A.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

struct Stack {

int stack[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*s) {

s->top = -1;

}

int is\_empty(struct Stack \*s) {

return s->top == -1;

}

int is\_full(struct Stack \*s) {

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

}

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

if (is\_full(s)) {

printf("Stack overflow. Cannot push element.\n");

} else {

s->top++;

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

printf("Pushed %d to the stack.\n", item);

}

}

int pop(struct Stack \*s) {

if (is\_empty(s)) {

printf("Stack underflow. Cannot pop element.\n");

return -1;

} else {

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

s->top--;

printf("Popped %d from the stack.\n", item);

return item;

}

}

int peek(struct Stack \*s) {

if (is\_empty(s)) {

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

return -1;

} else {

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

}

}

int size(struct Stack \*s) {

return s->top + 1;

}

int main() {

struct Stack stack;

initialize(&stack);

push(&stack, 1);

push(&stack, 2);

push(&stack, 3);

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

printf("Stack size: %d\n", size(&stack));

int popped\_item = pop(&stack);

printf("Popped item: %d\n", popped\_item);

printf("Is the stack empty? %s\n", is\_empty(&stack) ? "Yes" : "No");

return 0;

}

A screenshot of a phone

Description automatically generated

B.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

char stack[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*s) {

s->top = -1;

}

int is\_empty(struct Stack \*s) {

return s->top == -1;

}

void push(struct Stack \*s, char item) {

s->top++;

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

}

char pop(struct Stack \*s) {

if (!is\_empty(s)) {

char item = s->stack[s->top];

s->top--;

return item;

}

return '\0';

}

char peek(struct Stack \*s) {

if (!is\_empty(s)) {

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

}

return '\0';

}

int is\_operator(char ch) {

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

}

int precedence(char ch) {

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

return 1;

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

return 2;

return 0;

}

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

struct Stack stack;

initialize(&stack);

int i, j;

i = j = 0;

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

if (infix[i] >= 'A' && infix[i] <= 'Z' || infix[i] >= 'a' && infix[i] <= 'z') {

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

} else if (is\_operator(infix[i])) {

while (!is\_empty(&stack) && precedence(infix[i]) <= precedence(peek(&stack))) {

postfix[j++] = pop(&stack);

}

push(&stack, infix[i++]);

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

push(&stack, infix[i++]);

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

while (!is\_empty(&stack) && peek(&stack) != '(') {

postfix[j++] = pop(&stack);

}

if (!is\_empty(&stack) && peek(&stack) == '(') {

pop(&stack); // Discard the '('

}

i++;

} else {

// Ignore other characters like spaces

i++;

}

}

while (!is\_empty(&stack)) {

postfix[j++] = pop(&stack);

}

postfix[j] = '\0';

}

int main() {

char infix[100];

char postfix[100];

printf("Enter infix expression: ");

fgets(infix, sizeof(infix), stdin);

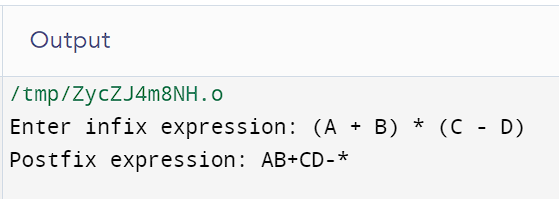
infix[strcspn(infix, "\n")] = '\0';

infixToPostfix(infix, postfix);

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

return 0;

}



C.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_SIZE 100

struct Stack {

int stack[MAX\_SIZE];

int top;

};

void initialize(struct Stack \*s) {

s->top = -1;

}

int is\_empty(struct Stack \*s) {

return s->top == -1;

}

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

s->top++;

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

}

int pop(struct Stack \*s) {

if (!is\_empty(s)) {

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

s->top--;

return item;

}

return 0; //

}

int evaluatePostfix(char postfix[]) {

struct 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 '/':

push(&stack, operand1 / operand2);

break;

}

}

i++;

}

return pop(&stack);

}

int main() {

char postfix[100];

printf("Enter postfix expression: ");

fgets(postfix, sizeof(postfix), stdin);

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

int result = evaluatePostfix(postfix);

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

return 0;

}



**Assignment-1**

**#include <stdio.h>**

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

**if (n == 1) {**

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

**return;**

**}**

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

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

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

**}**

**int main() {**

**int n = 4;**

**int source = 1;**

**int destination = 3;**

**int intermediate = 2;**

**move(n, source, destination, intermediate);**

**return 0;**

**}**

