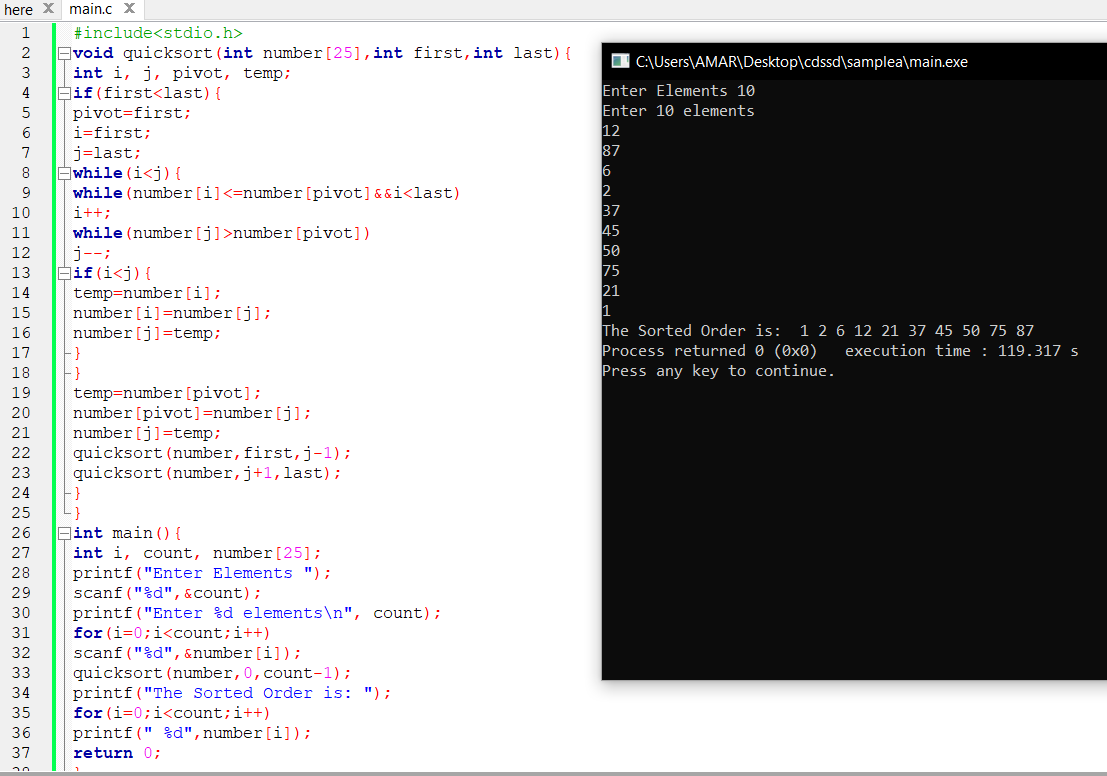
}

**OUTPUT**

Enter elements 10

Enter 10 elements 12 87 6 2 37 45 50 75 21 1

Sorted order is 1 2 6 12 21 37 45 50 75 87



**17.GRAPH TRAVERSAL**

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

typedef struct graph{

int nVertices;

int nEdges;

int \*\*adjMatrix;

}Graph;

typedef struct QueueNode{

int data;

struct QueueNode \*next;

}queueNode;

queueNode \*head = NULL;

void printGraph(Graph \*);

Graph \* createAdjMatrixOfGraph(Graph \*);

void enqueue( int item){

queueNode \*list = head;

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

temp->data = item;

temp->next = NULL;

if(list)

temp->next = list;

head = temp;

}

int dequeue(){

int itemDequeued;

queueNode \*list = head;

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

if(!list){

return INT\_MIN;

}

else if(!list->next){

itemDequeued = list->data;

head = NULL;

free(list);

return itemDequeued;

}

else{

while(list->next){

temp = list;

list = list->next;

}

}

itemDequeued = list->data;

temp->next = NULL;

free(list);

return itemDequeued;

}

Graph \* createAdjMatrixOfGraph(Graph \*g){

int i, u, v;

g = (Graph \*)malloc(sizeof(Graph));

printf("Enter the number of Vertices and Edges:");

scanf("%d%d", &g->nVertices, &g->nEdges);

g->adjMatrix = (int \*\*)malloc(sizeof(int \*) \* g->nVertices);

for(i = 0; i < g->nVertices; i++)

g->adjMatrix[i] = (int \*)malloc(sizeof(int) \* g->nVertices);

for(i = 0; i < g->nEdges; i++){

printf("Reading the endpoints of an edge(u v):\n");

scanf("%d %d", &u, &v);

g->adjMatrix[u][v] = 1;

g->adjMatrix[v][u] = 1;

}

return g;

}

void printGraph(Graph \*g){

int i, j;

printf("\n\*\*\*Adjacency Matrix of Graph\*\*\*\n");

for(i = 0 ; i < g->nVertices; i++ ){

printf("\n\t");

for(j = 0 ; j < g->nVertices; j++ ){

if( g->adjMatrix[i][j] ){

printf("1 ");

}

else

printf("0 ");

}

}

}

void breadthFirstTraversal(Graph \*g, int u, int \*visited){

int i, j;

int vertexDequeued;

visited[u] = 1;

enqueue(u);

for(j = 0; j < g->nVertices; j++){

vertexDequeued = dequeue();

if(vertexDequeued == INT\_MIN)

return;

printf(" %d", vertexDequeued);

for(i = 0; i < g->nVertices; i++){

if( (g->adjMatrix[vertexDequeued][i]) && ( !(visited[i]) ) ){

visited[i] = 1;

enqueue(i);

}

}

}

}

int main(void){

Graph \*g = NULL;

int \*visited;

g = createAdjMatrixOfGraph(g);

printGraph(g);

visited = (int \*)malloc(g->nVertices\*sizeof(int));

printf("\nBreadth First Traversal of the graph: ");

breadthFirstTraversal(g, 2, visited);

return 0;

}

