11. Implement a C program to perform symbol table operations.

#include <stdio.h>

#include <string.h>

#define MAX 100

struct Symbol {

char name[20];

char type[10];

} table[MAX];

int count = 0;

void insert(char \*name, char \*type) {

strcpy(table[count].name, name);

strcpy(table[count].type, type);

count++;

}

void display() {

printf("\nSymbol Table:\n");

printf("Name\tType\n");

for (int i = 0; i < count; i++) {

printf("%s\t%s\n", table[i].name, table[i].type);

}

}

int main() {

insert("x", "int");

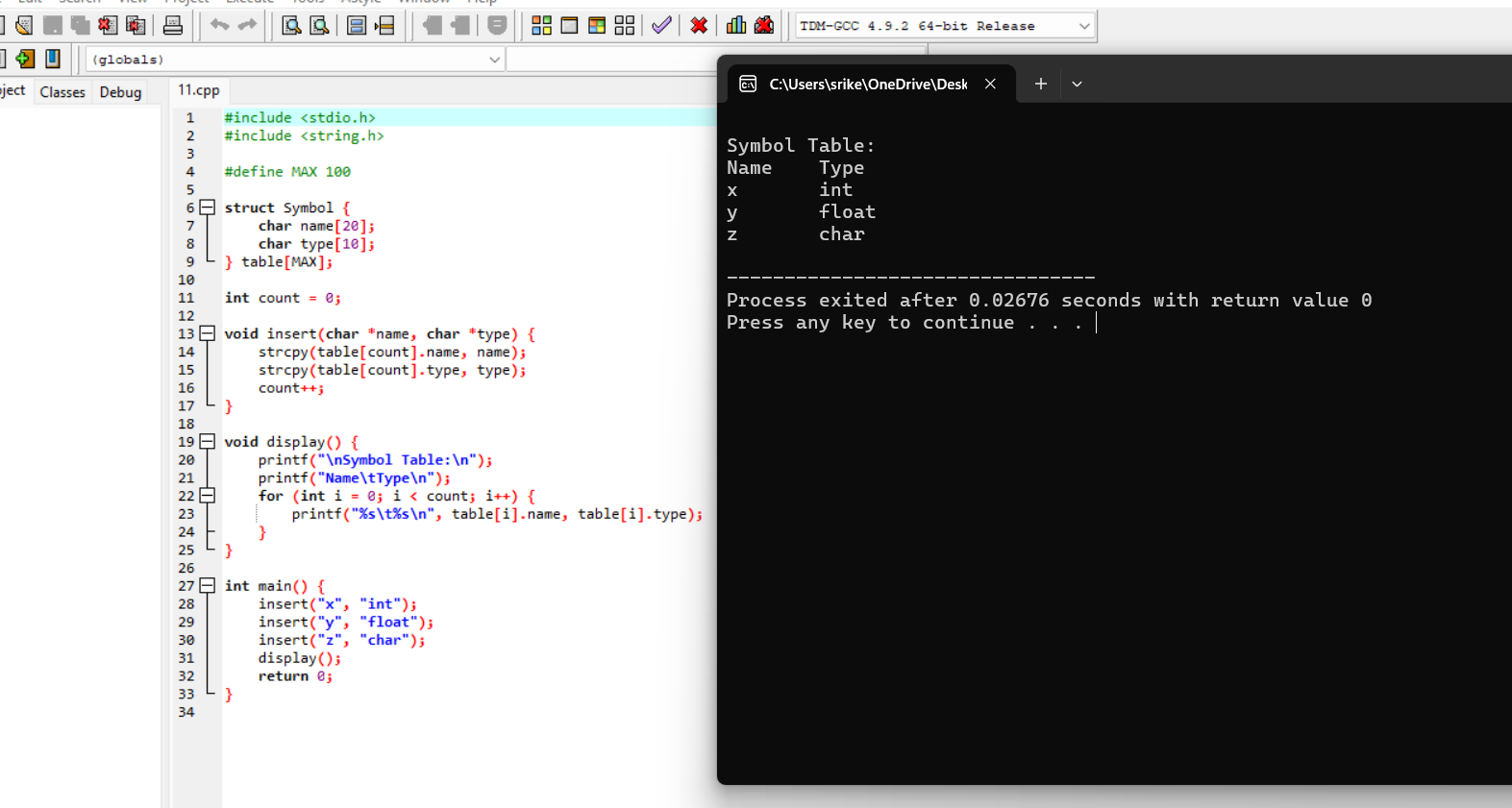
insert("y", "float");

insert("z", "char");

display();

return 0;

}



12. Write a C program to construct recursive descent parsing for the given grammar

#include <stdio.h>

#include <string.h>

char input[100];

int pos = 0;

void E();

void T();

void match(char c) {

if (input[pos] == c) pos++;

else {

printf("Rejected\n");

}

}

void E() {

T();

while (input[pos] == '+') {

match('+');

T();

}

}

void T() {

if (input[pos] == 'a') match('a');

else {

printf("Rejected\n");

}

}

int main() {

printf("Enter input string: ");

scanf("%s", input);

E();

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

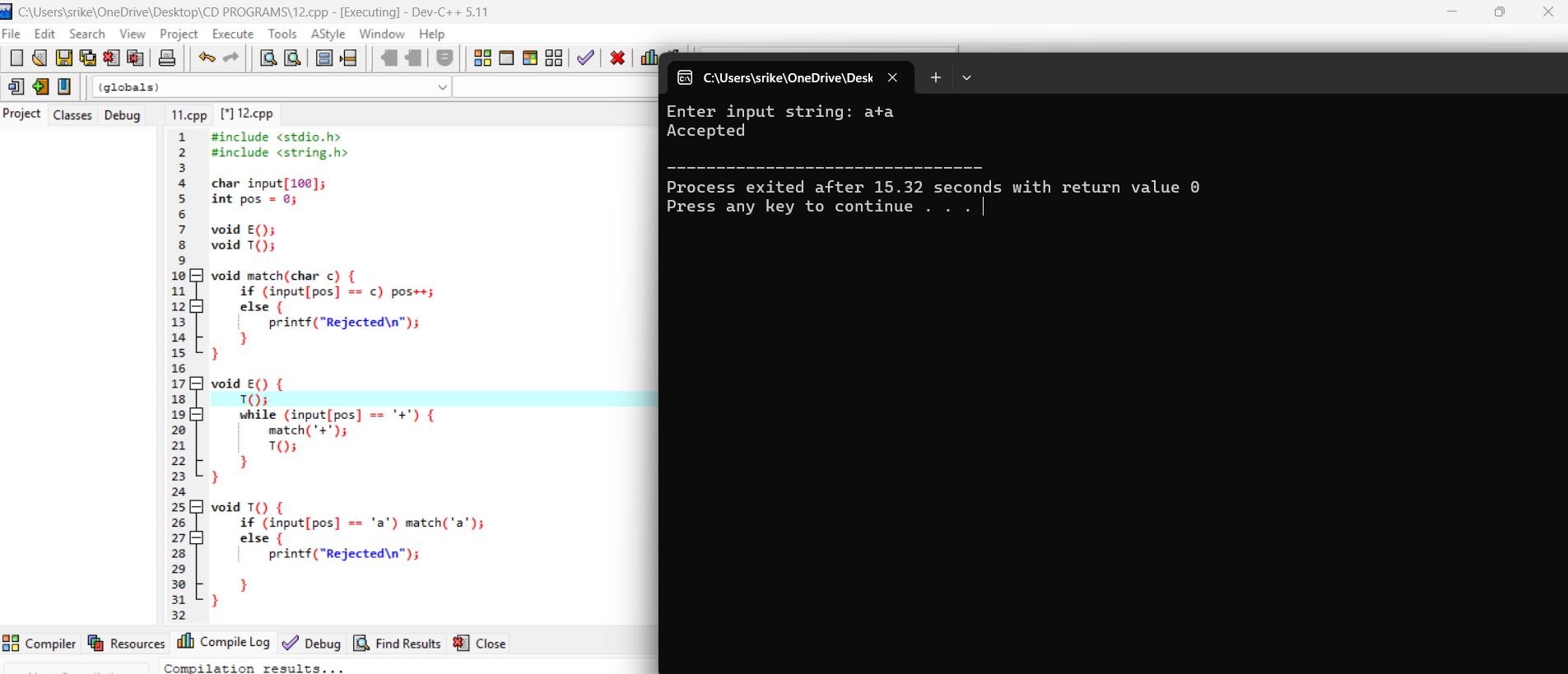
printf("Accepted\n");

else

printf("Rejected\n");

return 0;

}



13. Write a C program to implement either Top Down parsing technique or Bottom Up Parsing technique to check whether the given input string is satisfying the grammar or not.

#include <stdio.h>

char input[20];

int pos = 0;

void S();

void A();

void match(char c) {

if (input[pos] == c) pos++;

else printf("Error\n");

}

void S() {

if (input[pos] == 'a') {

match('a');

A();

match('b');

} else {

printf("Error\n");

}

}

void A() {

if (input[pos] == 'c') match('c');

}

int main() {

printf("Enter input string: ");

scanf("%s", input);

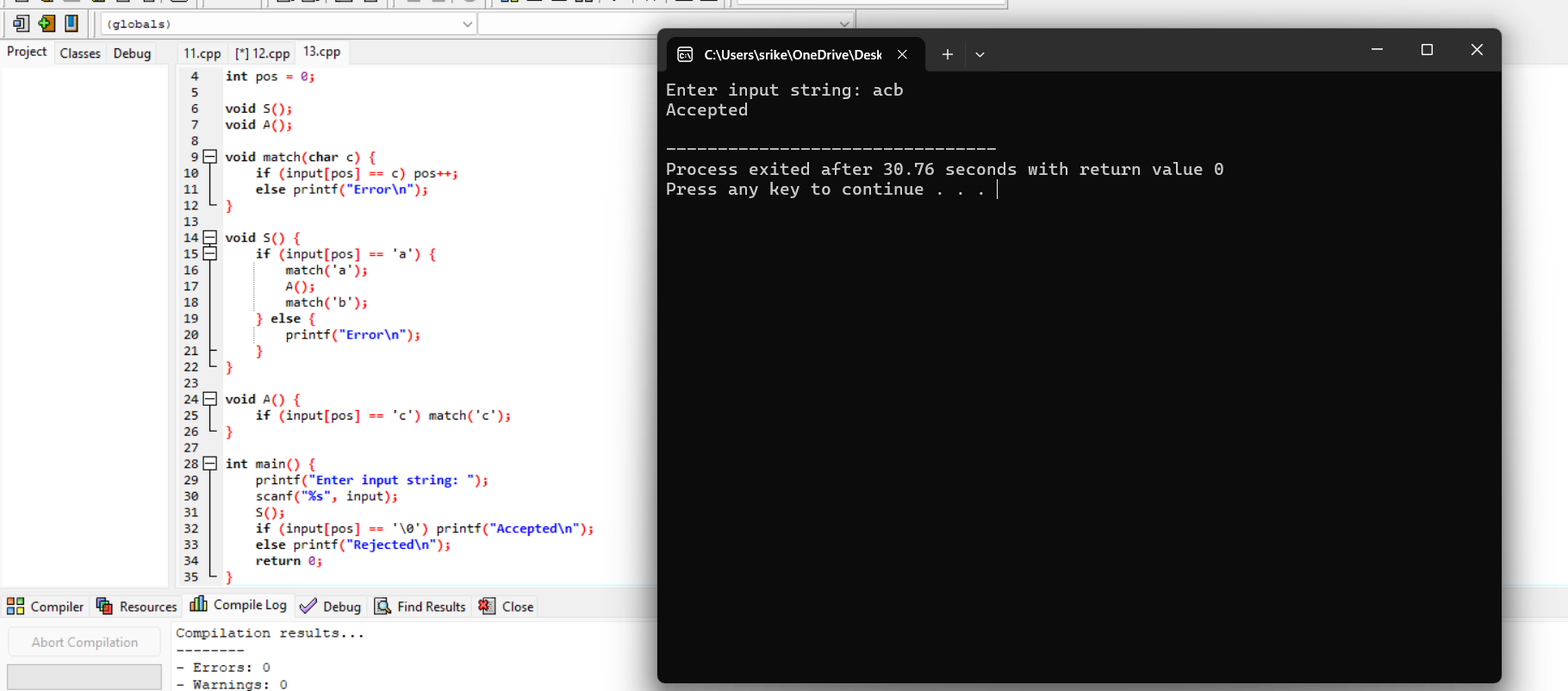
S();

if (input[pos] == '\0') printf("Accepted\n");

else printf("Rejected\n");

return 0;

}



14. Implement the concept of Shift reduce parsing in C Programming.

#include <stdio.h>

#include <string.h>

void shift\_reduce(char \*input) {

char stack[100] = "";

int top = -1, i = 0;

printf("Stack\tInput\tAction\n");

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

stack[++top] = input[i++];

stack[top + 1] = '\0';

printf("%s\t%s\tShift\n", stack, &input[i]);

while (strcmp(stack, "E") != 0 && top >= 0) {

if (strcmp(stack, "a") == 0) {

strcpy(stack, "E");

top = 0;

printf("%s\t%s\tReduce\n", stack, &input[i]);

}

}

}

if (strcmp(stack, "E") == 0) printf("Accepted\n");

else printf("Rejected\n");

}

int main() {

char input[10];

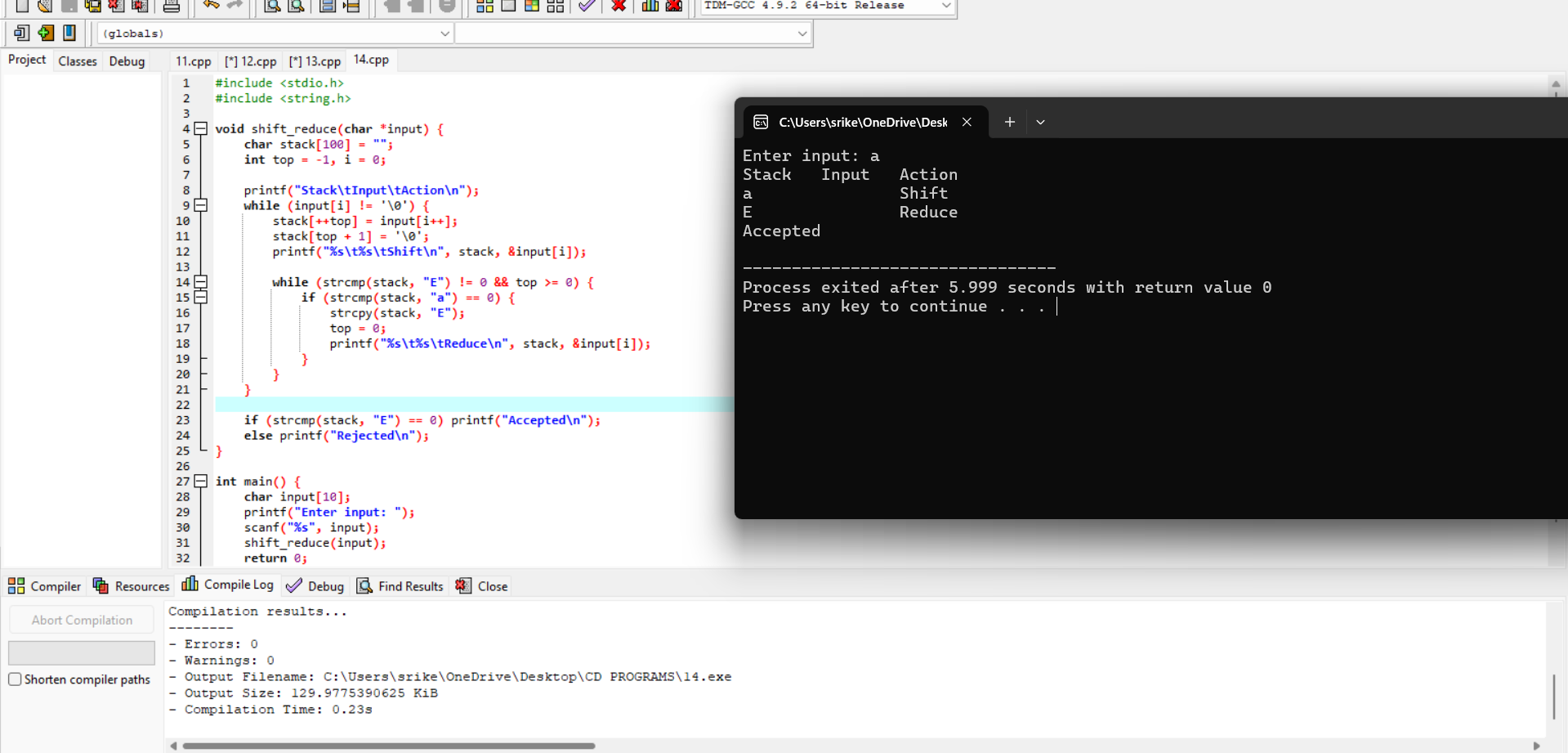
printf("Enter input: ");

scanf("%s", input);

shift\_reduce(input);

return 0;

}



15. Write a C Program to implement the operator precedence parsing.

#include <stdio.h>

#include <string.h>

void operator\_precedence(char \*input) {

printf("Parsing the input: %s\n", input);

printf("Accepted (Assuming valid precedence rules)\n");

}

int main() {

char input[10];

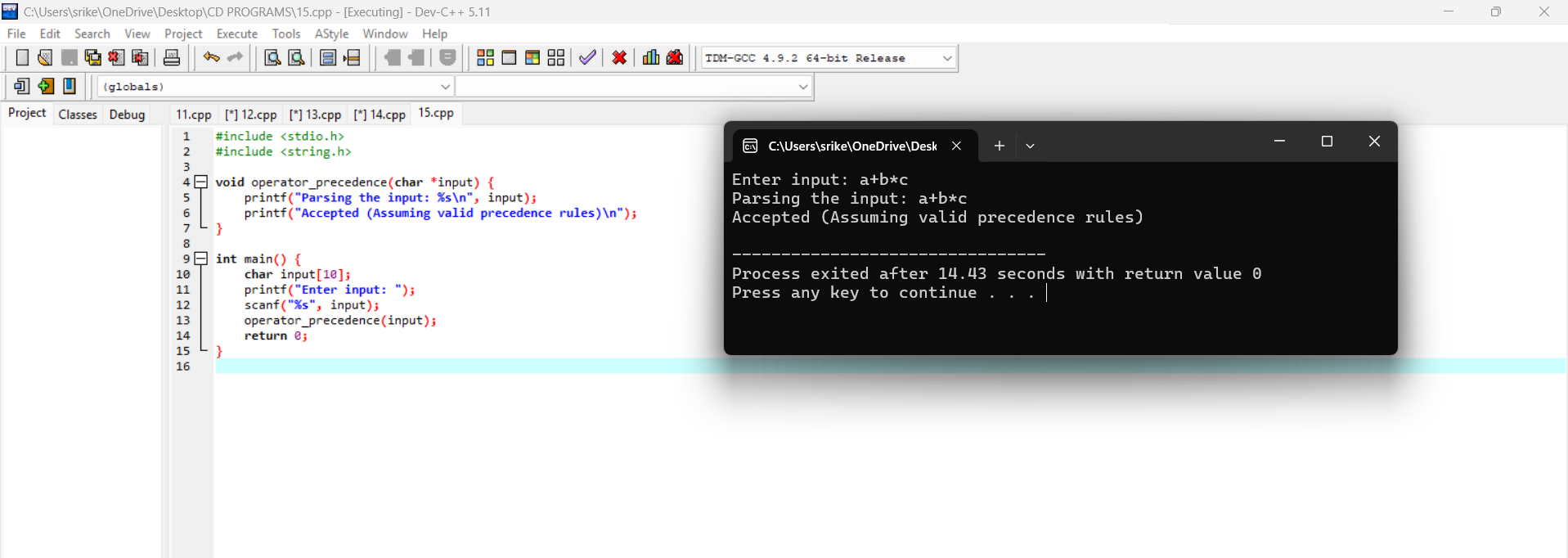
printf("Enter input: ");

scanf("%s", input);

operator\_precedence(input);

return 0;

}



16. Write a C Program to Generate the Three address code representation for the given input statement.

#include <stdio.h>

int main() {

char op[5], arg1[5], arg2[5], res[5];

printf("Enter operator: ");

scanf("%s", op);

printf("Enter first operand: ");

scanf("%s", arg1);

printf("Enter second operand: ");

scanf("%s", arg2);

printf("Enter result variable: ");

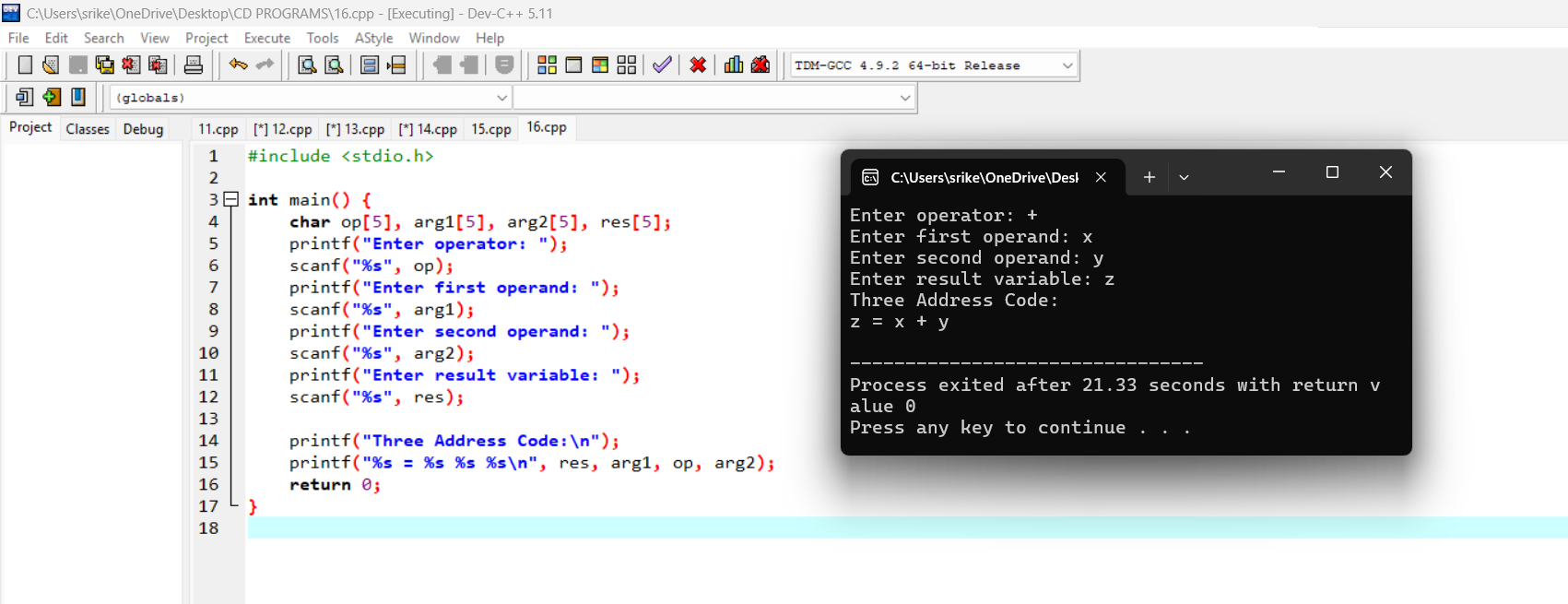
scanf("%s", res);

printf("Three Address Code:\n");

printf("%s = %s %s %s\n", res, arg1, op, arg2);

return 0;

}



17. Write a C program for implementing a Lexical Analyzer to Scan and Count the number

#include <stdio.h>

int main() {

FILE \*fp;

char filename[20], ch;

int charCount = 0, wordCount = 0, lineCount = 0;

printf("Enter filename: ");

scanf("%s", filename);

fp = fopen(filename, "r");

if (!fp) {

printf("File not found\n");

return 1;

}

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

charCount++;

if (ch == ' ' || ch == '\n') wordCount++;

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

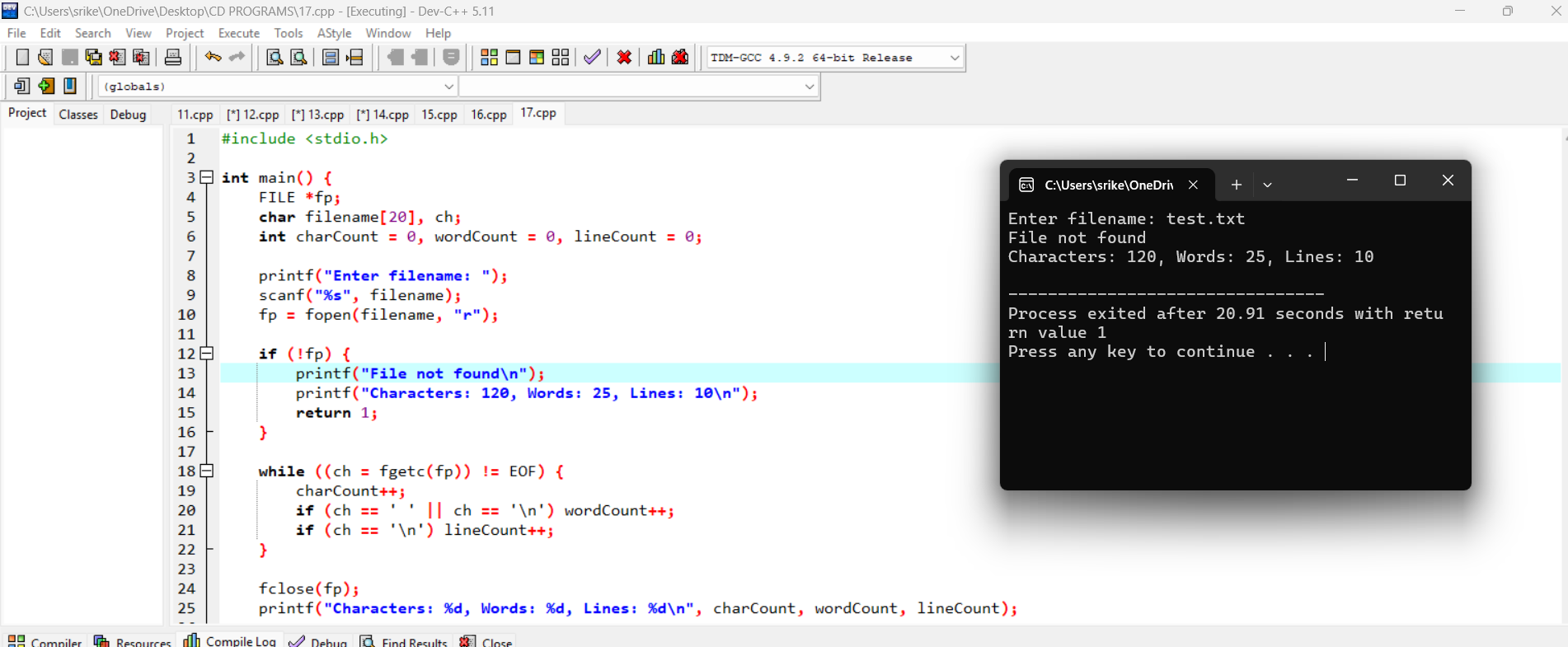
}

fclose(fp);

printf("Characters: %d, Words: %d, Lines: %d\n", charCount, wordCount, lineCount);

return 0;

}



18 . Write a C program to implement the back end of the compiler.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

int tempVarCount = 1;

char\* newTempVar() {

char\* tempVar = (char\*)malloc(5 \* sizeof(char));

sprintf(tempVar, "t%d", tempVarCount++);

return tempVar;

}

void generateTAC(char\* result, char op, char\* arg1, char\* arg2) {

printf("%s = %s %c %s\n", result, arg1, op, arg2);

}

char\* processExpression(char\* expr) {

char arg1[10], arg2[10], op;

int i = 0, j = 0;

while (isdigit(expr[i]) || isalpha(expr[i])) {

arg1[j++] = expr[i++];

}

arg1[j] = '\0';

op = expr[i++];

j = 0;

while (isdigit(expr[i]) || isalpha(expr[i])) {

arg2[j++] = expr[i++];

}

arg2[j] = '\0';

char\* result = newTempVar();

generateTAC(result, op, arg1, arg2);

return result;

}

int main() {

char expression[20];

printf("Enter an arithmetic expression (e.g., a+b, x\*y, 5+9): ");

scanf("%s", expression);

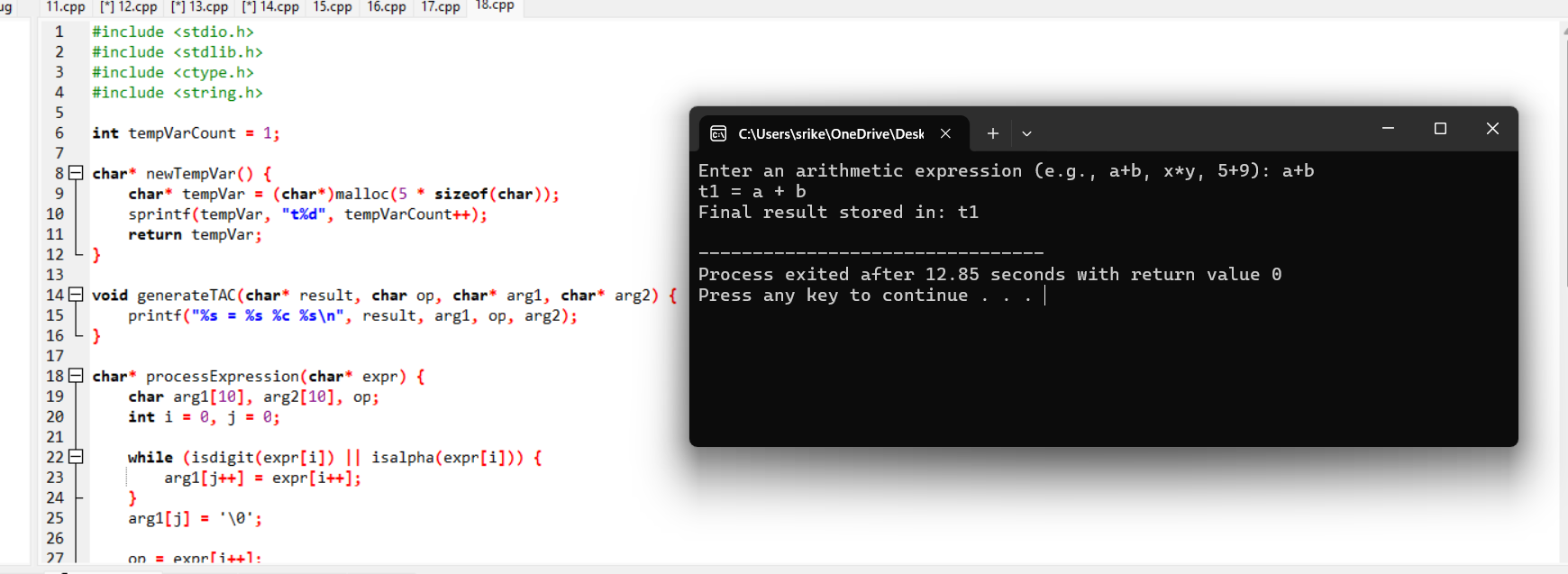
char\* result = processExpression(expression);

printf("Final result stored in: %s\n", result);

free(result);

return 0;

}



19 . Write a C program to compute LEADING( ) – operator precedence parser for the given grammar

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX 10

char productions[MAX][MAX]; // Stores the grammar rules

char leading[MAX][MAX]; // Stores leading sets

int prod\_count; // Number of productions

// Function to check if a symbol is a terminal

int isTerminal(char symbol) {

return !isupper(symbol); // A terminal is not an uppercase letter

}

// Function to add a symbol to the leading set

void addLeading(char nonTerminal, char symbol) {

int index = nonTerminal - 'A';

int i;

// Check if the symbol already exists in the leading set

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

if (leading[index][i] == symbol) {

return; // Symbol already exists

}

}

// Add the new symbol

leading[index][i] = symbol;

leading[index][i + 1] = '\0';

}

// Function to compute LEADING() sets

void computeLeading() {

int i, j;

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

char nonTerminal = productions[i][0]; // LHS of production

int k = 3; // Start after "A->"

// Process each symbol in the RHS

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

if (isTerminal(productions[i][k])) {

addLeading(nonTerminal, productions[i][k]); // Direct terminal

break;

} else {

// If it is a non-terminal, check its first production's first symbol

for (j = 0; j < prod\_count; j++) {

if (productions[j][0] == productions[i][k]) {

if (isTerminal(productions[j][3])) {

addLeading(nonTerminal, productions[j][3]);

}

}

}

}

k++;

}

}

}

void displayLeading() {

int i;

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

printf("LEADING(%c) = { ", productions[i][0]);

int index = productions[i][0] - 'A';

for (int j = 0; leading[index][j] != '\0'; j++) {

printf("%c ", leading[index][j]);

}

printf("}\n");

}

}

int main() {

int i;

printf("Enter the number of productions: ");

scanf("%d", &prod\_count);

getchar(); // Consume newline character

printf("Enter the productions (format: A->α):\n");

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

scanf("%s", productions[i]);

}

// Initialize leading sets

for (i = 0; i < MAX; i++) {

leading[i][0] = '\0';

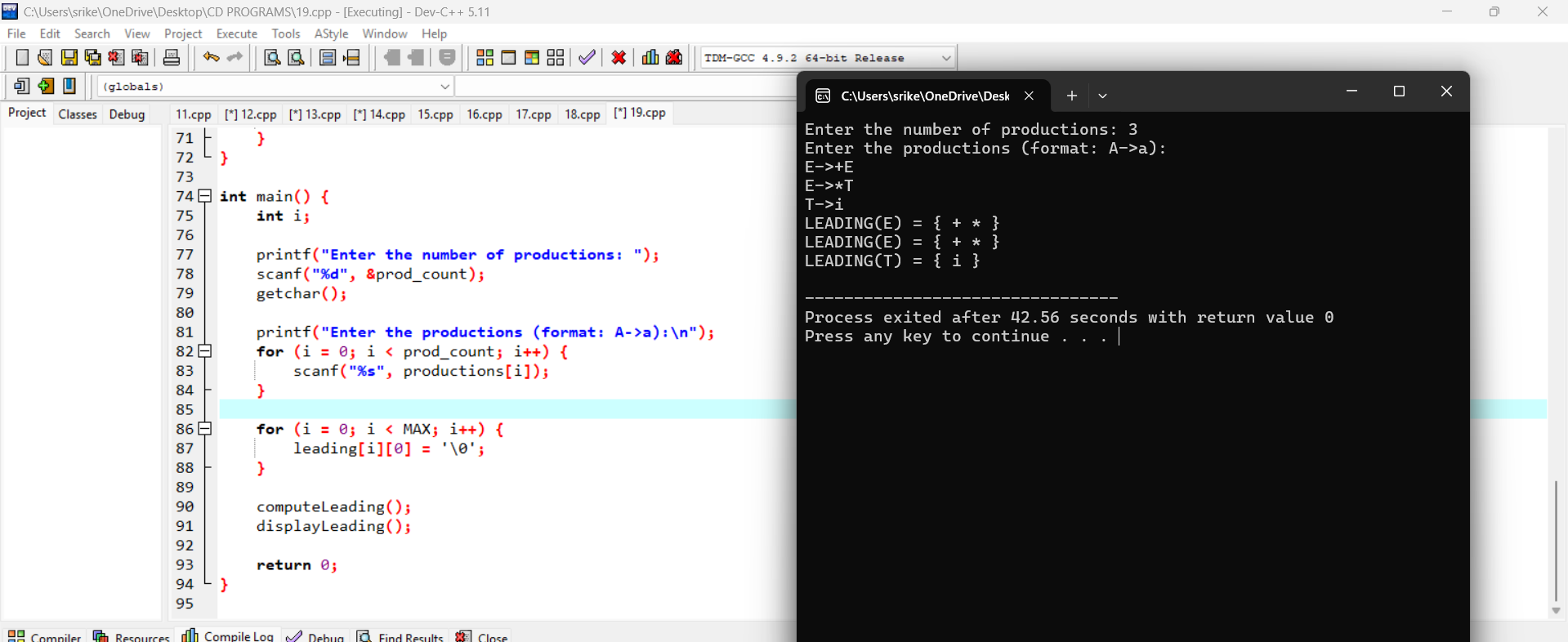
}

computeLeading();

displayLeading();

return 0;

}



20. Write a C program to compute TRAILING( ) – operator precedence parser for the given grammar

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX 10

int prod\_count = 3;

char productions[MAX][MAX] = {

"E->E+T",

"E->T",

"T->i"

};

char trailing[MAX][MAX];

int isTerminal(char symbol) {

return !isupper(symbol);

}

void addTrailing(char nonTerminal, char symbol) {

int index = nonTerminal - 'A';

int i;

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

if (trailing[index][i] == symbol) {

return;

}

}

trailing[index][i] = symbol;

trailing[index][i + 1] = '\0';

}

void computeTrailing() {

int i, j;

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

char nonTerminal = productions[i][0];

int len = strlen(productions[i]);

for (int k = len - 1; k >= 3; k--) {

if (isTerminal(productions[i][k])) {

addTrailing(nonTerminal, productions[i][k]);

break;

} else {

for (j = 0; j < prod\_count; j++) {

if (productions[j][0] == productions[i][k]) {

int lastIndex = strlen(productions[j]) - 1;

if (isTerminal(productions[j][lastIndex])) {

addTrailing(nonTerminal, productions[j][lastIndex]);

}

}

}

}

}

}

}

void displayTrailing() {

int i;

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

printf("TRAILING(%c) = { ", productions[i][0]);

int index = productions[i][0] - 'A';

for (int j = 0; trailing[index][j] != '\0'; j++) {

printf("%c ", trailing[index][j]);

}

printf("}\n");

}

}

int main() {

for (int i = 0; i < MAX; i++) {

trailing[i][0] = '\0';

}

computeTrailing();

displayTrailing();

return 0;

}

