CSA1428 - Compiler Design

LAB ACTIVITY-1

Qn 1) The lexical analyzer should ignore redundant spaces, tabs, and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, and operators using the C program.

192311044

Girija B

17.02.2025

CSA1428 - Compiler Design

LAB ACTIVITY-1

Qn 1) The lexical analyzer should ignore redundant spaces, tabs, and new lines. It

should also ignore comments. Although the syntax specification states that identifiers

can be arbitrarily long, you may restrict the length to some reasonable value. Develop a

lexical Analyzer to identify identifiers, constants, and operators using the C program.

Code:

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_IDENTIFIER\_LENGTH 31

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 isOperator(char ch) {

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

}

void lexicalAnalyzer(char \*code) {

char token[100];

int index = 0;

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

if (isalnum(code[i]) || code[i] == '\_') {

index = 0;

while (isalnum(code[i]) || code[i] == '\_') {

token[index++] = code[i++];

}

token[index] = '\0';

i--;

if (isValidIdentifier(token)) {

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

} else {

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

}

} else if (isOperator(code[i])) {

printf("Operator: %c\n", code[i]);

}

}

}

int main() {

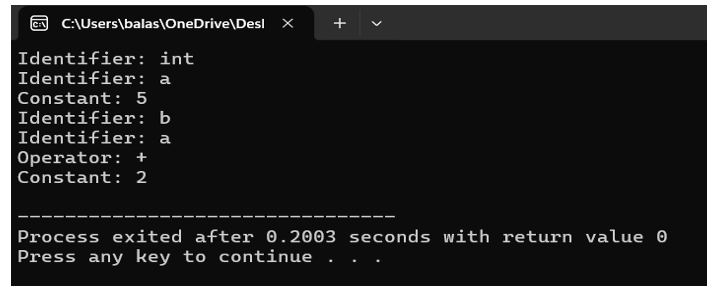
char code[] = "int a = 5; b = a + 2;";

lexicalAnalyzer(code);

return 0;

}

Output:



Qn2) Extend the lexical Analyzer to Check comments, dened as follows in C:

a) A comment begins with // and includes all characters until the end of that line.

b) A comment begins with /\* and includes all characters through the next occurrence of

the character sequence \*/Develop a lexical Analyzer to identify whether a given line is a

comment or not.

Code:

#include <stdio.h>

#include <string.h>

int isComment(char \*line) {

if (strncmp(line, "//", 2) == 0) return 1;

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

return 0;

}

void analyzeComments(char \*code) {

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

if (code[i] == '/' && code[i + 1] == '/') {

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

return;

}

if (code[i] == '/' && code[i + 1] == '\*') {

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

return;

}

}

}

int main() {

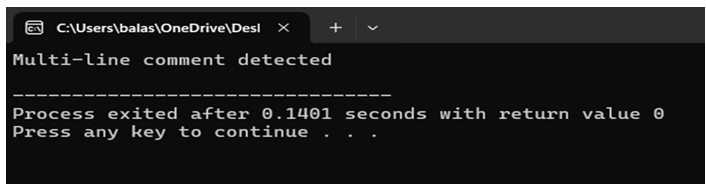
char code[] = "/\* This is a multi-line comment \*/\n// This is a single-line comment";

analyzeComments(code);

return 0;

}

Output:



Qn3) Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/

using regular Arithmetic operators .

Code:

#include <stdio.h>

int isOperator(char ch) {

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

}

void analyzeOperators(char \*code) {

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

if (isOperator(code[i])) {

printf("Operator: %c\n", code[i]);

}

}

}

int main() {

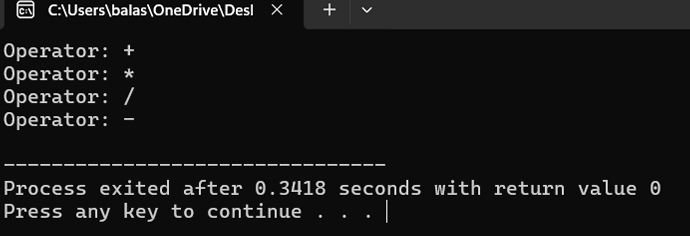
char code[] = "a = b + c \* d / e - f;";

analyzeOperators(code);

return 0;

}

Output:



Qn 4) Design a lexical Analyzer to find the number of whitespaces and newline

characters.

Code:

#include <stdio.h>

#include <ctype.h>

void countWhitespaces(char \*code) {

int whitespaceCount = 0, newlineCount = 0;

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

if (isspace(code[i])) {

if (code[i] == '\n') newlineCount++;

else whitespaceCount++;

}

}

printf("Whitespaces: %d, Newlines: %d\n", whitespaceCount, newlineCount);

}

int main() {

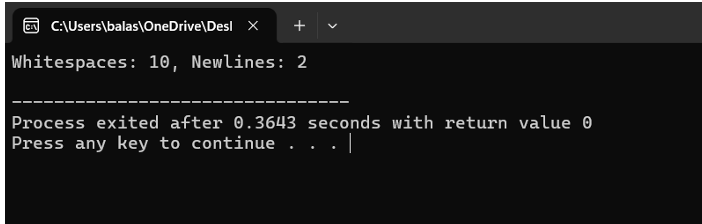
char code[] = "int a = 5; \n b = a + 2; \n";

countWhitespaces(code);

return 0;

}

Output:



Qn 5) Develop a lexical Analyzer to test whether a given identifier is valid or not.

Code:

#include <stdio.h>

#include <ctype.h>

#include <string.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;

}

void checkIdentifier(char \*identifier) {

if (isValidIdentifier(identifier)) {

printf("\"%s\" is a valid identifier.\n", identifier);

} else {

printf("\"%s\" is not a valid identifier.\n", identifier);

}

}

int main() {

char id1[] = "var\_name";

char id2[] = "2ndVariable";

checkIdentifier(id1);

checkIdentifier(id2);

return 0;

}

Output:

