#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_STACK\_SIZE 100

// Structure to represent stack

struct Stack {

int top;

char items[MAX\_STACK\_SIZE];

};

// Function to initialize stack

void initialize(struct Stack \*s) {

s->top = -1;

}

// Function to push element onto stack

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

if (s->top == MAX\_STACK\_SIZE - 1) {

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

s->items[++(s->top)] = c;

}

// Function to pop element from stack

char pop(struct Stack \*s) {

if (s->top == -1) {

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return s->items[(s->top)--];

}

// Function to check if a character is an operator

int is\_operator(char c) {

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

}

// Function to get precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/' || op == '%') return 2;

return 0;

}

// Function to convert infix expression to postfix (RPN) expression

void infix\_to\_rpn(char \*infix, char \*rpn) {

struct Stack operator\_stack;

initialize(&operator\_stack);

int i, j = 0;

for (i = 0; infix[i]; i++) {

if (isdigit(infix[i])) {

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

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

while (operator\_stack.top != -1 &&

precedence(operator\_stack.items[operator\_stack.top]) >= precedence(infix[i])) {

rpn[j++] = pop(&operator\_stack);

}

push(&operator\_stack, infix[i]);

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

push(&operator\_stack, infix[i]);

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

while (operator\_stack.top != -1 && operator\_stack.items[operator\_stack.top] != '(') {

rpn[j++] = pop(&operator\_stack);

}

if (operator\_stack.top == -1) {

printf("Mismatched parentheses\n");

exit(EXIT\_FAILURE);

}

pop(&operator\_stack);

}

}

while (operator\_stack.top != -1) {

if (operator\_stack.items[operator\_stack.top] == '(') {

printf("Mismatched parentheses\n");

exit(EXIT\_FAILURE);

}

rpn[j++] = pop(&operator\_stack);

}

rpn[j] = '\0';

}

int main() {

char infix[100], rpn[100];

printf("Enter an infix expression: ");

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

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

infix\_to\_rpn(infix, rpn);

printf("Equivalent RPN expression: %s\n", rpn);

return 0;

}