1.Implementation of stack using array

Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

// Structure to represent the stack

struct Stack {

int array[MAX\_SIZE];

int top;

};

// Function to initialize the stack

void initialize(struct Stack \*stack) {

stack->top = -1;

}

// Function to check if the stack is empty

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

// Function to check if the stack is full

int isFull(struct Stack \*stack) {

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

}

// Function to push an element onto the stack

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

if (isFull(stack)) {

printf("Stack Overflow\n");

return;

}

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

}

// Function to pop an element from the stack

int pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack Underflow\n");

exit(1);

}

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

}

// Function to peek the top element of the stack

int peek(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

exit(1);

}

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

}

int main() {

struct Stack stack;

initialize(&stack);

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

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

printf("Elements popped from stack:\n");

while (!isEmpty(&stack)) {

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

}

return 0;

}

2.Infinix to postfix

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

// Structure to represent the stack

struct Stack {

char array[MAX\_SIZE];

int top;

};

// Function to initialize the stack

void initialize(struct Stack \*stack) {

stack->top = -1;

}

// Function to check if the stack is empty

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

// Function to check if the stack is full

int isFull(struct Stack \*stack) {

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

}

// Function to push an element onto the stack

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

if (isFull(stack)) {

printf("Stack Overflow\n");

exit(1);

}

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

}

// Function to pop an element from the stack

char pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack Underflow\n");

exit(1);

}

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

}

// Function to return the precedence of operators

int precedence(char op) {

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

return 1;

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

return 2;

return 0;

}

// Function to convert infix expression to postfix expression

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

struct Stack stack;

initialize(&stack);

int i = 0, j = 0;

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

if (isdigit(infix[i]) || isalpha(infix[i])) {

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

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

push(&stack, infix[i]);

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

while (!isEmpty(&stack) && stack.array[stack.top] != '(') {

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

}

if (!isEmpty(&stack) && stack.array[stack.top] != '(') {

printf("Invalid expression\n");

exit(1);

} else {

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

}

} else { // Operator

while (!isEmpty(&stack) && precedence(infix[i]) <= precedence(stack.array[stack.top])) {

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

}

push(&stack, infix[i]);

}

i++;

}

while (!isEmpty(&stack)) {

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

}

postfix[j] = '\0';

}

int main() {

char infix[MAX\_SIZE], postfix[MAX\_SIZE];

printf("Enter infix expression: ");

fgets(infix, MAX\_SIZE, stdin);

infix[strlen(infix) - 1] = '\0'; // Remove newline character

infixToPostfix(infix, postfix);

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

return 0;

}

3.Evaluating postfix

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

// Structure to represent the stack

struct Stack {

int array[MAX\_SIZE];

int top;

};

// Function to initialize the stack

void initialize(struct Stack \*stack) {

stack->top = -1;

}

// Function to check if the stack is empty

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

// Function to check if the stack is full

int isFull(struct Stack \*stack) {

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

}

// Function to push an element onto the stack

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

if (isFull(stack)) {

printf("Stack Overflow\n");

exit(1);

}

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

}

// Function to pop an element from the stack

int pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack Underflow\n");

exit(1);

}

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

}

// Function to evaluate postfix expression

int evaluatePostfix(char \*postfix) {

struct Stack stack;

initialize(&stack);

int i, operand1, operand2, result;

for (i = 0; postfix[i] != '\0'; i++) {

if (isdigit(postfix[i])) {

push(&stack, postfix[i] - '0'); // Convert char to int

} else {

operand2 = pop(&stack);

operand1 = pop(&stack);

switch (postfix[i]) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

if (operand2 == 0) {

printf("Division by zero error\n");

exit(1);

}

result = operand1 / operand2;

break;

default:

printf("Invalid operator\n");

exit(1);

}

push(&stack, result);

}

}

return pop(&stack);

}

int main() {

char postfix[MAX\_SIZE];

printf("Enter postfix expression: ");

fgets(postfix, MAX\_SIZE, stdin);

postfix[strlen(postfix) - 1] = '\0'; // Remove newline character

int result = evaluatePostfix(postfix);

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

return 0;

}

4.Tower of Hanoi

Code:

#include <stdio.h>

// Function to perform the move

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 disks = 4; // Number of disks

int source = 1; // Source rod

int destination = 3; // Destination rod

int intermediate = 2; // Intermediate rod

move(disks, source, destination, intermediate);

return 0;

}