LAB TASK-4

1.

#include <stdio.h>

#define MAX\_SIZE 100

typedef struct {

int arr[MAX\_SIZE];

int top;

} Stack;

void initialize(Stack \*s) {

s->top = -1;

}

int isEmpty(Stack \*s) {

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

}

int isFull(Stack \*s) {

return (s->top == MAX\_SIZE - 1);

}

void push(Stack \*s, int item) {

if (isFull(s)) {

printf("Stack Overflow!\n");

return;

}

s->top++;

s->arr[s->top] = item;

}

int pop(Stack \*s) {

if (isEmpty(s)) {

printf("Stack Underflow!\n");

return -1;

}

int item = s->arr[s->top];

s->top--;

return item;

}

int peek(Stack \*s) {

if (isEmpty(s)) {

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

return -1;

}

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

}

int main() {

Stack s;

initialize(&s);

push(&s, 7);

push(&s, 29);

push(&s, 67);

printf("Top element of the stack: %d\n", peek(&s));

printf("Popped element: %d\n", pop(&s));

printf("Top element of the stack: %d\n", peek(&s));

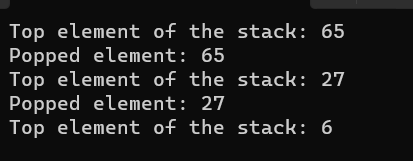
printf("Popped element: %d\n", pop(&s));

printf("Top element of the stack: %d\n", peek(&s));

return 0;

}

OUTPUT:



2.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

int precedence(char op) {

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

return 1;

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

return 2;

return 0;

}

void infixToPostfix(char \*infix, char \*postfix) {

char stack[MAX\_SIZE];

int top = -1;

int i, j;

for (i = 0, j = 0; infix[i] != '\0'; i++) {

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

} else if (infix[i] == '(') {

stack[++top] = infix[i];

} else if (infix[i] == ')') {

while (top != -1 && stack[top] != '(') {

postfix[j++] = stack[top--];

}

if (top == -1) {

printf("Invalid expression: Unbalanced parenthesis\n");

exit(EXIT\_FAILURE);

}

top--;

} else {

while (top != -1 && precedence(stack[top]) >= precedence(infix[i])) {

postfix[j++] = stack[top--];

}

stack[++top] = infix[i];

}

}

while (top != -1) {

if (stack[top] == '(') {

printf("Invalid expression: Unbalanced parenthesis\n");

exit(EXIT\_FAILURE);

}

postfix[j++] = stack[top--];

}

postfix[j] = '\0';

}

int main() {

char infix[MAX\_SIZE];

char postfix[MAX\_SIZE];

printf("Enter infix expression: ");

fgets(infix, MAX\_SIZE, stdin);

infix[strcspn(infix, "\n")] = '\0';

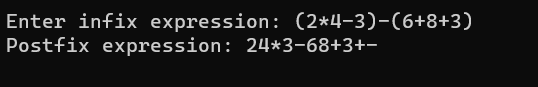
infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

OUTPUT:



3.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

typedef struct {

int data[MAX\_SIZE];

int top;

} Stack;

void initialize(Stack \*stack) {

stack->top = -1;

}

int isEmpty(Stack \*stack) {

return stack->top == -1;

}

void push(Stack \*stack, int element) {

stack->data[++stack->top] = element;

}

int pop(Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack underflow\n");

exit(EXIT\_FAILURE);

}

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

}

int evaluatePostfix(char \*postfix) {

Stack stack;

initialize(&stack);

int i = 0;

while (postfix[i] != '\0') {

if (isdigit(postfix[i])) {

push(&stack, postfix[i] - '0');

} else {

int operand2 = pop(&stack);

int operand1 = pop(&stack);

switch (postfix[i]) {

case '+':

push(&stack, operand1 + operand2);

break;

case '-':

push(&stack, operand1 - operand2);

break;

case '\*':

push(&stack, operand1 \* operand2);

break;

case '/':

if (operand2 != 0)

push(&stack, operand1 / operand2);

else {

printf("Error: Division by zero\n");

exit(EXIT\_FAILURE);

}

break;

default:

printf("Invalid character in expression\n");

exit(EXIT\_FAILURE);

}

}

i++;

}

return pop(&stack);

}

int main() {

char postfix[MAX\_SIZE];

printf("Enter a postfix expression: ");

fgets(postfix, MAX\_SIZE, stdin);

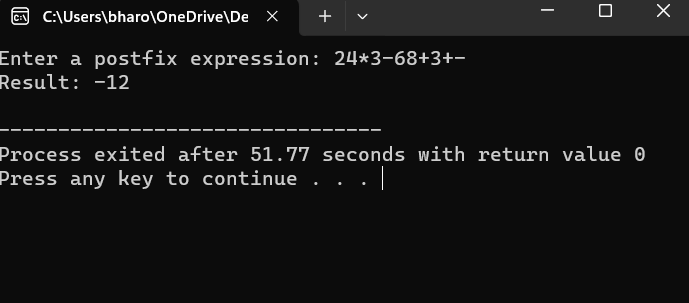
postfix[strlen(postfix) - 1] = '\0';

int result = evaluatePostfix(postfix);

printf("Result: %d\n", result);

return 0;

}



4.

#include <stdio.h>

void move(int n, int source, int destination, int intermediate) {

if (n == 1) {

printf("Move disk 1 from rod %d to rod %d\n", source, destination);

return;

}

move(n - 1, source, intermediate, destination);

printf("Move disk %d from rod %d to rod %d\n", n, source, destination);

move(n - 1, intermediate, destination, source);

}

int main() {

int num\_disks = 4;

move(num\_disks, 1, 3, 2);

return 0;

}

Output:

