# North Western University

Compiler Design

(User manual)

Course Code: CSE-4104

**Developed By:**

Mushfiqul Islam

20201104010

Tanishat Islam Tazmim

20201093010

Sanjida Hossain

20201073010

Department: CSE

4th year, 1st semester

## Contents

## Abstract:

The purpose of this lab project was to design and implement a lexical analyzer, also known as a lexer, for a programming language. The lexer is an essential component of a compiler or interpreter, responsible for breaking down the source code into tokens that can be processed further. This report discusses the design choices, implementation details, and showing symbol table of  this project.

## Introduction:

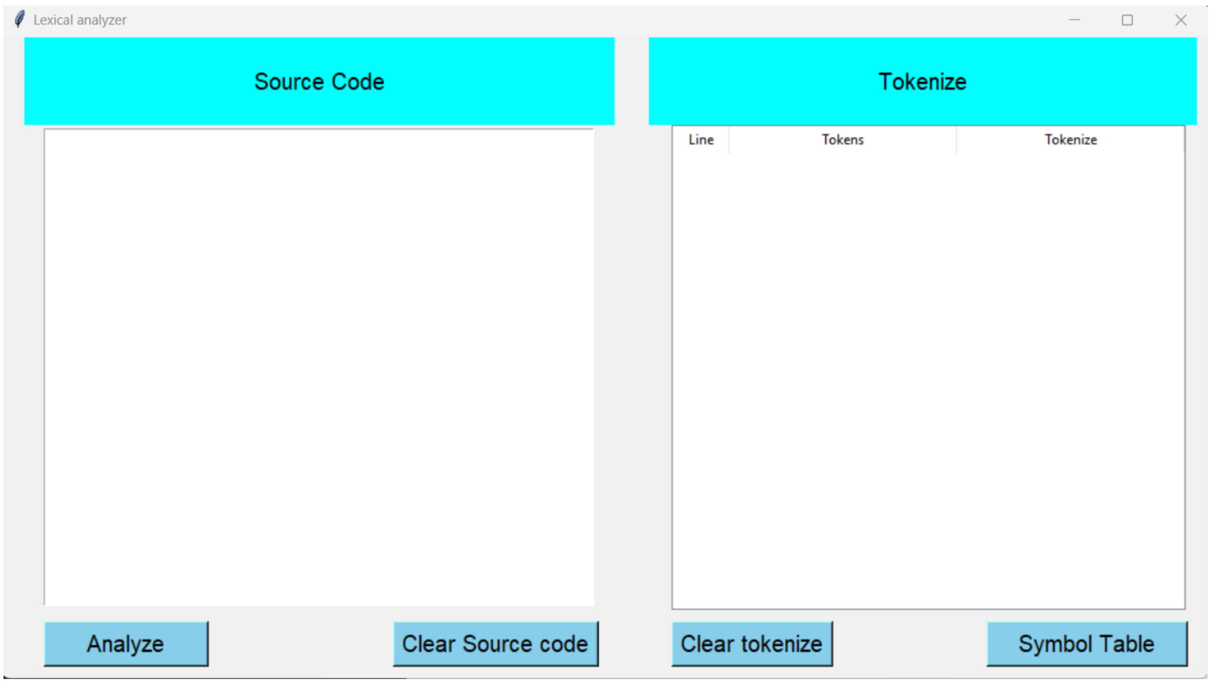
The lexical analyzer is the first phase of a compiler or interpreter that converts the source code into a more manageable form for subsequent analysis. The main objective of the lexical analyzer is to recognize and categorize the lexical units, or tokens, in the source code. These tokens include keywords, identifiers, numbers, operators, datatypes and punctuation symbols.

## Objectives:

The objective of this project is to create a program that can analyze the source code of a programming language and break it down into individual tokens. The program should accurately recognize and categorize tokens such as keywords, identifiers, literals, operators, and punctuation symbols. It should also handle errors and provide informative error messages. The project aims to create an efficient and reliable tool that can integrate with other components of a compiler or interpreter for further analysis and processing of the source code.

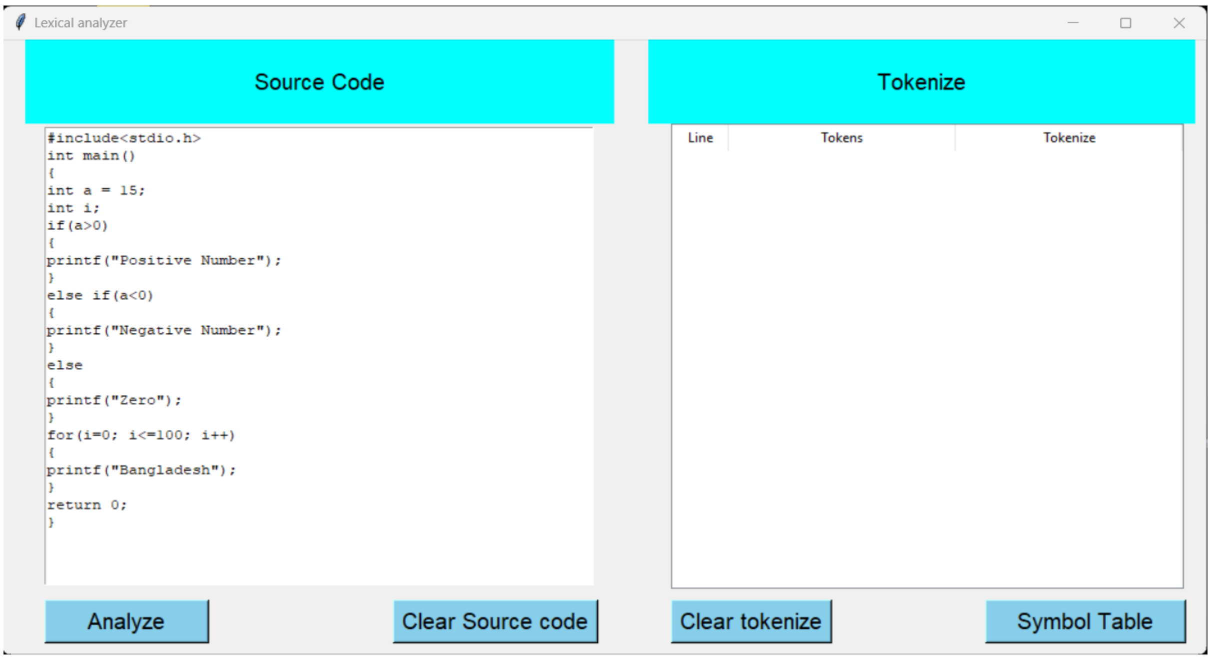
## Design And Implementation:

The lexer was implemented using the Python programming language and special use of python tkinter library. The design involved creating a set of c language code to match the different token types and then applying them to the source code in a sequential manner. The lexer followed the longest match rule, where it selects the longest matching token from the input stream. There are thress steps are input source code, show tokenization and show symbol table.



**Project Interface**

## Input Buffering:

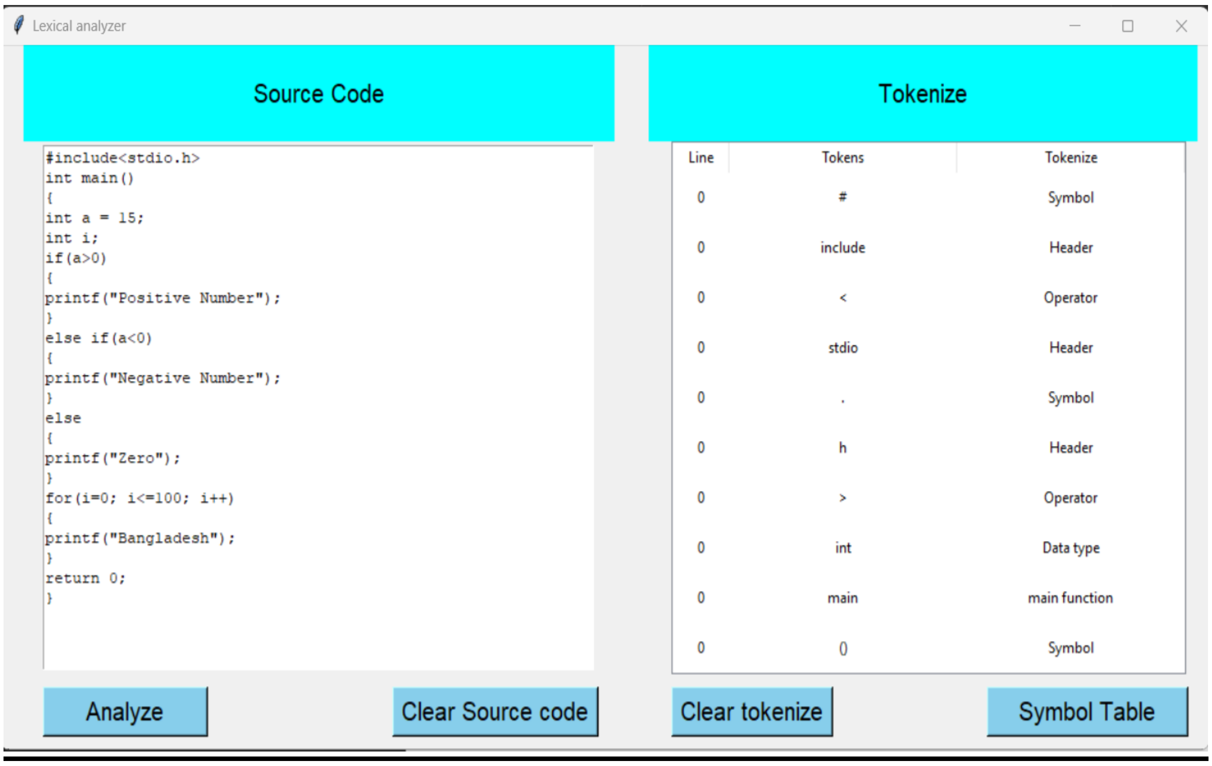


```
#include<stdio.h>
int main()
{
int a = 15;
int i;
if(a>0)
{
printf("Positive Number");
}
else if(a<0)
{
printf("Negative Number");
}
else
{
printf("Zero");
}
for(i=0; i<=100; i++)
{
printf("Bangladesh");
}
return 0;
}
```

The source code has been input in the input field. Only C language is supported as source code.

Clear source code button is used for clear the input field.

## Tokenization:

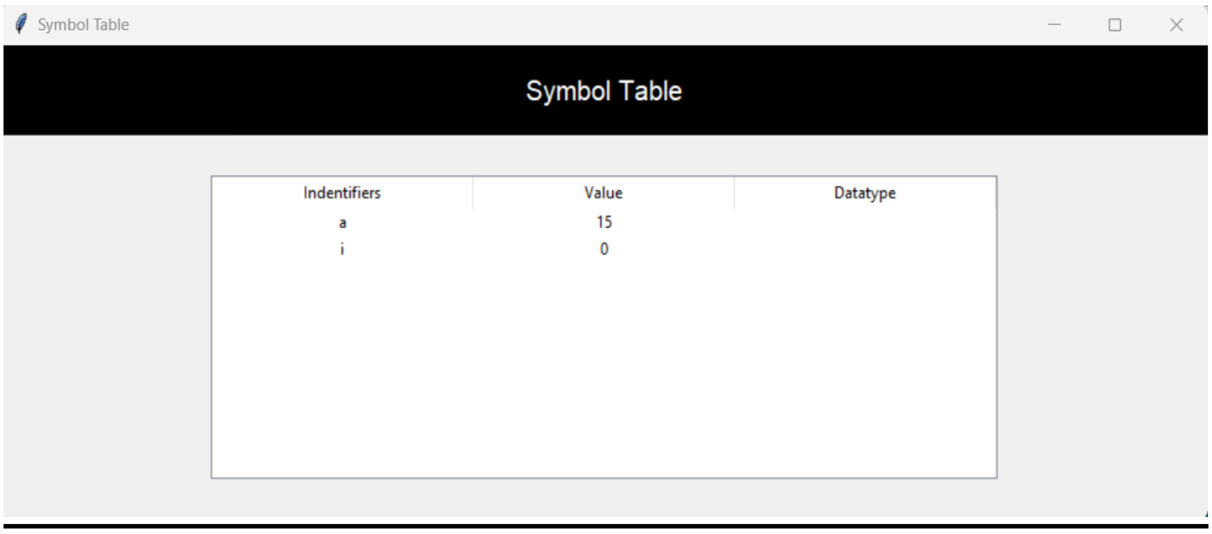


```
#include<stdio.h>
int main()
{
int a = 15;
int i;
if(a>0)
{
printf("Positive Number");
}
else if(a<0)
{
printf("Negative Number");
}
else
{
printf("Zero");
}
for(i=0; i<=100; i++)
{
printf("Bangladesh");
}
return 0;
}
```

| Line | Tokens | Tokenize |
|---|---|---|
| 0 | # | Symbol |
| 0 | include | Header |
| 0 | < | Operator |
| 0 | stdio | Header |
| 0 | . | Symbol |
| 0 | h | Header |
| 0 | > | Operator |
| 0 | int | Data type |
| 0 | main | main function |
| 0 | () | Symbol |

The lexer iterated over the input buffer character by character, recognizing and categorizing each token it encountered. Source code were utilized to define the patterns for various tokens, such as keywords, identifiers, number, strings symbols etc. Whole process handled after clicking analyze button. And there is clear tokenize button for clear the tokenize table. The lexer maintained a symbol table to store and manage identifiers encountered during tokenization.

## Symbol Table:



Symbol Table

| Indentifiers | Value | Datatype |
|---|---|---|
| a | 15 | |
| i | 0 | |

Symbol is used for hold the identifiers of the source code. So in this symbol table hold the identifiers and the value of each identifiers.

## Conclusion:

In this project successfully implemented a lexer for a programming language. The design and implementation of the lexer allowed for accurate and efficient token recognition. The project achieved its objective of breaking down the source code into meaningful tokens, laying the foundation for subsequent phases of the compiler.

## Thanks to

Md. Shymon Islam

Lecturer

Department Of CSE

North Western University Khulna