

**NAME: S Venkata Sai Naveen Kumar (192425176)**

**COURSE NAME : COMPILER DESIGN FOR CODE ANALYSIS**

**COURSE CODE : CSA1423**

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

C PROGRAMMING CODE:

#include <stdio.h> #include <ctype.h> int main() { char line[1000]; int characters = 0, words = 0, lines = 1; // lines = 1 for single input line int i = 0, inWord = 0; printf("Enter a line of text:\n"); fgets(line, sizeof(line), stdin); while (line[i] != '\0') { characters++; if (isspace(line[i])) { inWord = 0; if (line[i] == '\n') { lines++; // if line has newline (usually last char)

}

} else if (inWord == 0) { inWord = 1; words++;

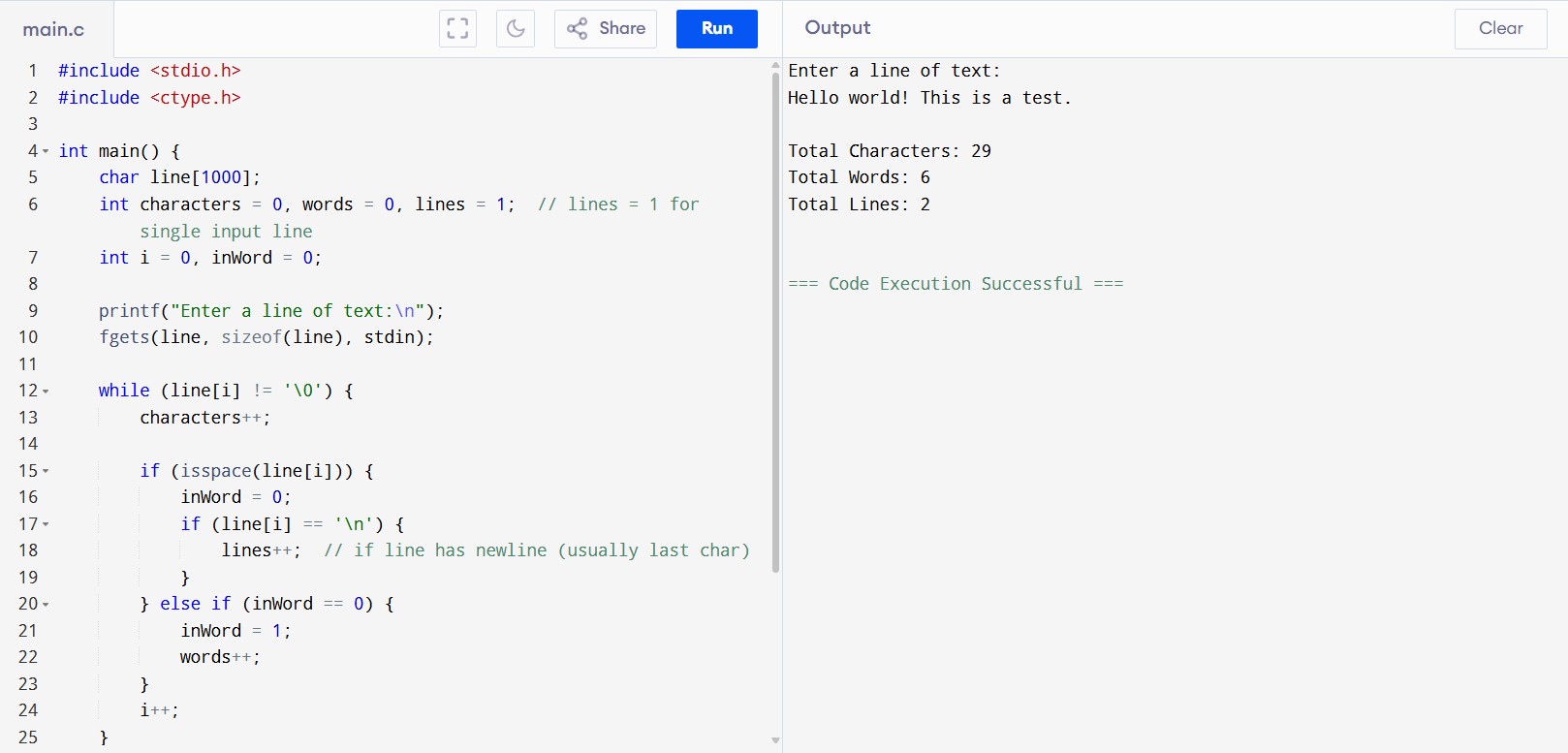
} i++;

}

printf("Total Characters: %d\n", characters); printf("Total Words: %d\n", words); printf("Total Lines: %d\n", lines); return 0;

}

**OUTPUT:**



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

C PROGRAMMING CODE:

#include <stdio.h> #include <string.h> struct Symbol { char name[20]; char type[20];

}; struct Symbol table[100]; int count = 0; void insert() { char name[20], type[20]; printf("Enter symbol name: "); scanf("%s", name); for (int i = 0; i < count; i++) { if (strcmp(table[i].name, name) == 0) { printf("Symbol already exists!\n");

return;

}

}

printf("Enter symbol type: "); scanf("%s", type); strcpy(table[count].name, name); strcpy(table[count].type, type); count++; printf("Symbol inserted.\n");

}

void search() { char name[20]; printf("Enter symbol name to search: "); scanf("%s", name); for (int i = 0; i < count; i++) { if (strcmp(table[i].name, name) == 0) { printf("Found: %s - %s\n", table[i].name, table[i].type);

return;

}

}

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

} void display() { if (count == 0) { printf("Symbol table is empty.\n"); return;

}

printf("Symbol Table:\nName\tType\n"); for (int i = 0; i < count; i++) { printf("%s\t%s\n", table[i].name, table[i].type);

}

}

int main() { int choice; while (1) { printf("\n1. Insert\n2. Search\n3. Display\n4. Exit\n"); printf("Enter choice: "); scanf("%d", &choice);

if (choice == 1) { insert();

} else if (choice == 2) { search();

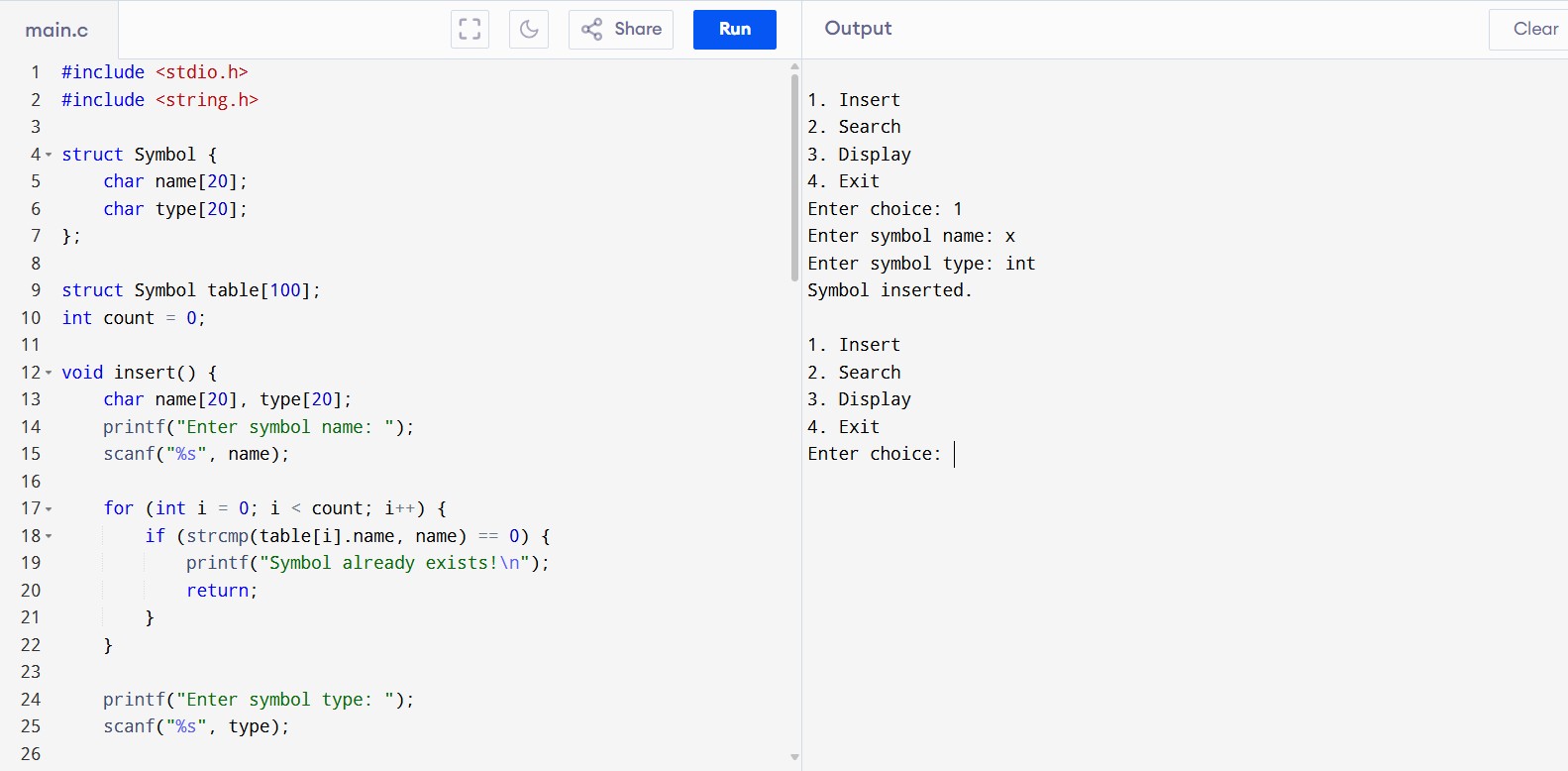
} else if (choice == 3) { display();

} else if (choice == 4) { break; } else { printf("Invalid choice.\n");

} } return 0;

}

**OUTPUT:**



**10. 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**

**.**

C PROGRAMMING CODE:

#include <stdio.h> char input[100]; int pos = 0; int expr(); int term(); int factor(); int expr() { if (!term()) return 0; while (input[pos] == '+' || input[pos] == '-') { pos++; if (!term()) return 0;

} return 1;

} int term() { if (!factor()) return 0; while (input[pos] == '\*' || input[pos] == '/') { pos++; if (!factor()) return 0;

} return 1;

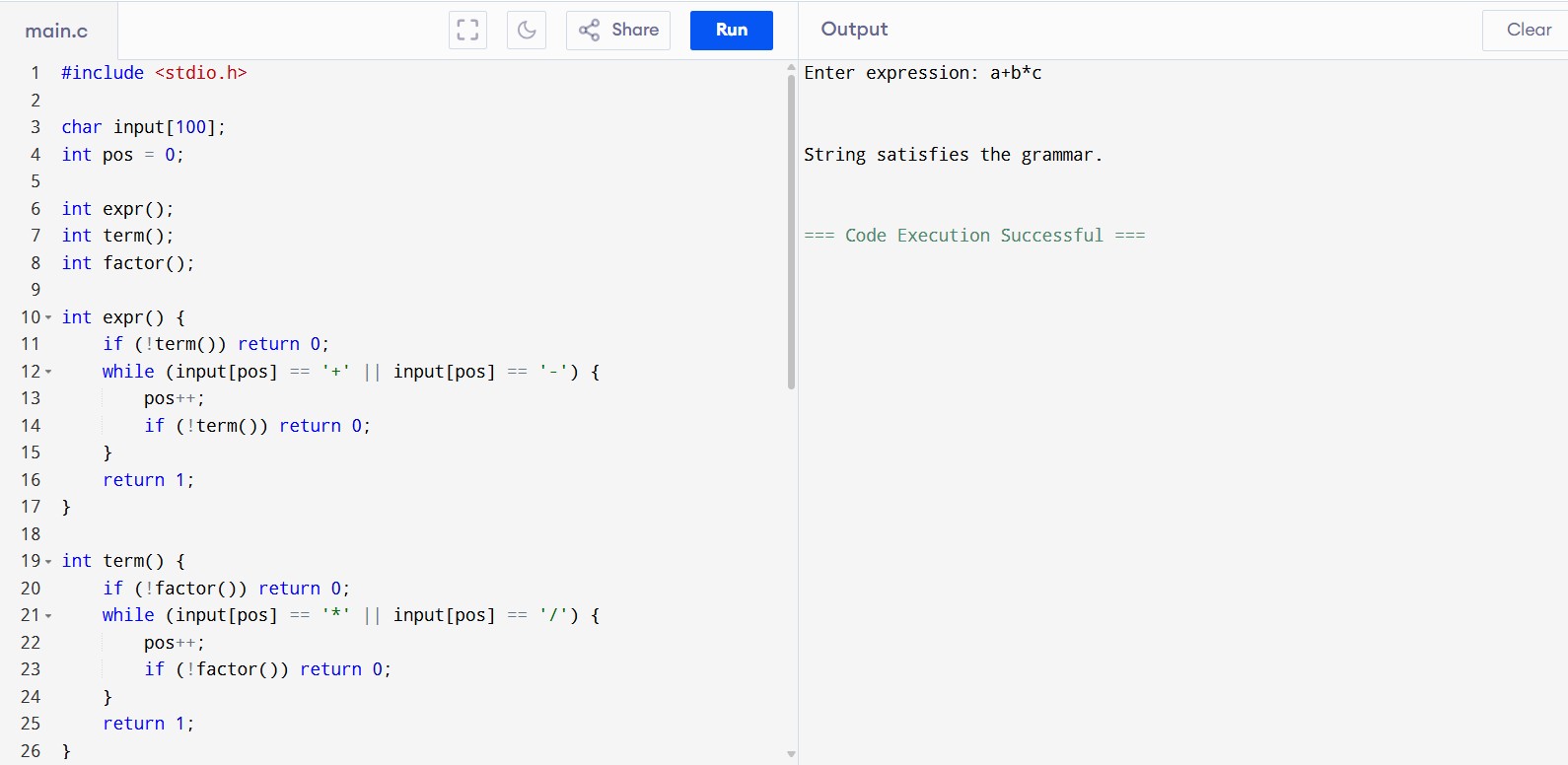
}

int factor() { if (input[pos] == '(') { pos++; if (!expr()) return 0; if (input[pos] == ')') { pos++; return 1; } return 0;

} else if (input[pos] >= 'a' && input[pos] <= 'z') { pos++; return 1; } return 0; } int main() { printf("Enter expression: "); scanf("%s", input); if (expr() && input[pos] == '\0') printf("String satisfies the grammar.\n"); else printf("String does not satisfy the grammar.\n"); return 0;

}

**OUTPUT:**



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

C PROGRAMMING CODE:

#include <stdio.h> char input[100]; int pos = 0; int expr(); int term(); int factor(); int expr() { if (!term()) return 0; while (input[pos] == '+' || input[pos] == '-') { pos++; if (!term()) return 0

} return 1; } int term() { if (!factor()) return 0; while (input[pos] == '\*' || input[pos] == '/') { pos++; if (!factor()) return 0;

} return 1; } int factor() { if (input[pos] == '(') { pos++; if (!expr()) return 0; if (input[pos] == ')') { pos++; return 1; } return 0;

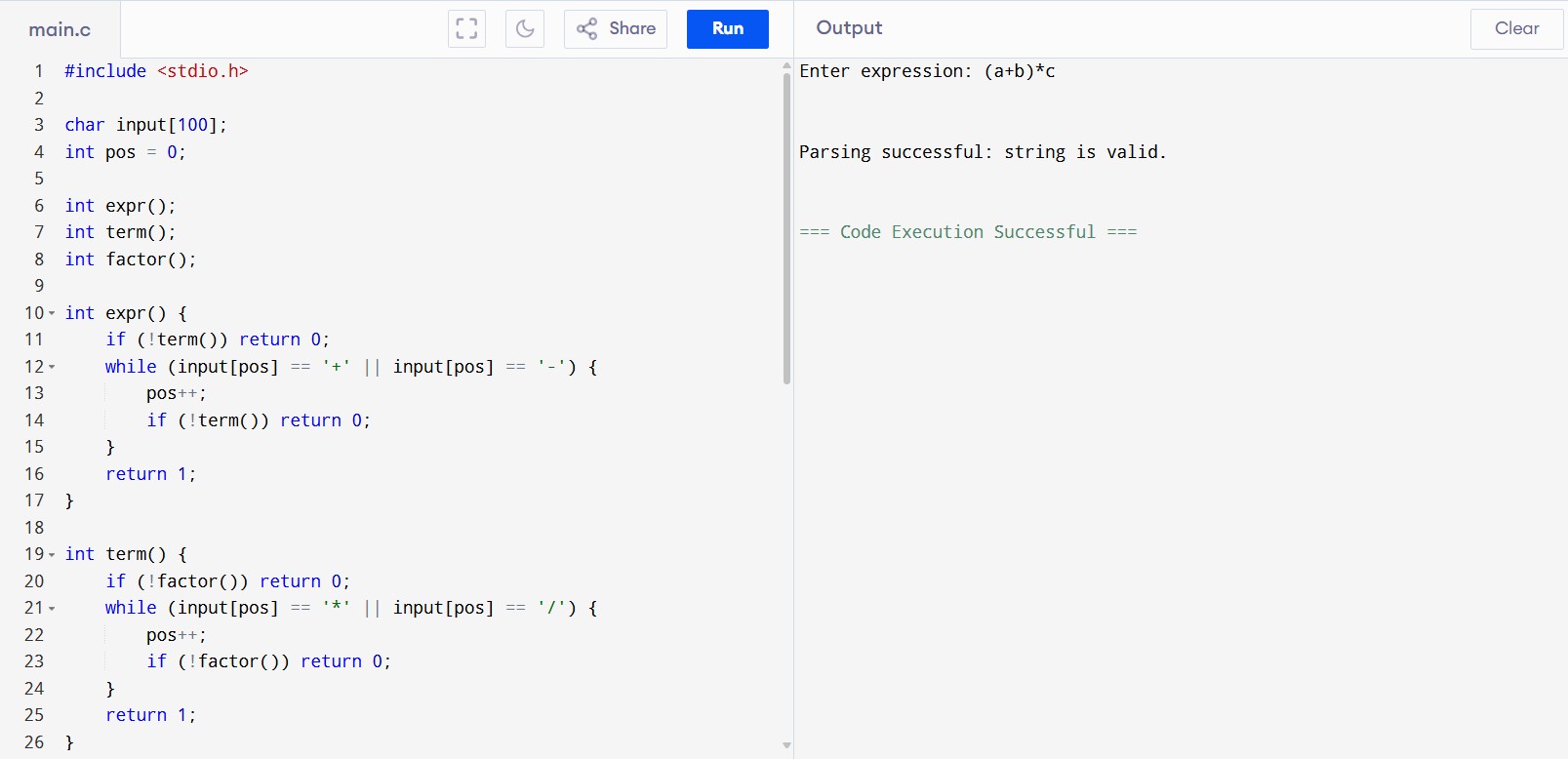
} else if (input[pos] >= 'a' && input[pos] <= 'z') { pos++; return 1; } return 0;

}

int main() { printf("Enter expression: "); scanf("%s", input); if (expr() && input[pos] == '\0') printf("Parsing successful: string is valid.\n"); else printf("Parsing failed: string is invalid.\n"); return 0;

}

OUTPUT:



**12. 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.**

C PROGRAMMING CODE:

#include <stdio.h> #include <ctype.h> char expr[100]; int pos = 0; int expression(); int term(); int factor(); int exponent(); int expression() { int val = term(); while (expr[pos] == '+' || expr[pos] == '-') { char op = expr[pos++]; int val2 = term(); if (op == '+') val += val2; else val -= val2;

} return val;

}

int term() { int val = exponent(); while (expr[pos] == '\*' || expr[pos] == '/') { char op = expr[pos++]; int val2 = exponent(); if (op == '\*') val \*= val2; else val /= val2;

}

return val; } int exponent() { int val = factor(); if (expr[pos] == '^') { pos++; int val2 = exponent(); // right associative int result = 1; for (int i = 0; i < val2; i++) result \*= val; val = result; } return val;

}

int factor() { int val = 0; if (expr[pos] == '(') { pos++; val = expression(); if (expr[pos] == ')') pos++; } else if (isdigit(expr[pos])) { while (isdigit(expr[pos])) { val = val \* 10 + (expr[pos] - '0'); pos++;

} } return val;

}

int main() { printf("Enter expression: "); fgets(expr, sizeof(expr), stdin); for (int i = 0; expr[i] != '\0'; i++) { if (expr[i] == '\n') { expr[i] = '\0'; break;

}

}

int result = expression(); if (expr[pos] == '\0') printf("Result = %d\n", result); else printf("Invalid expression.\n"); return 0;

}

**OUTPUT:**

