**16. Write a C program to implement Stack operations such as PUSH, POP and PEEK**

Input:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 10

struct Stack {

int arr[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*stack) {

stack->top = -1;

}

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

int isFull(struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

void push(struct Stack \*stack, int value) {

if (isFull(stack)) {

printf("Stack Overflow: Cannot push element %d\n", value);

} else {

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

printf("%d pushed to the stack\n", value);

}

}

int pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack Underflow: Cannot pop from empty stack\n");

return -1;

} else {

int value = stack->arr[stack->top--];

return value;

}

}

int peek(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

return -1;

} else {

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

}

}

int main() {

struct Stack stack;

initialize(&stack);

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

printf("Peek: %d\n", peek(&stack));

printf("Pop: %d\n", pop(&stack));

printf("Peek: %d\n", peek(&stack));

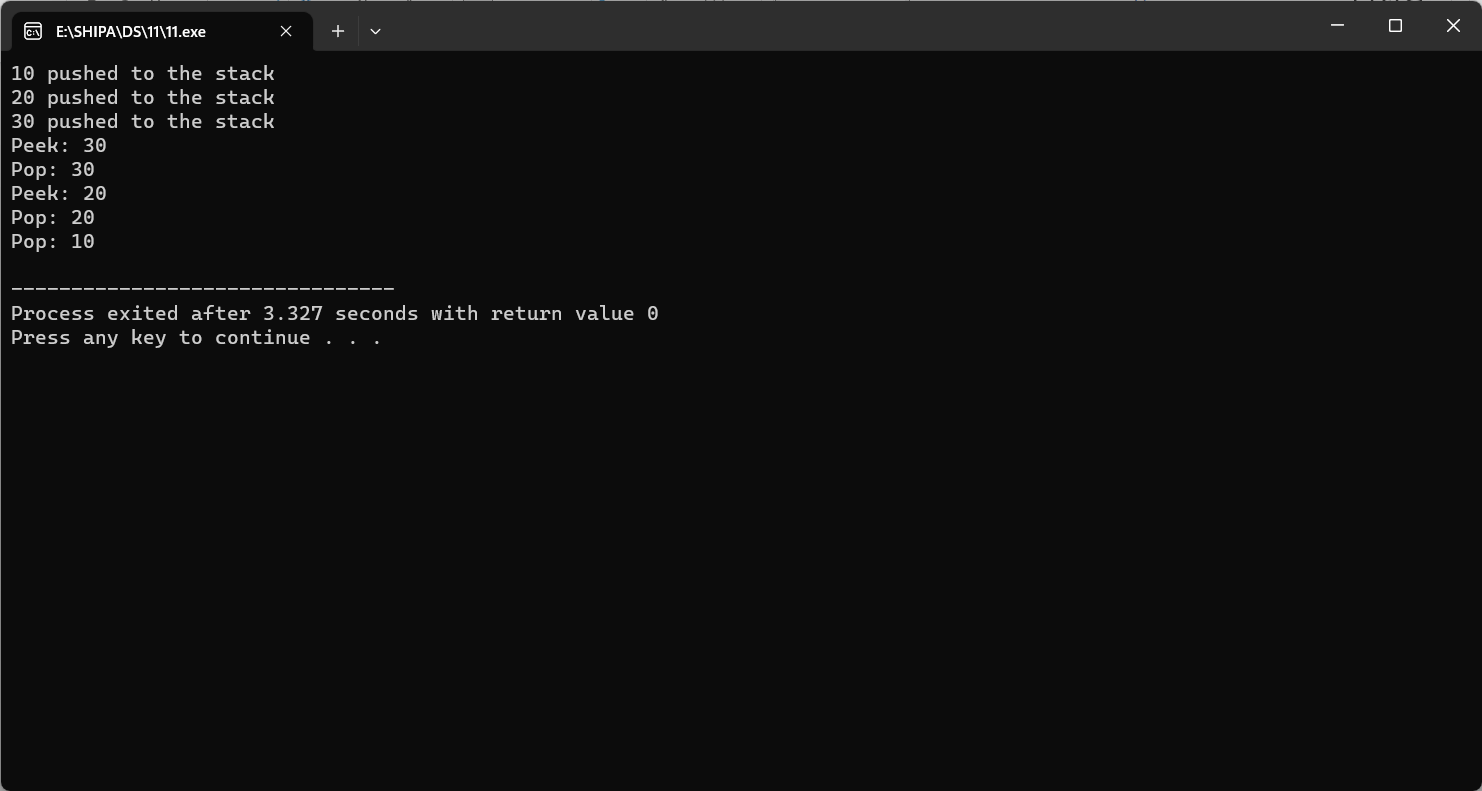
printf("Pop: %d\n", pop(&stack));

printf("Pop: %d\n", pop(&stack));

return 0;

}

Output:



**12. Write a C program to implement the application of Stack (Notations)**

Input:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

char arr[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*stack) {

stack->top = -1;

}

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

int isFull(struct Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

void push(struct Stack \*stack, char value) {

if (isFull(stack)) {

printf("Stack Overflow: Cannot push element %c\n", value);

} else {

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

}

}

char pop(struct Stack \*stack) {

if (isEmpty(stack)) {

return '\0'; // Return a special character to indicate stack underflow

} else {

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

}

}

int isBalanced(char expression[]) {

struct Stack stack;

initialize(&stack);

int len = strlen(expression);

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

if (expression[i] == '(' || expression[i] == '[' || expression[i] == '{') {

push(&stack, expression[i]);

} else if (expression[i] == ')' || expression[i] == ']' || expression[i] == '}') {

if (isEmpty(&stack)) {

return 0;

}

char popped = pop(&stack);

if ((expression[i] == ')' && popped != '(') ||

(expression[i] == ']' && popped != '[') ||

(expression[i] == '}' && popped != '{')) {

return 0;

}

}

}

return isEmpty(&stack);

}

int main() {

char expression[MAX\_SIZE];

printf("Enter an expression: ");

scanf("%s", expression);

if (isBalanced(expression)) {

printf("The expression is balanced.\n");

} else {

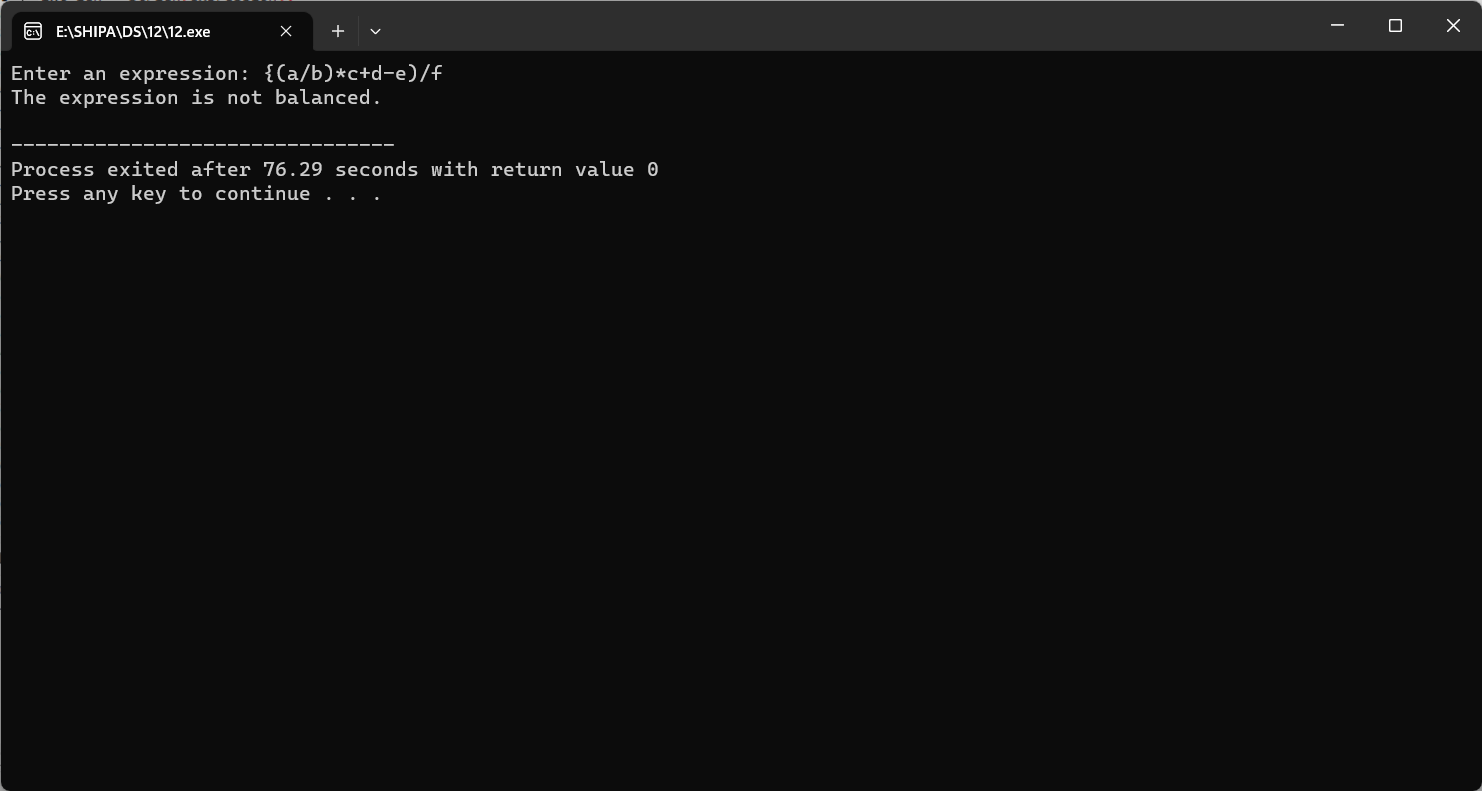
printf("The expression is not balanced.\n");

}

return 0;

}

Output:



**13. Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display**

Input:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 10

struct Queue {

int arr[MAX\_SIZE];

int front, rear;

};

void initialize(struct Queue \*queue) {

queue->front = -1;

queue->rear = -1;

}

int isEmpty(struct Queue \*queue) {

return queue->front == -1;

}

int isFull(struct Queue \*queue) {

return (queue->rear + 1) % MAX\_SIZE == queue->front;

}

void enqueue(struct Queue \*queue, int value) {

if (isFull(queue)) {

printf("Queue is full: Cannot enqueue element %d\n", value);

} else {

if (isEmpty(queue)) {

queue->front = 0;

}

queue->rear = (queue->rear + 1) % MAX\_SIZE;

queue->arr[queue->rear] = value;

printf("%d enqueued to the queue\n", value);

}

}

int dequeue(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty: Cannot dequeue\n");

return -1;

} else {

int value = queue->arr[queue->front];

if (queue->front == queue->rear) {

queue->front = -1;

queue->rear = -1;

} else {

queue->front = (queue->front + 1) % MAX\_SIZE;

}

return value;

}

}

void display(struct Queue \*queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

} else {

printf("Queue elements: ");

int i = queue->front;

while (i != queue->rear) {

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

i = (i + 1) % MAX\_SIZE;

}

printf("%d\n", queue->arr[i]);

}

}

int main() {

struct Queue queue;

initialize(&queue);

enqueue(&queue, 10);

enqueue(&queue, 20);

enqueue(&queue, 30);

display(&queue);

printf("Dequeued: %d\n", dequeue(&queue));

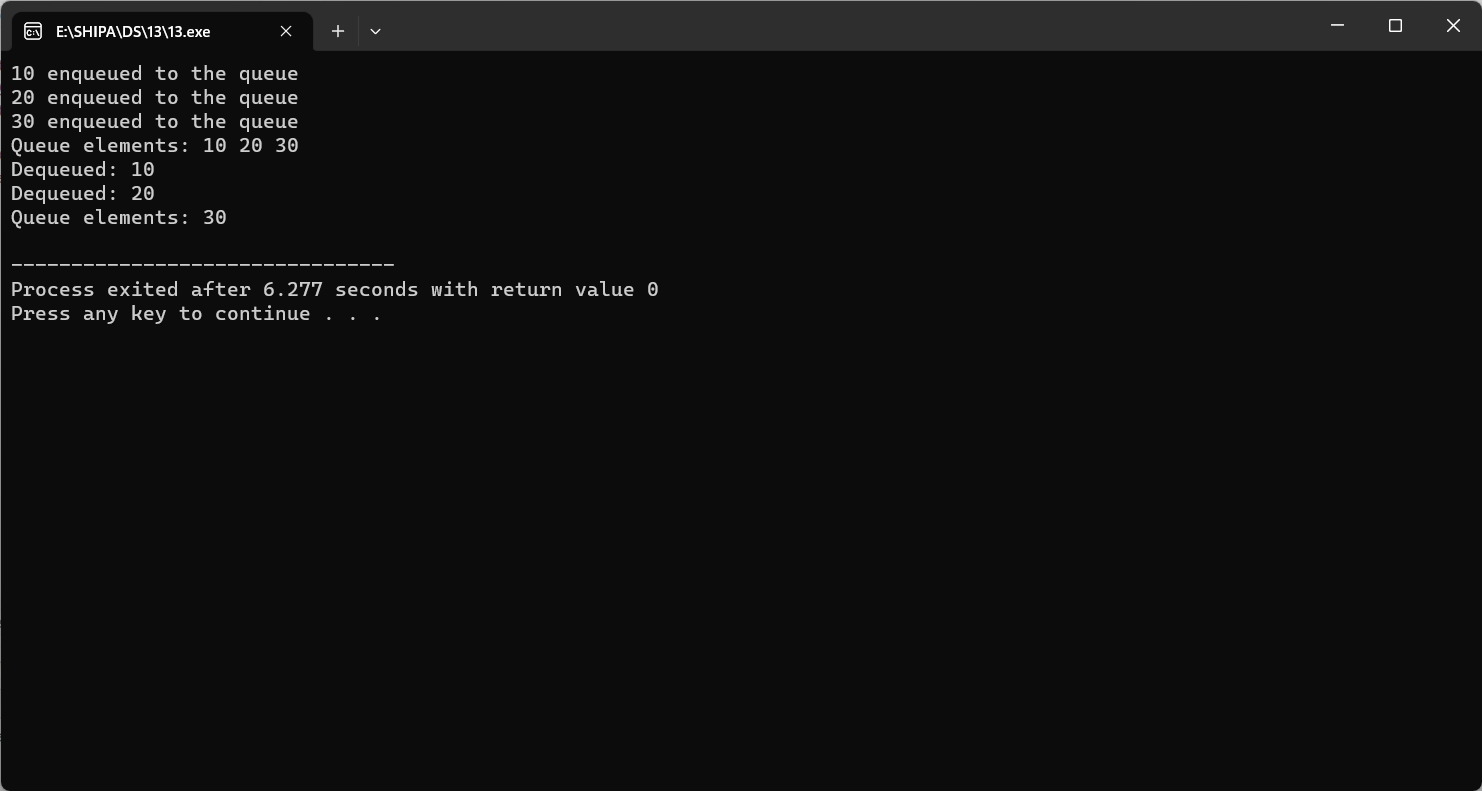
printf("Dequeued: %d\n", dequeue(&queue));

display(&queue);

return 0;

}

Output:



**14. Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder)**

Input:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*left;

struct Node \*right;

};

struct Node \*createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void inorderTraversal(struct Node \*root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

void preorderTraversal(struct Node \*root) {

if (root == NULL) {

return;

}

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

void postorderTraversal(struct Node \*root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

int main() {

struct Node \*root = createNode(1);

root->left = createNode(2);

root->right = createNode(3);

root->left->left = createNode(4);

root->left->right = createNode(5);

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

printf("Preorder Traversal: ");

preorderTraversal(root);

printf("\n");

printf("Postorder Traversal: ");

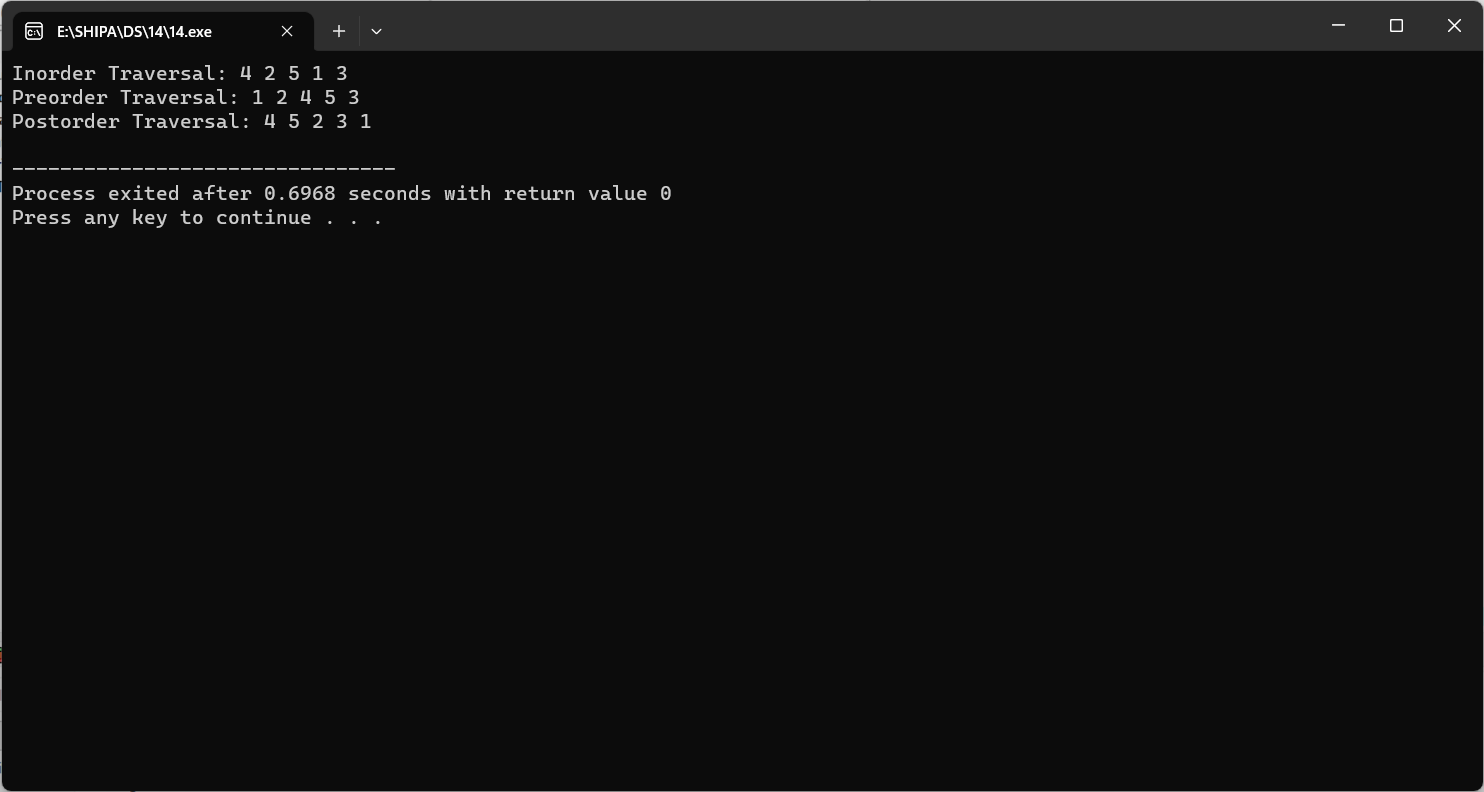
postorderTraversal(root);

printf("\n");

return 0;

}

Output:



**15. Write a C program to implement hashing using Linear Probing method**

Input:

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

struct HashTable {

int table[TABLE\_SIZE];

};

void initializeHashTable(struct HashTable \*hashTable) {

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable->table[i] = -1; // Initialize all slots as empty (-1)

}

}

int hashFunction(int key) {

return key % TABLE\_SIZE;

}

void insert(struct HashTable \*hashTable, int value) {

int index = hashFunction(value);

while (hashTable->table[index] != -1) {

index = (index + 1) % TABLE\_SIZE; // Move to the next slot using linear probing

}

hashTable->table[index] = value;

}

void display(struct HashTable \*hashTable) {

printf("Hash Table:\n");

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable->table[i] != -1) {

printf("Index %d: %d\n", i, hashTable->table[i]);

}

}

}

int main() {

struct HashTable hashTable;

initializeHashTable(&hashTable);

int values[] = {25, 45, 36, 77, 82, 19, 50, 38, 29};

for (int i = 0; i < sizeof(values) / sizeof(values[0]); i++) {

insert(&hashTable, values[i]);

}

display(&hashTable);

return 0;

}

Output:

