**1. Lexical Analyzer for Identifiers, Constants, and Operators**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_ID\_LEN 31

// Function to check if a character is an operator

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '=');

}

// Function to check if a word is a keyword

int isKeyword(const char \*word) {

const char \*keywords[] = {"int", "float", "if", "else", "return", "while", "for"};

for (int i = 0; i < 7; i++) {

if (strcmp(word, keywords[i]) == 0) return 1;

}

return 0;

}

// Function to analyze tokens

void lexicalAnalyzer(FILE \*file) {

char ch, buffer[MAX\_ID\_LEN + 1];

int index = 0;

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

// Ignore whitespace

if (isspace(ch)) continue;

// Ignore single-line comments

if (ch == '/' && (ch = fgetc(file)) == '/') {

while ((ch = fgetc(file)) != EOF && ch != '\n');

continue;

}

// Ignore multi-line comments

if (ch == '/' && (ch = fgetc(file)) == '\*') {

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

if (ch == '\*' && (ch = fgetc(file)) == '/') break;

}

continue;

}

// Identifiers & Keywords

if (isalpha(ch) || ch == '\_') {

index = 0;

do {

if (index < MAX\_ID\_LEN) buffer[index++] = ch;

ch = fgetc(file);

} while (isalnum(ch) || ch == '\_');

buffer[index] = '\0';

ungetc(ch, file);

if (isKeyword(buffer))

printf("Keyword: %s\n", buffer);

else

printf("Identifier: %s\n", buffer);

}

// Constants

else if (isdigit(ch)) {

index = 0;

do {

buffer[index++] = ch;

ch = fgetc(file);

} while (isdigit(ch) || ch == '.');

buffer[index] = '\0';

ungetc(ch, file);

printf("Constant: %s\n", buffer);

}

// Operators

else if (isOperator(ch)) {

printf("Operator: %c\n", ch);

}

// Other Symbols

else {

printf("Symbol: %c\n", ch);

}

}

}

int main() {

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

if (!file) {

printf("Error opening file.\n");

return 1;

}

lexicalAnalyzer(file);

fclose(file);

return 0;

}



**2. Detecting Comments**

**#include <stdio.h>**

**#include <string.h>**

**void checkComment(char \*line) {**

**if (strncmp(line, "//", 2) == 0)**

**printf("Single-line comment detected.\n");**

**else if (strncmp(line, "/\*", 2) == 0 && strstr(line, "\*/"))**

**printf("Multi-line comment detected.\n");**

**else**

**printf("Not a comment.\n");**

**}**

**int main() {**

**char line[100];**

**printf("Enter a line: ");**

**fgets(line, sizeof(line), stdin);**

**checkComment(line);**

**return 0;**

**}**

****

**3. Validating Arithmetic Operators**

**#include <stdio.h>**

**int isValidOperator(char op) {**

**return (op == '+' || op == '-' || op == '\*' || op == '/');**

**}**

**int main() {**

**char op;**

**printf("Enter an operator: ");**

**scanf(" %c", &op);**

**if (isValidOperator(op))**

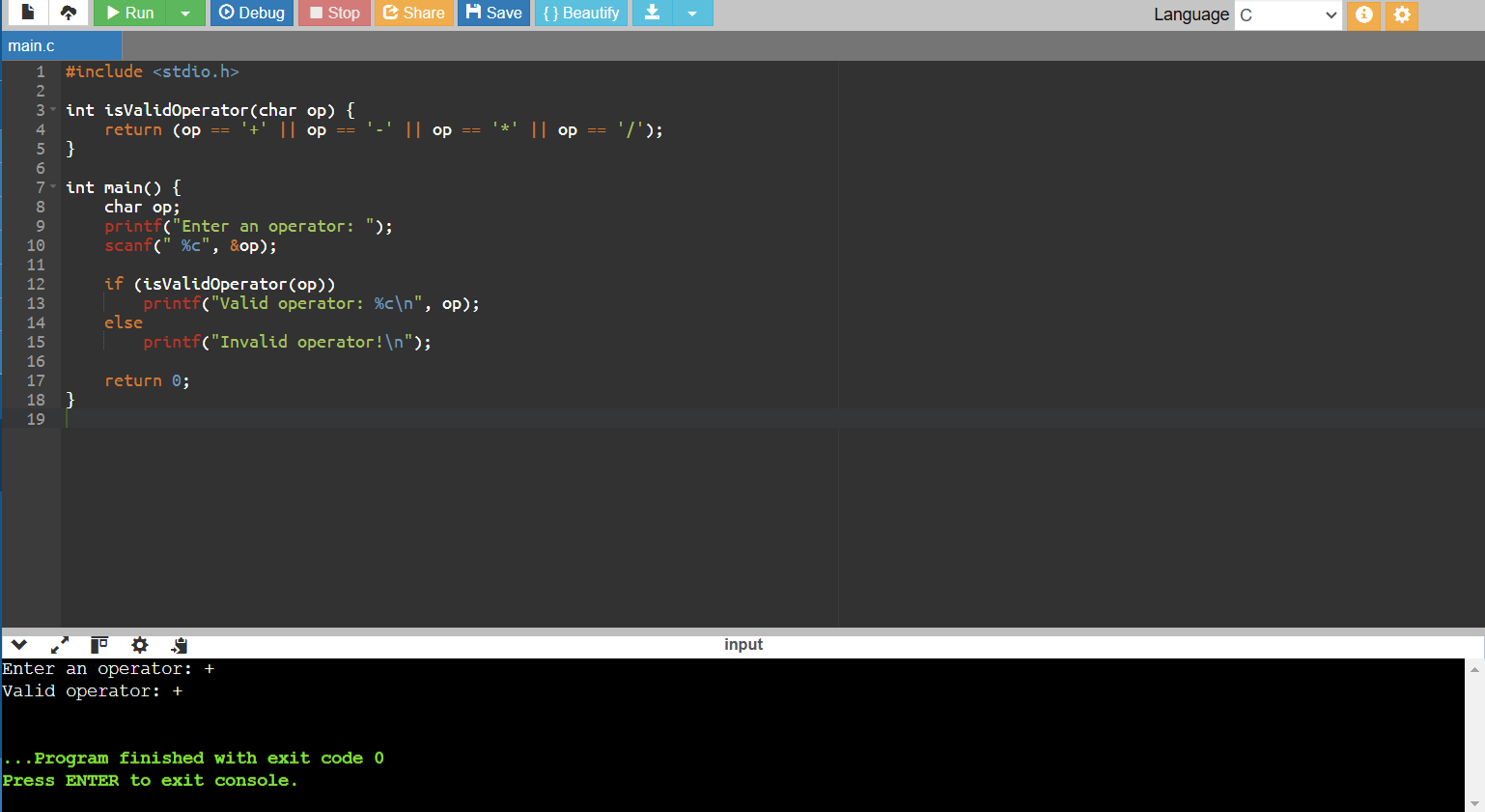
**printf("Valid operator: %c\n", op);**

**else**

**printf("Invalid operator!\n");**

**return 0;**

**}**

****

**4. Counting Whitespaces and Newlines**

**#include <stdio.h>**

**int main() {**

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

**if (!file) {**

**printf("Error opening file.\n");**

**return 1;**

**}**

**char ch;**

**int spaces = 0, newlines = 0;**

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

**if (ch == ' ' || ch == '\t')**

**spaces++;**

**else if (ch == '\n')**

**newlines++;**

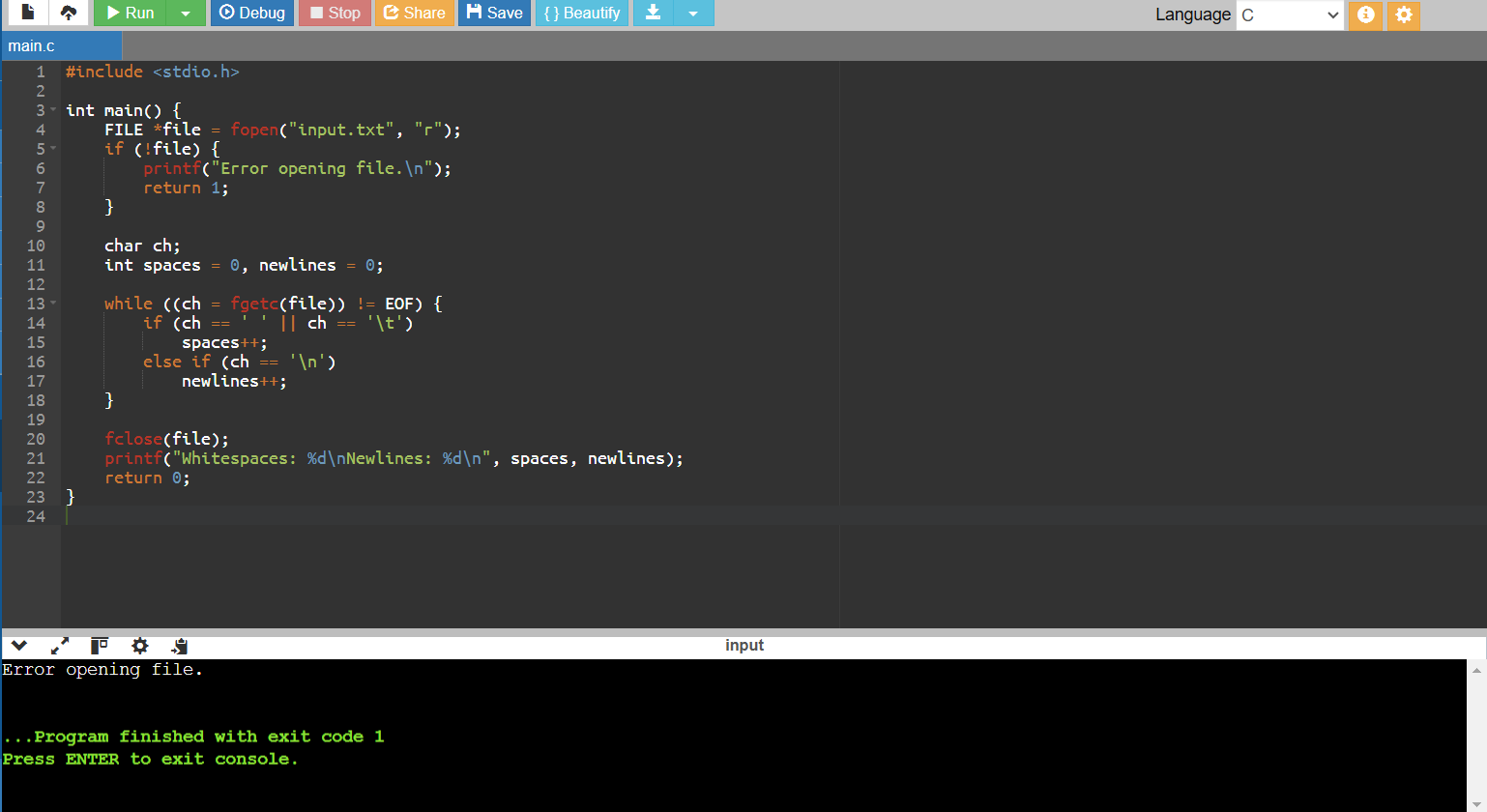
**}**

**fclose(file);**

**printf("Whitespaces: %d\nNewlines: %d\n", spaces, newlines);**

**return 0;**

**}**

****

**5. Valid Identifier Checker**

**#include <stdio.h>**

**#include <ctype.h>**

**int isValidIdentifier(char \*str) {**

**if (!isalpha(str[0]) && str[0] != '\_') return 0;**

**for (int i = 1; str[i] != '\0'; i++) {**

**if (!isalnum(str[i]) && str[i] != '\_') return 0;**

**}**

**return 1;**

**}**

**int main() {**

**char identifier[50];**

**printf("Enter an identifier: ");**

**scanf("%s", identifier);**

**if (isValidIdentifier(identifier))**

**printf("Valid identifier.\n");**

**else**

**printf("Invalid identifier.\n");**

**return 0;**

**}**

****

**6. Left Recursion Elimination**

**#include <stdio.h>**

**#include <string.h>**

**void eliminateLeftRecursion(char \*grammar) {**

**char A = grammar[0];**

**char alpha[10], beta[10];**

**sscanf(grammar + 3, "%[^|]|%s", alpha, beta);**

**printf("Without Left Recursion:\n");**

**printf("%c -> %s%c'\n", A, beta, A);**

**printf("%c' -> %s%c' | ε\n", A, alpha, A);**

**}**

**int main() {**

**char grammar[50];**

**printf("Enter grammar (A->Aα|β): ");**

**scanf("%s", grammar);**

**eliminateLeftRecursion(grammar);**

**return 0;**

**}**

****

**7. Left Factoring Elimination**

**#include <stdio.h>**

**#include <string.h>**

**void eliminateLeftFactoring(char \*grammar) {**

**char A = grammar[0];**

**char alpha[10], beta[10];**

**sscanf(grammar + 3, "%[^|]|%s", alpha, beta);**

**printf("Without Left Factoring:\n");**

**printf("%c -> %s%c'\n", A, alpha, A);**

**printf("%c' -> %s | ε\n", A, beta);**

**}**

**int main() {**

**char grammar[50];**

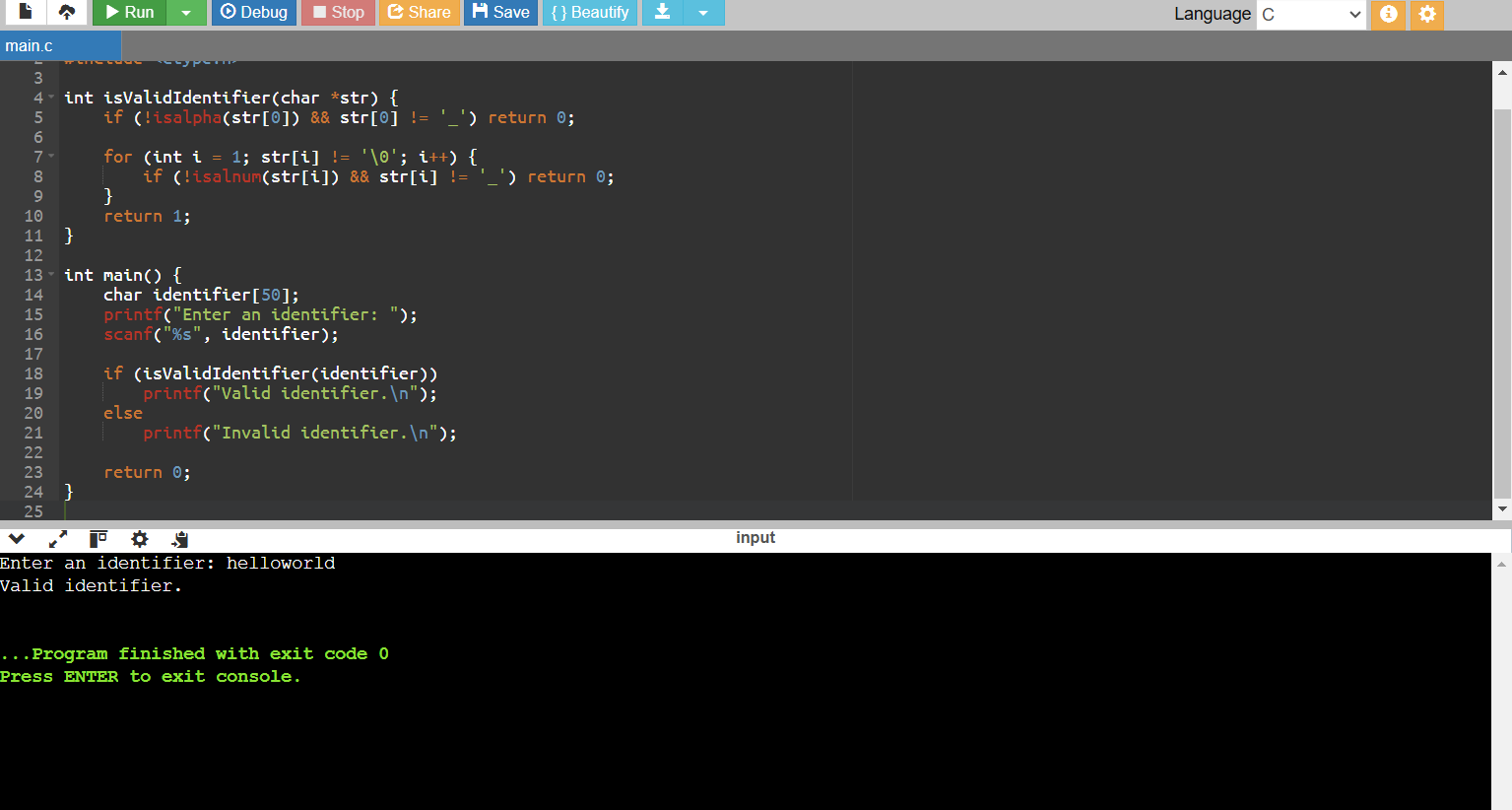
**printf("Enter grammar (A->αB|αC): ");**

**scanf("%s", grammar);**

**eliminateLeftFactoring(grammar);**

**return 0;**

**}**

****