**1a. Implementation of stack using arrays:**

Source\_code:

#include <stdio.h>

#include<stdlib.h>

#define MAX 9

int arr\_stack[MAX];

int top = -1;

void push(int ele){

if(top == MAX - 1){

printf("\n stack is full !"); //Overflow

}

else {

top++;

arr\_stack[top] = ele;

}

}

int pop(){

if(top == -1){

printf("\n stack is empty"); //Underflow

exit(1);

}

else{

int item;

item = arr\_stack[top];

top--;

return item;

}

}

void display(){

if(top == -1){

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

}

else{

printf("\n Elements in stack are: \n");

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

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

}

}

}

int main() //Implementation of stack using arrays

{

int ele, choice;

while(1)

{

printf("\n 1. Push\n 2. Pop \n 3. display \n 4. exit \n");

printf("\n enter your choice \n ");

scanf("%d",&choice);

if(choice == 1)

{

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

scanf("\n %d", &ele);

push(ele); //push the element

}

else if(choice == 2)

{

ele = pop(); //pop the element

printf("\n The popped element is %d \n",ele);

}

else if (choice == 3)

{

display(); //display the stack

}

else if(choice == 4)

{

exit(1);

}

else

{

printf("\n Check your choice again \n");

continue;

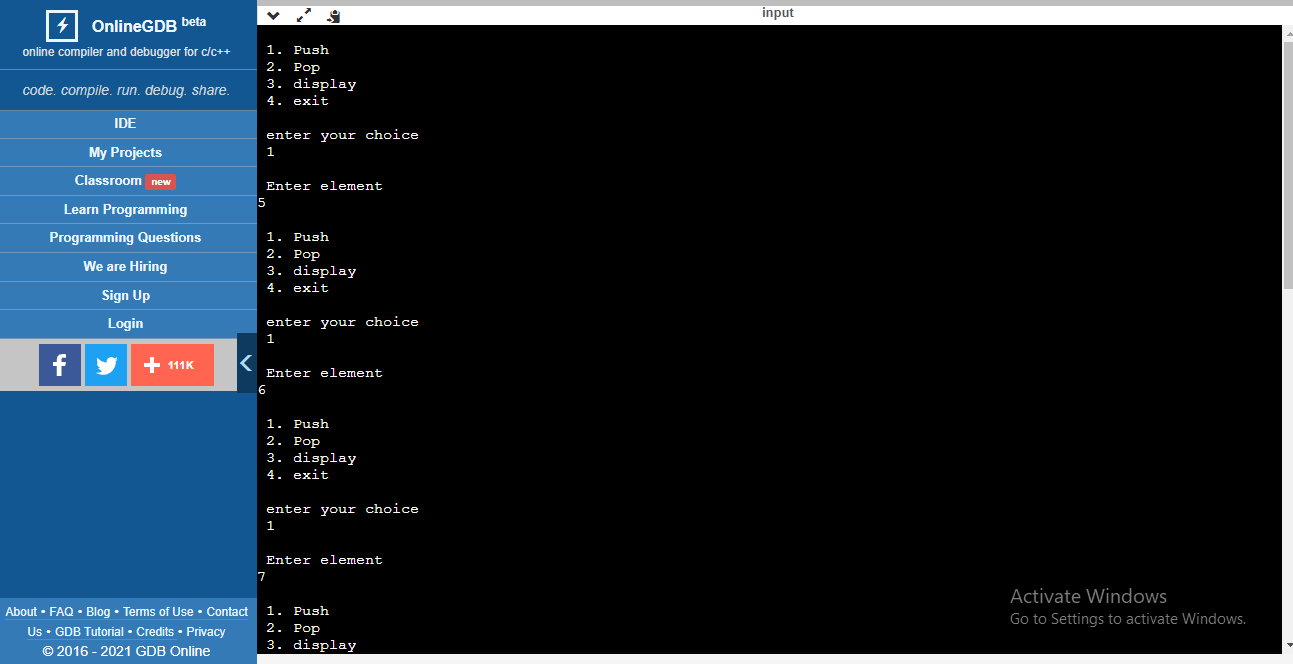
}

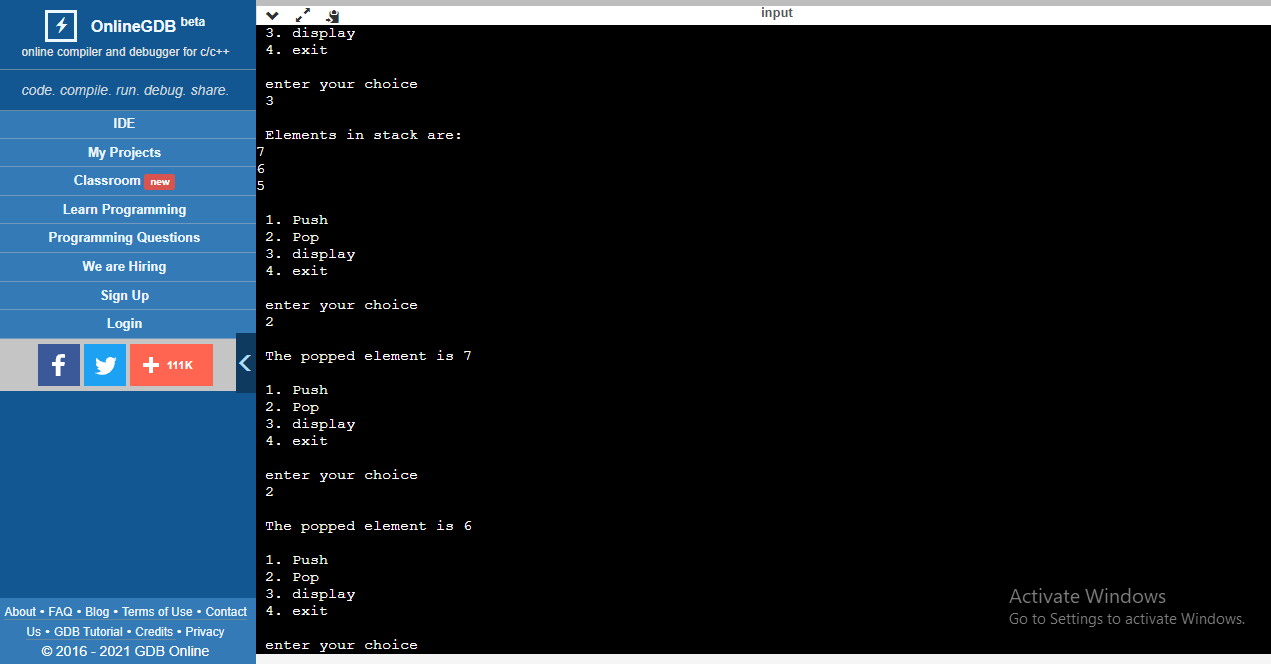
}

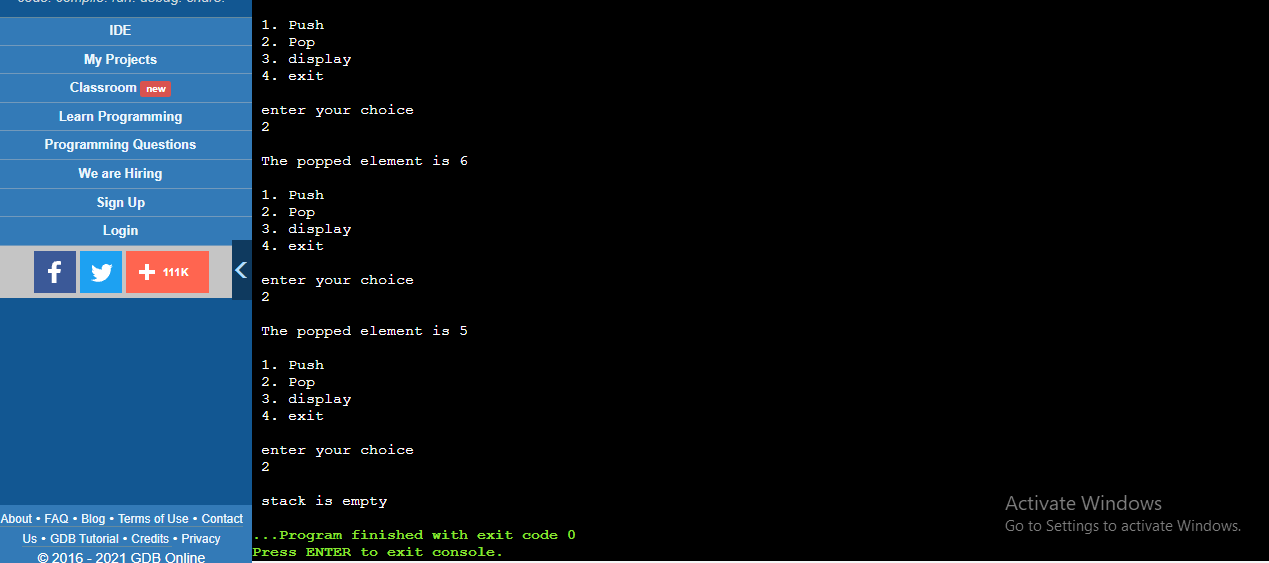
return 0;

}

**Output**:







**1b. Implementation of Stack using Linked List:**

Source\_code:

#include <stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*link;

} \*top = NULL;

void push(int ele){

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

tmp -> data = ele;

tmp -> link = top;

top = tmp;

}

int pop(){

struct node \*tmp;

int ele;

if(top == NULL){

printf("\n stack is empty. ");

exit(1);

}

tmp = top;

ele = tmp -> data;

top = top -> link;

free(tmp);

return ele;

}

void display(){

printf("\n Elements in list are: \n");

if(top == NULL){

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

exit(1);

}

struct node \*p = top;

while(p->link){

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

p=p->link;

}

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

}

int main() //Implementation of stack using LinkedLists

{

int ele, choice;

printf("\n Implementation using Linked List \n");

while(1)

{

printf(" \n 1. Push\n 2. Pop \n 3. display \n 4. exit \n");

printf("\n enter your choice \n ");

scanf("%d",&choice);

if(choice == 1)

{

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

scanf("\n %d", &ele);

push(ele); //push the element

}

else if(choice == 2)

{

ele = pop(); //pop the element

printf("\n The popped element is %d \n",ele);

}

else if (choice == 3)

{

display(); //display the stack

}

else if(choice == 4)

{

exit(1);

}

else

{

printf("\n Check your choice again \n");

continue;

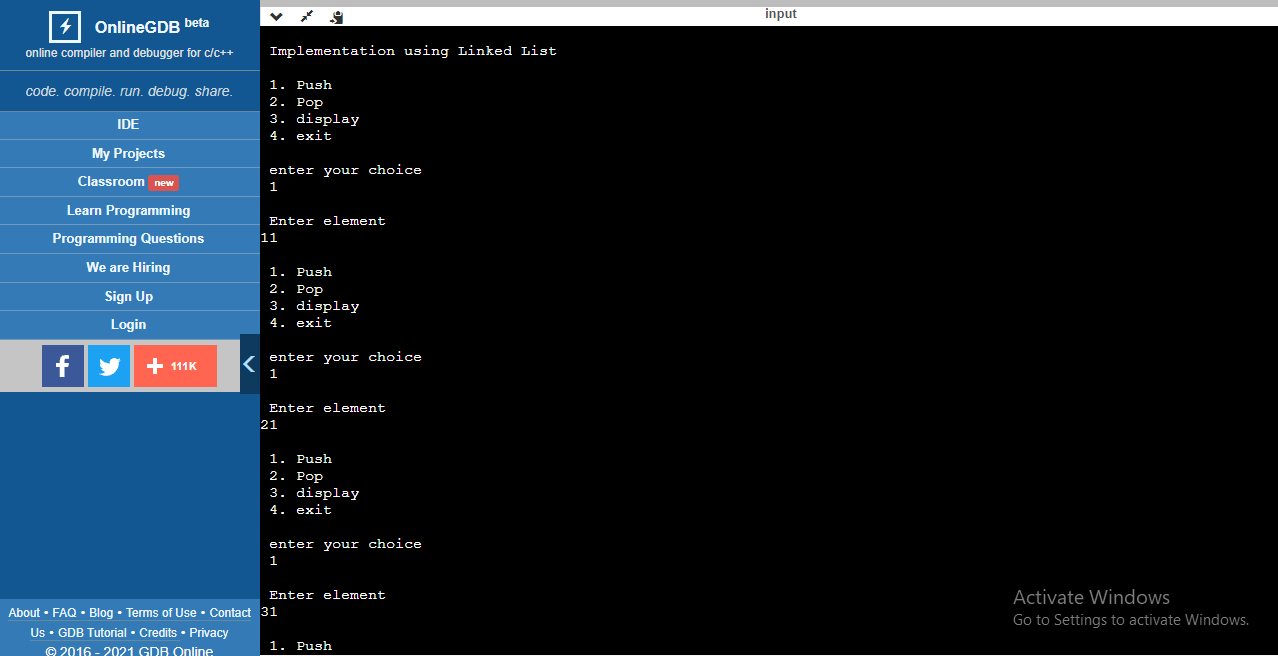
}

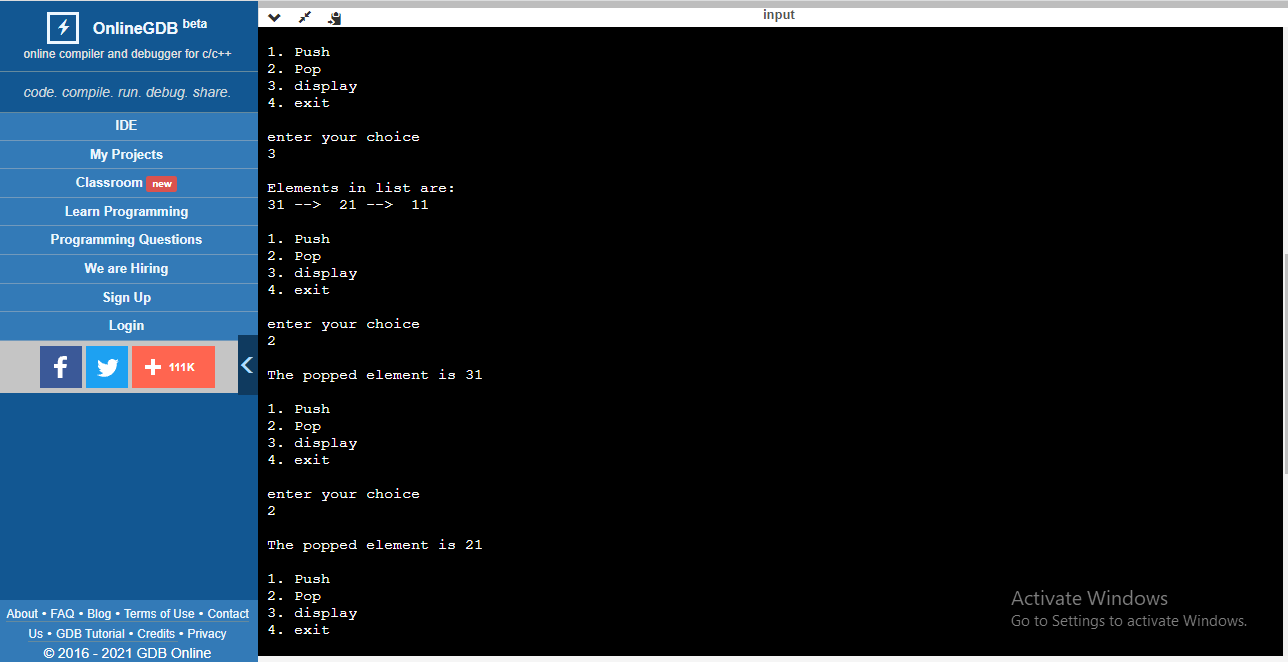
}

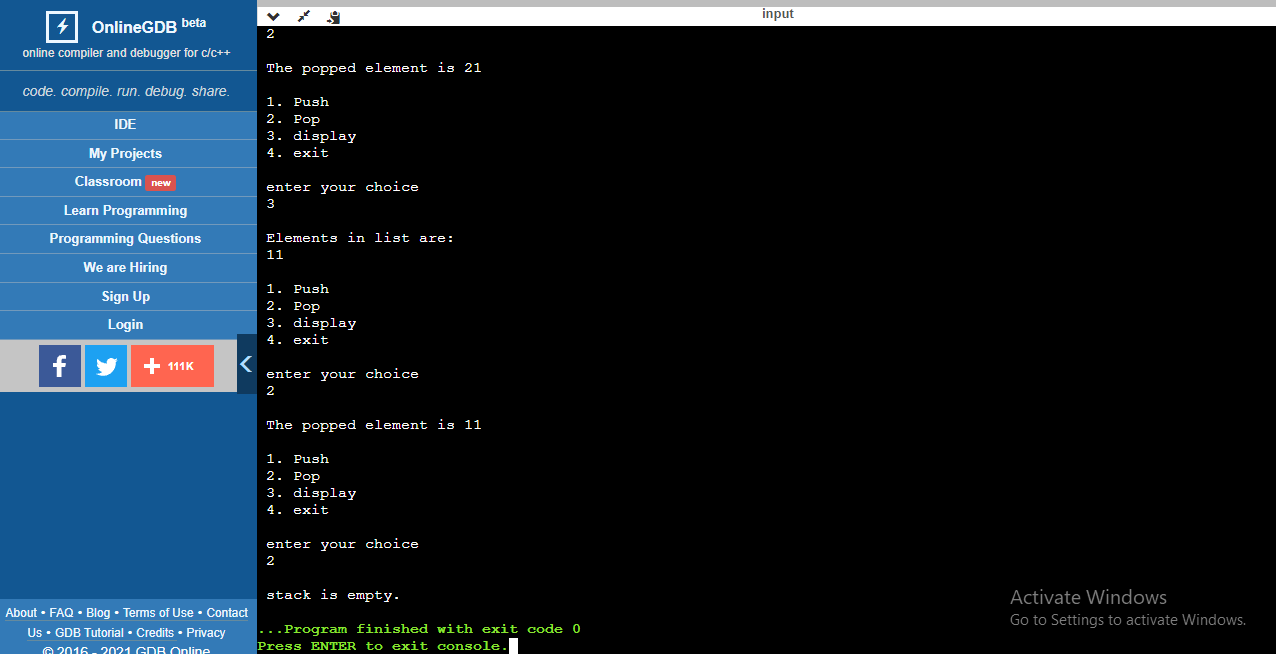
return 0;

}

**Output:**

****

****

****

**2a. Implementation of Queues using Arrays**:

Source\_code:

#include <stdio.h>

#include<stdlib.h>

#define MAX 9

int front=-1;

int rear =-1;

int q\_arr[MAX];

void enque(int ele)

{

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

printf("\n Que Oveflow");

return;

}

if(front == -1){

front = 0;

}

if(rear == MAX-1){

rear = 0;

}

else

rear++;

q\_arr[rear] = ele;

}

int deque()

{

int ele;

if (front ==-1){

printf("\n Que Underflow \n");

exit(1);

}

ele = q\_arr[front];

if(front==rear) // que has single element

{

front=-1;

rear=-1;

}

else if(front == MAX-1) //front reached end of array,so initialize it to start.

front=0;

else

front=front+1;

return ele;

}

void display(){

int i;

if(rear == -1){

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

}

else{

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

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

}

}

}

int main() //Implementation of que using array

{

int ele, choice;

while(1)

{

printf("\n 1. Enqueue\n 2. Dequeue \n 3. Display \n 4. Exit \n");

printf("enter your choice ");

scanf("%d",&choice);

if(choice ==1 ){

printf("\n Enter element ");

scanf("%d", &ele);

enque(ele); //insert the element

}

else if(choice == 2)

{

ele = deque(); //delete the element

printf("\n The deleted element is %d ",ele);

}

else if(choice == 3)

{

display();

}

else if(choice ==4)

{

exit(1);

}

else

{

printf("\n Check your choice again");

}

}

return 0;

}

**Output**:





**2b. Implementation of Queue using LinkedList:**

Source\_code:

#include <stdio.h>

#include<stdlib.h>

#define MAX 9

struct node{

int data;

struct node \*link;

}\*front =NULL, \*rear =NULL;

void enque(int ele){

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

if(tmp == NULL){

printf("\n Memory not exists");

return;

}

tmp->data =ele;

tmp->link = NULL;

if(front == NULL){ //When inserting for first time

front = tmp;

}

else{

rear->link =tmp;

}

rear = tmp; //changing the rear position after insertion.

}

int deque(){

int ele;

struct node \* tmp;

if(front == NULL){

printf("\n Que Underflow");

exit(1);

}

tmp =front;

ele = tmp->data;

front = front->link;

free(tmp);

return ele;

}

void display(){

struct node \*p;

p=front;

if(front == NULL)

{

printf("\n Queue is empty");

return;

}

while(p->link!=NULL) //traverse over each node

{

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

p=p->link;

}

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

}

int main() //Implementation of que using linked list

{

int ele, choice;

printf("\n Queue using LinkedList \n");

while(1)

{

printf("\n 1. Enqueue\n 2. Dequeue \n 3. Display \n 4. Exit \n");

printf("enter your choice ");

scanf("%d",&choice);

if(choice ==1 ){

printf("\n Enter element ");

scanf("%d", &ele);

enque(ele); //insert the element

}

else if(choice == 2)

{

ele = deque(); //delete the element

printf("\n The deleted element is %d ",ele);

}

else if(choice == 3)

{

display();

}

else if(choice ==4)

{

exit(1);

}

else

{

printf("\n Check your choice again");

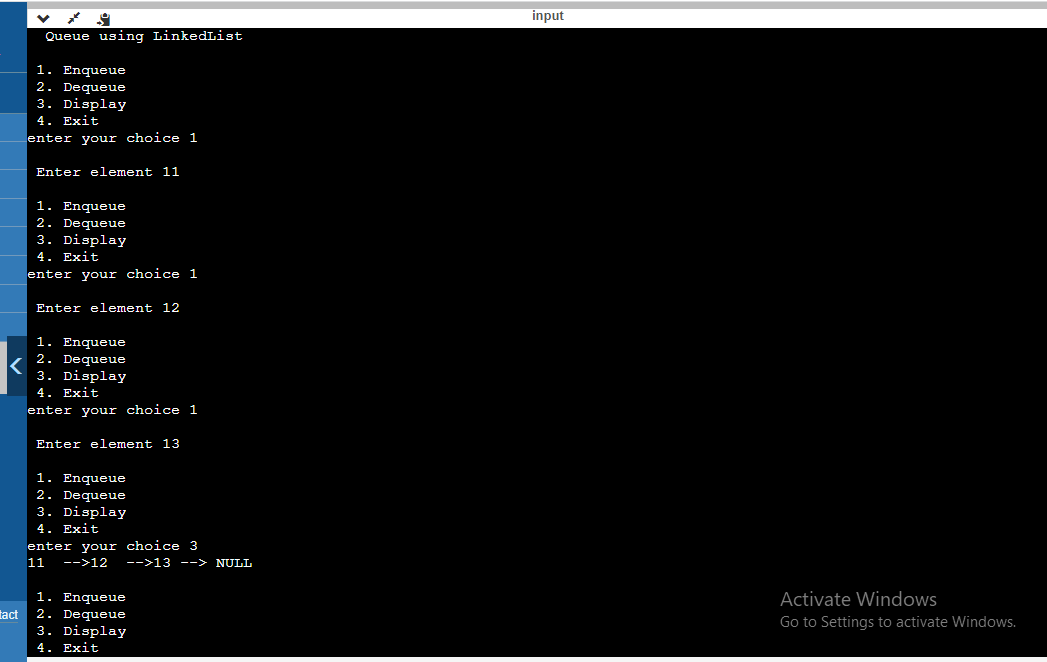
}

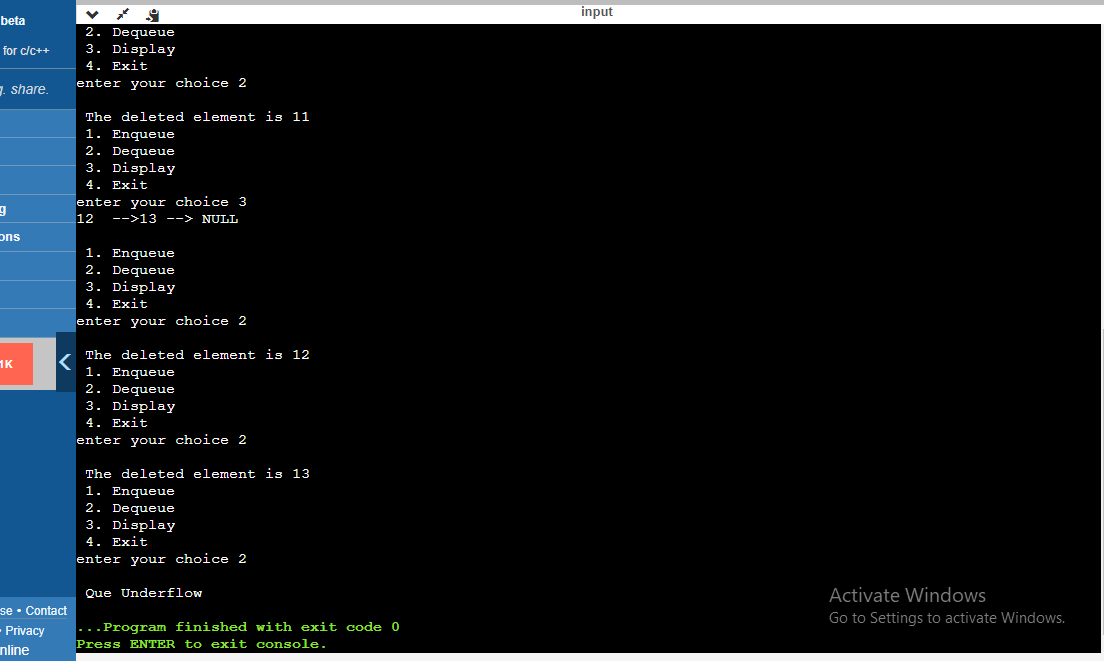
}

return 0;

}

Output:





**3. Write a program to implement Tower of Hanoi without recursion. Also Debug the code using gdb debugger.**

**Source\_code**:

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

// Structure for stack,represents pole

struct Stack

{

int size; //capacity

int top;

int \*arr;

};

// function for creating a stack of given size.

struct Stack\* Create\_stack(int size)

{

struct Stack\* stack =

(struct Stack\*) malloc(sizeof(struct Stack));

stack -> size = size;

stack -> top = -1;

stack -> arr = (int\*) malloc(stack -> size \* sizeof(int));

return stack;

}

int Check\_Full(struct Stack\* stack)

{

return (stack->top == stack->size - 1);

}

int Check\_Empty(struct Stack\* stack)

{

return (stack->top == -1);

}

void push(struct Stack \*stack, int item) //pushes the disk

{

if (Check\_Full(stack))

return;

stack -> arr[++stack -> top] = item;

}

int pop(struct Stack\* stack) //pops the disk

{

if (Check\_Empty(stack))

return -1;

return stack -> arr[stack -> top--];

}

//prints all the disk moves

void print\_DiskMove(char fromPole, char toPole, int disk)

{

printf("\n Move the disk %d from %c to %c \n",

disk, fromPole, toPole);

}

// Implementation of moves between two poles

void diskMovement(struct Stack \*src,

struct Stack \*dest, char s, char d)

{

int Top1\_Disk = pop(src);

int Top2\_Disk = pop(dest);

// When pole 1 is empty

if (Top1\_Disk == -1)

{

push(src, Top2\_Disk);

print\_DiskMove(d, s, Top2\_Disk);

}

// When pole2 pole is empty

else if (Top2\_Disk == -1)

{

push(dest, Top1\_Disk);

print\_DiskMove(s, d, Top1\_Disk);

}

// When top disk of pole1 > top disk of pole2

else if (Top1\_Disk > Top2\_Disk)

{

push(src, Top1\_Disk);

push(src, Top2\_Disk);

print\_DiskMove(d, s, Top2\_Disk);

}

// When top disk of pole1 < top disk of pole2

else

{

push(dest, Top2\_Disk);

push(dest, Top1\_Disk);

print\_DiskMove(s, d, Top1\_Disk);

}

}

//Function to implement TOH Iterative

void TowerOfHanoii(int disk\_count, struct Stack

\*src, struct Stack \*aux,

struct Stack \*dest)

{

int i, total\_moves;

char s = 'S', d = 'D', a = 'A';

//If number of disks is even, then exchange between destination pole and auxiliary pole

if (disk\_count % 2 == 0)

{

char tmp = d;

d = a;

a = tmp;

}

total\_moves = pow(2, disk\_count) - 1;

//Larger disks will be pushed first

for (i = disk\_count; i >= 1; i--)

push(src, i);

for (i = 1; i <= total\_moves; i++)

{

if (i % 3 == 0)

diskMovement(aux, dest, a, d);

else if (i % 3 == 1)

diskMovement(src, dest, s, d);

else if (i % 3 == 2)

diskMovement(src, aux, s, a);

}

}

int main()

{

int disc\_count;

// Input: number of disks

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

scanf("%d",&disc\_count);

printf("There are %d disk's \n",disc\_count);

struct Stack \*src, \*dest, \*aux;

// Creating three stacks of size 'disc\_count'

src = Create\_stack(disc\_count);

aux = Create\_stack(disc\_count);

dest = Create\_stack(disc\_count);

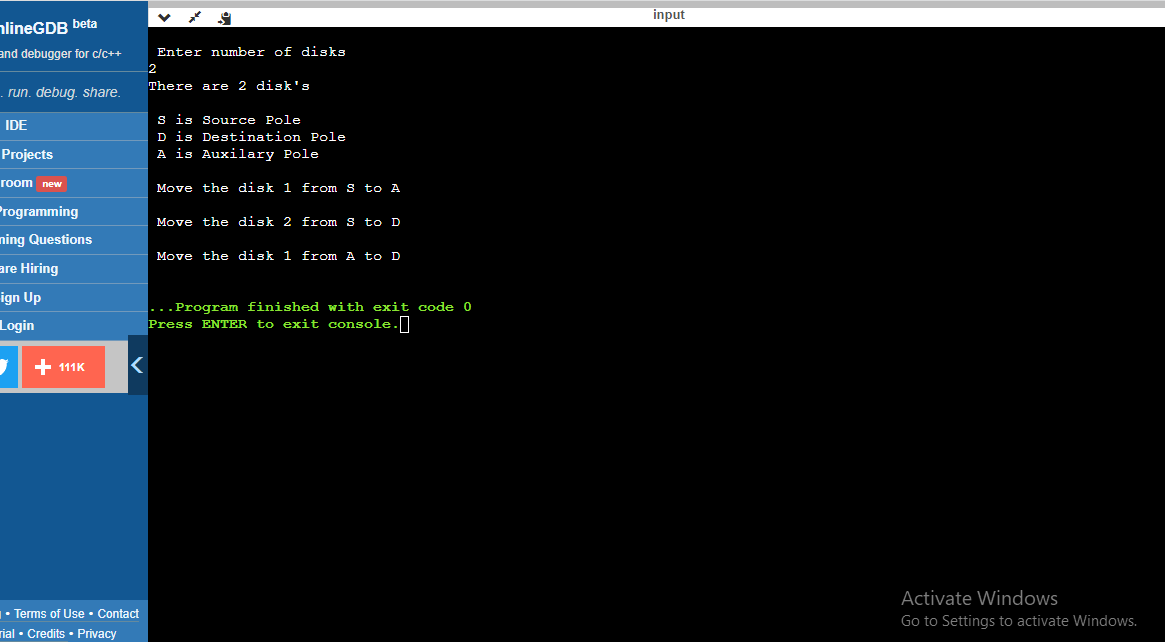
printf("\n S is Source Pole \n D is Destination Pole\n A is Auxilary Pole \n");

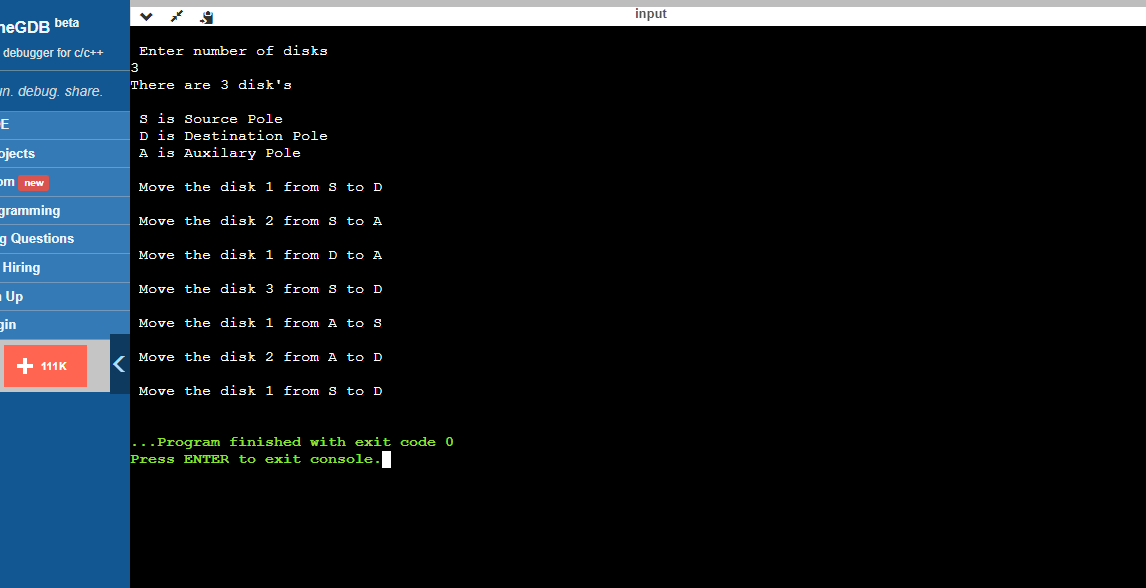
TowerOfHanoii(disc\_count, src, aux, dest);

return 0;

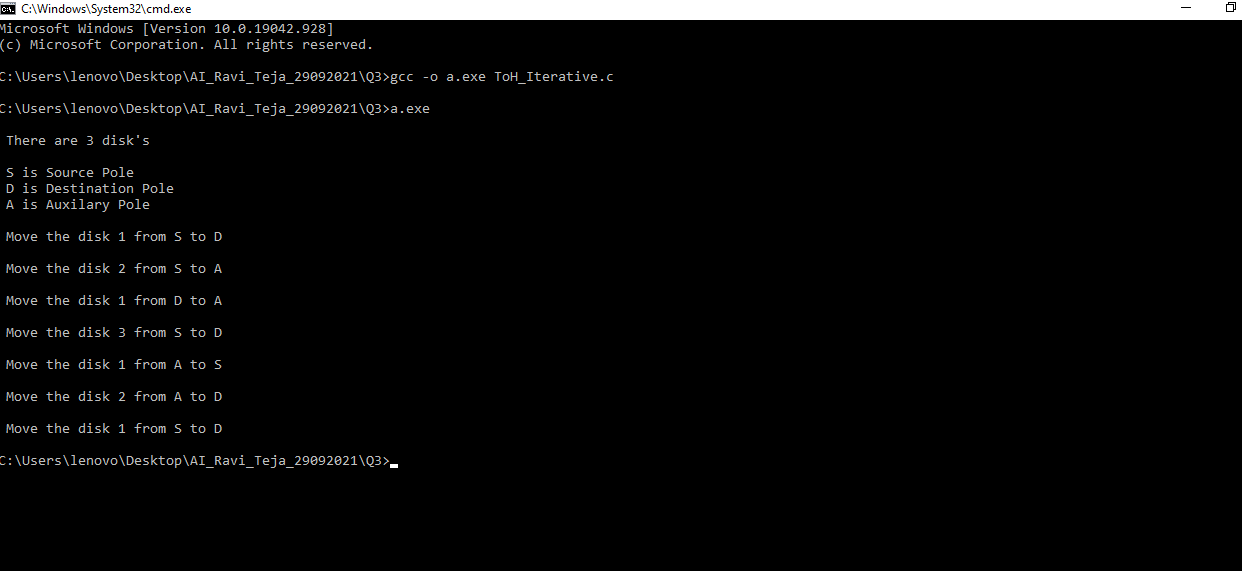
}

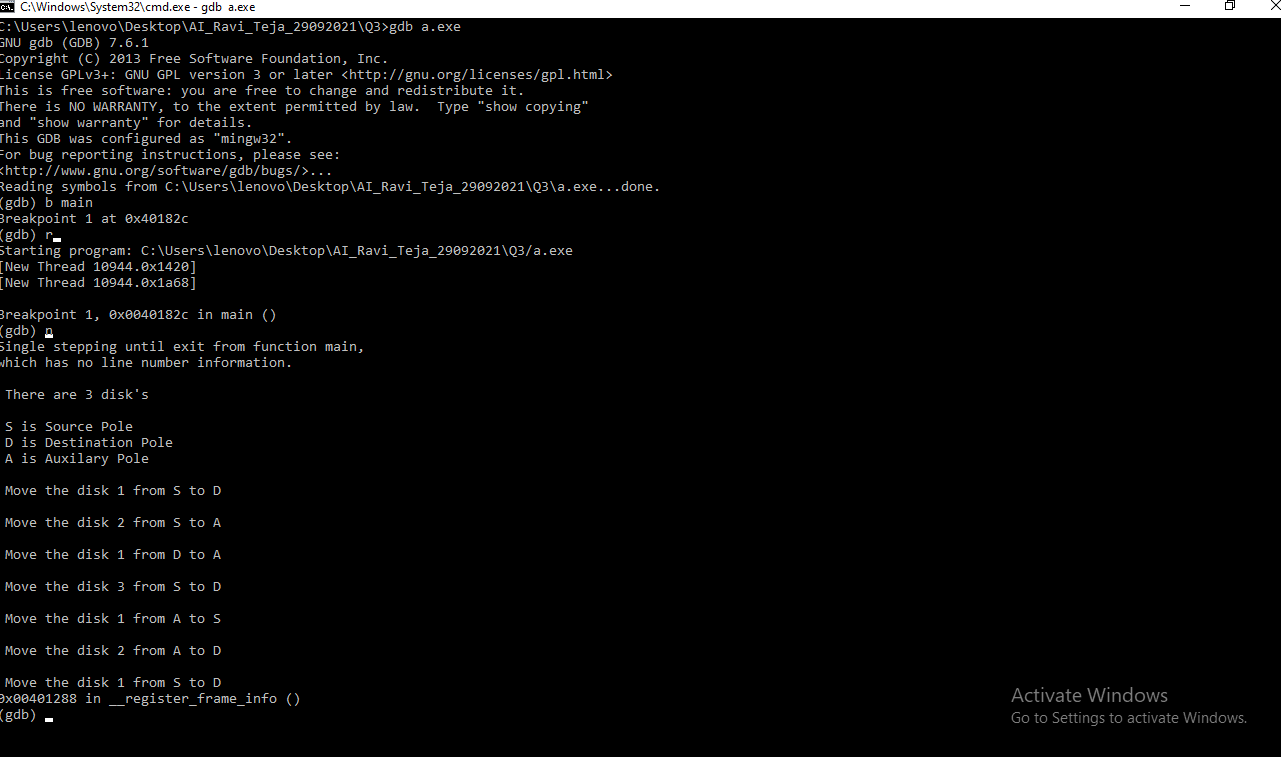
**Output**:





**Debug:**

****

****

**4. Write a program that convert the infix Expression into Postfix Expression and then evaluate the postfix expression using stacks. [Note: There should be atleast one two digit operand in the expression]**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

char stack[MAX];

int top = -1;

int stack1[MAX];

int top1 = -1;

int

isOperator (char symbol)

{

if (symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/'

|| symbol == '^')

{

return 1;

}

return 0;

}

int

precedenceValue (char symbol)

{

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

{

return 1;

}

else if (symbol == '\*' || symbol == '/')

{

return 2;

}

else if (symbol == '^')

{

return 3;

}

else

{

return 0;

}

}

void

push (char value)

{

if (top >= MAX - 1)

{

printf ("\nStack Overflow");

}

else

{

top++;

stack[top] = value;

}

}

void

push1 (int value)

{

if (top1 >= MAX - 1)

{

printf ("\nStack Overflow");

}

else

{

top1++;

stack1[top1] = value;

}

}

char

pop ()

{

char value;

if (top < 0)

{

printf ("\nStack Underflow");

getchar ();

exit (1);

}

else

{

value = stack[top];

top = top - 1;

return (value);

}

}

int

pop1 ()

{

int value;

if (top1 < 0)

{

printf ("\nStack Underflow");

getchar ();

exit (1);

}

else

{

value = stack1[top1];

top1 = top1 - 1;

return (value);

}

}

void

infixToPostfix (char infix[], char postfix[])

{

push ('('); /\* push '(' onto stack \*/

strcat (infix, ")");

//char convert[MAX];

int index = 0;

int postfixIndex = 0;

//int cIndex= 0;

char value = infix[index];

char holder;

while (value != '\0')

{

if (value == '(')

{

push (value);

}

else if (isOperator (value) == 1)

{

holder = pop (); //holder ayite stack lo undhi

while (isOperator (holder) == 1

&& precedenceValue (holder) >= precedenceValue (value))

{

postfix[postfixIndex] = holder;

postfixIndex++;

holder = pop ();

}

push (holder);

push (value);

postfix[postfixIndex] = 's';

postfixIndex++;

}

else if (value == ')')

{

holder = pop ();

while (holder != '(')

{

postfix[postfixIndex] = holder;

postfixIndex++;

holder = pop ();

}

}

else if (isdigit (value))

{

postfix[postfixIndex] = value;

postfixIndex++;

}

else

{

printf ("\nInvalid infix Expression"); /\* the it is illegeal symbol \*/

getchar ();

exit (1);

}

index++;

value = infix[index];

}

postfix[postfixIndex] = '\0';

}

int

postfix\_eval (char postfix[])

{

int a,b, temp, result, i, j = 0;

int number = 0;

/\*for(j=0,i=0;j<length;j++){

if((postfix[j]>='0' && postfix[j]<='9') && postfix[j]!='s'){

res\_postfix[j] = postfix[i];

j++;

}

else{

j++;

}

}

\*/

for (i = 0; i < strlen (postfix); i++)

{

if (postfix[i] <= '9' && postfix[i] >= '0')

{

j = i;

while (postfix[j] <= '9' && postfix[j] >= '0')

//while(isdigit(postfix[j]))

{

number = number \* 10 + (postfix[j] - '0');

j++;

}

push1(number);

i = j-1;

number = 0;

}

else if(postfix[i]=='s'){

continue;

}

else

{

a = pop1 ();

b = pop1 ();

//printf(" \n a is %d b is %d\n", a,b);

switch (postfix[i])

{

case '+':

temp = b + a;

break;

case '-':

temp = b - a;

break;

case '\*':

temp = b \* a;

break;

case '/':

temp = b / a;

break;

case '%':

temp = b % a;

break;

/\*case '^':

temp = pow (b, a);

break;\*/

}

push1(temp);

//printf(" \n %d %d\n", temp,i);

}

}

result = pop1();

return result;

}

int

main ()

{

//code

char infix[MAX];

char postfix[MAX];

char res\_postfix[MAX];

int length;

length = sizeof (postfix) / sizeof (int);

printf("\n Enter the expression \n");

gets(infix);

infixToPostfix (infix, postfix);

infix[strlen(infix)-1] = '\0';

printf("Infix expression is ");

puts(infix);

printf ("Postfix Expression: ");

puts (postfix);

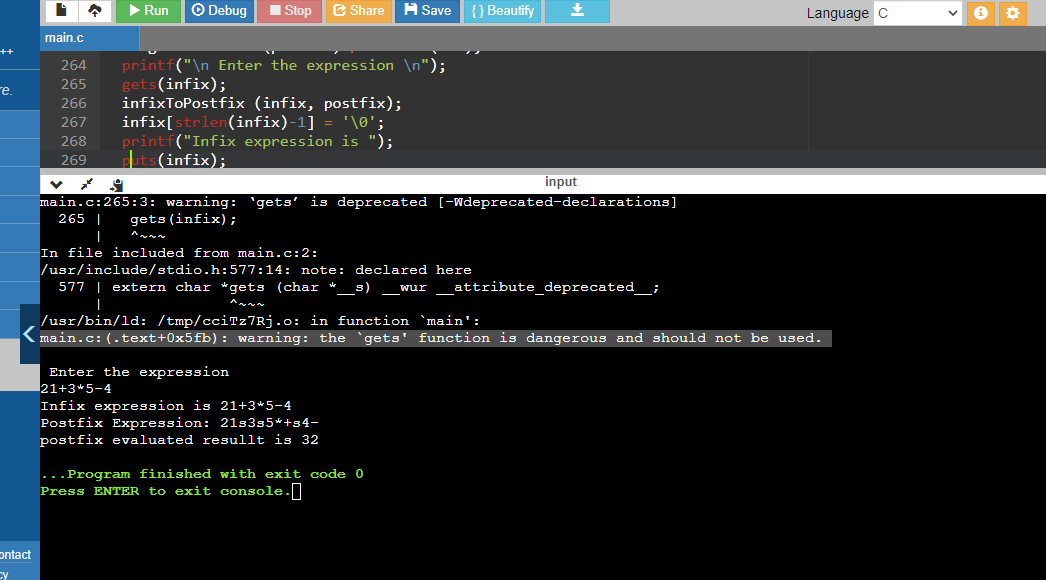
int answer = postfix\_eval (postfix);

printf("postfix evaluated resullt is %d", answer);

return 0;

}

**Output**:



**Note**: Here ‘s’ in postfix expression has nothing to do, it is just a delimiter, which helped in evaluating postfix expression.

**5. Given an array of integers and a number ‘n’, find the smallest subarray with sum greater than the given value. Display the smallest subarray by adding the size of the subarray to each element in the subarray.**

**Example - 1**

**Arr[] = {1,4,45,6,0,19}, n = 51**

**Smallest Subarray is {4,45,6}**

**Output is {7,48,9}**

**Example – 2**

**Arr[] = {1,11,100,1,0,200,3,2,1,250}, n = 280**

**Smallest subarray is {100,1,0,201}**

**Output is {104,5,4,204}**

**Source\_code:**

#include <stdio.h>

#include<stdlib.h>

int m;

// Returns length of smallest subarray with sum greater than x.

// If there is no subarray with given sum, then returns n+1

int Min\_len\_SubArray(int arr[], int n, int x)

{

// Initialize length of smallest subarray as n+1

int min\_len = n + 1,j;

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

{

int curr\_sum = arr[i]; // Initialize sum to starting element

if (curr\_sum > x) return 1; // If first element is greater

j = i+1; //varying the end points

while(j<n)

{

curr\_sum += arr[j]; // add element to present sum

if (curr\_sum > x && (j - i + 1) < min\_len){

min\_len = (j - i + 1); //update the smallest length

//printf("start is %d \n ",i);

//printf("end is %d \n", j);

m = i; //to keep track of where minimum length starts

}

j++;

}

}

return min\_len;

}

int main()

{

int arr1[] = {1,11,100,1,0,200,3,2,1,250}; //Input array

int arr2[30],giv\_num; //Result array, number

printf("\n Enter a value \n ");

scanf("%d",&giv\_num);

int arr\_size = sizeof(arr1)/sizeof(arr1[0]); // finding size of array

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

for(int i=0;i<arr\_size;i++){

printf(" %d ",arr1[i]);

}

int res = Min\_len\_SubArray(arr1, arr\_size,giv\_num);

if(res == arr\_size+1)

{

printf("\n Not possible to have \n");

}

else

{

printf("\n\n Final length of min subarray %d\n",res);

}

//printf("\n start index is %d \n",m);

printf("\n Resultant minimal sub array is \n");

for(int i=m;i<m+res;i++){

//arr2[j] =arr1[i];

printf(" %d ",arr1[i]);

}

printf("\n Resultant array after adding length %d is \n",res);

for(int i =m,j=0;i<m+res;i++,j++){

arr2[j] =arr1[i]+ res;

printf(" %d ",arr2[j]);

}

return 0;

}

**Output**:

