**8. Implement a C program to perform** **symbol table operations.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct {

char name[50];

char type[20];

int size;

int address;

} Symbol;

Symbol table[MAX];

int count = 0;

void insertSymbol(char \*name, char \*type, int size, int address) {

if (count < MAX) {

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

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

table[count].size = size;

table[count].address = address;

count++;

printf("Symbol inserted successfully!\n");

} else {

printf("Symbol Table is full!\n");

}

}

int searchSymbol(char \*name) {

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

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

return i;

}

}

return -1;

}

void displayTable() {

if (count == 0) {

printf("Symbol Table is empty!\n");

return;

}

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

printf("------------------------------------------------\n");

printf("Index | Name | Type | Size | Address\n");

printf("------------------------------------------------\n");

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

printf("%5d | %-10s | %-10s | %4d | %7d\n", i, table[i].name, table[i].type, table[i].size, table[i].address);

}

printf("------------------------------------------------\n");

}

int main() {

int choice, size, address;

char name[50], type[20];

while (1) {

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

printf("1. Insert Symbol\n");

printf("2. Search Symbol\n");

printf("3. Display Symbol Table\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter name, type, size, and address: ");

scanf("%s %s %d %d", name, type, &size, &address);

insertSymbol(name, type, size, address);

break;

case 2:

printf("Enter the symbol name to search: ");

scanf("%s", name);

int index = searchSymbol(name);

if (index != -1) {

printf("Symbol found at index %d.\n", index);

} else {

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

}

break;

case 3:

displayTable();

break;

case 4:

exit(0);

default:

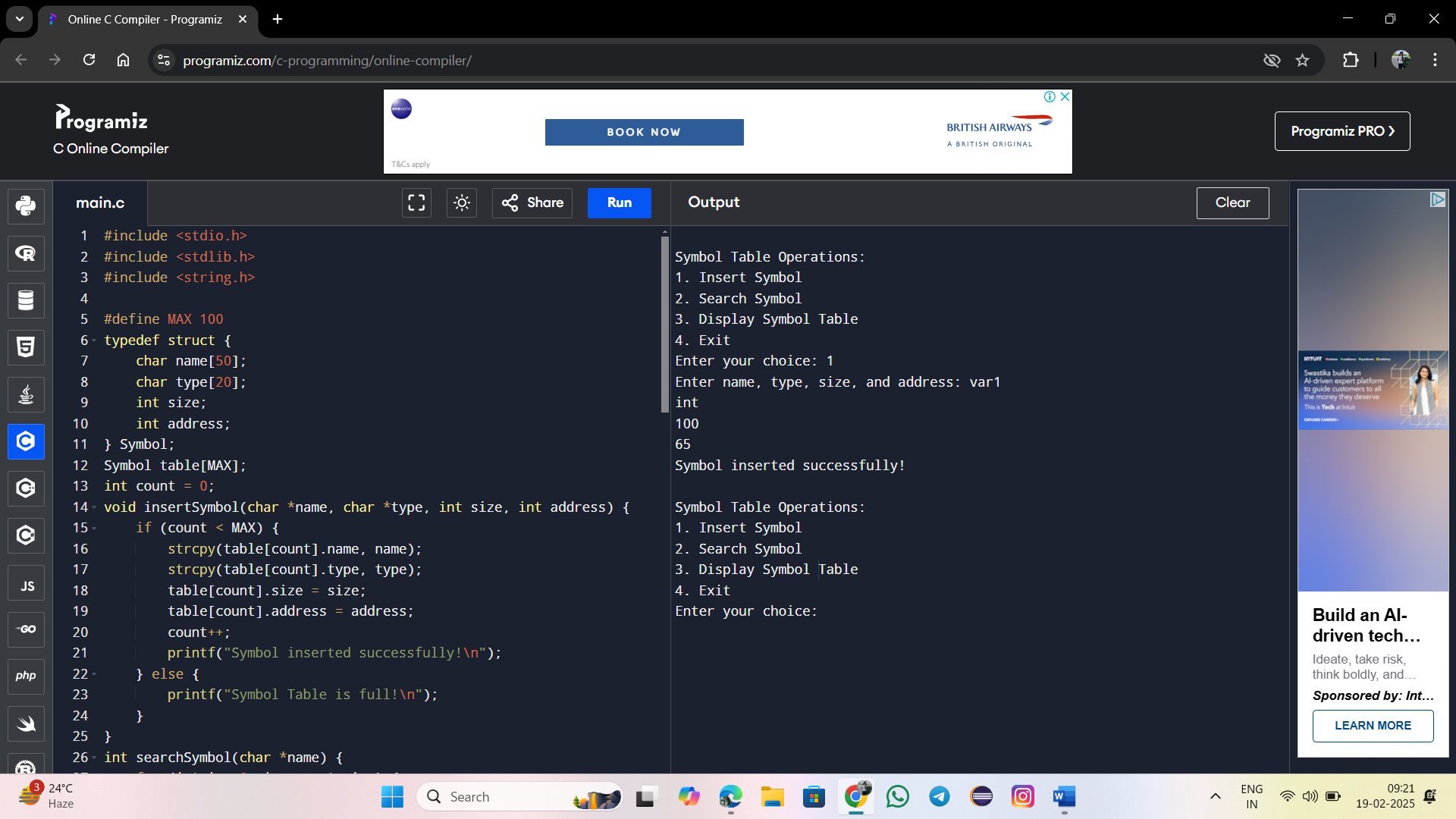
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



**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>

#include <stdbool.h>

#define MAX 100

const char \*subjects[] = {"I", "You", "He", "She", "They", "We"};

const char \*verbs[] = {"eat", "play", "read", "write", "like"};

const char \*objects[] = {"apple", "book", "game", "song"};

bool isValidWord(const char \*word, const char \*list[], int size) {

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

if (strcmp(word, list[i]) == 0) {

return true;

}

}

return false;

}

bool checkGrammar(char \*sentence) {

char \*words[3];

int wordCount = 0;

char \*token = strtok(sentence, " ");

while (token != NULL && wordCount < 3) {

words[wordCount++] = token;

token = strtok(NULL, " ");

}

if (wordCount != 3) {

return false;

}

return isValidWord(words[0], subjects, 6) &&

isValidWord(words[1], verbs, 5) &&

isValidWord(words[2], objects, 4);

}

int main() {

char sentence[MAX];

printf("Enter a sentence: ");

fgets(sentence, MAX, stdin);

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

if (checkGrammar(sentence)) {

printf("Valid sentence structure!\n");

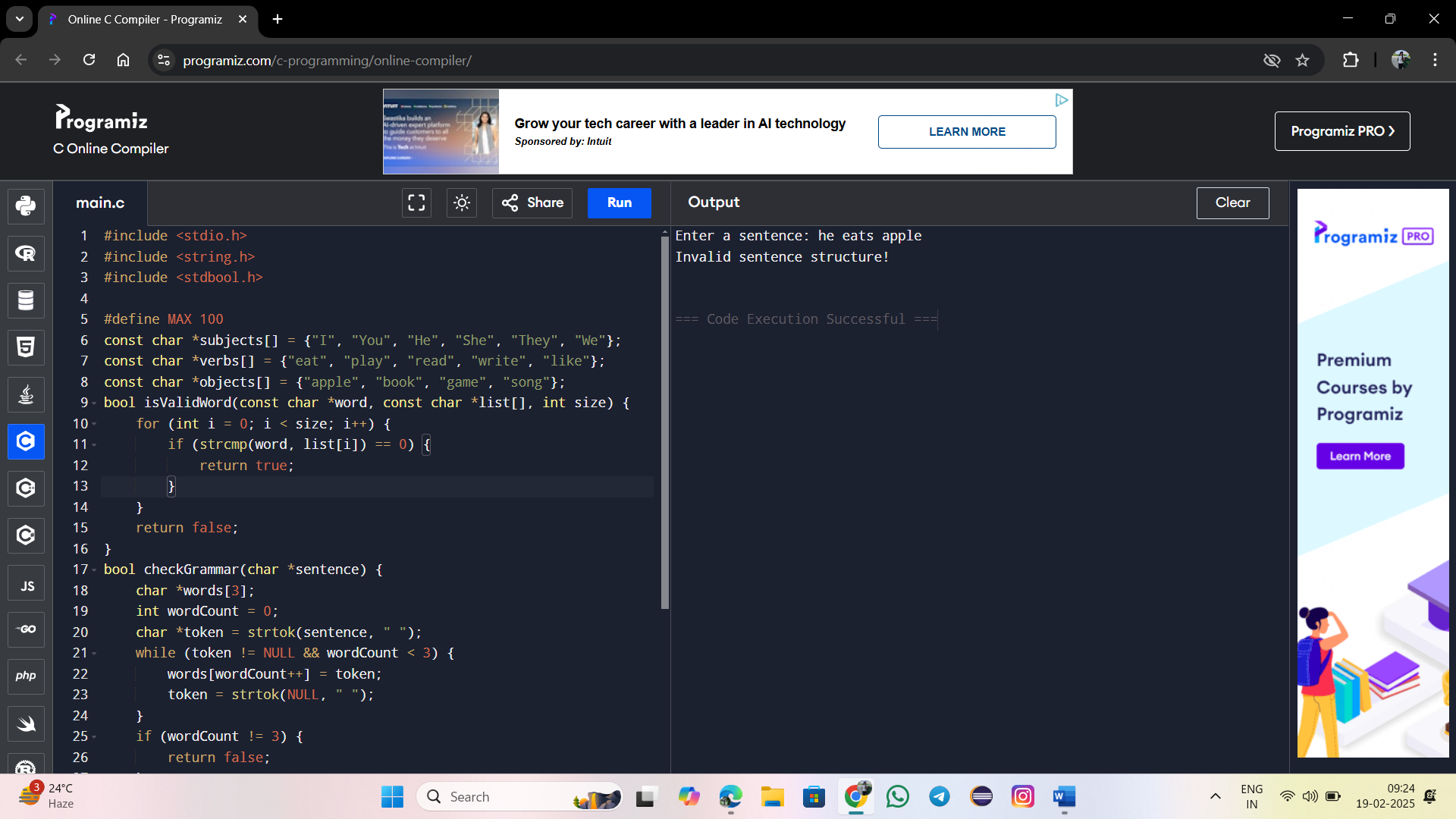
} else {

printf("Invalid sentence structure!\n");

}

return 0;

}



**10.Write a C program to construct recursive descent parsing**.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

char input[100];

int pos = 0;

void E();

void Eprime();

void T();

void Tprime();

void F();

// Function to match expected character

void match(char expected) {

if (input[pos] == expected) {

pos++;

} else {

printf("Syntax Error: Expected '%c' but found '%c'\n", expected, input[pos]);

exit(1);

}

}

// Parsing functions

void E() {

T();

Eprime();

}

void Eprime() {

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

match('+');

T();

Eprime();

}

// Else epsilon (do nothing)

}

void T() {

F();

Tprime();

}

void Tprime() {

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

match('\*');

F();

Tprime();

}

// Else epsilon (do nothing)

}

void F() {

if (isalnum(input[pos])) {

match(input[pos]); // Match identifier (id)

} else if (input[pos] == '(') {

match('(');

E();

match(')');

} else {

printf("Syntax Error: Unexpected character '%c'\n", input[pos]);

exit(1);

}

}

int main() {

printf("Enter an expression: ");

scanf("%s", input);

strcat(input, "$"); // End marker

E();

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

printf("Parsing successful! The input is valid.\n");

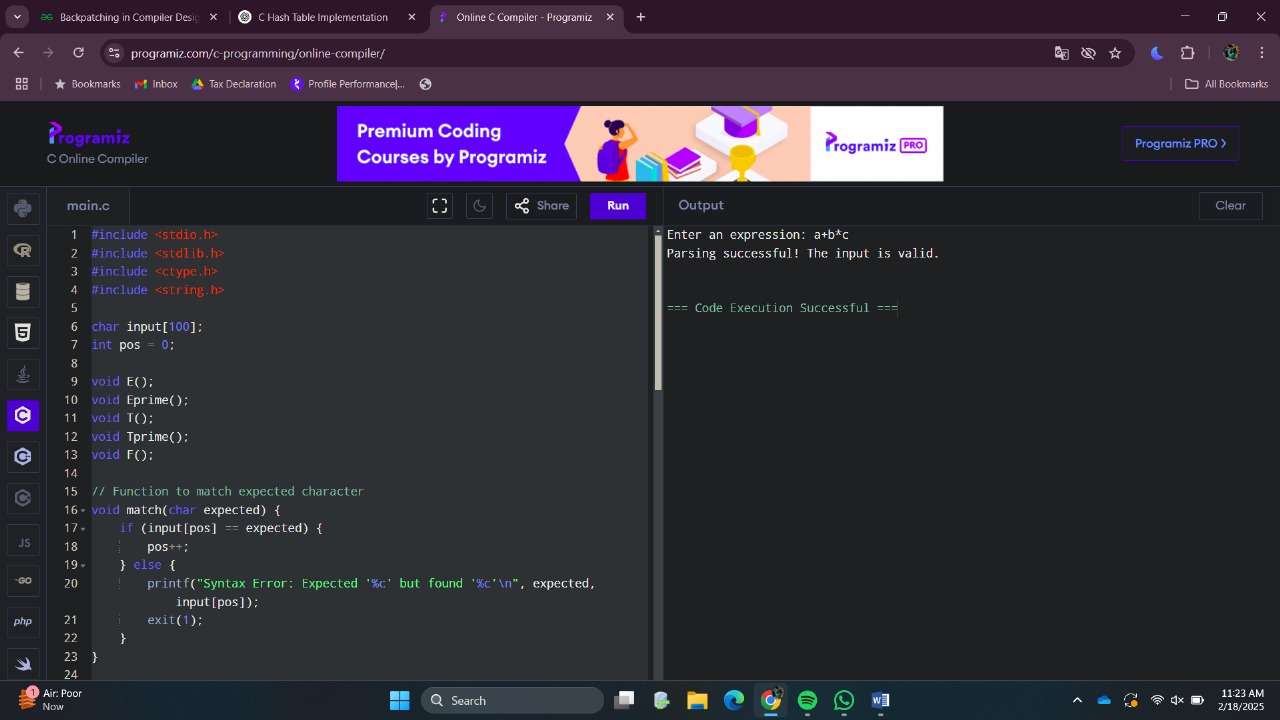
} else {

printf("Syntax Error: Unexpected end of input.\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 <stdlib.h>

#include <ctype.h>

#include <math.h>

char \*expr;

// Function prototypes

double parseExpression();

double parseTerm();

double parseFactor();

double parsePower();

// Function to parse exponentiation (highest precedence)

double parsePower() {

double value = parseFactor();

while (\*expr == '^') {

expr++;

value = pow(value, parseFactor());

}

return value;

}

// Function to parse factors (parentheses or numbers)

double parseFactor() {

double value;

if (\*expr == '(') {

expr++; // Skip '('

value = parseExpression();

if (\*expr == ')') {

expr++; // Skip ')'

} else {

printf("Error: Mismatched parentheses!\n");

exit(1);

}

} else if (isdigit(\*expr)) {

value = strtod(expr, &expr); // Convert number

} else {

printf("Error: Unexpected character '%c'\n", \*expr);

exit(1);

}

return value;

}

// Function to parse terms (Multiplication and Division)

double parseTerm() {

double value = parsePower();

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

char op = \*expr;

expr++;

if (op == '\*') {

value \*= parsePower();

} else {

double divisor = parsePower();

if (divisor == 0) {

printf("Error: Division by zero!\n");

exit(1);

}

value /= divisor;

}

}

return value;

}

// Function to parse expressions (Addition and Subtraction)

double parseExpression() {

double value = parseTerm();

while (\*expr == '+' || \*expr == '-') {

char op = \*expr;

expr++;

if (op == '+') {

value += parseTerm();

} else {

value -= parseTerm();

}

}

return value;

}

int main() {

char input[100];

printf("Enter an arithmetic expression: ");

scanf("%s", input);

expr = input;

double result = parseExpression();

if (\*expr == '\0') {

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

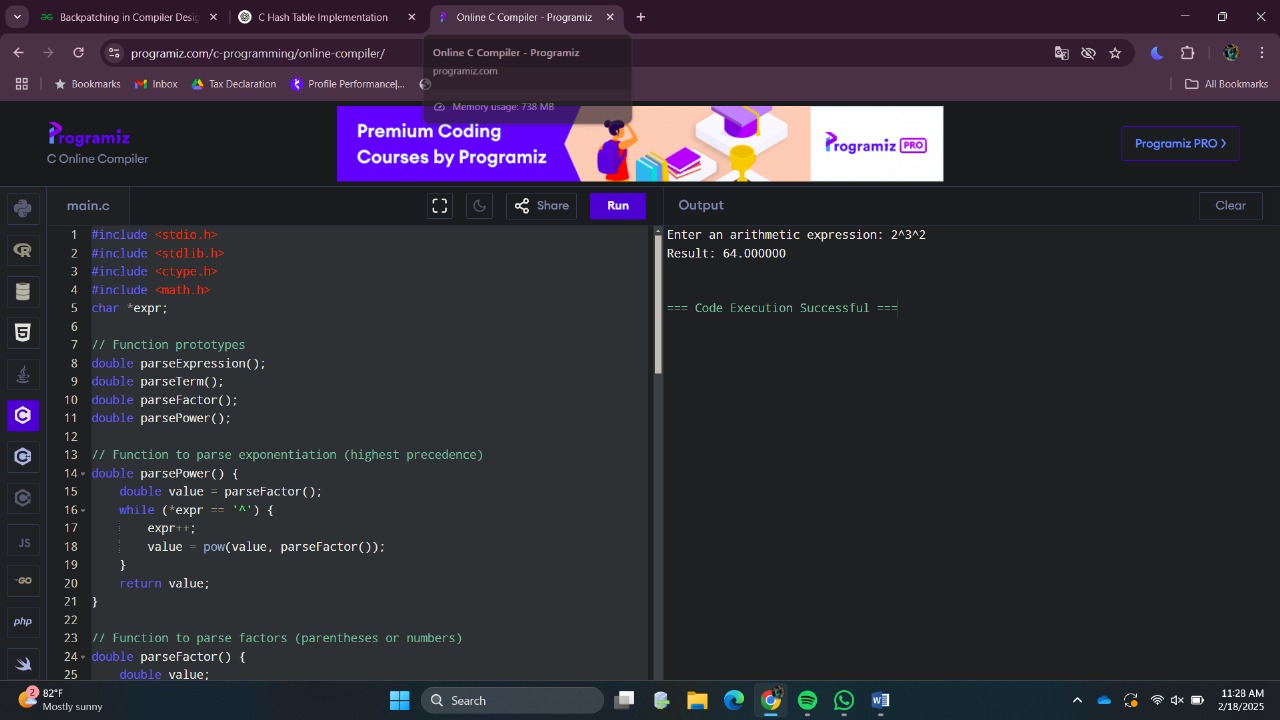
} else {

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

}

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 <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

int tempVarCount = 1;

char input[MAX];

int pos = 0;

void generateTAC(char result[], char op1[], char operator, char op2[]) {

printf("t%d = %s %c %s\n", tempVarCount, op1, operator, op2);

sprintf(result, "t%d", tempVarCount);

tempVarCount++;

}

void parseExpression(char result[]);

void parseTerm(char result[]);

void parseFactor(char result[]);

// Parses a Factor (numbers or variables)

void parseFactor(char result[]) {

if (isalnum(input[pos])) {

sprintf(result, "%c", input[pos]);

pos++;

} else if (input[pos] == '(') {

pos++; // Skip '('

parseExpression(result);

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

pos++; // Skip ')'

} else {

printf("Error: Mismatched parentheses!\n");

exit(1);

}

} else {

printf("Error: Invalid character '%c'\n", input[pos]);

exit(1);

}

}

// Parses a Term (handles multiplication and division)

void parseTerm(char result[]) {

char left[MAX], right[MAX];

parseFactor(left);

while (input[pos] == '\*' || input[pos] == '/') {

char op = input[pos];

pos++;

parseFactor(right);

generateTAC(result, left, op, right);

strcpy(left, result);

}

strcpy(result, left);

}

// Parses an Expression (handles addition and subtraction)

void parseExpression(char result[]) {

char left[MAX], right[MAX];

parseTerm(left);

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

char op = input[pos];

pos++;

parseTerm(right);

generateTAC(result, left, op, right);

strcpy(left, result);

}

strcpy(result, left);

}

int main() {

char result[MAX];

printf("Enter an arithmetic expression: ");

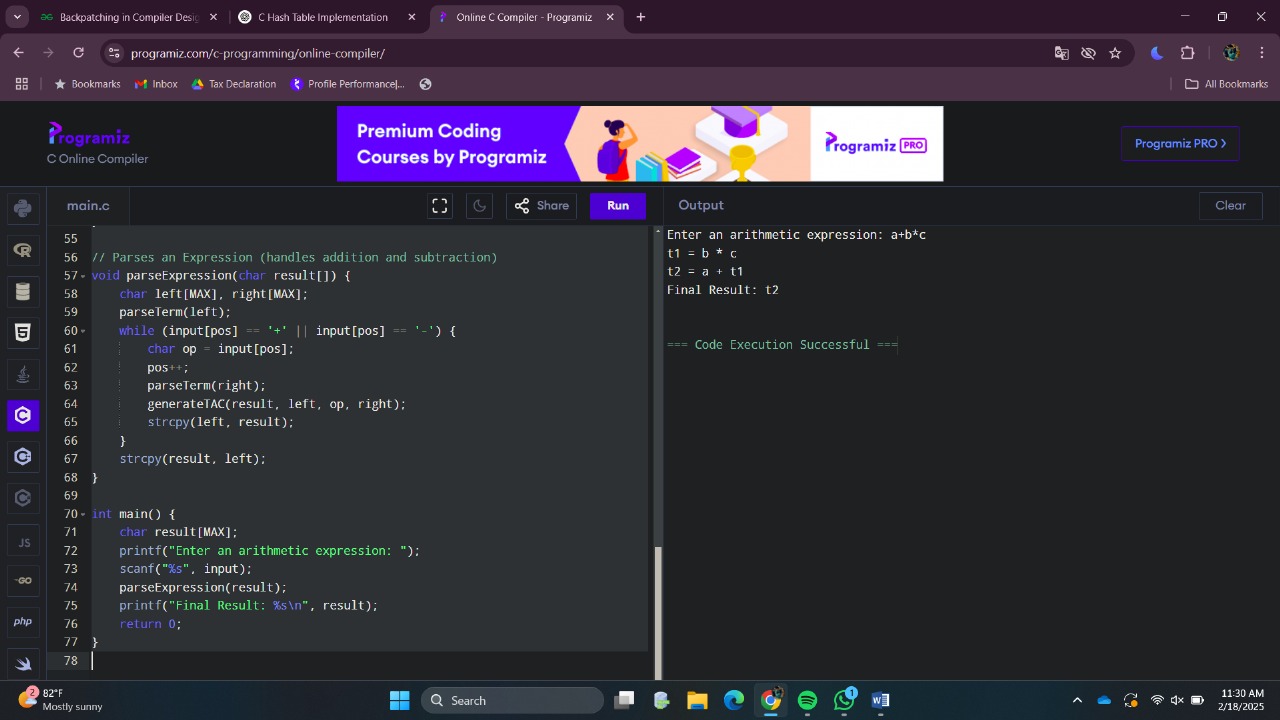
scanf("%s", input);

parseExpression(result);

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

return 0;

}



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

#include <stdio.h>

#include <ctype.h>

int main() {

char ch;

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

int inWord = 0; // Flag to track whether we are inside a word

printf("Enter text (Ctrl+D to end input):\n");

// Read input character by character until EOF (Ctrl+D)

while ((ch = getchar()) != EOF) {

charCount++; // Count the character

// If a newline character is found, increment line count

if (ch == '\n') {

lineCount++;

}

// If a space or newline is encountered, we consider the end of a word

if (isspace(ch)) {

if (inWord) {

wordCount++; // End of a word, so increment word count

inWord = 0; // Reset inWord flag

}

} else {

inWord = 1; // Inside a word

}

}

// If the last word was not followed by a space/newline, increment word count

if (inWord) {

wordCount++;

}

// If the file has no content, set line count to 1

if (charCount > 0 && lineCount == 0) {

lineCount = 1;

}

// Output the results

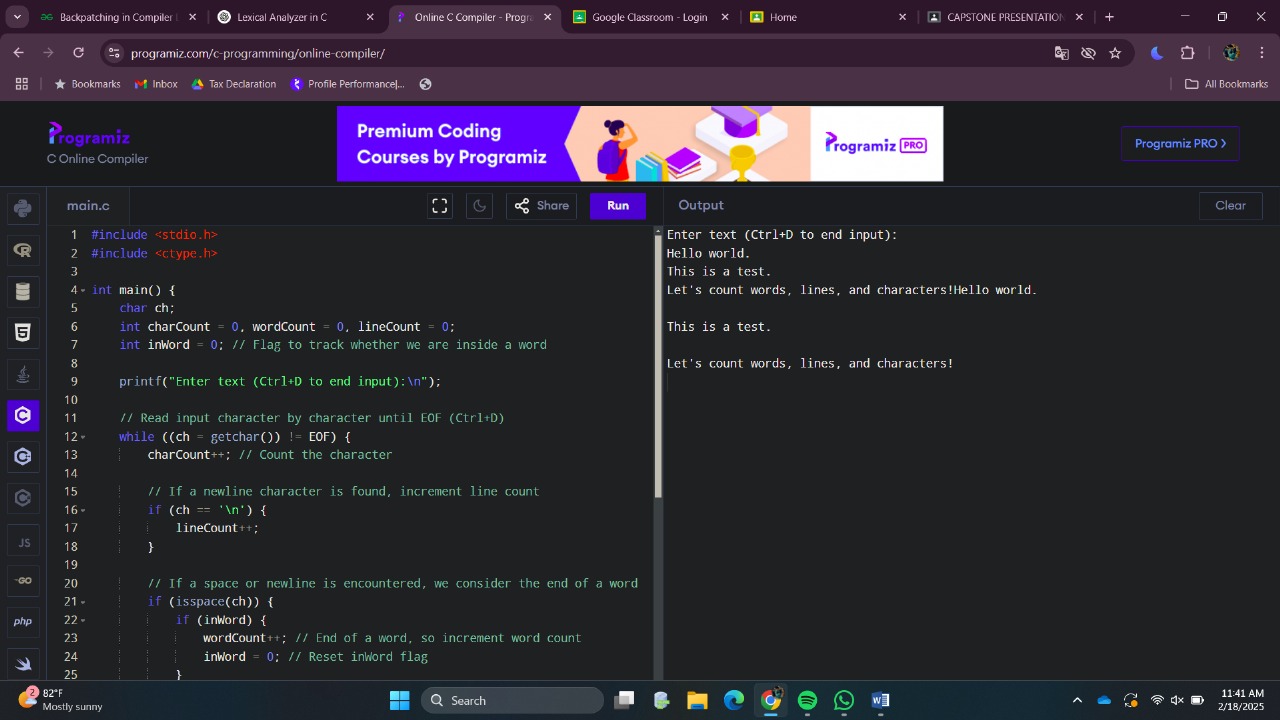
printf("\nCharacter Count: %d\n", charCount);

printf("Word Count: %d\n", wordCount);

printf("Line Count: %d\n", lineCount);

return 0;

}



**14.** **Write a C Program for code optimization to eliminate common subexpression.**

#include <stdio.h>

int main() {

// Declare variables

int a = 10, b = 20, c = 30, d = 40;

int result1, result2, result3;

// Without optimization

result1 = a \* b + c \* d;

result2 = a \* b + c \* d; // Redundant computation of the same subexpression

result3 = a \* b + c \* d; // Another redundant computation

printf("Without optimization:\n");

printf("Result1: %d\n", result1);

printf("Result2: %d\n", result2);

printf("Result3: %d\n\n", result3);

// Optimized version by eliminating common subexpression

int commonExpression = a \* b + c \* d; // Calculate once and reuse

result1 = commonExpression;

result2 = commonExpression;

result3 = commonExpression;

printf("With optimization (common subexpression eliminated):\n");

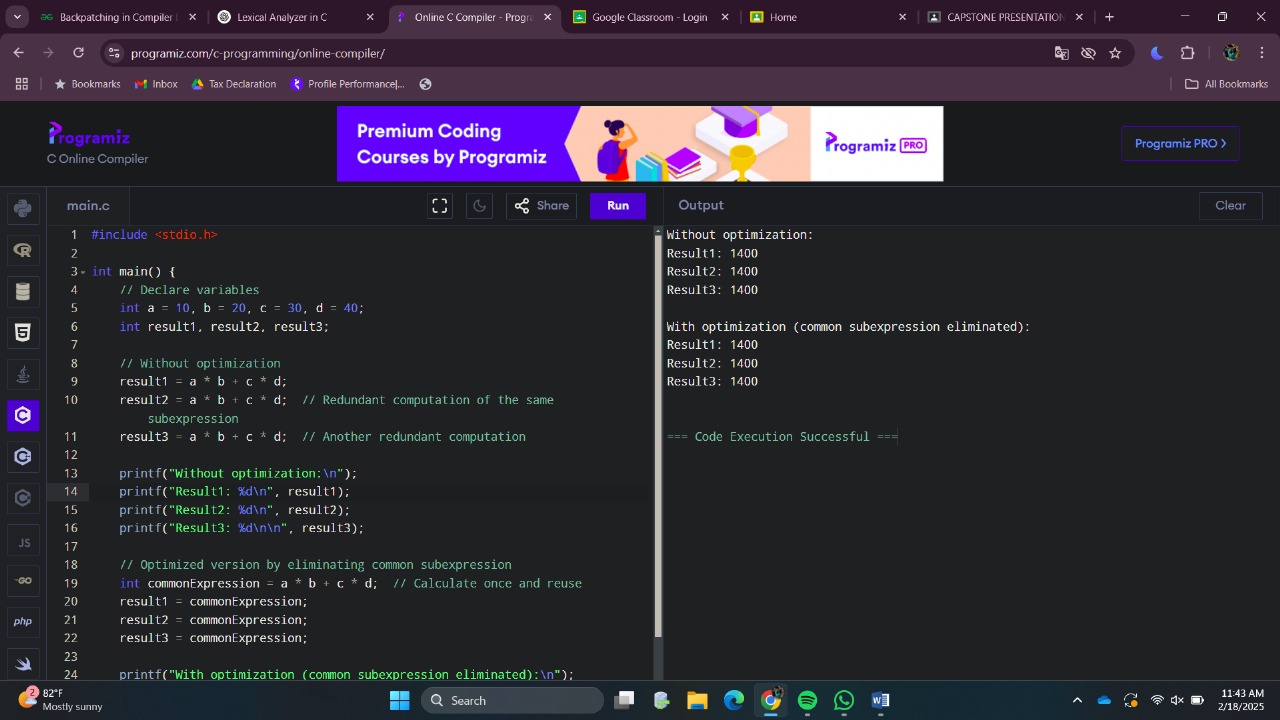
printf("Result1: %d\n", result1);

printf("Result2: %d\n", result2);

printf("Result3: %d\n", result3);

return 0;

}



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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Function to generate assembly code for a simple arithmetic expression

void generateAssembly(char\* expr) {

char \*token;

char \*operator;

int num1, num2;

// Print the start of the assembly code

printf(".intel\_syntax noprefix\n");

printf(".global \_start\n");

printf("\_start:\n");

// Tokenize the expression and handle basic operations

token = strtok(expr, " ");

num1 = atoi(token); // First operand

token = strtok(NULL, " ");

operator = token; // Operator

token = strtok(NULL, " ");

num2 = atoi(token); // Second operand

// Assembly code for addition or subtraction

if (strcmp(operator, "+") == 0) {

printf("mov eax, %d\n", num1); // Load first operand into eax

printf("add eax, %d\n", num2); // Add second operand to eax

} else if (strcmp(operator, "-") == 0) {

printf("mov eax, %d\n", num1); // Load first operand into eax

printf("sub eax, %d\n", num2); // Subtract second operand from eax

} else if (strcmp(operator, "\*") == 0) {

printf("mov eax, %d\n", num1); // Load first operand into eax

printf("mul eax, %d\n", num2); // Multiply eax by second operand

} else if (strcmp(operator, "/") == 0) {

printf("mov eax, %d\n", num1); // Load first operand into eax

printf("xor edx, edx\n"); // Clear edx before division

printf("div %d\n", num2); // Divide eax by second operand (result in eax)

} else {

printf("Error: Unsupported operator\n");

return;

}

// Exit the program

printf("mov ebx, 0\n");

printf("int 0x80\n");

}

int main() {

char expr[100];

// Input arithmetic expression from user (example: "5 + 3")

printf("Enter an arithmetic expression (e.g., 5 + 3): ");

fgets(expr, sizeof(expr), stdin);

// Remove trailing newline character if any

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

// Generate assembly code for the expression

generateAssembly(expr);

return 0;

}

