**NAME: Ajay Kumar J**

**REG NO: 192372052**

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

**Aim:**

To implement a symbol table in C that supports insertion, search, and display operations.

**Code:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define SIZE 100

struct Symbol {

char name[50];

char type[20];

int address;

} table[SIZE];

int count = 0;

void insert(char name[], char type[], int address) {

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

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

table[count].address = address;

count++;

}

int search(char name[]) {

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

if (strcmp(table[i].name, name) == 0)

return i;

}

return -1;

}

void display() {

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

printf("Name\tType\tAddress\n");

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

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

}

}

int main() {

insert("x", "int", 100);

insert("y", "float", 104);

insert("func", "function", 200);

display();

char searchName[50];

printf("\nEnter symbol to search: ");

scanf("%s", searchName);

int pos = search(searchName);

if (pos != -1)

printf("%s found at index %d\n", searchName, pos);

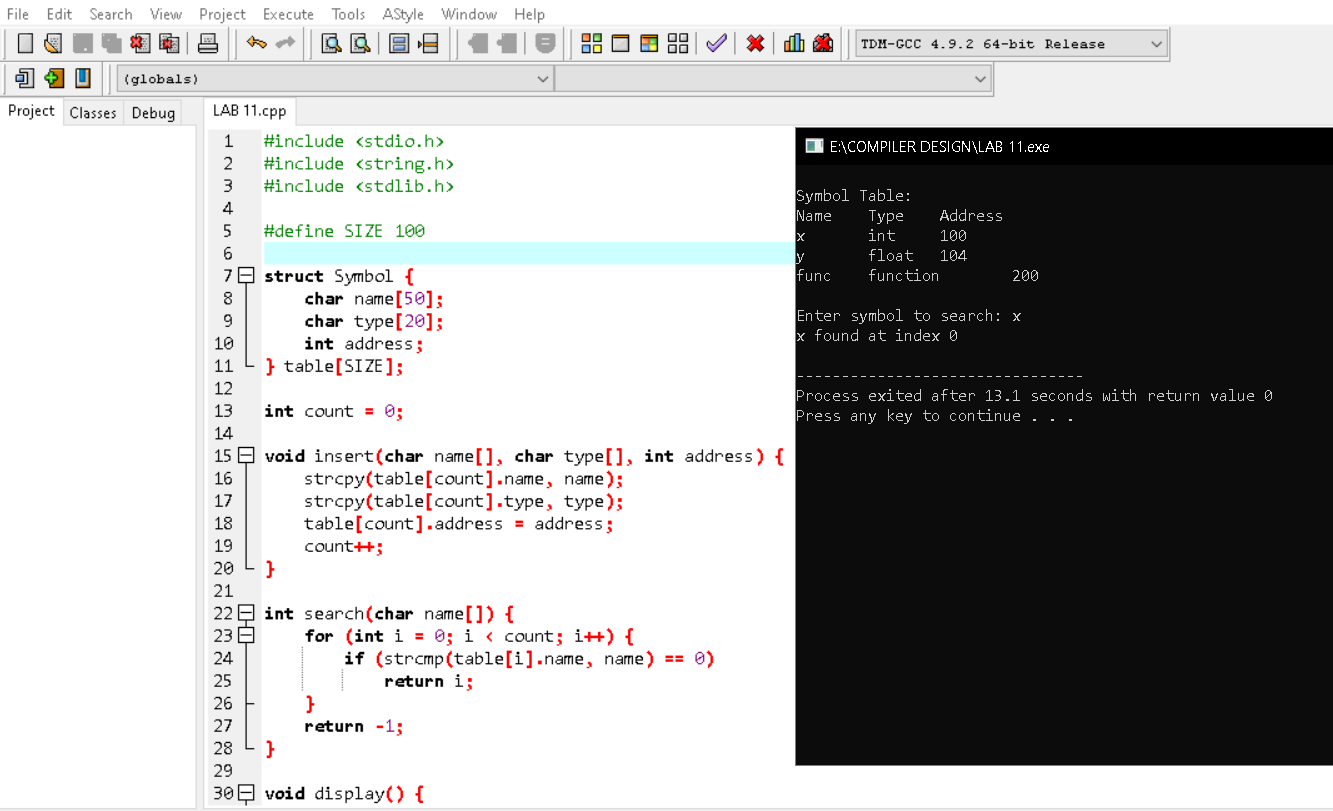
else

printf("%s not found in symbol table.\n", searchName);

return 0;

}

**Output:**

****

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

**Aim:**

To implement a recursive descent parser for the given grammar.

**Code:**

#include <stdio.h>

#include <string.h>

char input[100];

int index = 0;

void E(), T(), F(), E\_prime(), T\_prime();

void match(char expected) {

if (input[index] == expected) {

index++;

} else {

printf("Error in parsing\n");

exit(0);

}

}

void E() {

T();

E\_prime();

}

void E\_prime() {

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

match('+');

T();

E\_prime();

}

}

void T() {

F();

T\_prime();

}

void T\_prime() {

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

match('\*');

F();

T\_prime();

}

}

void F() {

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

match('(');

E();

match(')');

} else if (input[index] == 'i') {

match('i'); // Assuming 'i' represents an identifier

} else {

printf("Error in parsing\n");

exit(0);

}

}

int main() {

printf("Enter input string: ");

scanf("%s", input);

E();

if (input[index] == '\0') {

printf("Parsing successful\n");

} else {

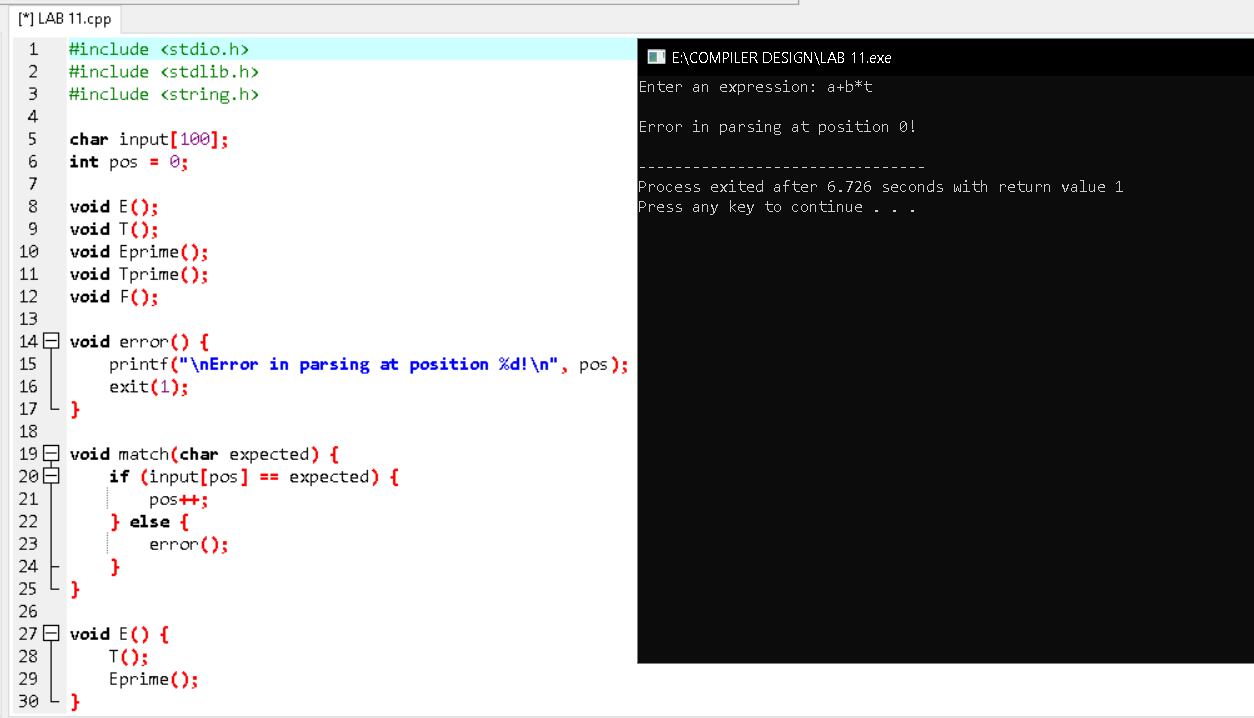
printf("Error: Unexpected input\n");

}

return 0;

}

**Output:**

****

**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.**

**Aim:**

To implement a Bottom-Up Parsing technique (Shift-Reduce Parsing) in C to check whether the given input string satisfies the specified grammar.

**C Code Implementation:**

#include <stdio.h>

#include <string.h>

char stack[50];

char input[50];

int top = -1;

int ip = 0;

void push(char c) {

stack[++top] = c;

}

void pop() {

top--;

}

void display() {

printf("\nStack: %s\t Input: %s", stack, input + ip);

}

int reduce() {

if (top >= 2) {

if (stack[top] == 'E' && stack[top - 1] == '+' && stack[top - 2] == 'E') {

printf("\nReduce by E -> E+E");

top -= 2;

return 1;

}

if (stack[top] == 'E' && stack[top - 1] == '\*' && stack[top - 2] == 'E') {

printf("\nReduce by E -> E\*E");

top -= 2;

return 1;

}

}

if (top >= 2) {

if (stack[top] == ')' && stack[top - 1] == 'E' && stack[top - 2] == '(') {

printf("\nReduce by E -> (E)");

top -= 2;

return 1;

}

}

if (top >= 0) {

if (stack[top] == 'a') {

printf("\nReduce by E -> a");

stack[top] = 'E';

return 1;

}

}

return 0;

}

int main() {

printf("Enter the input string ending with $: ");

scanf("%s", input);

push('$');

printf("\nBottom-Up Parsing (Shift-Reduce) Simulation:\n");

display();

while (1) {

if (input[ip] != '\0') {

push(input[ip++]);

printf("\nShift '%c'", stack[top]);

display();

}

while (reduce()) {

display();

}

if (input[ip] == '\0' && top == 1 && stack[top] == 'E' && stack[0] == '$') {

printf("\n\nThe input string is successfully parsed!\n");

break;

}

if (input[ip] == '\0' && top != 1) {

printf("\n\nError: The input string cannot be parsed by the given grammar.\n");

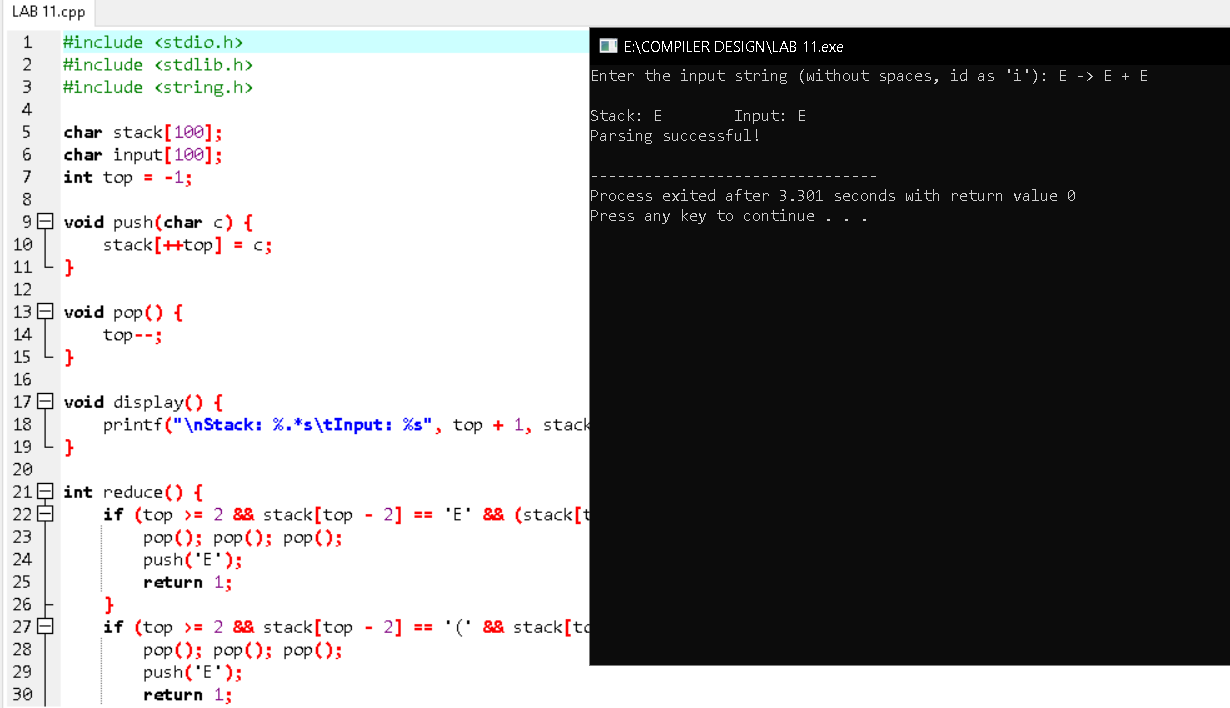
break;

}

}

return 0;

}



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

**Aim:**To implement the Shift-Reduce Parsing technique in C to parse a given input string based on a specified grammar.

**Code:**

#include <stdio.h>

#include <string.h>

char stack[50];

char input[50];

int top = -1, ip = 0;

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

void pop() { top--; }

int reduce() {

if (top >= 2) {

if (stack[top] == 'E' && stack[top - 1] == '+' && stack[top - 2] == 'E') {

top -= 2;

stack[top] = 'E';

return 1;

}

if (stack[top] == 'E' && stack[top - 1] == '\*' && stack[top - 2] == 'E') {

top -= 2;

stack[top] = 'E';

return 1;

}

}

if (top >= 2 && stack[top] == ')' && stack[top - 1] == 'E' && stack[top - 2] == '(') {

top -= 2;

stack[top] = 'E';

return 1;

}

if (top >= 0 && stack[top] == 'a') {

stack[top] = 'E';

return 1;

}

return 0;

}

int main() {

printf("Enter input string (end with $): ");

scanf("%s", input);

push('$');

while (1) {

if (input[ip] != '\0') {

push(input[ip++]);

}

while (reduce());

if (input[ip] == '\0' && top == 1 && stack[0] == '$' && stack[1] == 'E') {

printf("String is successfully parsed!\n");

break;

}

if (input[ip] == '\0' && top != 1) {

printf("Error: Parsing failed.\n");

break;

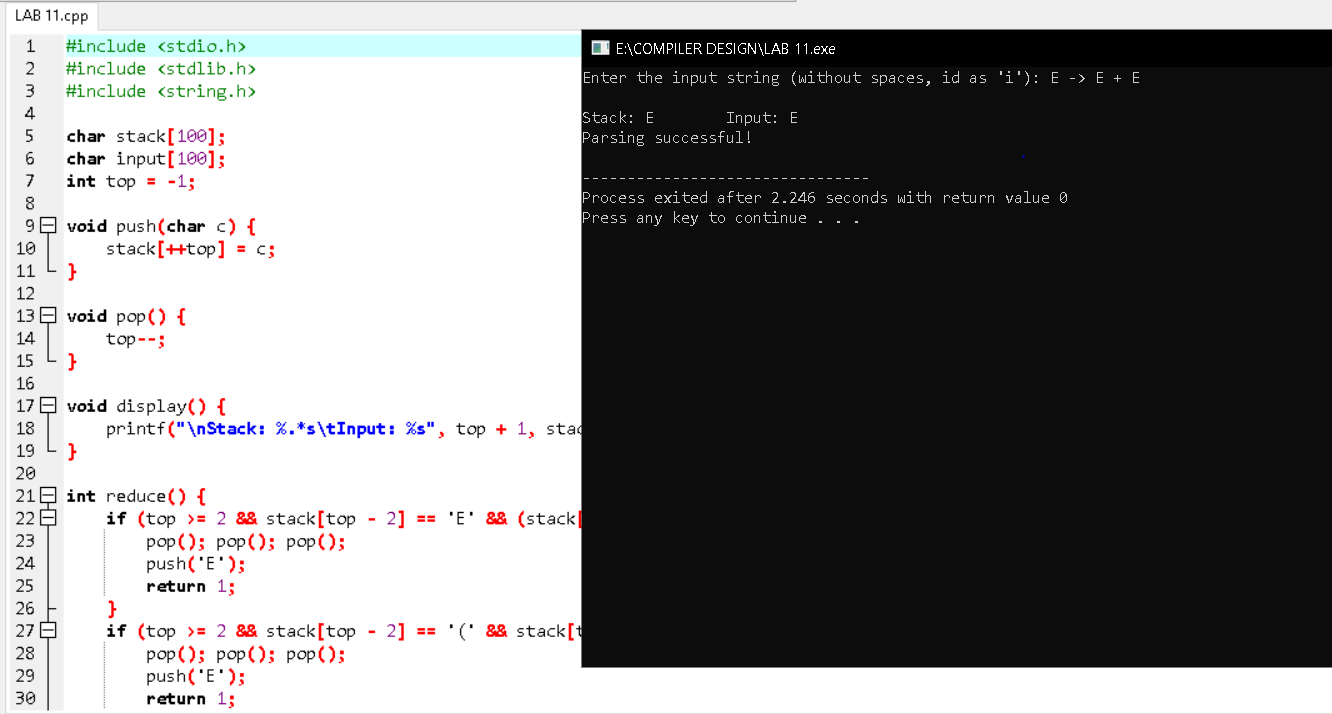
}

}

return 0;

}

**Output:**

****

**15. Write a C Program to implement Operator Precedence Parsing.**

**Aim:**  
To implement Operator Precedence Parsing in C to parse and evaluate a given input expression using operator precedence relations.

**Code:**

#include <stdio.h>

#include <string.h>

int prec(char c) {

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

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

return 0;

}

void parse(char \*exp) {

char stack[50];

int top = -1, i = 0;

stack[++top] = '$'

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

while (top >= 0 && prec(stack[top]) >= prec(exp[i])) {

printf("Reduce by popping %c\n", stack[top--]);

}

printf("Shift '%c'\n", exp[i]);

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

}

while (top > 0) {

printf("Reduce by popping %c\n", stack[top--]);

}

printf("Parsing Successful!\n");

}

int main() {

char exp[50];

printf("Enter expression: ");

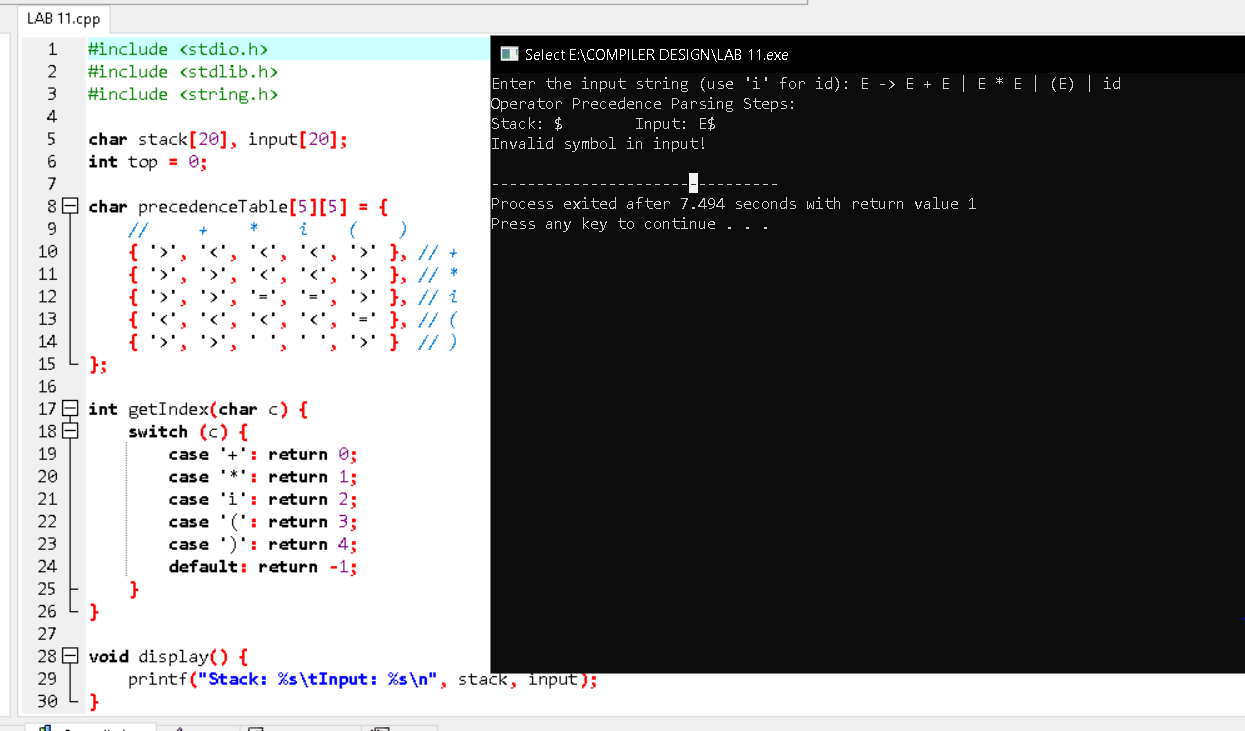
scanf("%s", exp);

parse(exp);

return 0;

}

**Output:**

****

**16. Write a C Program to Generate the Three-Address Code representation for the given input statement.**

**Aim:**  
To implement a C program that generates three-address code for arithmetic expressions.

**Code:**

#include <stdio.h>

#include <string.h>

int main() {

char expr[50], temp[5] = "t";

int i = 0, j = 1, k = 0;

printf("Enter an arithmetic expression: ");

scanf("%s", expr);

printf("\nThree-Address Code:\n");

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

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

printf("t%d = %c %c %c\n", k++, expr[i - 1], expr[i], expr[i + 1]);

i++;

}

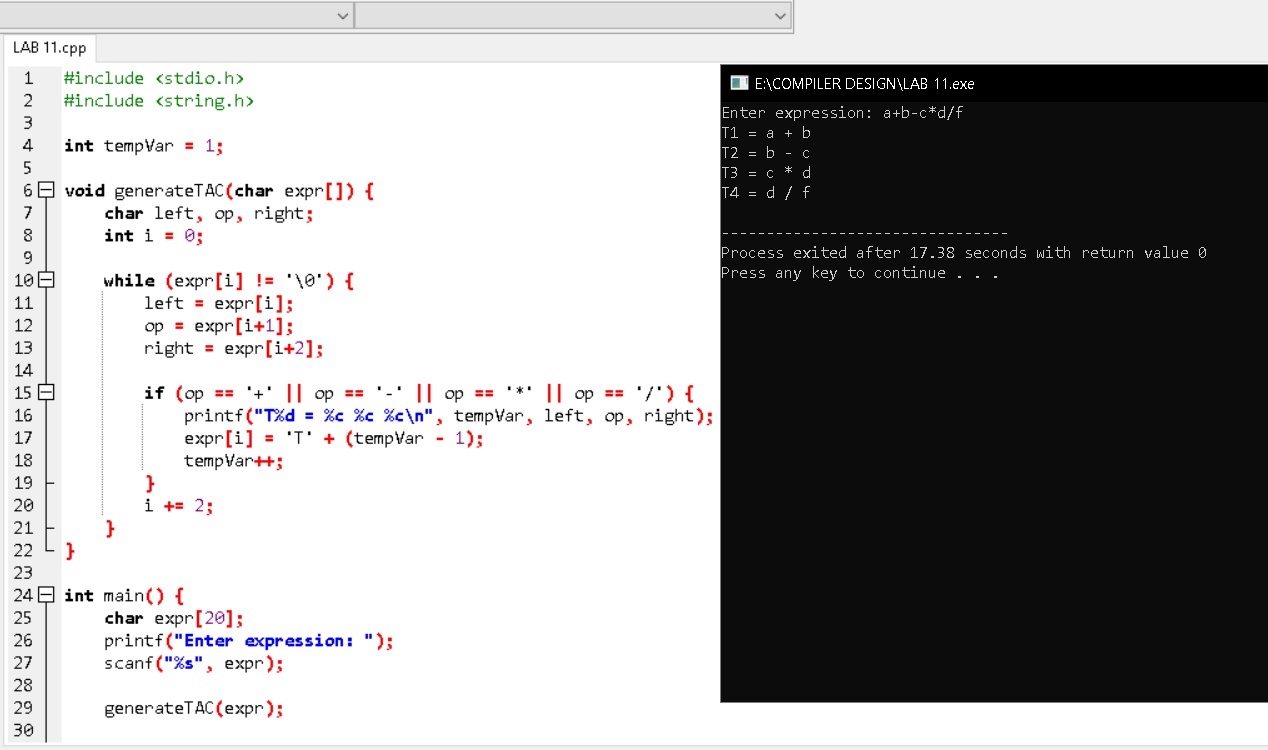
i++;

}

return 0;

}

**Output:**

****

**17. Write a C program for implementing a Lexical Analyzer to Scan and Count the number of characters, words, and lines in a file.**

**Aim:**To implement a lexical analyzer in C that scans and counts characters, words, and lines from a given file.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

int main() {

FILE \*file;

char filename[50], ch;

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

printf("Enter filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("File not found!\n");

return 1;

}

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

characters++;

if (ch == ' ' || ch == '\t' || ch == '\n') words++;

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

}

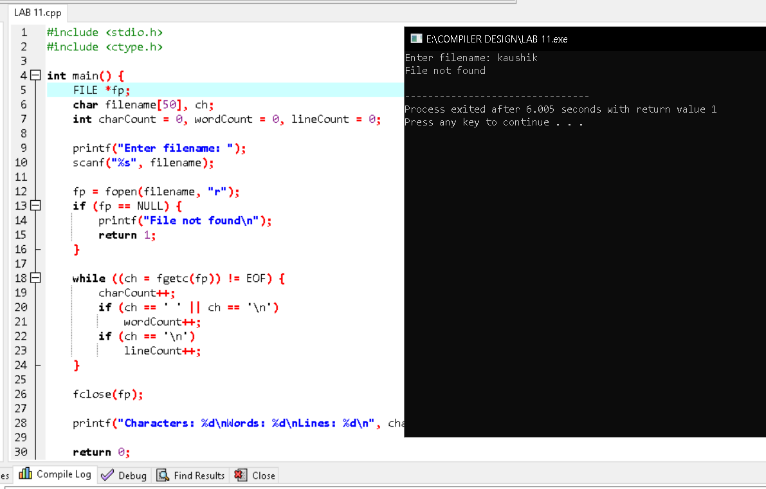
fclose(file);

printf("Characters: %d\nWords: %d\nLines: %d\n", characters, words, lines);

return 0;

}

**Output:**

****

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

**Aim:**To implement the back end of a compiler in C that generates machine code for arithmetic expressions.

**Code:**

#include <stdio.h>

int main() {

char expr[50];

printf("Enter arithmetic expression: ");

scanf("%s", expr);

printf("\nGenerated Assembly Code:\n");

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

if (expr[i] == '+') printf("ADD\n");

else if (expr[i] == '-') printf("SUB\n");

else if (expr[i] == '\*') printf("MUL\n");

else if (expr[i] == '/') printf("DIV\n");

else printf("PUSH %c\n", expr[i]);

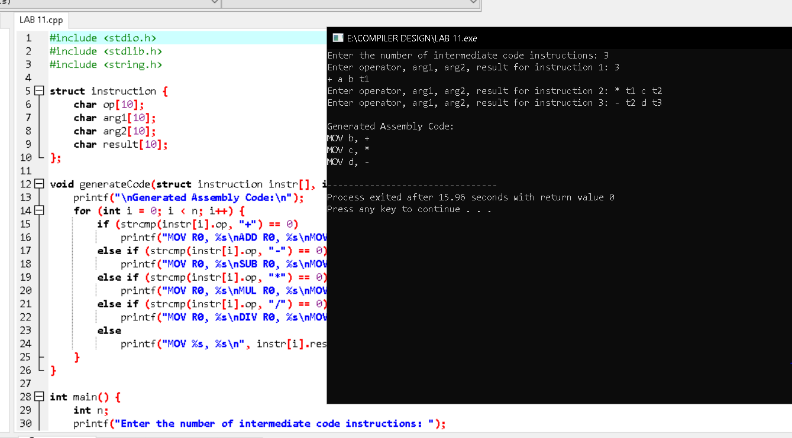
}

printf("POP RESULT\n");

return 0;

}

**Output:**

****

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

**Aim:**To implement a C program that computes the LEADING sets for operators in a given grammar.

**Code:**

#include <stdio.h>

#include <string.h>

void findLeading(char \*grammar, char terminal) {

printf("Leading(%c) = {", terminal);

for (int i = 0; i < strlen(grammar); i++) {

if (grammar[i] == terminal && grammar[i + 1] != '\0') {

printf(" %c ", grammar[i + 1]);

}

}

printf("}\n");

}

int main() {

char grammar[50], terminal;

printf("Enter grammar (like E->E+T|T): ");

scanf("%s", grammar);

printf("Enter terminal symbol: ");

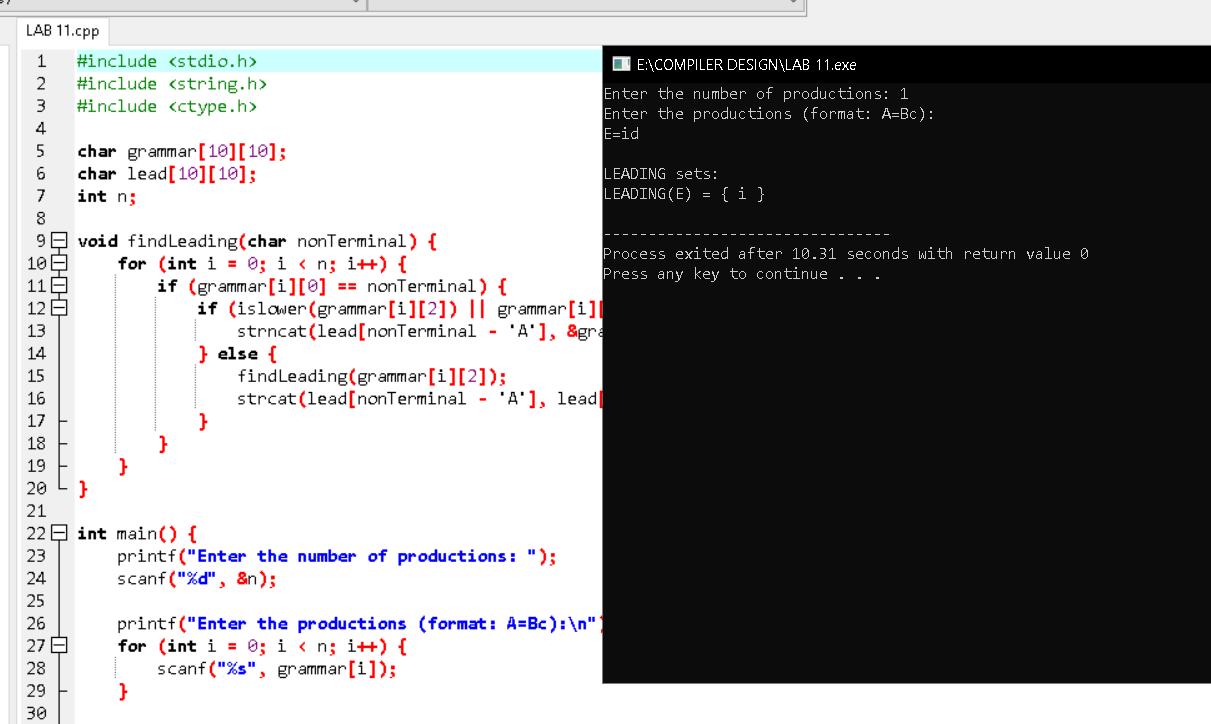
scanf(" %c", &terminal);

findLeading(grammar, terminal);

return 0;

}

**Output:**

****

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

**Aim:**

To implement a C program that computes the TRAILING sets for operators in a given grammar.

**Code:**

#include <stdio.h>

#include <string.h>

void findTrailing(char \*grammar, char terminal) {

printf("Trailing(%c) = {", terminal);

for (int i = 0; i < strlen(grammar); i++) {

if (grammar[i] == terminal && i > 0) {

printf(" %c ", grammar[i - 1]);

}

}

printf("}\n");

}

int main() {

char grammar[50], terminal;

printf("Enter grammar (like E->E+T|T): ");

scanf("%s", grammar);

printf("Enter terminal symbol: ");

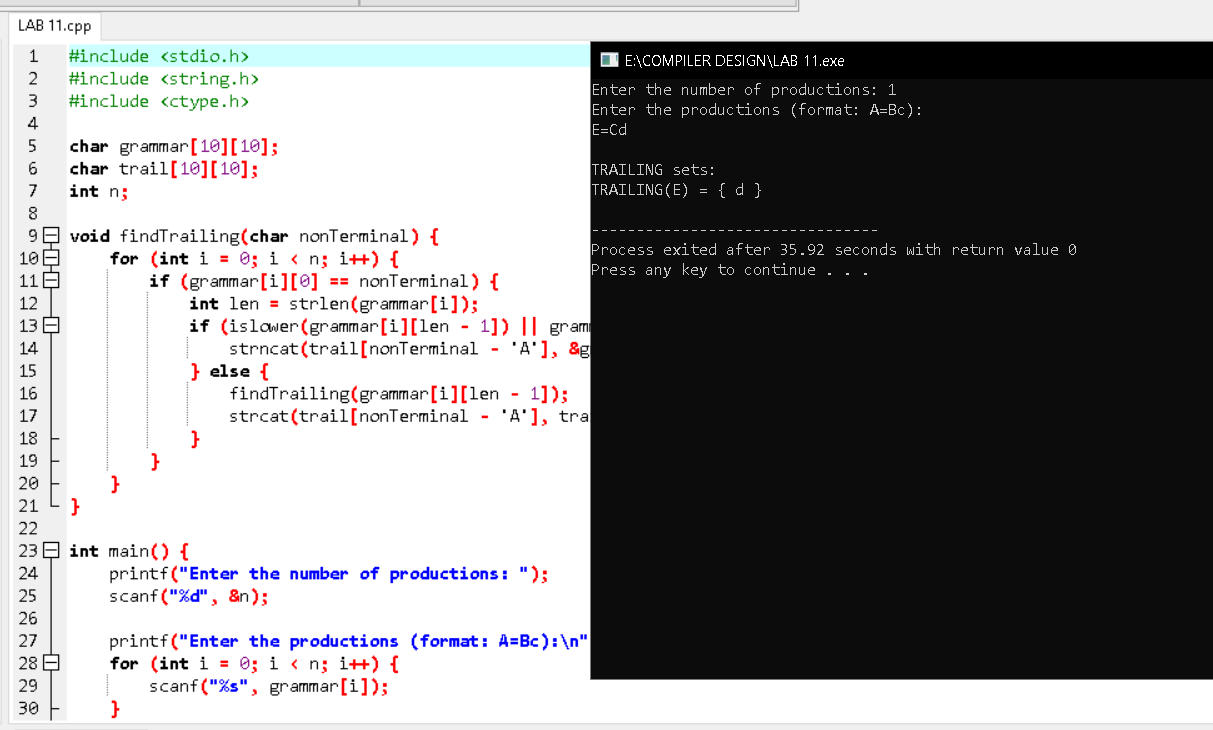
scanf(" %c", &terminal);

findTrailing(grammar, terminal);

return 0;

}

**Output:**

****