EasyChair Preprint

№ 10366

Study on Tokenization and Tool Fabrication

Singh Shubham Rajan, Tushar Lahaik, Prince Verma,
Rishab Kumar, Akash Deep and Battula Naga Prasanth Reddy

June 9, 2023

Study on Tokenization and Tool Fabrication

Singh Shubham Rajan
Student,
Computer Science and Engineering
Lovely Professional University
Punjab, India
shubhamsinghk8@gmail.com

Tushar Lahaik
Student,
Computer Science and Engineering
Lovely Professional University
Punjab, India
tusharlahaik@gmail.com

Prince Verma
Assistant Professor,
Computer Science and Engineering
Lovely Professional University
Punjab, India
prince.29516@lpu.co.in

Rishab Kumar
Student,
Computer Science and Engineering (Hons)
Lovely Professional University
Punjab, India
rishabkumar6805@gmail.com

Akashdeep
Student,
Computer Science and Engineering
Lovely Professional University
Punjab, India
akash2002guru@gmail.com

Battula Naga Prasanth Reddy
Student,
Computer Science and Engineering
Lovely Professional University
Punjab, India
battulanagaprasanth2324@gmail.com

Abstract: Tokenization is an important step in compilation, as it enables machines to understand human language more effectively. During tokenization, a lexical analyzer reads the input character by character and groups them into meaningful tokens based on some predefined. These tokens may include keywords, strings, variables, operators, constants, and special symbols. The tokens are then passed on to the next stage of the compilation process. The Tokens Identifier Model is a web-based platform that can assist learners in classifying and identifying different tokens in the C++ and C computer languages. For newcomers and college students with experience in computer applications, the tool seeks to make learning easier. The structure and syntax of these programming languages can be better understood by learners by offering a web platform that accepts input in the form of.cpp and.c files and outputs the categorized tokens in the form of tables.

Keywords: Lexical Analysis, Lexical Analyzer, Compiler, Tokens, Programming Language Processing.

1. Introduction

Tokenization is the process of splitting a sequence of computer programs or software that contains instructions that the computer follows to do something (typically a source code), into smaller units called tokens [1]. A token is the smallest unit of computer languages, such as a keyword, special symbols, variables, operators, strings, or constants that is meaningful to the compiler. The important abstract of tokenization or lexical analysis is 1. Processing and analyzing large amounts of data in a structured and efficient manner 2. Tokenization or lexical analysis involves splitting down program source code, into smaller units called tokens.

The concept of tokenization or lexical analysis is used in many applications. It is used in many applications some of them are data analysis, natural language processing, and programming language processing [2]. In computer language, tokenization is the first step of the compilation process, where a program's source code is split into separate tokens or lexemes. Once the source code has been tokenized, each token is typically assigned a value that indicates its type or function. By splitting a source code into its individual units, tokenization makes it easier to analyze and process. Tokenization is a critical component of the compilation process and is used in a wide range of programming languages and software development tools.

Keywords	**Constants**	**Strings**
Keywords are predefined or reserved words whose meaning is already defined by Compiler. It is used to define control structures, data types, and other programming elements**, Ex. 'if', 'else',**	Constant is a fixed value or identifier that cannot be changed during the execution of the program. It is used to define fixed values that are used throughout the program, **Ex. 'const**	Strings is a collection of characters that is used to represent text or data, names, addresses, and messages. in computer programming, **Ex.' class (std:string)'.**

'switch' etc	**double PI = 3.14; const int MAX_SIZE = 100;'**.	
Variables Variable is named as a storage location or a reference to an object that is used to store data or manipulate data during execution in computer programming**, Ex. 'int num = 10;', 'name = "XYZ"'.**	**Operator** Operators are symbols that perform operations or actions on variables and values in one or more operands like arithmetic, comparison, logical, and assignment operators**., Ex. '&&', '=='.**	**Special Symbols:** Special symbols have a specified meaning in computer language used to group and organize code, specify function parameters, and access elements in data structures like mathematical, logical, and relational operators, **Ex. ' + ',' - ',' * ' etc.**

Table 1: Various Tokens

2. Previous Work

Tokenization is a technique used in natural language processing (NLP) to break up a sentence or a code file into individual words or tokens. There have been many previous works on tokenization, ranging from simple rule-based methods to more complex statistical and learning-based models. Some remarkable works include:

1. Penn Treebank Tokenization: The rule-based tokenization method developed by the University of Pennsylvania [9]. The Penn Treebank Tokenizer splits the sentence into individual tokens, including punctuation marks, while also handling contractions, abbreviations, and other special symbols.
2. NLTK Tokenization: Natural Language Toolkit is an open-source library for natural language processing in Python [10]. Analyze a large amount of textual data into parts and perform analysis on the character of the text and split it into tokens.

3. Proposed Work

Code is written in JavaScript and is used to perform lexical analysis of a given input file. The algorithm used in the code is a simple implementation of lexical analysis, which includes splitting the input file into individual words and identifying the language's keywords, data types, operators, special characters, variables, and constants.

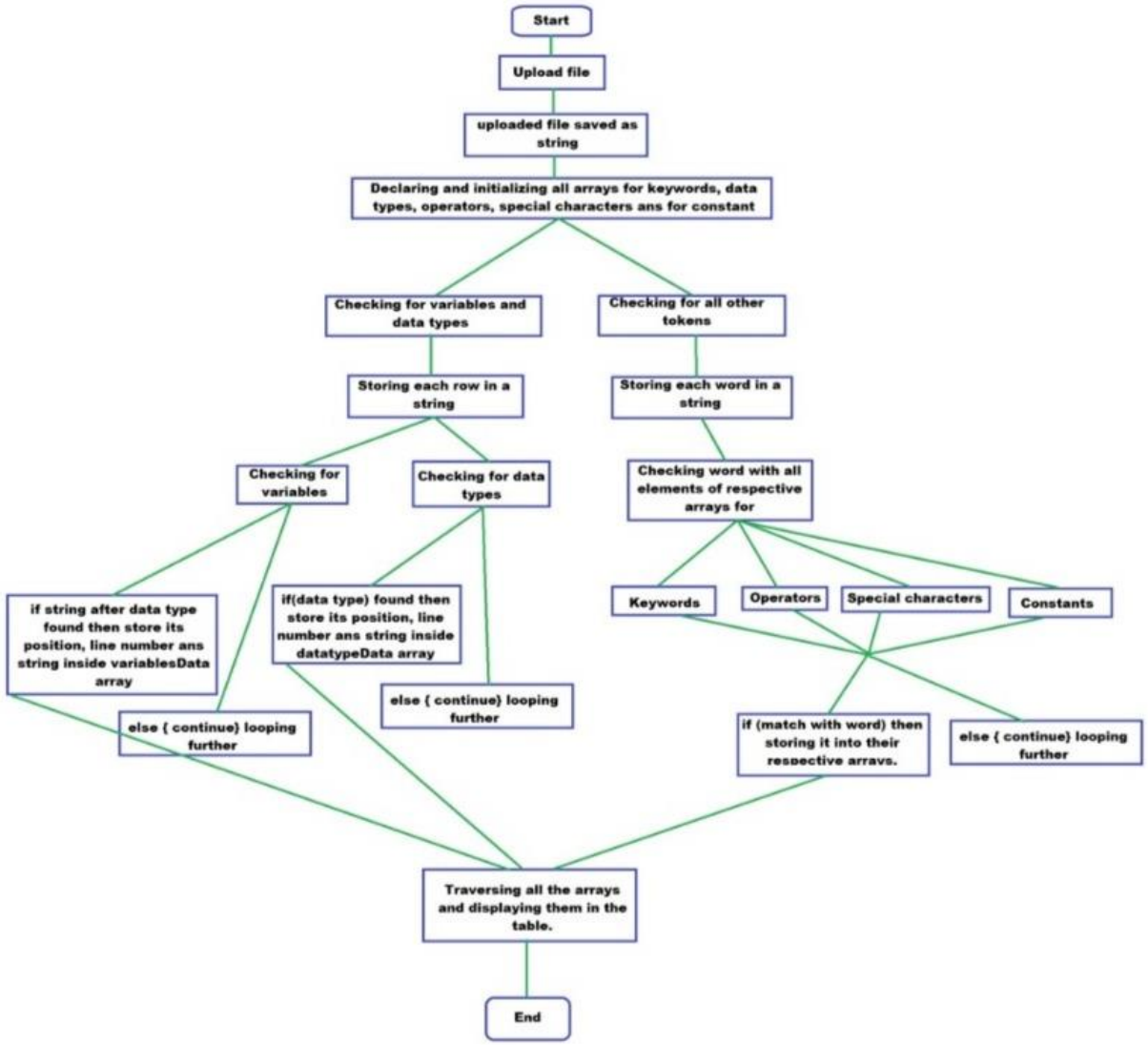

Figure 1: Tokenization

3.1 Methodology

The code reads the file using the File Reader object and splits the file's contents into individual lines using the split () method. It then loops through each line, splits each line into individual words, and checks each word against the pre-defined lists of keywords, data types, operators, special characters, and constants to identify the tokens in the input file. The regular expression used in the code is:

```
const regex = new RegExp(`${data Types[i]} \\s+(\\w+)`);
```

Equation (1)

- This regular expression is constructed using the RegExp [4] constructor in JavaScript. It takes a string as input, which is constructed using string interpolation with the current element of the data Types array and the \s+ [6] and (\w+) special characters.
- The \\s+ special character matches one or more whitespace characters, while the (\w+) [6] special characters match one or more word characters. Word characters include any letter, digit, or underscore.
- The regular expression is then used to match against a string list item, which presumably contains code. If there is a match, the code extracts the first captured group (i.e., the (\w+) [6] part of the regex) and stores it in an object along with other data about the variable.

- This regular expression is looking for variable declarations of the form data Type variable Name. For example, if data Types[i] is "int", then the regular expression would match against strings like "int x", "int y = 5", or "int z, a;" [7].

Regular expressions are a powerful tool for working with text data in programming. They allow you to define patterns of characters to search for or manipulate. For more information on regular expressions in JavaScript, you can refer to the following resources:

The code uses a for Each () loop to iterate through each line and an inner for Each () loop to iterate through each word on the line. For each word, it checks against each pre-defined list to identify the token type, and it adds the token and its properties to the corresponding array. The regular expression used in the above code matches any of the keywords present in the text by searching for word boundaries around the keywords and making the search case insensitive. Overall, the code performs a basic implementation of a lexical analyzer algorithm.

3.2 Implementation

The code is written in React and is used for reading a file and analyzing its content to extract data and information related to programming constructs like data types, operators, keywords, special characters, constants, and variables. The analysis is performed by parsing each line of the file and breaking it down into individual tokens or words. The code starts by defining four arrays - keywords, operators, special characters, and Data Types. These arrays contain the keywords, operators, special characters, and data types commonly used in programming languages. These arrays are used later in the code to identify and extract these constructs from the file. The code defines two states using the use State hook - file and data. file stores the uploaded file, and data stores the extracted data from the file.

The handle File Upload function is called when a file is uploaded using the input element. It reads the uploaded file using a File Reader object and extracts the data by splitting the file into individual lines using the split method. Then, for each line, the function analyzes the content and extracts the relevant information. The extracted information is then stored in an array and assigned to the data state. The function analyzes each line by splitting it into individual tokens using the split method. Then it performs a series of checks to identify and extract the programming constructs from the tokens.

First, the function checks for data types. It joins the tokens using the join method to create a single string and checks if it contains any of the data types defined in the data Types array. If it finds a match, it creates an object containing the line number, data type name, and position, and pushes it into the data Types Data array. Next, the function checks for variables. It loops through the data Types array and uses a regular expression to match the variable name with the data type. If it finds a match, it creates an object containing the line number, variable name, and position, and pushes it into the variables Data array.

After that, the function checks for operators and special characters. It loops through each token and checks if it contains any of the operators or special characters defined in the operators and special Characters arrays. If it finds a match, it creates an object containing the line number, operator or special character, and position, and pushes it into the operators' Data or special characters Data array, respectively. The function also checks for keywords by looping through the keywords array and checking if any of the tokens match. If it finds a match, it creates an object containing the line number, keyword, and position, and pushes it into the string Data array. After all the lines have been analyzed, the extracted data is combined into a single object and assigned to the data state.

Overall, the code provides a simple way to extract and analyze programming constructs from a file using JavaScript and React. It could be useful for various applications like code analysis, syntax highlighting, and code completion.

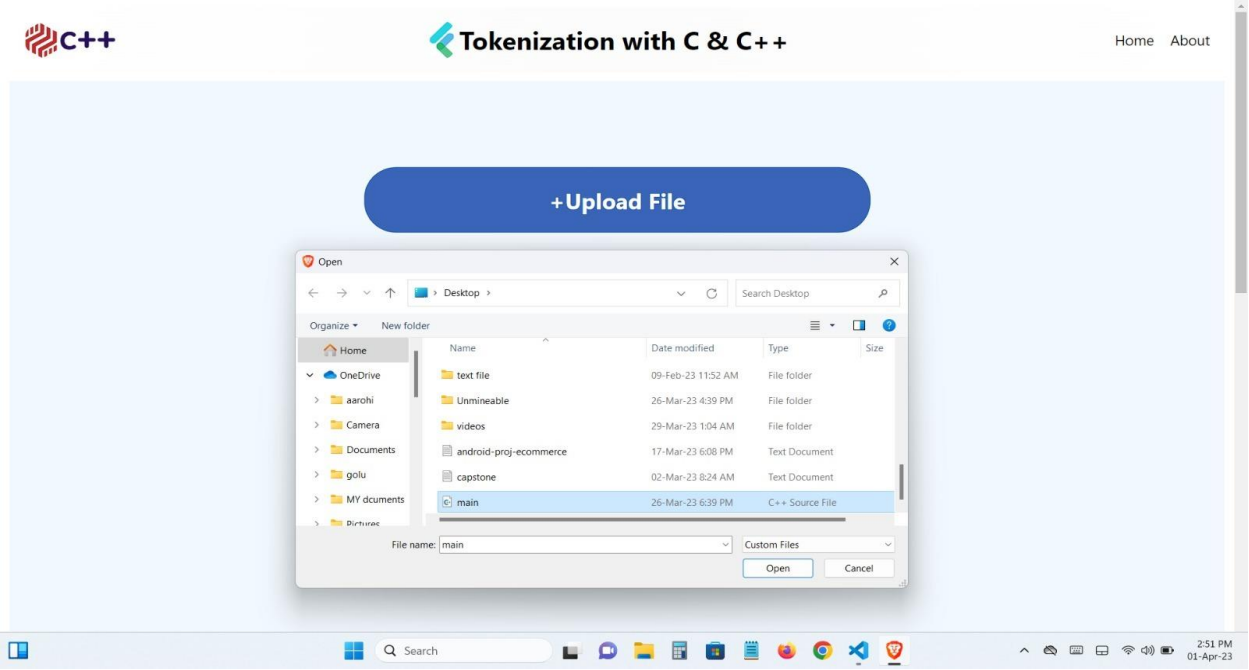

Figure 1: Uploading files for tokenization

Figure 2: Output of Tokenization for Code file 1

4. Conclusion and Future scope

In conclusion, the goal of the study was to create a web platform utilizing React JS and CSS as a tool for token detection and classification using C++ and C programming. To detect various token kinds, including keywords, identifiers, constants, strings, special symbols, and operators, an algorithm is used. The tool processed the .cpp and .c files used as input files successfully, and it produced tables that categorized the various tokens used in the program as output.

Project has the potential to help programmers learn C++ and C, especially those who are just starting out. The tool can assist these students in recognizing and understanding the many tokens that are present in their programs, which will improve their understanding of the code and their capacity to create effective and efficient programs.

Even if the effort was successful in producing a C++ and C programming tool for token detection and categorization, there is still room for development [10]. Future study in this field may take several different avenues, including:

1. Incorporating machine learning methods to increase the classification and token identification's accuracy.
2. Adding support for additional programming languages, such as Python or Java, in the tool.
3. Making the online platform's interface more user-friendly by adding features like a code editor or a visual depiction of the token types.
4. Integrating the tool with an online learning environment to give C++ and C programming students a more interactive and engaging learning environment.

Overall, this initiative has laid a strong platform for further study in this field, and it can be anticipated that the creation of tools that can facilitate better programming language learning and comprehension will continue to advance.

5. References:

[1] Lexical analysis https://en.wikipedia.org/wiki/Lexical_analysis (21 March 2023).

[2] Lexical Analysis: Norms and Exploitations (The MIT Press) Hardcover – January 25, 2013.

[3] A Systematic Literature Review of Lexical Analyzer Implementation Techniques in Compiler Design https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3770588 (Posted: 18 Feb 2021).

[4] Regular Expressions - Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

[5] Regular Expression Tester - regex101: https://regex101.com/

[6] Regular-Expressions.info: https://www.regular-expressions.info/javascript.html

[7] Regular-Expressions.info: https://www.regular-expressions.info

[8] Lexical analysis of student research drafts in computing: https://www.sciencedirect.com/science/article/abs/pii/S1044028319303552

[9] The Penn Treebank: https://repository.upenn.edu/cis_reports/237/

[10] Review on Natural Language Processing Trends and Techniques Using NLTK: https://link.springer.com/chapter/10.1007/978-981-13-9187-3_53

[11] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf

[12] Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing (3rd ed.). Pearson.

[13] Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

[14] Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv preprint arXiv:1508.01991. https://arxiv.org/pdf/1508.01991.pdf

[15] Ruder, S. (2017). An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098. https://arxiv.org/pdf/1706.05098.pdf

[16] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12, 2493-2537. http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf

[17] Lafferty, J. D., McCallum, A., & Pereira, F. C. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001), 282-289. https://www.ics.uci.edu/~welling/teaching/ICS273Afall11/Lafferty01.pdf