**Department of CSE**

**Compiler Lab (CSE 352)**

**Lab Report 04**

Simulation of a Lexical Analyzer to Validate Operators

**Submitted By :**

**Fahima Abida Chowdhury**

ID: 0432220005101135

Semester: Spring-2025

Batch: 52(6B1)

**Submitted To :**

**Md. Ismail**

Lecturer

Department of CSE

University of Information Technology and Sciences

**Experiment No.: 4**

**Experiment Name: Simulation of a Lexical Analyzer to Validate Operators**

**Problem Statement**

The objective of this experiment is to write a C program that simulates a **lexical analyzer**, specifically designed to identify and validate different **operators** in a programming language. The lexical analyzer recognizes relational, logical, arithmetic, assignment, and bitwise operators by analyzing the input string character-by-character.

**Implementation**

#include <stdio.h>

#include <string.h>

int main() {

char s[5];

printf("Enter any operator: ");

fgets(s, sizeof(s), stdin);

size\_t len = strlen(s);

if (len > 0 && s[len - 1] == '\n') {

s[len - 1] = '\0';

}

switch (s[0]) {

case '>':

if (s[1] == '=') printf("Greater than or equal\n");

else printf("Greater than\n");

break;

case '<':

if (s[1] == '=') printf("Less than or equal\n");

else printf("Less than\n");

break;

case '=':

if (s[1] == '=') printf("Equal to\n");

else printf("Assignment\n");

break;

case '!':

if (s[1] == '=') printf("Not equal\n");

else printf("Bitwise NOT (Invalid operator!)\n");

break;

case '&':

if (s[1] == '&') printf("Logical AND\n");

else printf("Bitwise AND\n");

break;

case '|':

if (s[1] == '|') printf("Logical OR\n");

else printf("Bitwise OR\n");

break;

case '+':

if (s[1] == '+') printf("Increment operator\n");

else printf("Addition\n");

break;

case '-':

if (s[1] == '-') printf("Decrement operator\n");

else printf("Subtraction\n");

break;

case '\*':

printf("Multiplication\n");

break;

case '/':

printf("Division\n");

break;

case '%':

printf("Modulus\n");

break;

default:

printf("Not a valid operator\n");

}

return 0;

}

**Input**

The program takes a single operator or operator pair as input (e.g., +, >=, !=, ++).

**Example Inputs:**

>=

+

!=

&&

\*\*

**Output**

Based on the input, the output describes the type of operator:

**Example Outputs:**

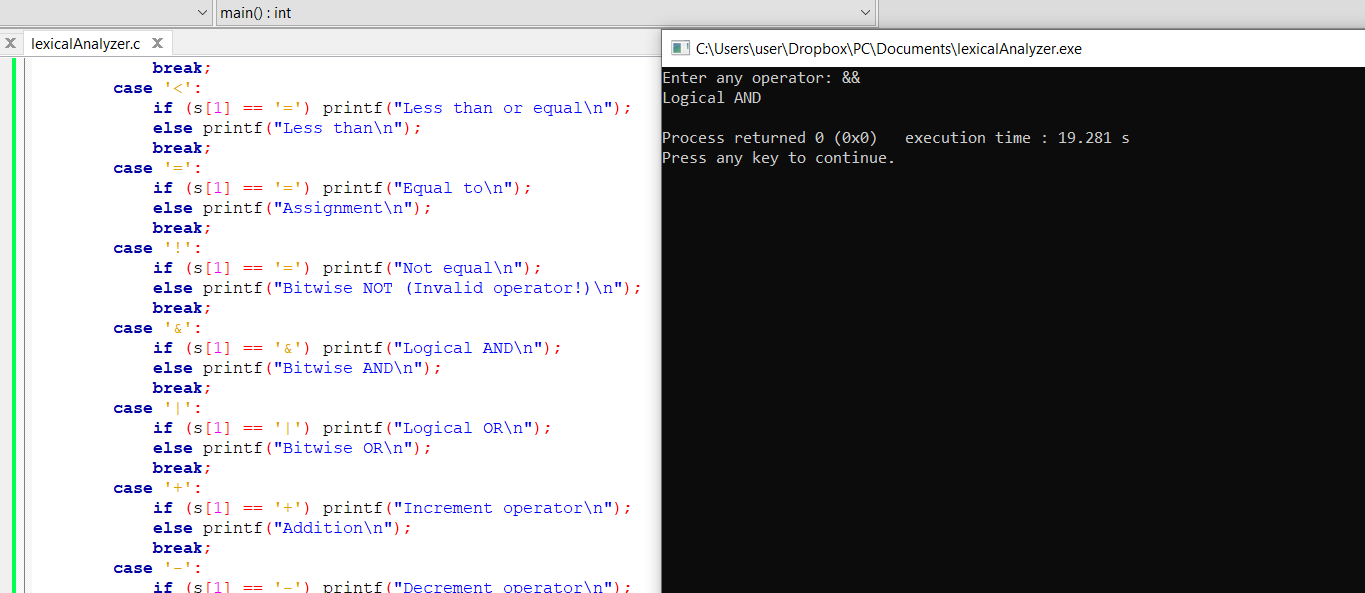
Greater than or equal

Addition

Not equal

Logical AND

Not a valid operator

****

**Conclusion**

This lab effectively simulates a simple lexical analyzer by validating various C languageoperators through a switch-case structure. It distinguishes between:

* **Relational operators** (<, <=, >, >=)
* **Equality operators** (==, !=)
* **Logical operators** (&&, ||)
* **Arithmetic operators** (+, -, \*, /, %)
* **Bitwise operators** (&, |)
* **Unary operators** (++, --)

This experiment enhances the understanding of how a lexical analyzer interprets tokens and symbols, a foundational concept in compiler design.