NORTH WESTERN UNIVERSITY

Compiler Design

(User Manual)

Course Code-CSE-4104

Developed By:

Sneara parvin

20201074010

Parinita Roy Papri

20201076010

Umme Habiba Tuly

20201124010

Department: Bsc in cse 4-year

4th year,1st semester

Table of contents

Abstract:

The abstract of a lexical analyzer acts as the interface between the source code and the subsequent stages of the compiler or interpreter. It produces a stream of tokens that serve as input for the parser, which performs further syntactic analysis and constructs an abstract syntax tree (AST) representing the structure of the program. The lexer's output helps facilitate subsequent compilation or interpretation processes, enabling the compiler or interpreter to analyze and execute the program.

Introduction:

The lexical analyzer operates on a character stream and applies a set of predefined rules to recognize and classify tokens. These rules are typically defined using regular expressions or finite automata. As the lexer processes the input, it scans the text character by character, recognizing patterns that correspond to different token types. characters into tokens and assigns each token a specific category or class.

Objectives:

The main objectives of a lexical analyzer, also known as a lexer or scanner, are as follows:

1. Tokenization: The lexical analyzer breaks down the input source code into a sequence of tokens. Tokens are the smallest meaningful units of a programming language, such as keywords, identifiers, operators, literals, and punctuation symbols. The lexer identifies and categorizes each token based on predefined rules.

2. Removal of Whitespaces and Comments: The lexer eliminates unnecessary whitespace characters, including spaces, tabs, and newline characters, from the source code

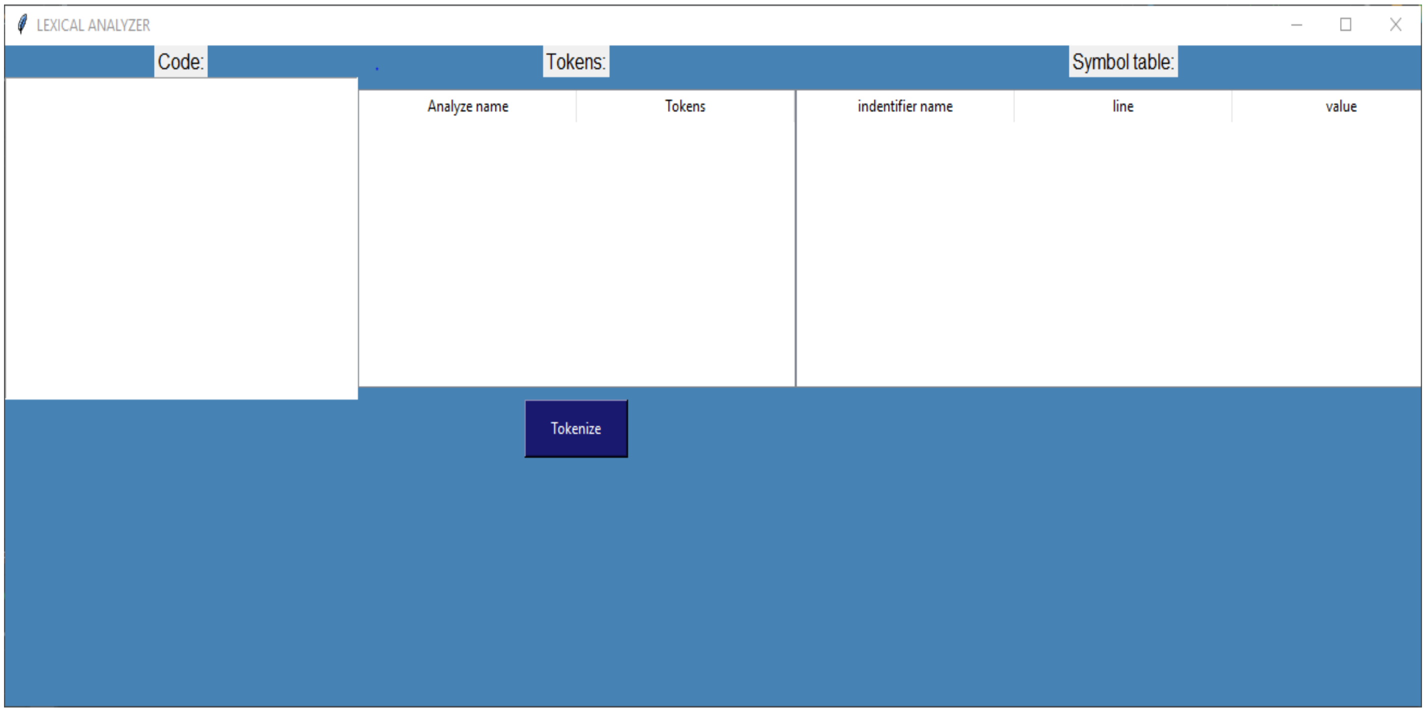
4. Symbol Table Generation: The lexer may generate a symbol table, which is a data structure used to store information about identifiers (variables, functions, classes) encountered during tokenization.

Overall, the main objective of a lexical analyzer is to transform the input source code into a stream of tokens, providing a foundation for the subsequent phases of a compiler or interpreter.

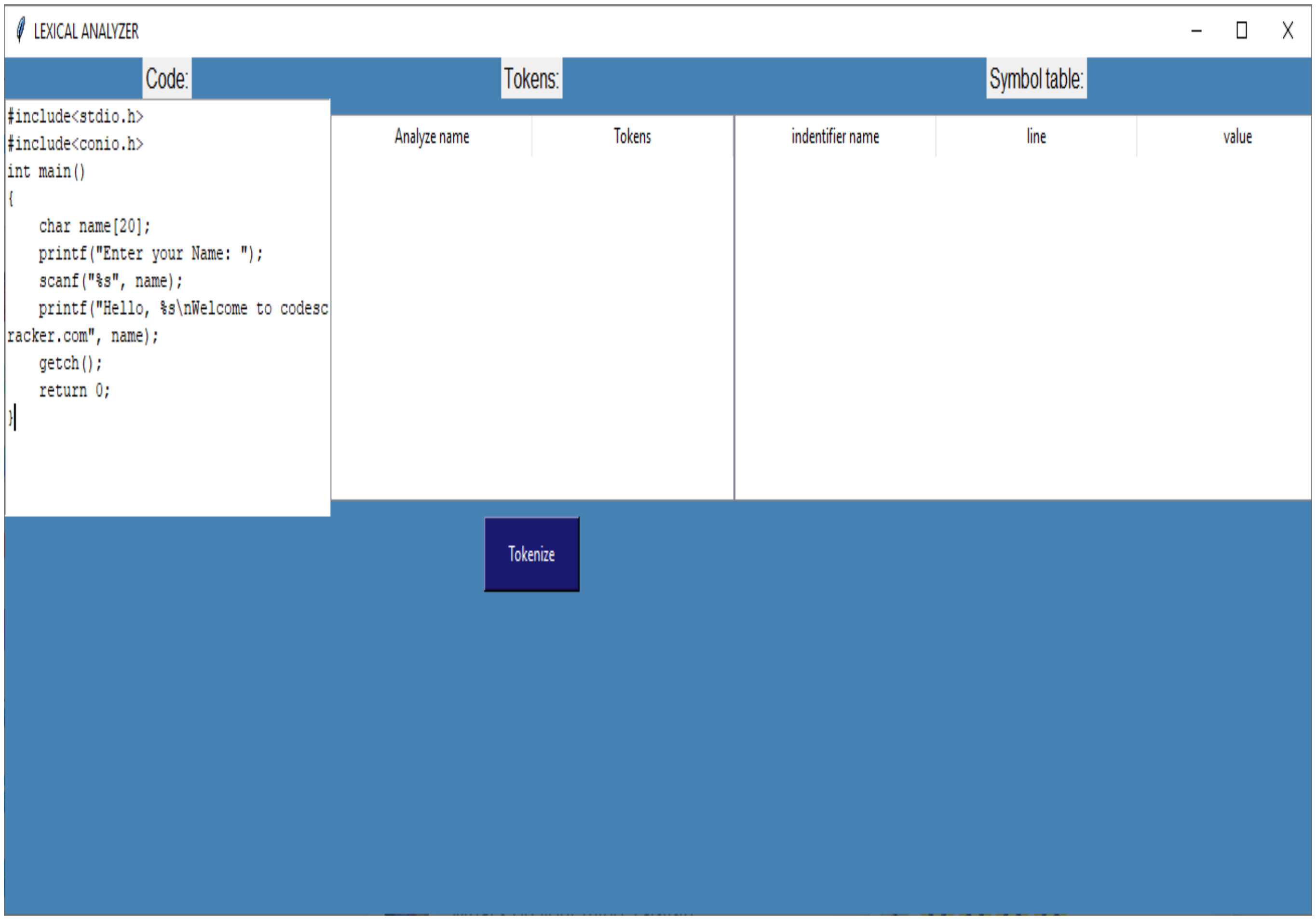
Design and Implementation:

A lexical analyzer, also known as a scanner, is responsible for breaking down the input source code into a sequence of tokens, which are the smallest meaningful units of a programming language. Here's a general outline of the design and implementation process:

Project interface



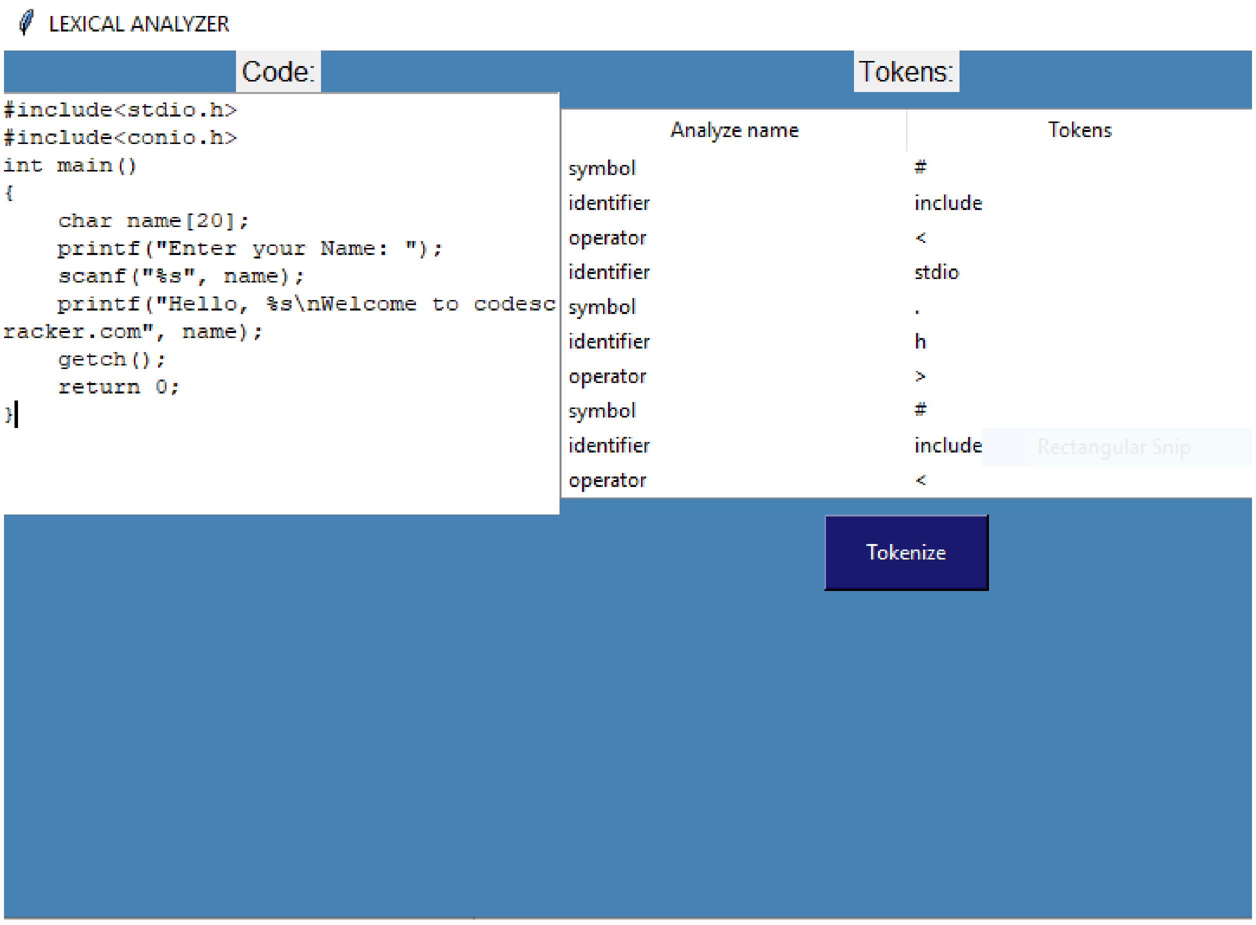
Input Buffering:



```
#include<stdio.h>
#include<conio.h>
int main()
{
    char name[20];
    printf("Enter your Name: ");
    scanf("%s", name);
    printf("Hello, %s\nWelcome to codesc
racker.com", name);
    getch();
    return 0;
}
```

The source code has been taken c language in input.

<u>Tokenization:</u>



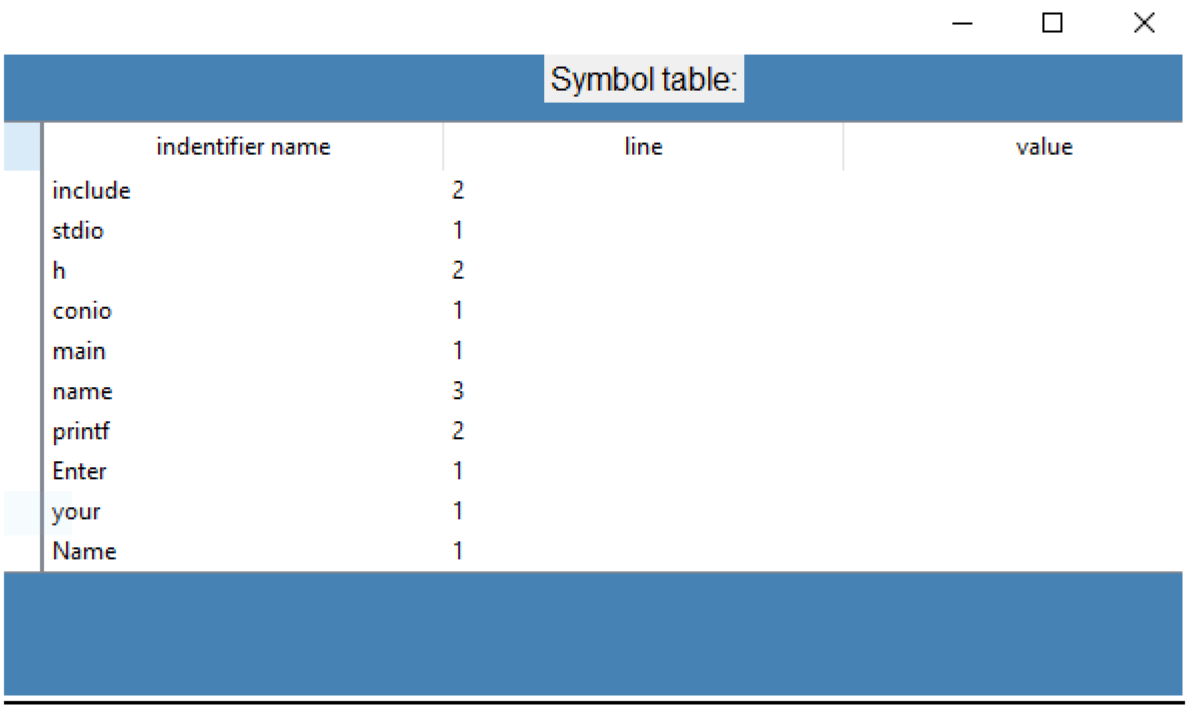
```
#include<stdio.h>
#include<conio.h>
int main()
{
    char name[20];
    printf("Enter your Name: ");
    scanf("%s", name);
    printf("Hello, %s\nWelcome to codesc
racker.com", name);
    getch();
    return 0;
}
```

Analyze name	Tokens
symbol	#
identifier	include
operator	<
identifier	stdio
symbol	.
identifier	h
operator	>
symbol	#
identifier	include
operator	<

Tokenization is the process of breaking down a sequence of characters, such as source code, into smaller units called tokens. In the context of a lexical analyzer, tokenization refers to identifying and categorizing the different types of tokens present in the source code. Each token represents a meaningful unit, such as a keyword, identifier, operator, or literal value.

Symbol Table:

The symbol table only here has a identifier name,line,value tokenize of c dode.



indentifier name	line	value
include	2	
stdio	1	
h	2	
conio	1	
main	1	
name	3	
printf	2	
Enter	1	
your	1	
Name	1	

Conclusion:

The lexical analyzer plays a crucial role in the compilation process by breaking down the source code into meaningful tokens, detecting lexical errors, managing the symbol table, and generating the token stream for further processing. It serves as the initial step in transforming human-readable source code

into a format that can be understood and processed by the compiler or interpreter.

Special Thanks to:

Md. Shymon Islam

Lecturer

Department of CSE

North Western University

Khulna, Bangladesh.