9. All languages have Grammar. When people frame a sentence we usually say whether the sentence is framed as per the rules of the grammar or not .similarly use the same ideology ,implement to check whether the given input string is satisfying the grammar or not .  
#include <stdio.h>

#include <string.h>

int check(char str[], int i, int j) {

if (i > j) return 0;

if (str[i] == 'a' && str[j] == 'b') {

if (i + 1 == j) return 1;

return check(str, i + 1, j - 1);

}

return 0;

}

int main() {

char str[50];

printf("Enter the string: ");

scanf("%s", str);

if (check(str, 0, strlen(str) - 1))

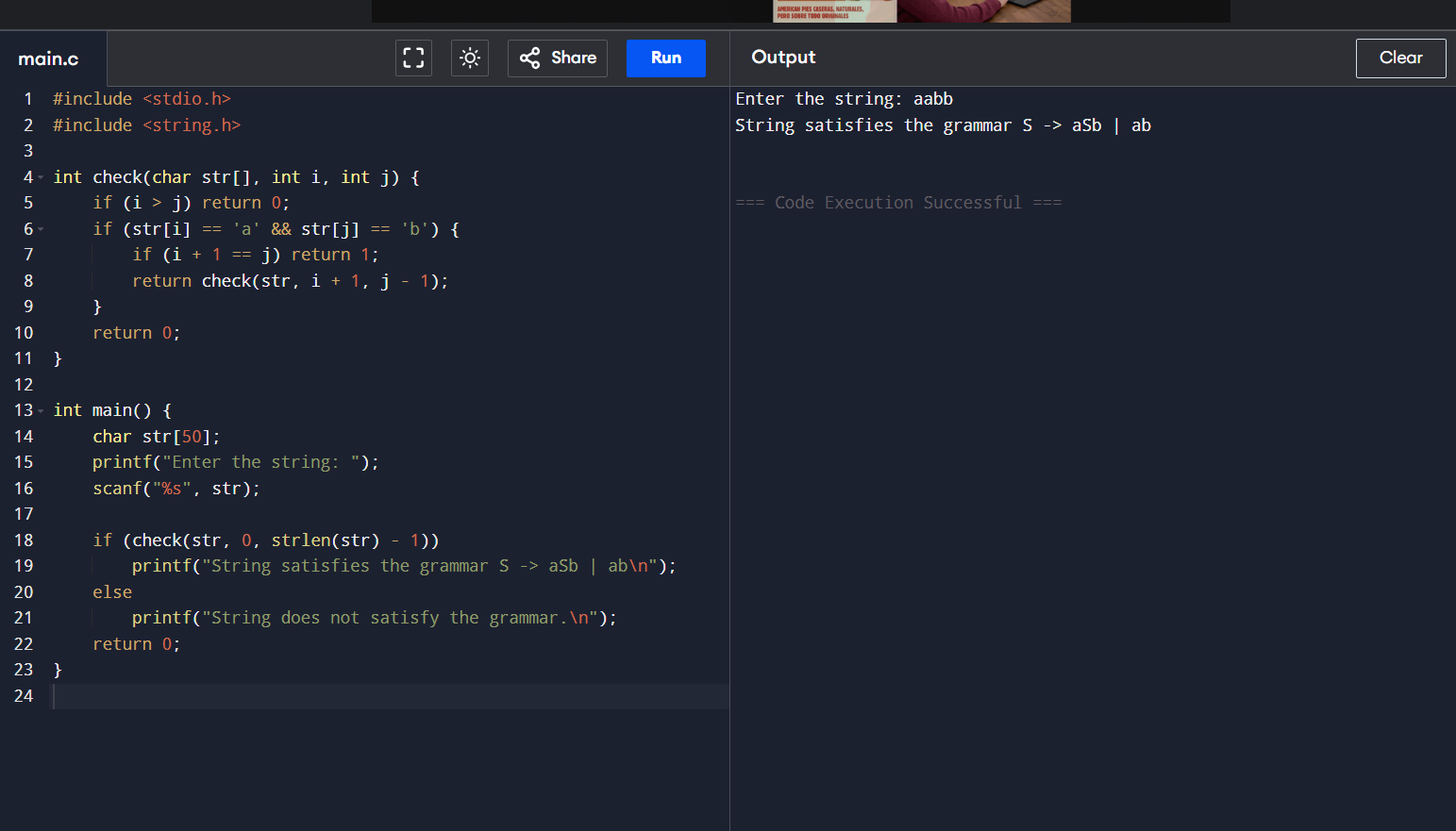
printf("String satisfies the grammar S -> aSb | ab\n");

else

printf("String does not satisfy the grammar.\n");

return 0;

}



10.Write a c program to construct recursive descent parsing.

#include <stdio.h>

#include <string.h>

char input[20];

int i = 0;

void E();

void T();

void F();

void E() {

T();

if (input[i] == '+') {

i++;

E();

}

}

void T() {

F();

if (input[i] == '\*') {

i++;

T();

}

}

void F() {

if (input[i] == '(') {

i++;

E();

if (input[i] == ')') i++;

else printf("Error: missing ')'\n");

} else if (input[i] == 'i' && input[i+1] == 'd') {

i += 2;

} else {

printf("Error: invalid factor\n");

}

}

int main() {

printf("Enter an expression: ");

scanf("%s", input);

E();

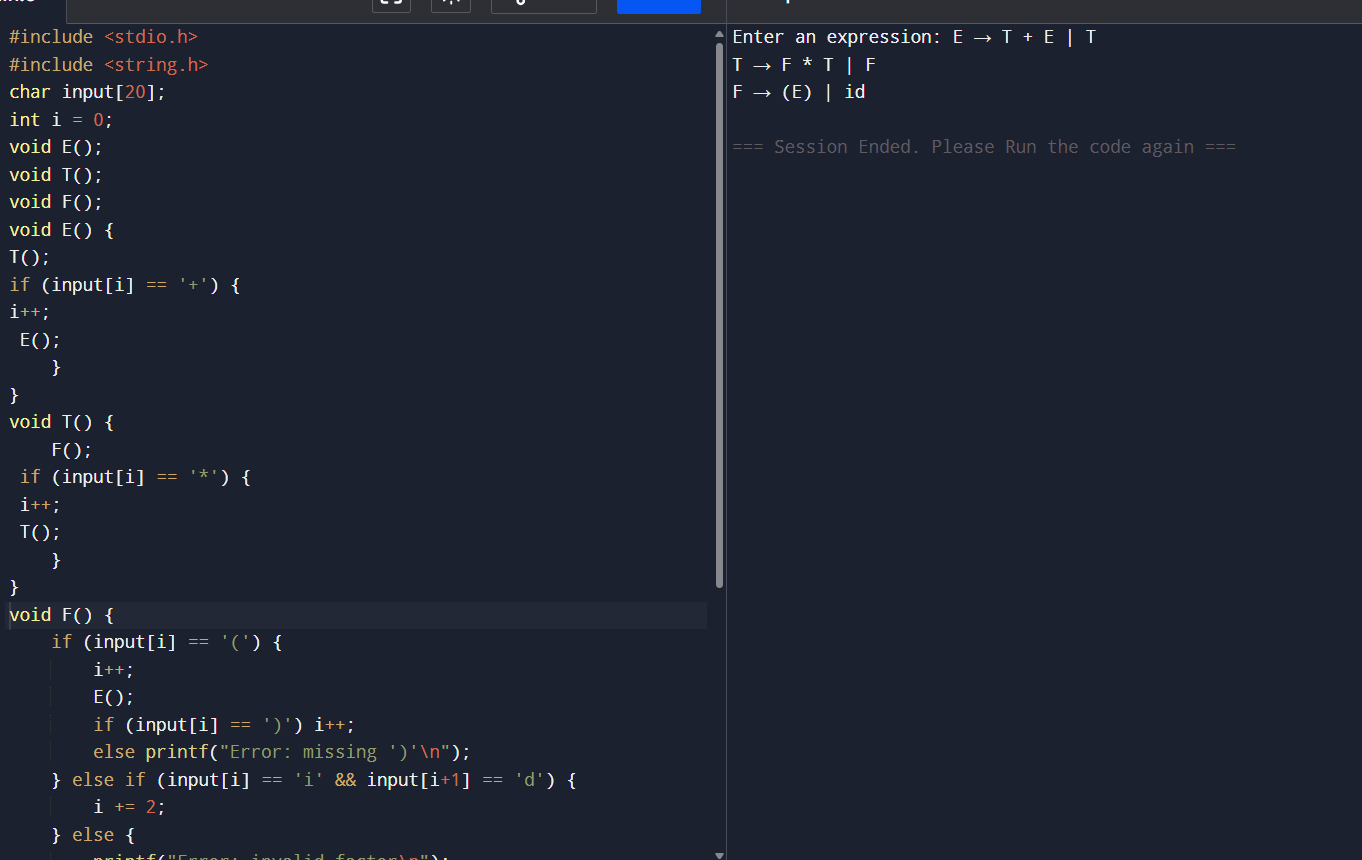
if (input[i] == '\0')

printf("String accepted.\n");

else

printf("String rejected.\n");

return 0;

}

11. In a class of Grade 3,Mathematics  
Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A  
smart kid of the class Kishore of the class says it is Parentheses,  
Exponentiation, Multiplication, Division, Addition, Subtraction. Can you write  
a C Program to help the students to understand about the operator  
precedence parsing for an expression containing more than one operator, the  
order of evaluation depends on the order of operations.

#include <stdio.h>

#include <ctype.h>

char stack[50];

int top = -1;

void push(char c) { stack[++top] = c; }

char pop() { return stack[top--]; }

int precedence(char c) {

if (c == '^') return 3;

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

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

return 0;

}

int main() {

char exp[50], output[50];

int i, k = 0;

printf("Enter infix expression: ");

scanf("%s", exp);

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

if (isalnum(exp[i])) output[k++] = exp[i];

else if (exp[i] == '(') push(exp[i]);

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

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

output[k++] = pop();

top--; // remove '('

} else {

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

output[k++] = pop();

push(exp[i]);

}

}

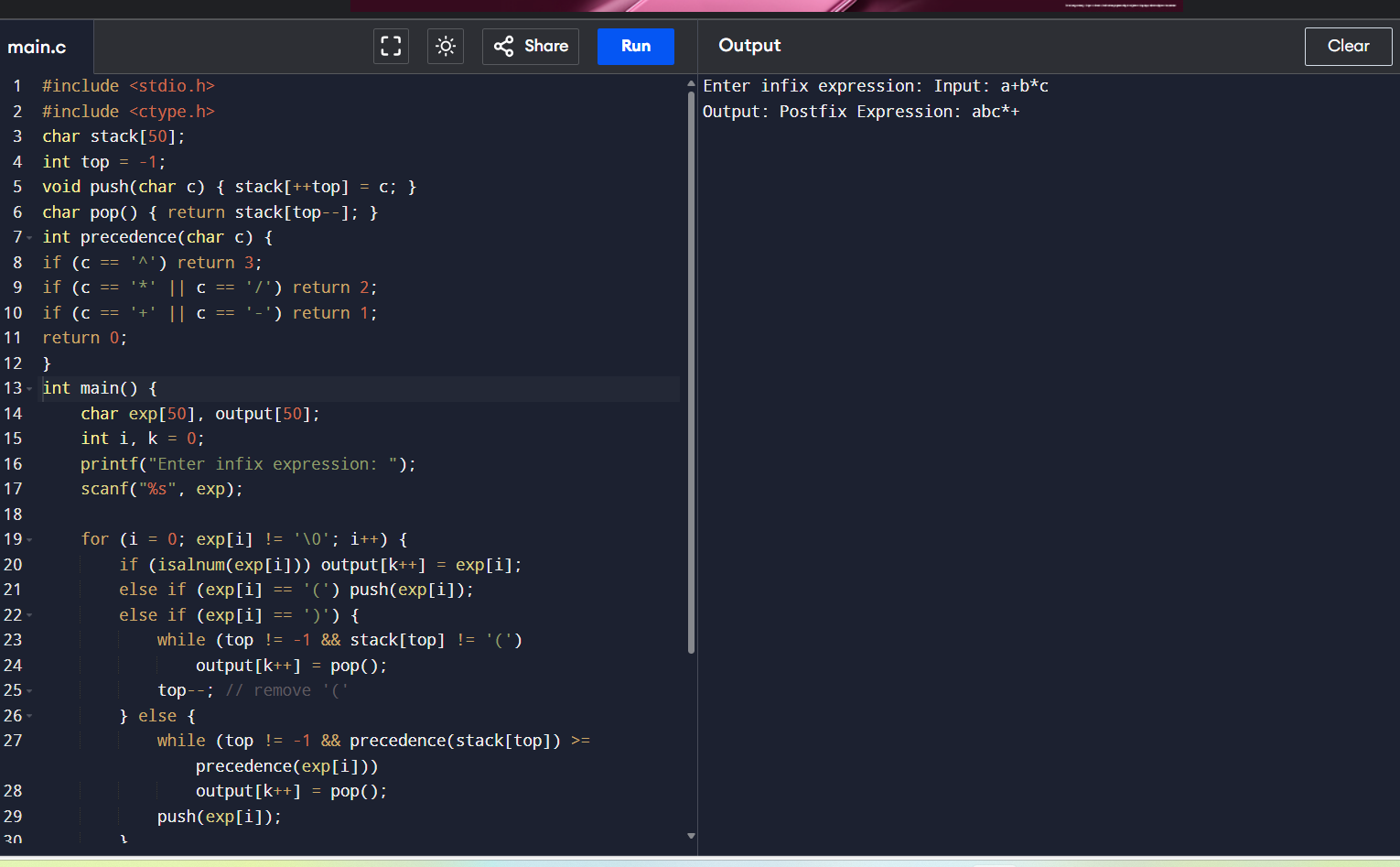
while (top != -1)

output[k++] = pop();

output[k] = '\0';

printf("Postfix Expression: %s\n", output);

return 0;

}

12. The main function of the  
Intermediate code generation is producing three address code statements for a  
given input expression. The three address codes help in determining the  
sequence in which operations are actioned by the compiler. The key work of  
Intermediate code generators is to simplify the process of Code Generator.  
Write a C Program to Generate the Three address code representation for the  
given input statement.

#include <stdio.h>

#include <string.h>

int main() {

char expr[20], temp = '1';

char op1, op2, op;

printf("Enter expression (like a=b+c\*d): ");

scanf("%s", expr);

int i = 2; // skip 'a='

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

if (expr[i] == '+' || expr[i] == '-' || expr[i] == '\*' || expr[i] == '/') {

op1 = expr[i - 1];

op = expr[i];

op2 = expr[i + 1];

printf("t%c = %c %c %c\n", temp, op1, op, op2);

expr[i - 1] = 't';

expr[i] = temp++;

}

i++;

}

printf("%c = t%c\n", expr[0], temp - 1);

return 0;

}