



**THEORY OF AUTOMATA**

**Lexical analyzer for any CPP code, mathematical expression, including parentheses**

***Project Members:***

Hassan Ali (22K-4637)

Abdul Sami (22K-4354)

***Course Instructor:***

Dr Shahzad

***Section:***

4-H

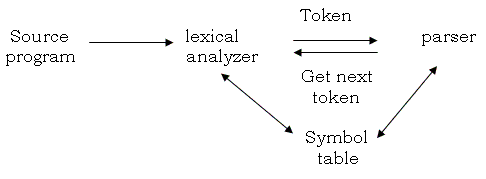
**Title:** Design of Lexical Analyzer

**Introduction:**

Our aim of developing this project is to utilize and furnish our programming skills which we have learned throughout our computer science degree and to implement the understanding of Theory of Automata course knowledge. A lexical analyzer plays a very important part in the compilation of a program.

**Background:**

Compiler is responsible for converting high level language in machine language. There are several phases involved in this and lexical analysis is the first phase.



Lexical analyser reads the characters from source code and convert it into tokens. A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer).

**Project Specifications:**

Our project is a lexical analyzer which has been developed using the c++ programming language. It perform a variety of functions such tokenization, finding and reporting errors to the users and telling some important details of the program such number of lines or number of tokens and whether a token used is valid or not.

**Problem Analysis:**

A compiler does not immediately convert a high-level language into binary – it takes time to complete! ****During the compilation process, the first step that is undertaken is called lexical analysis.**** During this process, the program typed by the user is shredded to pieces and every token that is a part of it is extracted and stored separately (tokens are the smallest indivisible parts of a program). These tokens need to be classified into particular types before the compilation process can begin.

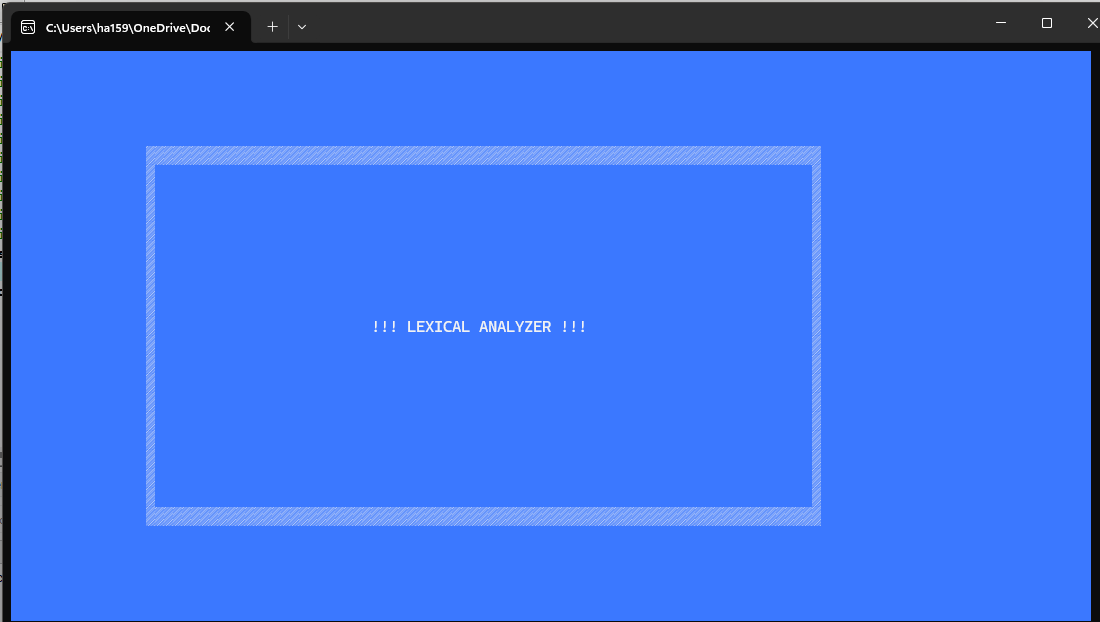
A few different tokens (aka lexemes) are:

* **Keywords**
* **Identifiers**
* **Operators**
* **Constants (Literals)**
* **Separators**

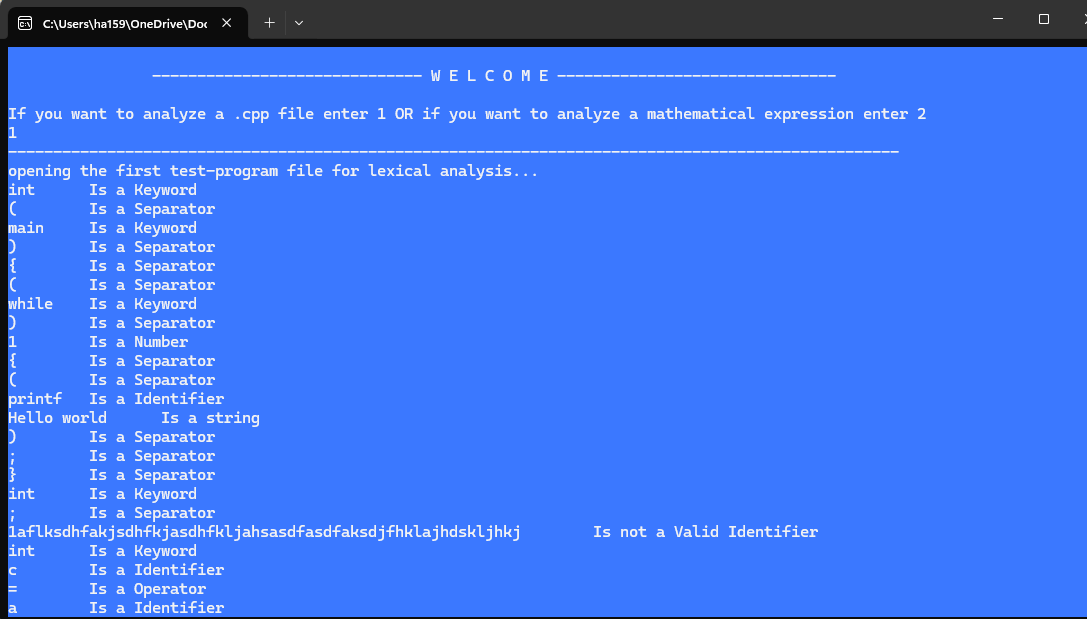
**Contributions:**

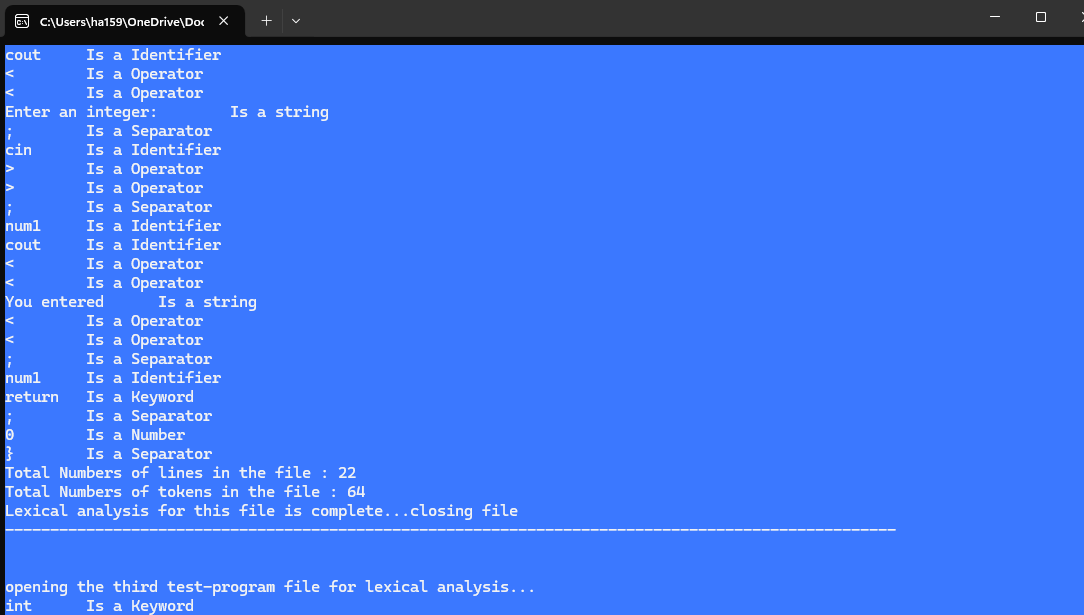
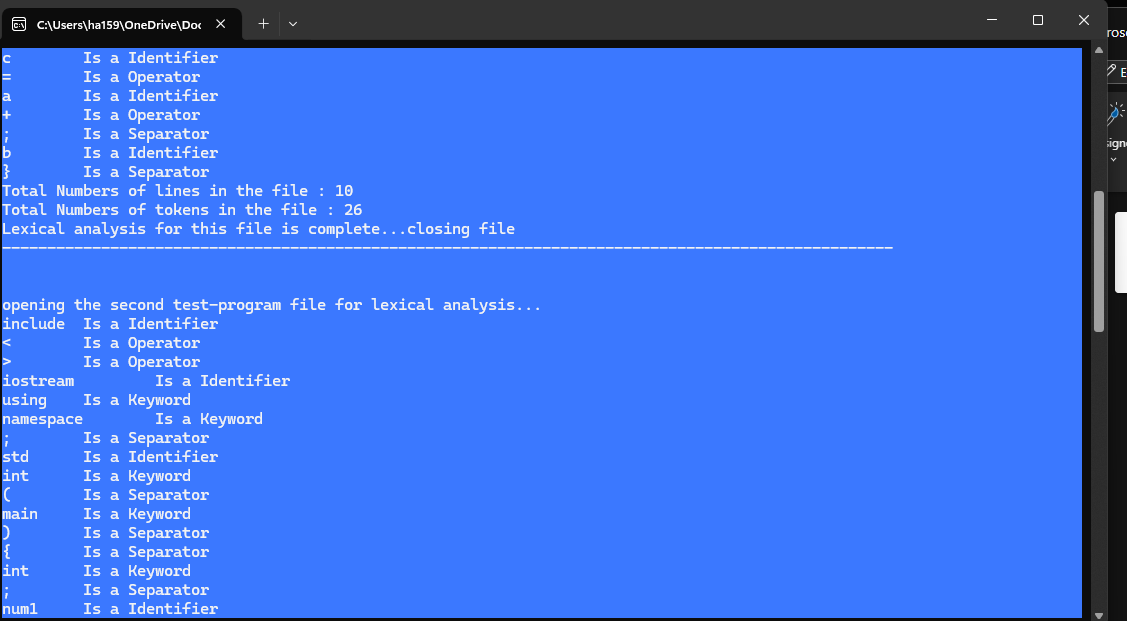
When we started me (Hassan Ali) started working on parsing mathematical expressions but we noticed that the project did not had any type of complexity in it thus my member (Abdul Sami) suggested that not only we should work on mathematical expressions we should also parse programs in CPP, as actual analyzing takes place in parsing programming statements in tokenizing them. Thus he started working on evaluating cpp codes and i worked on parsing mathematical expressions and on GUI (kind of) to make look things a little better.

**implementation and Testing:**

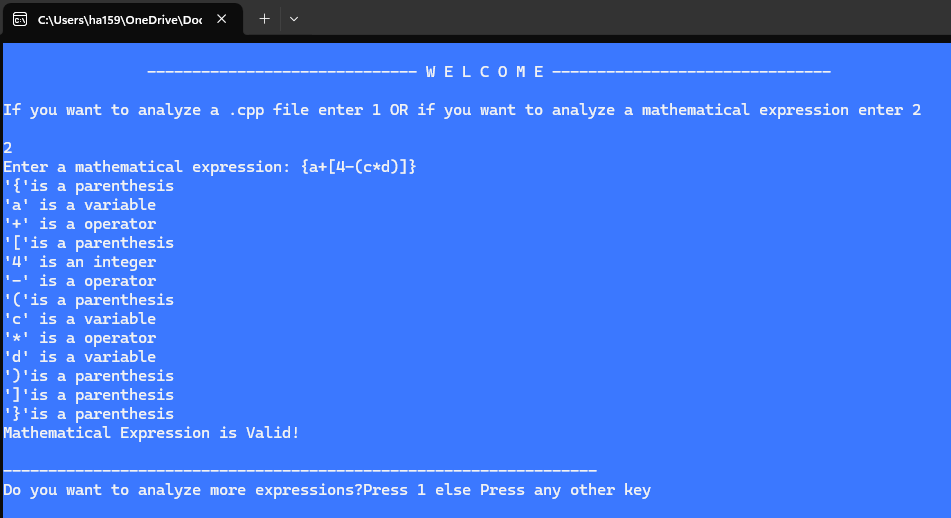


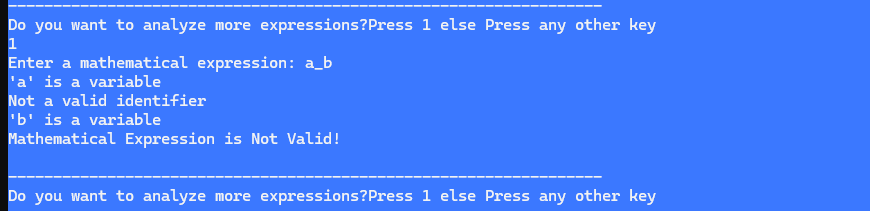
-> We had three test programs which we tested.





-> Now we will parse mathematical expressions:





# **Results:**

# After working for quite some time on this, we have successfully developed this lexical analyzer. It works perfectly but I am sure it can still be modified, and new features can be added to increase its working.

# **Conclusion:**

Our implementation of a C++ lexical analyzer should be enough to demonstrate how it works as part of the compiler. I have also explained what a compiler, interpreter, and the difference between them is. I learned a great deal of work going on in behind while we are compiling our programs. Hope this helped you in understanding the lexical analysis in C++ programming. This was the project for my Theory of Automata course.