LAB TASK-4

#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));

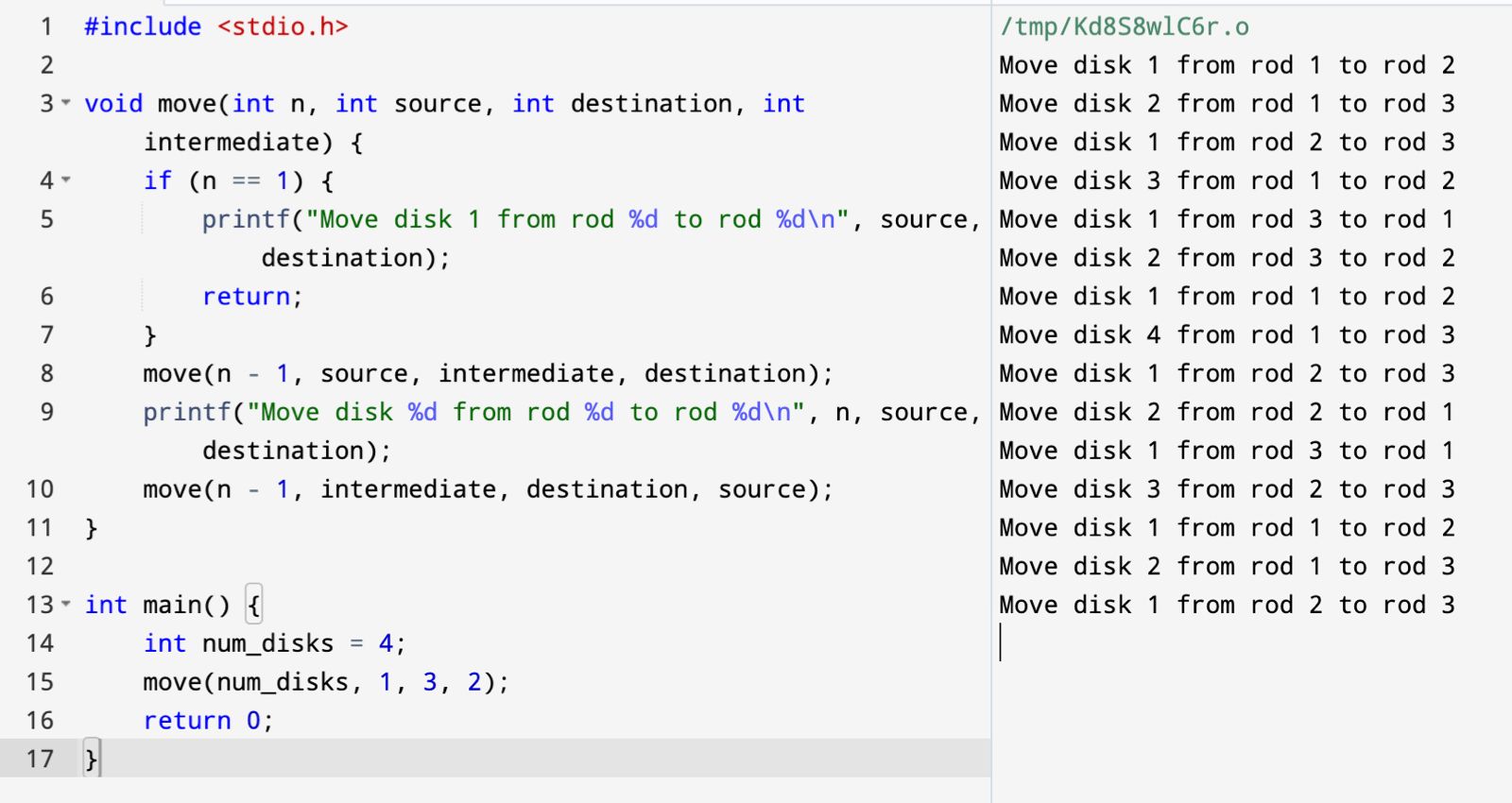
printf("Top element of the stack: %d\n", peek(&s));

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

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

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

int top;

unsigned capacity;

char \*array;

};

struct Stack \*createStack(unsigned capacity) {

struct Stack \*stack = (struct Stack \*)malloc(sizeof(struct Stack));

if (!stack) return NULL;

stack->top = -1;

stack->capacity = capacity;

stack->array = (char \*)malloc(stack->capacity \* sizeof(char));

if (!stack->array) return NULL;

return stack;

}

int isEmpty(struct Stack \*stack) {

return stack->top == -1;

}

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

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

}

char pop(struct Stack \*stack) {

if (!isEmpty(stack))

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

return '$';

}

int precedence(char op) {

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

return 1;

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

return 2;

return 0;

}

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

struct Stack \*stack = createStack(strlen(infix));

int i, k;

for (i = 0, k = -1; infix[i]; ++i) {

if (isalnum(infix[i]))

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

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

push(stack, infix[i]);

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

while (!isEmpty(stack) && stack->array[stack->top] != '(')

postfix[++k] = pop(stack);

if (!isEmpty(stack) && stack->array[stack->top] != '(')

return;

else

pop(stack);

}

else {

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

postfix[++k] = pop(stack);

push(stack, infix[i]);

}

}

while (!isEmpty(stack))

postfix[++k] = pop(stack);

postfix[++k] = '\0';

}

int main() {

char infix[MAX\_SIZE];

char postfix[MAX\_SIZE];

printf("Enter the infix expression: ");

fgets(infix, MAX\_SIZE, stdin);

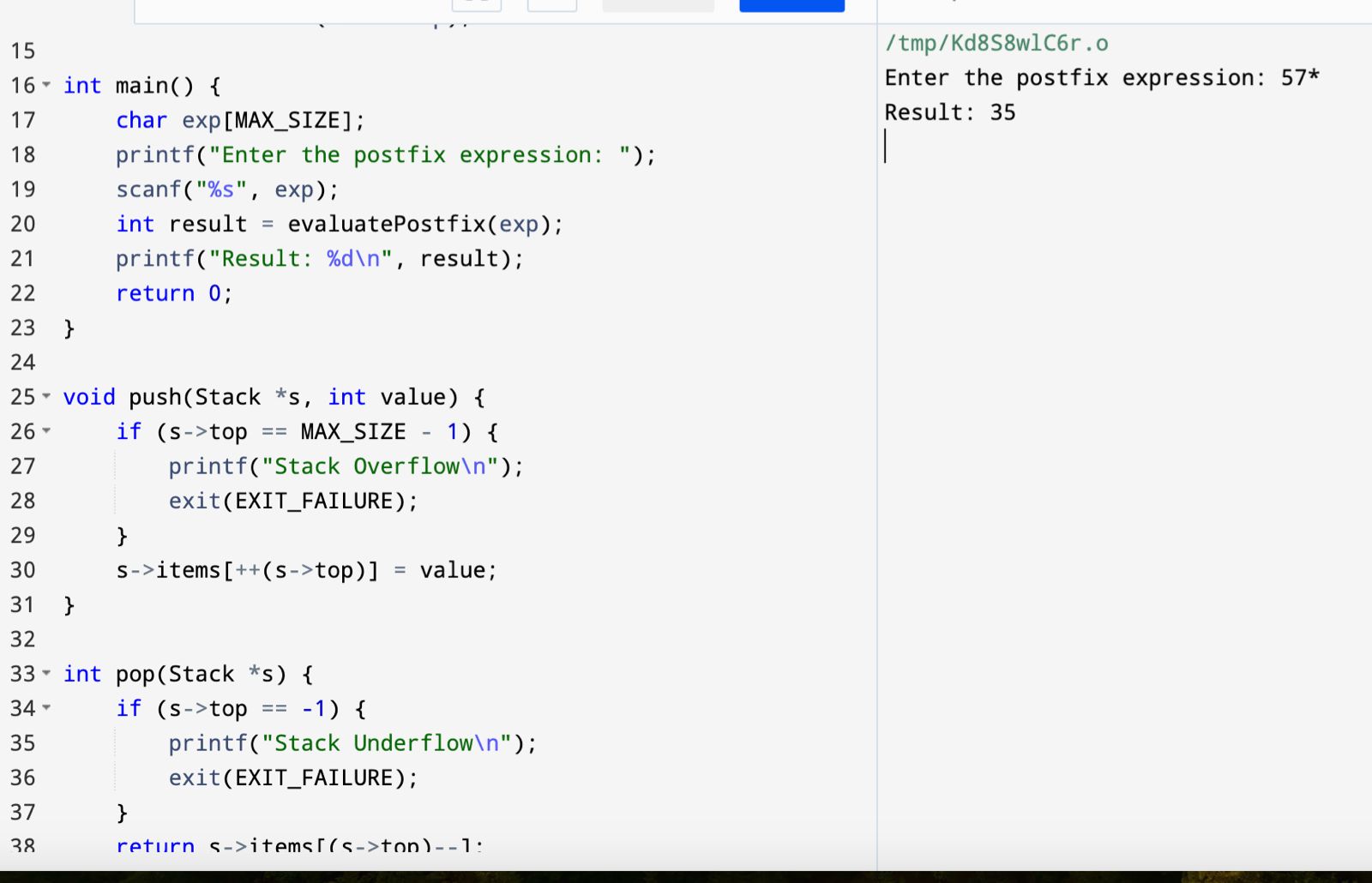
infix[strcspn(infix, "\n")] = 0;

infixToPostfix(infix, postfix);

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

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_SIZE 100

typedef struct {

int top;

int items[MAX\_SIZE];

} Stack;

void push(Stack \*s, int value);

int pop(Stack \*s);

int evaluatePostfix(char \*exp);

int main() {

char exp[MAX\_SIZE];

printf("Enter the postfix expression: ");

scanf("%s", exp);

int result = evaluatePostfix(exp);

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

return 0;

}

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

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

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

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

}

int pop(Stack \*s) {

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

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

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

}

int evaluatePostfix(char \*exp) {

Stack s;

s.top = -1;

int i, op1, op2, result;

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

if (isdigit(exp[i])) {

push(&s, exp[i] - '0');

} else {

op2 = pop(&s);

op1 = pop(&s);

switch (exp[i]) {

case '+':

push(&s, op1 + op2);

break;

case '-':

push(&s, op1 - op2);

break;

case '\*':

push(&s, op1 \* op2);

break;

case '/':

push(&s, op1 / op2);

break;

default:

printf("Invalid operator\n");

exit(EXIT\_FAILURE);

}

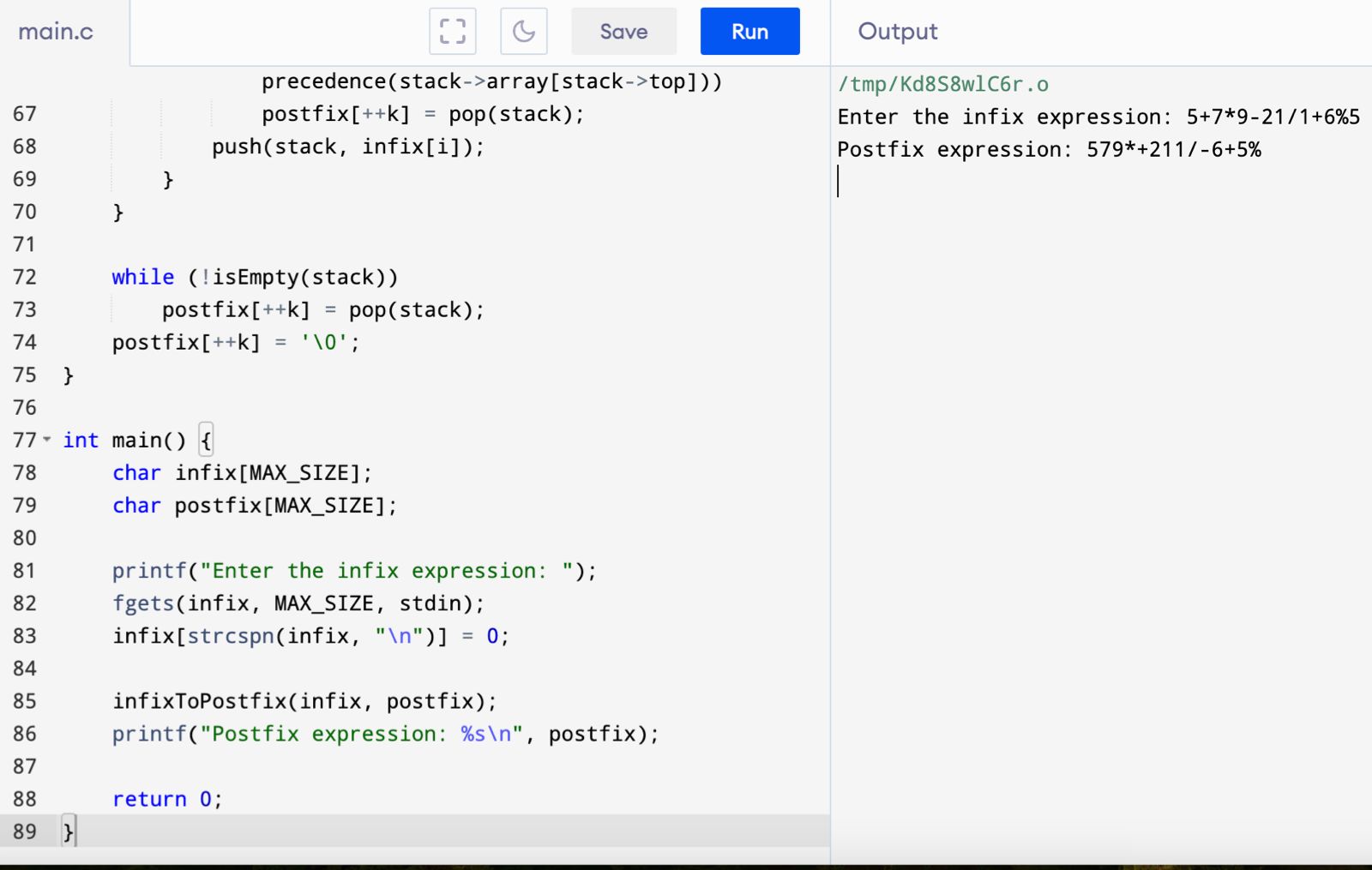
}

}

result = pop(&s);

return result;

}



#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;

}

