Activity based

Project Report on

System Programming

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

Yashashree Sagar Bedmutha

SRN No : 202100177

Roll No : 02

Div : E

Third Year Engineering

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2023-2024

1.1. Abstract

The main goal of the project is to develop a flexible lexical analyzer for the English language. Tokenization, classification, design and implementation, and its wide range of applications in language processing are all included in this comprehensive project. English text is effectively broken up into structured tokens by the analyzer, providing an essential basis for more complicated natural language processing tasks. A crucial aspect is its capacity to recognize and classify tokens for a range of applications, including text categorization, sentiment analysis, and information retrieval. The goal of this project is to produce a reliable and versatile tool that can be used in a variety of language processing contexts. The method simplifies the process of obtaining insightful information from unstructured English text data by use of rigorous design and execution procedures.

1.2.Introduction

Within the field of programming and natural language processing, the main objective of this project is to develop a basic, yet flexible, lexical analyzer that is specifically designed for the English language. A thorough examination of the analyzer's lifetime, including crucial stages like design and implementation, tokenization, classification, and its extensive application in the field of language processing, is required for this project.

Careful design considerations are prioritized from the beginning. The project begins with a strategic blueprint that carefully shapes the data structures, algorithms, and architectural elements of the lexical analyzer. This design step converts abstract design principles into tangible code, laying the groundwork for the implementation that follows. The goal is to give the analyzer the capacity to parse input text, recognize and classify tokens quickly, and follow pre-established lexical rules.

To improve analyzability, tokenization—the division of English text into meaningful units like words, numbers, punctuation, and operators—is a crucial component. Classification and recognition play equally important roles in allowing the analyzer to recognize and classify different kinds of tokens according to pre-established guidelines. Due to its adaptability, the tool can be easily integrated into a wide range of language processing applications, such as information retrieval, text categorization, and sentiment analysis.

The analyzer serves as the cornerstone for more complex natural language processing activities as well. It supports tasks like named entity recognition, sentiment analysis, and part-of-speech tagging, which enable the extraction of insightful information from unstructured English text data.

1.3. Literature Summary

Several external libraries and jar files were used in the "phase2" Java program's implementation to carry out natural language processing operations. The Stanford NLP (Natural Language Processing) library is the main resource used. It is well-known for having extensive and effective text analysis features. Tokenization, sentence splitting, and part-of-speech tagging are all supported by this library. The following particular libraries and parts were utilized in the project

1. Stanford NLP Library

- The Stanford NLP library is an open-source natural language processing library developed by Stanford University. It offers various tools and models for text processing, including tokenization, part-of-speech tagging, named entity recognition, and more.

2. Java Util Package

- The Java **util** package is used to work with data structures like ArrayList, Properties, and other core Java utilities. It is essential for handling text data and managing program configurations.

3. Logging Properties

- A custom logging configuration is used to suppress INFO messages and ensure that only relevant log data is displayed. This configuration is provided in an external logging properties file.

These libraries and other sections work together to enable the "phase2" program to analyze text input thoroughly, recognizing operators, special characters, punctuations, and different portions of speech.

1.4. Methodology

The pseudo-code for the main "phase2" program function is shown below, along with an explanation of the high-level procedures and reasoning involved:

```
1. Function createPipeline():
    1. Create a StanfordCoreNLP pipeline with specific annotators for tokenization,
       sentence splitting, and part-of-speech tagging.
    2. Return the pipeline for further text analysis.
2. Function extractTokens(document):
    1. Initialize an empty list to store Token objects.
    2. Extract sentences from the annotated document.
    3. For each sentence, iterate through the tokens and extract their part-of-speech and
       text.
    4. Create Token objects and add them to the list.
    5. Return the list of tokens.
3. Function displayTokensOfType(tokens, posTags):
    1. Initialize counters and lists for specific token categories (e.g., nouns, verbs,
       pronouns).
    2. Iterate through the list of tokens and classify tokens into specific categories based
       on their part-of-speech tags.
    3. Display the count and list of tokens for each category.
4. Function displayPunctuation(tokens):
    1. Initialize counters and lists for punctuation tokens.
    2. Iterate through the list of tokens and identify punctuation tokens.
    3. Display the count and list of punctuation tokens.
5. Function displayOperators(tokens):
    1. Initialize counters and lists for operator tokens.
    2. Iterate through the list of tokens and identify operator tokens.
    3. Display the count and list of operator tokens.
6. Function displaySpecialTokens(tokens):
    1. Initialize counters and lists for special character tokens.
    2. Iterate through the list of tokens and identify special character tokens.
    3. Display the count and list of special character tokens.
7. Function isPunctuation(word):
    1. Define a regular expression to match punctuation.
    2. Check if the given word matches the defined regular expression and return a
       boolean value.
8. Function isSpecialCharacter(word):
    1. Define a regular expression to match special characters.
    2. Check if the given word matches the defined regular expression and return a
       boolean value.
9. Function isOperator(word):
    1. Define a regular expression to match operators.
    2. Check if the given word matches the defined regular expression and return a
       boolean value.
10. Function configureLogging():
```

1.5.

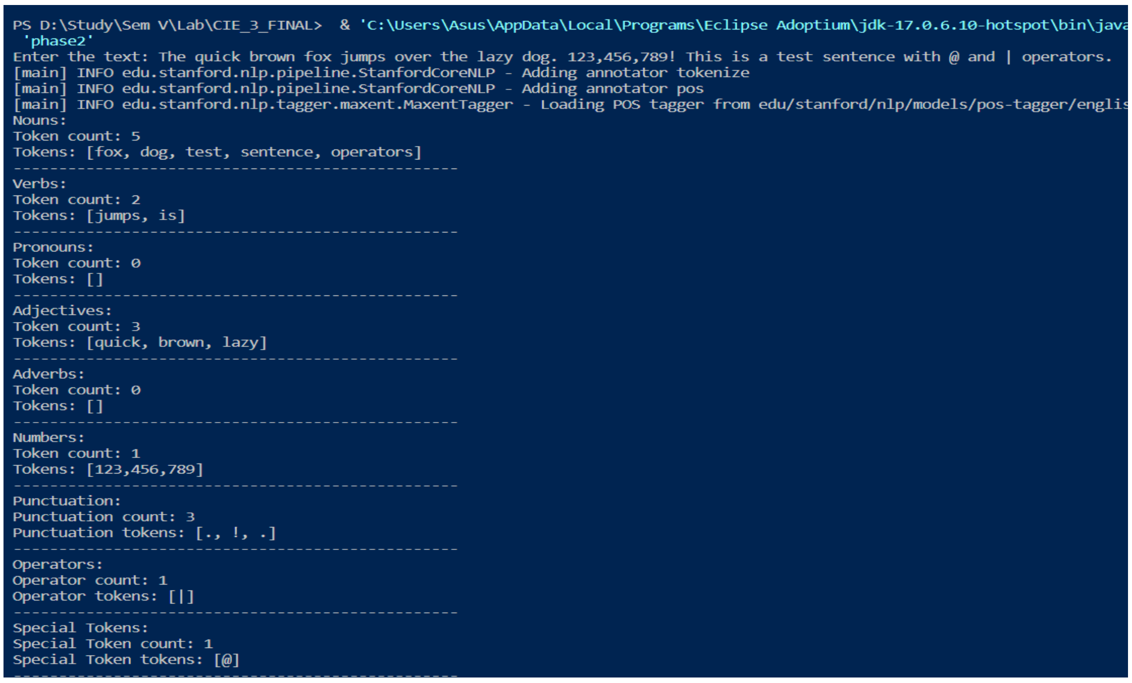
1.5 Result

Insightful Presentation: The program doesn't merely provide a raw list of these token categories; it offers insight by providing the count of tokens within each category, along with the corresponding lists of tokens. This presentation aids users in quickly grasping the distribution and significance of various linguistic elements in the input text.

Enhanced Understanding: By displaying these categorized tokens and their counts, the "phase2" program significantly enhances the user's understanding of the linguistic components and special characters within the input text. It simplifies the task of identifying and assessing the usage of different word types, numbers, punctuations, operators, and special characters in the text.

Analytical Utility: The result is not just a list of tokens but a valuable analytical tool for linguists, researchers, and programmers who need to gain an in-depth understanding of the linguistic structure and special elements within a given text. It facilitates efficient textual analysis and supports decision-making based on the composition of the input text.

Output:



```
PS D:\Study\Sem V\Lab\CIE_3_FINAL>  & 'C:\Users\Asus\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java
 'phase2'
Enter the text: The quick brown fox jumps over the lazy dog. 123,456,789! This is a test sentence with @ and | operators.
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator tokenize
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator pos
[main] INFO edu.stanford.nlp.tagger.maxent.MaxentTagger - Loading POS tagger from edu/stanford/nlp/models/pos-tagger/englis
Nouns:
Token count: 5
Tokens: [fox, dog, test, sentence, operators]
-------------------------------------------------
Verbs:
Token count: 2
Tokens: [jumps, is]
-------------------------------------------------
Pronouns:
Token count: 0
Tokens: []
-------------------------------------------------
Adjectives:
Token count: 3
Tokens: [quick, brown, lazy]
-------------------------------------------------
Adverbs:
Token count: 0
Tokens: []
-------------------------------------------------
Numbers:
Token count: 1
Tokens: [123,456,789]
-------------------------------------------------
Punctuation:
Punctuation count: 3
Punctuation tokens: [., !, .]
-------------------------------------------------
Operators:
Operator count: 1
Operator tokens: [|]
-------------------------------------------------
Special Tokens:
Special Token count: 1
Special Token tokens: [@]
-------------------------------------------------
```

1.6. References

- Stanford NLP library's official documentation
- Hersh, W. R., Campbell, E. M., & Malveau, S. E. (Year). *Assessing the feasibility of large-scale natural language processing in a corpus of ordinary medical records: a lexical analysis.*
- Lee, S., Binkley, D., Gold, N., Islam, S., Krinke, J., & Yoo, S. (Year). *Evaluating lexical approximation of program dependence.*
- Tjabaka, L. H. (Year). *The rise of lexical subjects in English covert infinitives.*