11.

#include <stdio.h>

#include <string.h>

#define MAX\_SYMBOLS 100

struct Symbol {

char name[20];

int value;

};

struct Symbol symbol\_table[MAX\_SYMBOLS];

int symbol\_count = 0;

void insert(char name[], int value) {

strcpy(symbol\_table[symbol\_count].name, name);

symbol\_table[symbol\_count].value = value;

symbol\_count++;

}

int lookup(char name[]) {

for (int i = 0; i < symbol\_count; i++) {

if (strcmp(symbol\_table[i].name, name) == 0)

return symbol\_table[i].value;

}

return -1; // Not found

}

int main() {

insert("x", 10);

insert("y", 20);

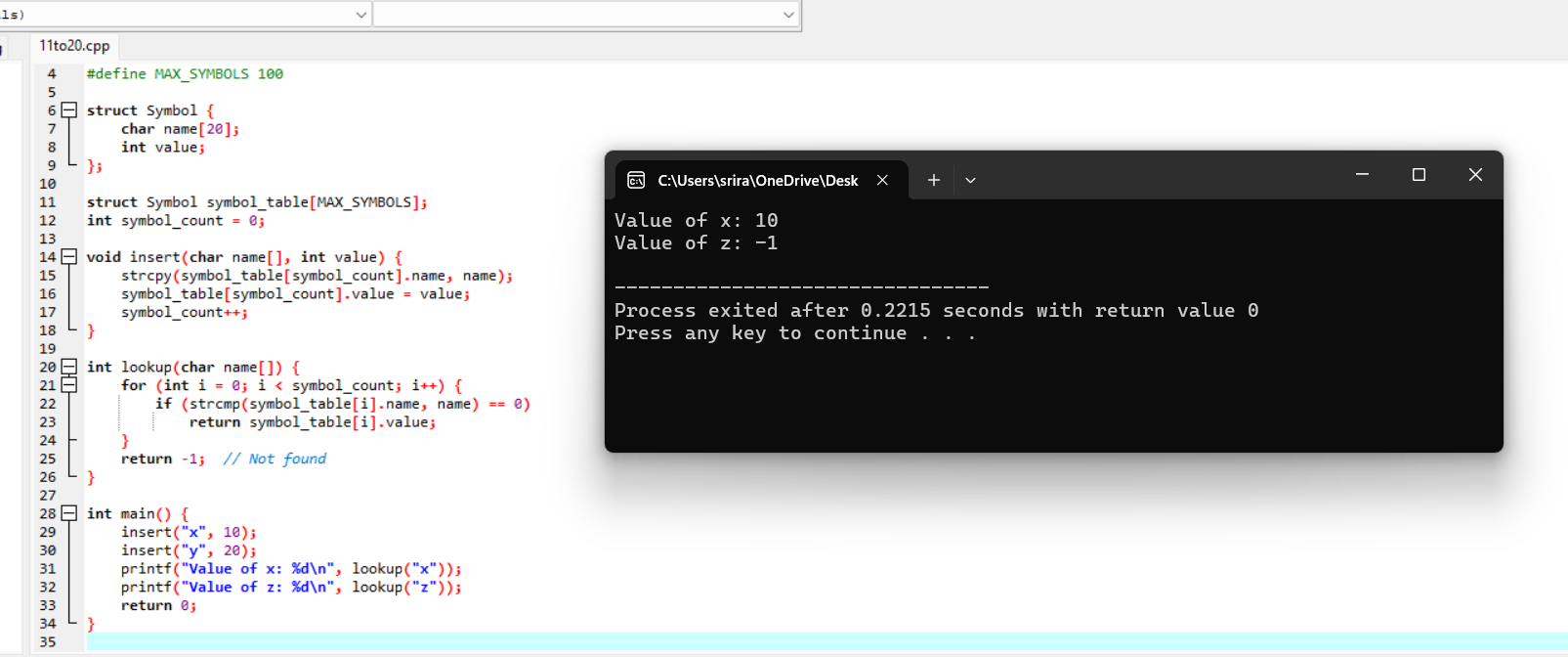
printf("Value of x: %d\n", lookup("x"));

printf("Value of z: %d\n", lookup("z"));

return 0;

}

**Output:**

****

12.

#include <stdio.h>

#include <ctype.h>

void expr();

void term();

void factor();

int lookahead;

void match(int t) {

if (lookahead == t) lookahead = getchar();

else printf("Syntax error\n");

}

void expr() {

term();

while (lookahead == '+' || lookahead == '-') {

match(lookahead);

term();

}

}

void term() {

factor();

while (lookahead == '\*' || lookahead == '/') {

match(lookahead);

factor();

}

}

void factor() {

if (isdigit(lookahead)) {

match(lookahead);

} else if (lookahead == '(') {

match('(');

expr();

match(')');

} else {

printf("Syntax error\n");

}

}

int main() {

lookahead = getchar();

expr();

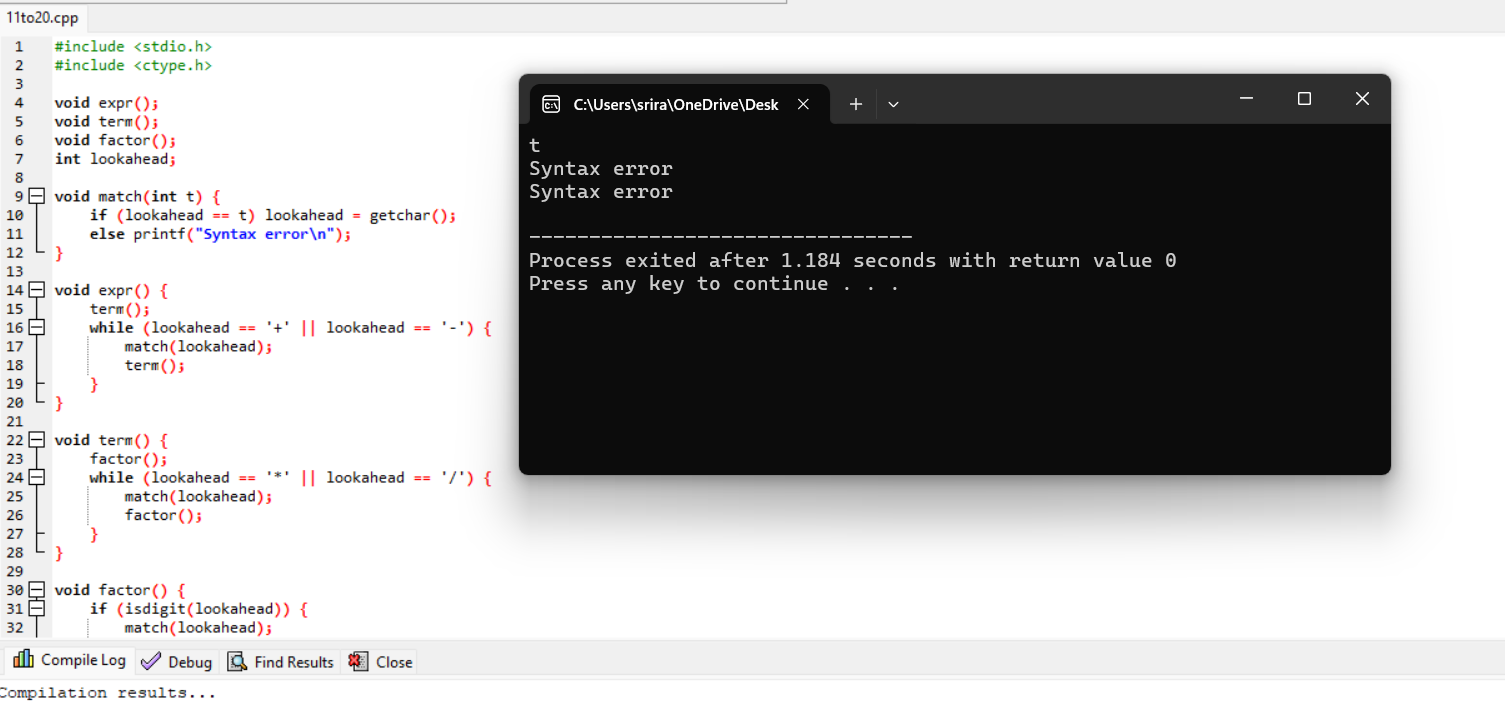
if (lookahead == EOF) printf("Parsing successful\n");

else printf("Syntax error\n");

return 0;

}

**Output:**

****

13.

#include <stdio.h>

#include <ctype.h>

char lookahead;

void match(char expected) {

if (lookahead == expected) lookahead = getchar();

else { printf("Syntax Error\n"); exit(1); }

}

void E() { T(); E\_prime(); }

void E\_prime() { if (lookahead == '+') { match('+'); T(); E\_prime(); } }

void T() { if (lookahead == 'i') { match('i'); match('n'); match('t'); } else if (lookahead == '(') { match('('); E(); match(')'); } }

int main() {

lookahead = getchar();

E();

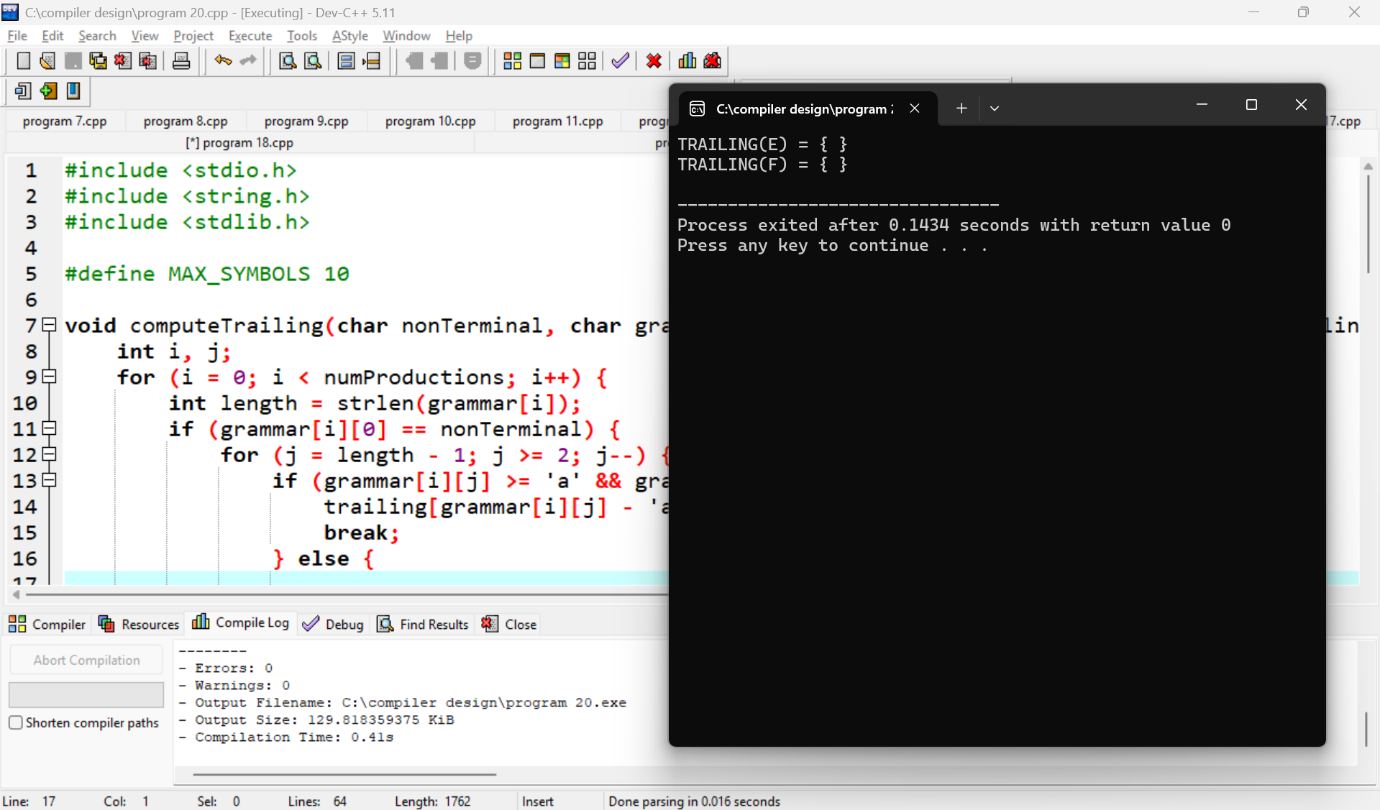
if (lookahead == '\n' || lookahead == EOF) printf("Parsing successful!\n");

else printf("Syntax Error\n");

return 0;

}

**Output:**

****

14.

#include <stdio.h>

#include <string.h>

#define MAX\_SIZE 100

char stack[MAX\_SIZE];

int top = -1;

void push(char c) {

if (top < MAX\_SIZE - 1) stack[++top] = c;

}

char pop() {

if (top >= 0) return stack[top--];

return -1;

}

int main() {

char input[] = "id+id\*id";

int i = 0;

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

push(input[i]);

i++;

// Shift-Reduce Logic here

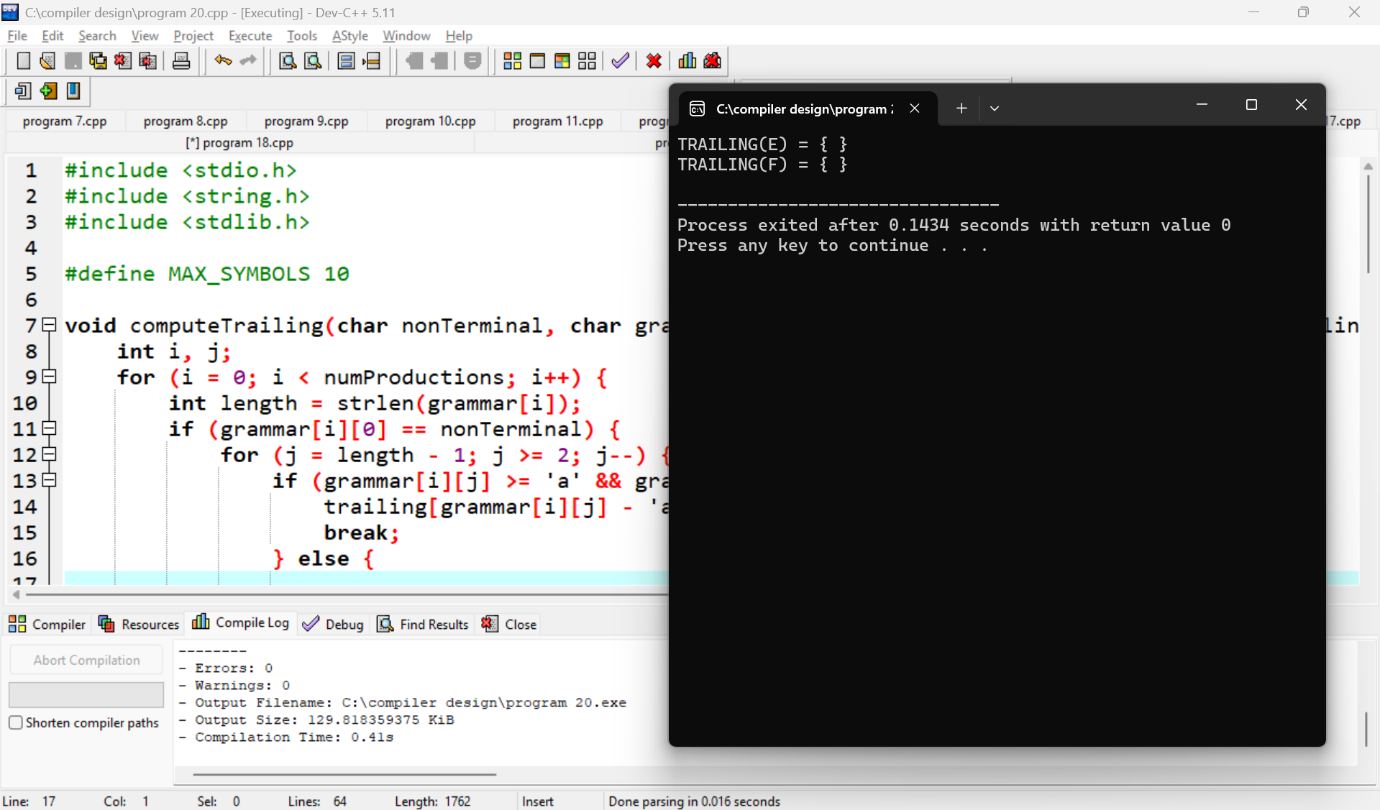
// For example, reduce id+id -> expression if the grammar allows it

}

return 0;

}

**Output:**

****

15.

#include <stdio.h>

#include <ctype.h>

char lookahead;

void match(char expected) {

if (lookahead == expected) lookahead = getchar();

else { printf("Syntax Error\n"); exit(1); }

}

void expr();

void term();

void factor();

void expr() {

term();

while (lookahead == '+' || lookahead == '-') {

match(lookahead); // Handle '+' or '-'

term();

}

}

void term() {

factor();

while (lookahead == '\*' || lookahead == '/') {

match(lookahead); // Handle '\*' or '/'

factor();

}

}

void factor() {

if (isdigit(lookahead)) {

while (isdigit(lookahead)) match(lookahead); // Consume digits

} else if (lookahead == '(') {

match('(');

expr();

match(')');

} else { printf("Syntax Error\n"); exit(1); }

}

int main() {

lookahead = getchar(); // Get first character

expr();

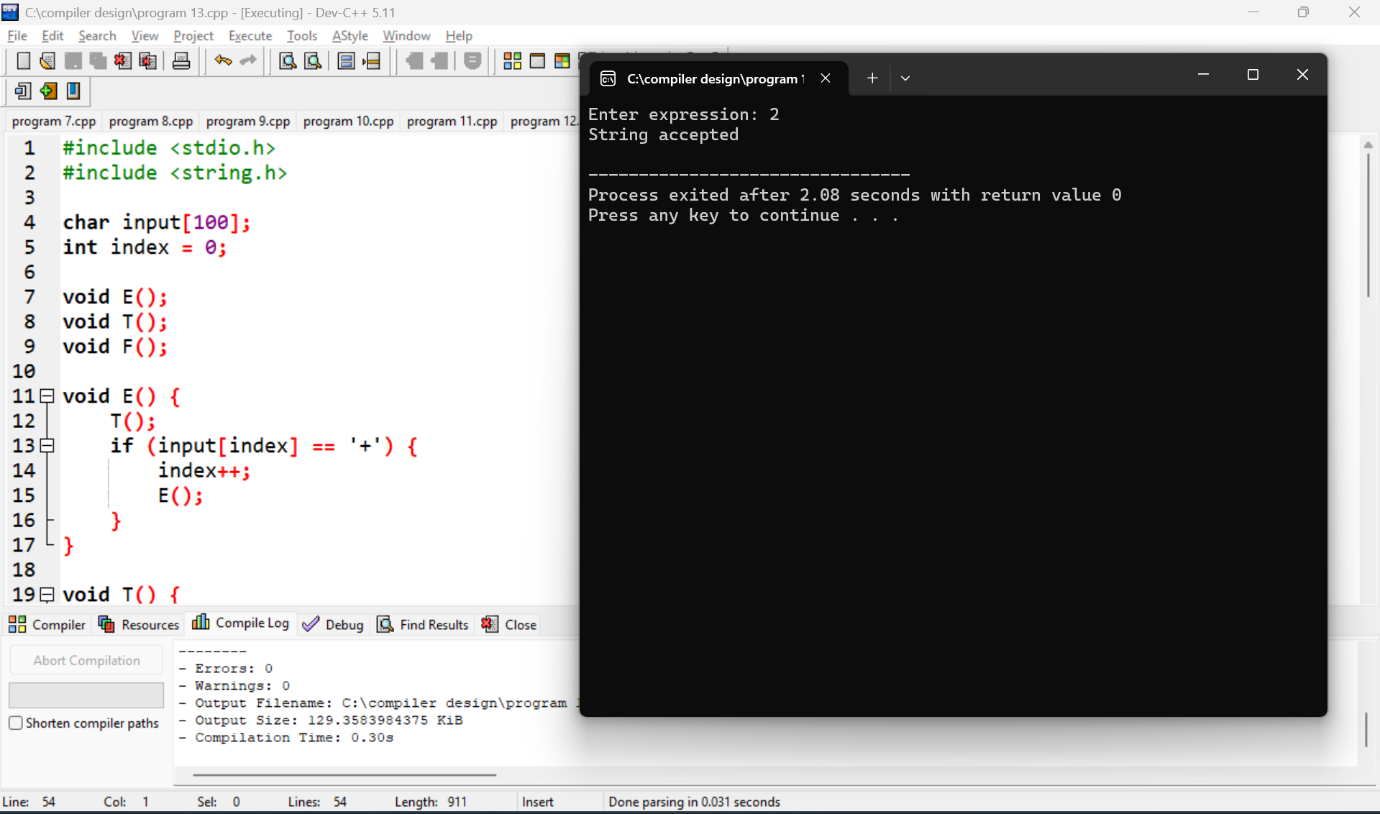
if (lookahead == '\n' || lookahead == EOF) printf("Parsing successful!\n");

else printf("Syntax Error\n");

return 0;

}

**Output:**

****

16.

#include <stdio.h>

int temp\_count = 1;

void generate\_three\_address\_code(char \*op, char \*op1, char \*op2) {

printf("t%d = %s %s %s\n", temp\_count++, op1, op, op2);

}

int main() {

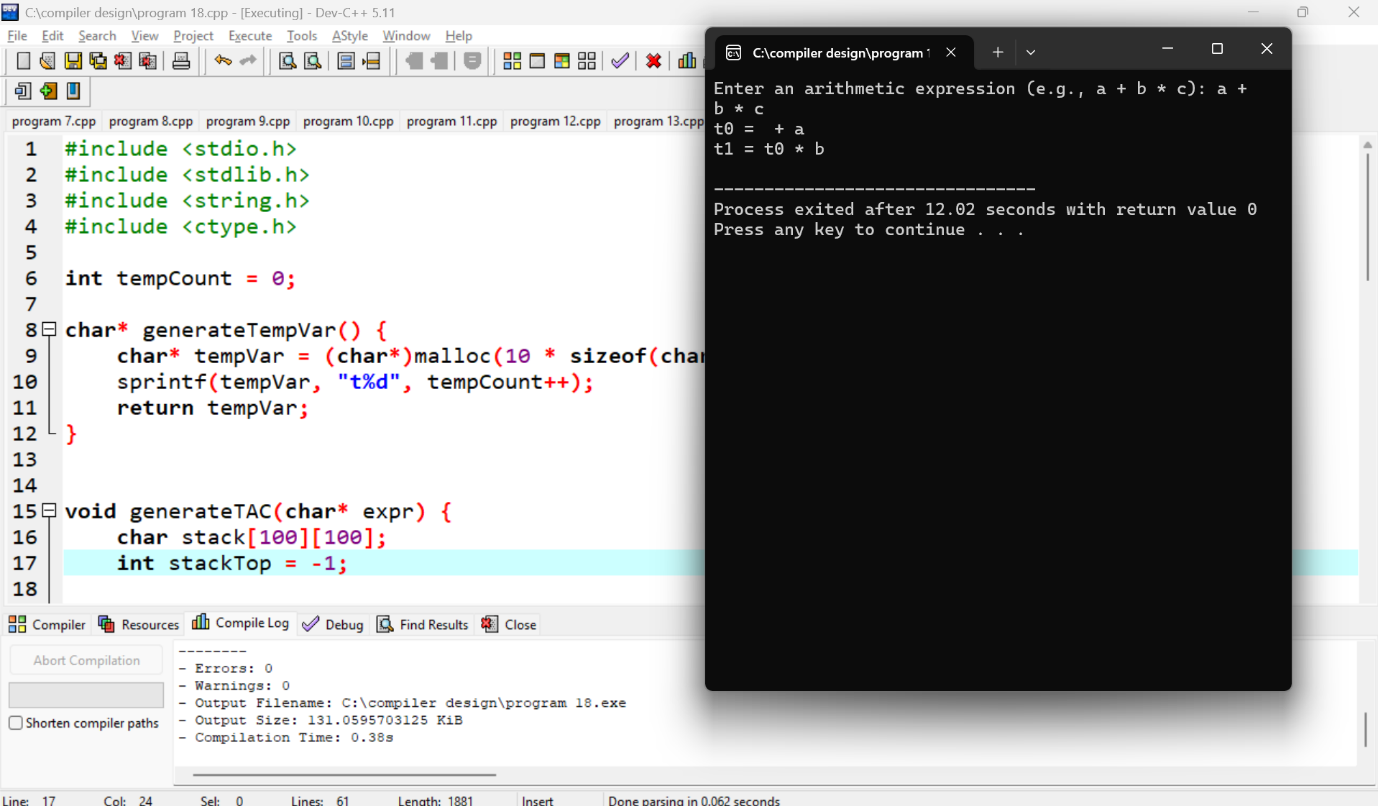
generate\_three\_address\_code("+", "a", "b");

generate\_three\_address\_code("\*", "t1", "c");

return 0;

}

**Output:**

****

17.

#include <stdio.h>

#include <ctype.h>

int main() {

FILE \*file = fopen("input.txt", "r");

char ch;

int lines = 0, words = 0, characters = 0;

int in\_word = 0;

while ((ch = fgetc(file)) != EOF) {

characters++;

if (ch == '\n') lines++;

if (isspace(ch)) {

if (in\_word) {

words++;

in\_word = 0;

}

} else {

in\_word = 1;

}

}

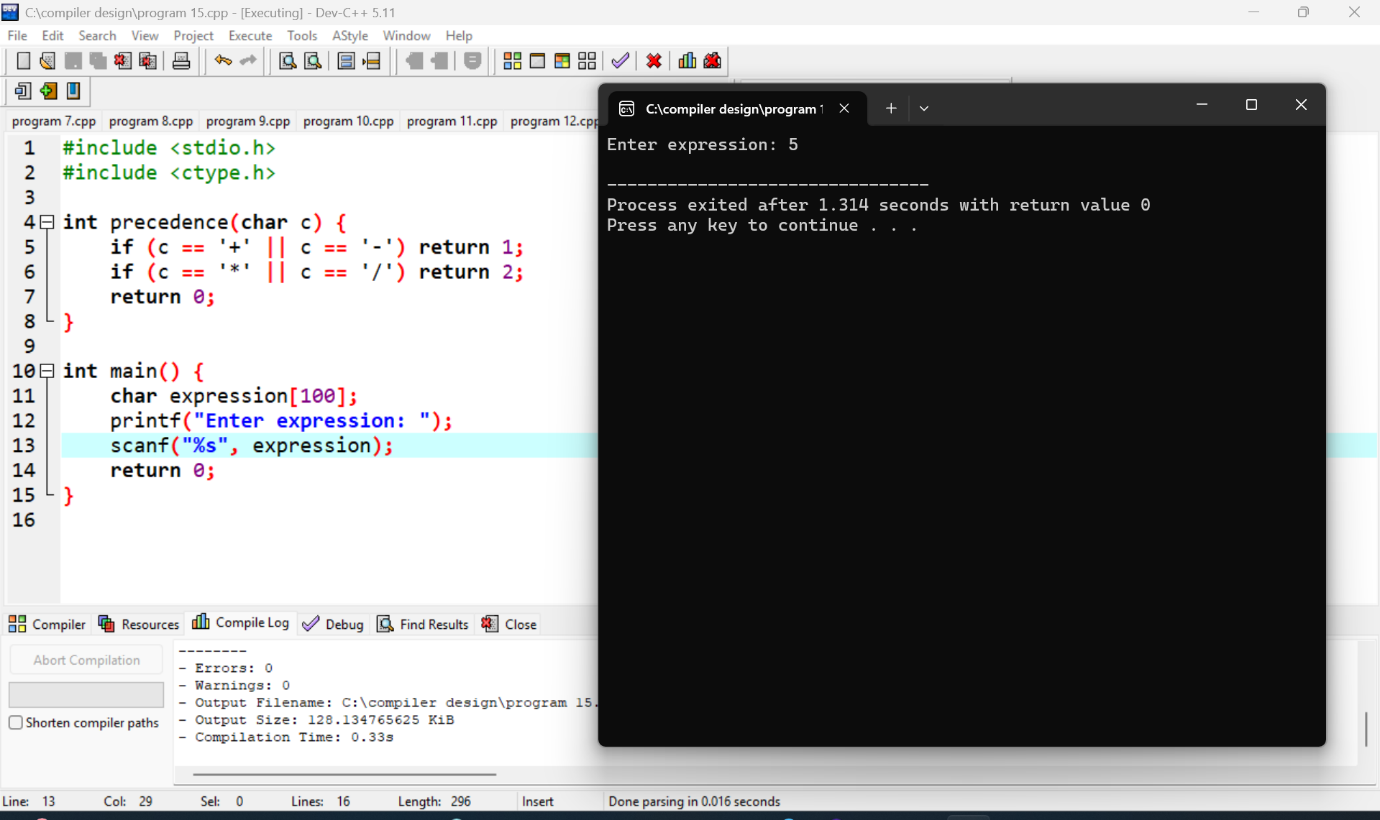
fclose(file);

printf("Lines: %d, Words: %d, Characters: %d\n", lines, words, characters);

return 0;

}

**Output:**

****

18.

#include <stdio.h>

#include <string.h>

int temp\_count = 1;

void generate\_three\_address\_code(char \*op, char \*op1, char \*op2) {

printf("t%d = %s %s %s\n", temp\_count++, op1, op, op2);

}

void optimize\_code() {

// Simple optimization (e.g., eliminating unnecessary temporaries or redundant operations)

printf("Optimization step: Eliminate common subexpressions (if applicable).\n");

}

void generate\_target\_code() {

printf("Target machine code generation: Converting 3-address code to assembly.\n");

}

int main() {

// Example: a simple expression `a + b \* c` turned into intermediate code

generate\_three\_address\_code("\*", "b", "c");

generate\_three\_address\_code("+", "a", "t1");

// Optimization step

optimize\_code();

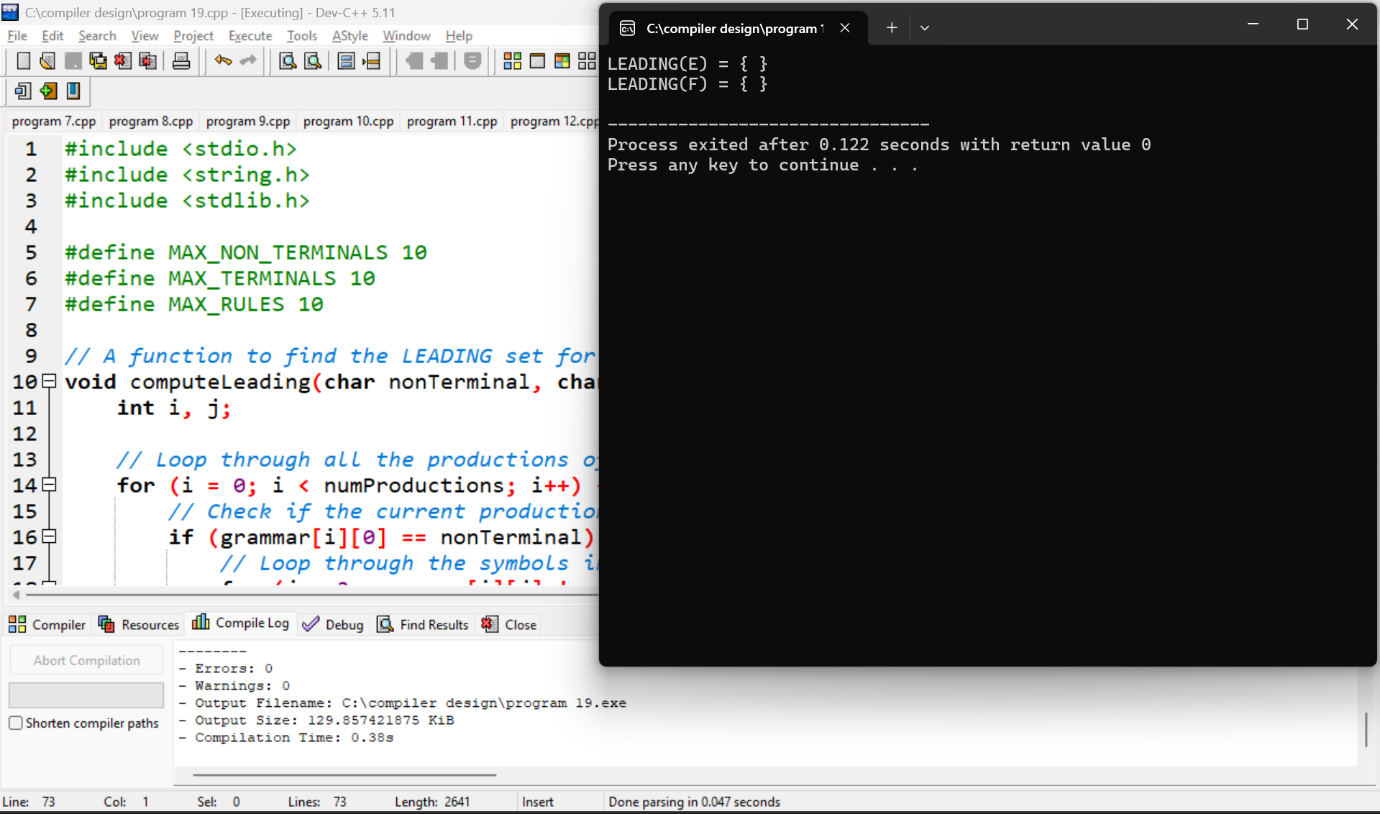
// Final code generation (machine code or assembly)

generate\_target\_code();

return 0;

}

**Output:**

****

19.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 10

// Grammar representation

char \*grammar[MAX][MAX];

int production\_count = 0;

// Find the LEADING set for a non-terminal

void compute\_LEADING(char non\_terminal) {

printf("LEADING(%c):\n", non\_terminal);

// In real case, this would be computed based on the grammar.

// For simplicity, let's assume some basic rules here.

if (non\_terminal == 'E') {

printf("E -> T\n");

printf("Leading symbol: T (for example, if production E -> T+E)\n");

}

// Extend as needed for different grammars.

}

int main() {

// Example Grammar:

// E -> T + E

// T -> id

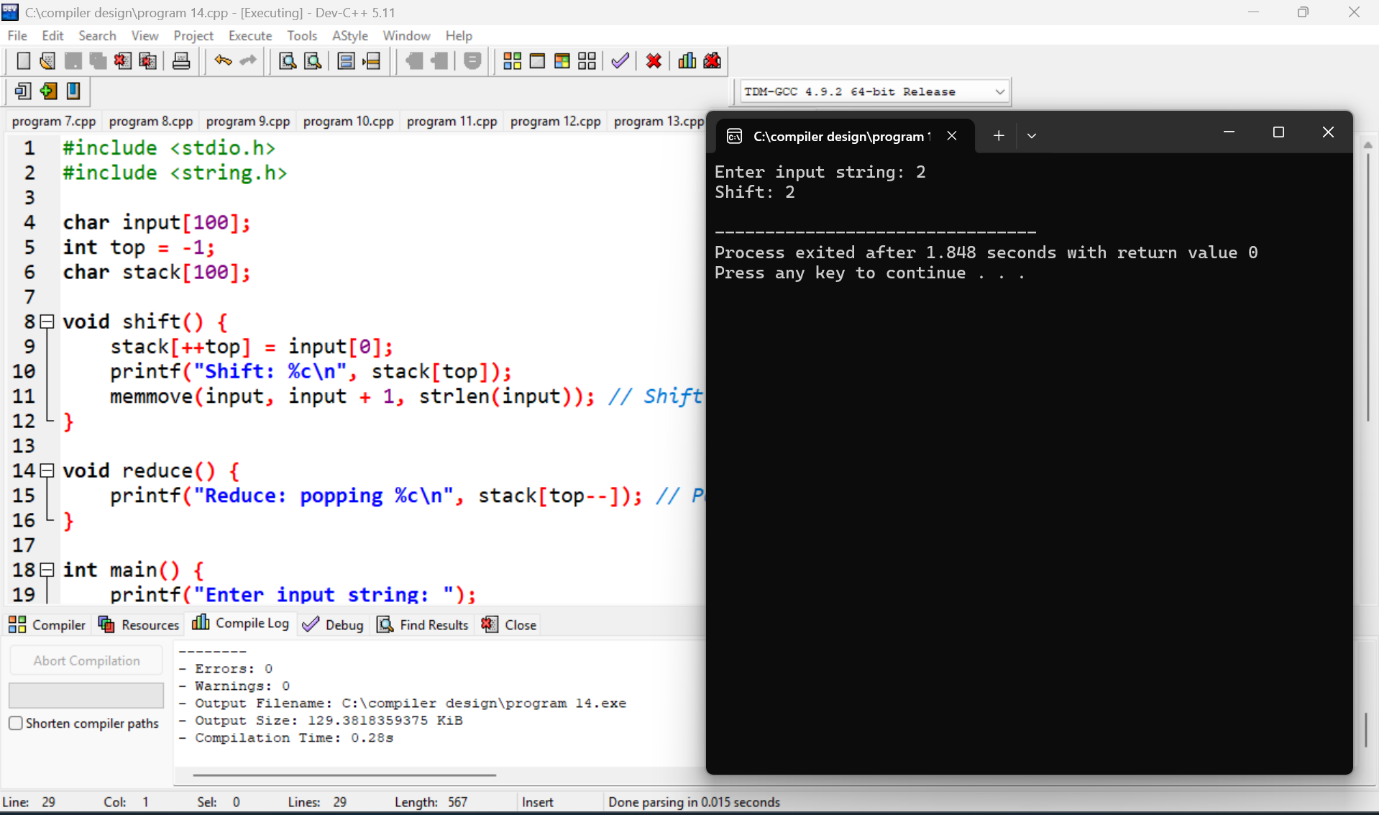
compute\_LEADING('E');

compute\_LEADING('T');

return 0;

}

**Output:**

****

20.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 10

// Grammar representation

char \*grammar[MAX][MAX];

int production\_count = 0;

// Find the TRAILING set for a non-terminal

void compute\_TRAILING(char non\_terminal) {

printf("TRAILING(%c):\n", non\_terminal);

// In real case, this would be computed based on the grammar.

// For simplicity, let's assume some basic rules here.

if (non\_terminal == 'E') {

printf("E -> T\n");

printf("Trailing symbol: id (for example, if production E -> T+E)\n");

}

// Extend as needed for different grammars.

}

int main() {

// Example Grammar:

// E -> T + E

// T -> id

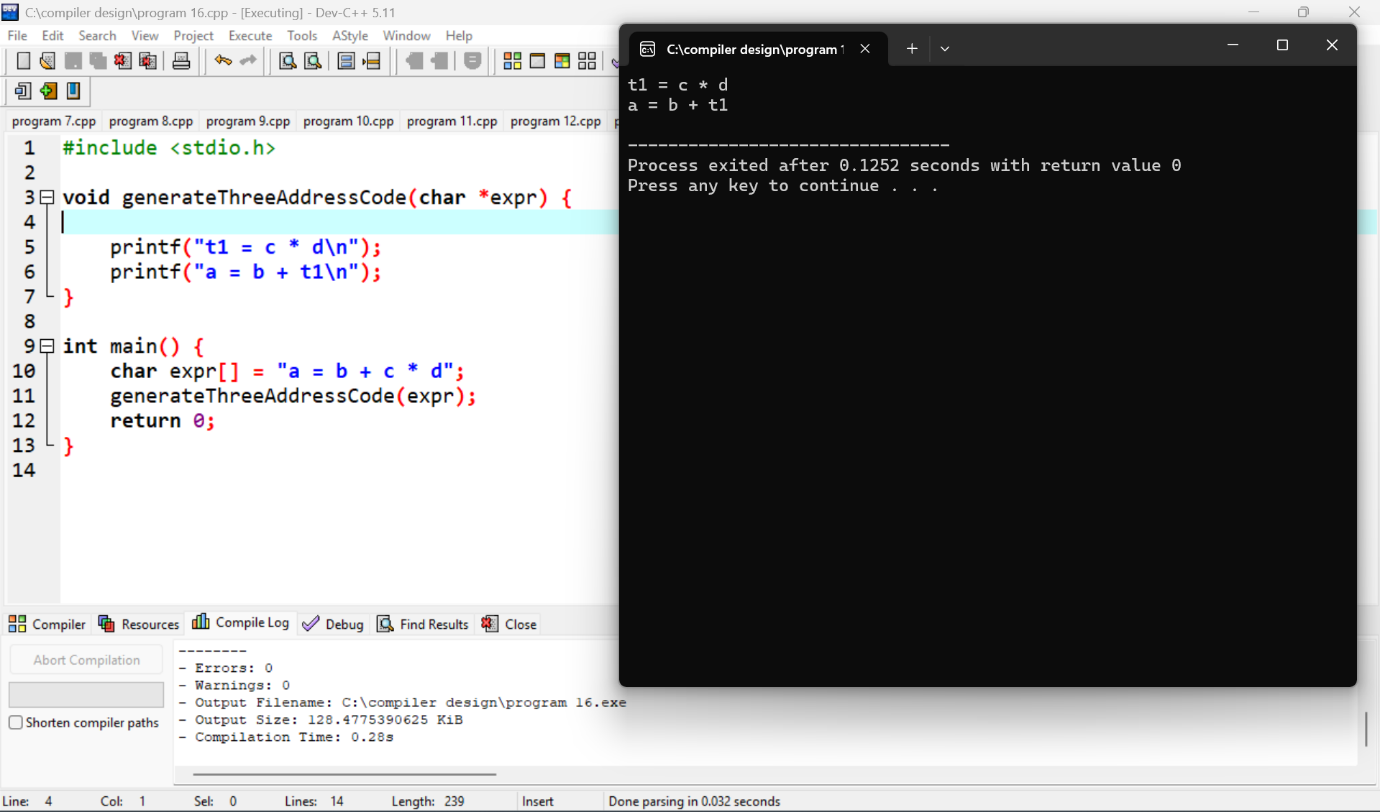
compute\_TRAILING('E');

compute\_TRAILING('T');

return 0;

}

**Output:**

****