FINAL PROJECT REPORT - COMP6062001

"Piglatin Translator Using Regular Expression, Context Free Grammar, And Lexer"

Christopher Owen (2502019180)

Justin Theofilus Yonathan (2502036382)

Pandya Limawan (2502022433)

Rendy Gunawan (2502024666)

BINUS INTERNATIONAL

FACULTY OF COMPUTING AND MEDIA

COMPUTER SCIENCE

2023

TABLE OF CONTENTS

FINAL PROJECT REPORT - COMP6062001 1

TABLE OF CONTENTS 2

INTRODUCTION 3

RELATED WORK 4

- Regular Expressions in Natural Language Processing 4
- Natural Language Processing 4
- Pig Latin Translation 4

IMPLEMENTATION 6

- RE Specification 6
- CFG Specification 6
- Scanner Sketch 7
- Parser Sketch 7

EVALUATION AND DISCUSSIONS 9

- Code Implementation 9
- Input and Output 14

CONCLUSION AND RECOMMENDATION 17

REFERENCES 18

APPENDICES 19

- Program Manual 19
- Demo Video 19
- GitHub Link 19
- Canva Link 19

INTRODUCTION

As we live in the modern world, the way humans communicate evolves each day to become more diverse and challenging to experiment with. There are many languages spread across the world, not only the country's official or main language but also other unofficial old or newly made languages. One of them is called Pig Latin. It is not a normal natural language that people can learn and master to communicate daily. It has a little twist that makes them sound unique and difficult to understand. Pig Latin is a code language or pseudo language that has a secret way to be understood and can be humorous due to its rules.

Pig Latin was first introduced in the 16th century, as it was called Hog Latin (Tirosh, 2023b). Then, it became popular in late 19th-century English periodicals because the entertainment world was interested in it and used it to entertain people through songs, films, and "new" interactions that they introduced to their fans. Through this, the language spread, and more people tried to figure out how to learn and master Pig Latin. Moreover, learning Pig Latin is not as easy as people think. It is based on the English language, but its adaptation can confuse people. There are several rules that complicate English words, and only people who understand them can know what they mean. One of the rules is to move the first consonant syllable to the end and add "ay." For example, book becomes *ook-bay*, duck becomes *uck-day*, find becomes *ind-fay*, and many other words have similar patterns. Another rule is that if a word begins with vowels, the Pig Latin word needs to add "way" at the end of it. For example, eye becomes *eye-way*, apple becomes *apple-way*, and there are other similar pattern words. This can be difficult to learn if people do not pay attention to the rules implemented.

Due to its complexity, we introduce the Pig Latin translator to ease learners' mastery of the Pig Latin language. It is similar to other translators, but this software is specific for Pig Latin because of the twist it has. It is very unique, considering its adaptation from English, which can also make it an adaptive and versatile language. Not only is it effective for communication platforms but also for entertainment and educational purposes, which can further experiment with the language. In the future, people can intend to implement these rules in their original language, which can broaden the usage of Pig Latin. Lastly, it is very secure, considering the difficulty with which people can learn. Its effectiveness in obscuring information and maintaining confidentiality can benefit those who understand the language well.

RELATED WORK

The successful application of regular expressions takes center stage in the landscape of computational linguistics and language processing, providing an effective toolkit for tasks such as string manipulation and pattern recognition. The combination of regular expressions, context-free grammar, and Lexer technology provides a solid foundation for developing advanced language processors. This chapter is an introduction to the beneficial application of regular expressions, context-free grammar, and lexer techniques, with a special emphasis on their integration within a Pig Latin translator. We hope that by delving into the complexities of these linguistic tools, we will be able to reveal their collective potential in addressing the unique challenges posed by Pig Latin translation, such as capturing distinct language patterns and ensuring accurate structural representation.

Regular Expressions in Natural Language Processing

Bird goes deeply into the role of regular expressions in natural language processing, bringing light on their versatile applications in linguistic tasks. Notably, the authors emphasize the universality of strings in language processing, emphasizing the importance of addressing not just content but also formatting and markup within texts. The work introduces regular expressions' fundamental building blocks, emphasizing their utility in matching character sequences and navigating complex language structures. The study includes discussions on wildcards, optionality, repeatability, options, and other topics, demonstrating how these elements contribute to efficient string processing (Bird, 2006). The study provided examples to illustrate the practical application of regular expressions in tasks like spell-checking, format validation, and markup extraction. By studying and understanding this work, researchers can gain valuable insights into leveraging regular expressions for enhanced natural language processing.

Natural Language Processing

Parsing stands out as a critical component in the vast field of Natural Language Processing (NLP), rapidly evolving to encompass extensive coverage, high accuracy, and efficiency. Despite successful attempts to convert sentences from various languages into parse tree forms, the emphasis on Indonesian language parsing has remained limited. This highlights the importance of developing an Indonesian parser to support language processing applications and advance NLP research. Part-of-speech tagging, named entity recognition, and context interpretation are all stages in the NLP process of parsing, with parse trees serving as syntactic structures. This final project fills a research gap by developing a system for Indonesian sentence-to-parse-tree conversion using Bottom-Up Parsing, making an important contribution to NLP and language processing (Tubagus, 2020).

Pig Latin Translation

The translation process of Pig Latin involves a straightforward modification of English words, transforming them into a playful and coded language. The procedure includes

moving the initial consonant or consonant cluster of an English word to the end and appending "ay." For example, the word "dictionary" becomes "ictionary-day." This linguistic game, primarily utilized by children but also embraced by some adults, does not constitute a genuine language but rather a creative means of communication. Pig Latin has no connection to the Latin language, and its origin remains uncertain, though references to it date back to the late nineteenth century, with anecdotal claims of Thomas Jefferson composing letters in Pig Latin. Some Pig Latin words, like ixnay and amscray, have found their way into English slang, demonstrating the game's influence on language. (Dictionary.com, 2010)

IMPLEMENTATION

RE Specification

- Words = [a-zA-Z]
- Punctuations = [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]
- Numbers = [0-9]

The project implements regular expressions for the input that it will receive from the user, which are words or sentences with punctuation and numbers. The regular expressions are not written in the code explicitly, but the team uses a Python library named NLTK, which stands for Natural Language Toolkit. It allows Python programs to work with human language data containing English words usable for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum (*NLTK :: Natural Language Toolkit*, n.d.).

CFG Specification

- Sentence → WordList
- WordList → Word WordList | Word
- Word → ConsonantWord | VowelWord
- Cluster → Consonant Cluster | Consonant
- ConsonantWord → Consonant Cluster 'ay'
- VowelWord → Vowel 'yay'
- Consonant → 'b' | 'c' | 'd' | 'f' | 'g' | 'h' | 'j' | 'k' | 'l' | 'm' | 'n' | 'p' | 'q' | 'r' | 's' | 't' | 'v' | 'w' | 'x' | 'y' | 'z'
- Vowel → 'a' | 'e' | 'i' | 'o' | 'u'

The project implements several context free grammar rules, which are the foundation of the code and have better readability and concept for developers. Using CFG, people can determine which are valid or invalid strings. Examples of valid strings are “The cat is fast.” translated into “eThay atcay isyay astfay.”,

Some examples of invalid strings are “asdf asdf”, which prints an error message.

Scanner Sketch

The scanner or lexer class of the code will categorize input into tokens and validate English words using NLTK.

Token Class:

The Token class encapsulates the properties of a token, including its type and value.

Lexer Class:

The Lexer class initializes with the input text and maintains the current position and character during tokenization.

Error Handling:

The error() method raises an exception for invalid syntax, ensuring robust error detection in the scanning process.

Tokenization Methods:

- advance(): Moves the lexer to the next character in the input.
- skip_whitespace(): Skips over whitespace characters.
- word(): Captures a sequence of alphabetic characters, representing a word.
- get_next_token(): Returns the next token in the input, categorizing words and punctuation.

Parser Sketch

The parser class of the code will translate English text into Pig Latin by grabbing the tokenized words (terms and expressions) from the input and converting them using the formula. The parser's structure has different methods that handle different grammar rules for translating text.

Translation Logic:

The parser utilizes a translation function, translate_word_to_pig_latin(), to convert English words into Pig Latin. Translation follows Pig Latin rules, such as moving the first consonant cluster to the end of the word and adding "ay" or "way" based on whether the word starts with a vowel.

Vowel Detection:

The translation function employs a list of vowels to determine whether a word starts with a vowel.

Parsing Methods:

- factor(): Processes individual tokens, translating words into Pig Latin and returning punctuation marks as-is.
- term(): Handles sequences of factors, concatenating their translations.
- expr(): Manages the overall translation process, incorporating terms separated by spaces.

EVALUATION AND DISCUSSIONS

Code Implementation

This section explains the code that was implemented to create the English to Pig Latin translator program. In order to create this program, the NLTK corpus library was used to get a list of English words to act as a reference. Additionally, in this implementation, the team decided on a couple of classes that were created, which are the TokenType, Token, Lexer, and Parser classes. Firstly, the TokenType and Token class functions allow the Lexer to tokenize each of the identifiers into respective token types, which are defined in the TokenType class and initialized through the Token class.

```
Comment Code
class TokenType:
    WORD = 'WORD'
    NUMBER = 'NUMBER'
    PUNCTUATION = 'PUNCTUATION'
    EOF = 'EOF'

Comment Code
class Token:
    def __init__(self, type, value):
        self.type = type
        self.value = value
```

For this program, the tokens that were implemented are words, which are English words taken from the NLTK corpus library; numbers and punctuation, which are taken from the default Python library; and an EOF token, which helps to signify the end of a translated sentence. Secondly, the Lexer class works to define the token types of each character, and if possible, it will bundle some into certain tokens, such as words or numbers.

```
class Lexer:
    #initializes important variables for current text such as current char position and
    #defining set of valid english words from the nltk corpus
    def __init__(self, text):
        self.text = text
        self.pos = 0
        self.current_char = self.text[self.pos] if self.pos < len(self.text) else None
        self.english_words = set(word.lower() for word in nltk_words.words())
```

First of all, the Lexer class will define a few variables that are going to play an important role, such as the current character position and the set of valid English words from the NLTK corpus.

```
#Main lexer function which integrates other functions into one that reads
#the input text and iterates each char.
#Detecting words, numbers, punctuations, EOF, and skipping whitespaces.
def get_next_token(self):
    while self.current_char is not None:
        if self.current_char.isspace():
            self.skip_whitespace()
            continue

        if self.current_char.isalpha():
            word = self.word()
            if word.lower() in self.english_words:
                return Token(TokenType.WORD, word)

        if self.current_char.isdigit():
            num = self.number()
            if num:
                return Token(TokenType.NUMBER, num)

        if self.current_char in '!"#$%&()*+,-./:;<=>?@[\]^_`{|}~':
            token_value = self.current_char
            self.advance()
            return Token(TokenType.PUNCTUATION, token_value)

        self.error()
```

Afterwards, the most important function in the lexer class is the get_next_token() function, which integrates other functions belonging to the lexer class in order to iterate through each character to give each bundle of characters individual tokens from the tokens defined in the TokenType class. The get_next_token() function is the function that will help during the parsing process as well. Third, the parser class works to convert the inputted

English text into Pig Latin by incorporating the result of the lexer class and the formula that is defined in the parser class.

```
class Parser:
    #Initializing important variables such as creating
    #lexer class for inputted text and utilizing
    #get_next_token() function to get current token
    def __init__(self, lexer):
        self.lexer = lexer
        self.current_token = self.lexer.get_next_token()
```

Similar to the lexer class, the parser class starts by initializing some variables that are going to be used on the functions of the class later on. The parser class initializes an instance of the lexer class and also utilizes the get_next_token() function to record the current recorded token.

```
#Function that handles which action to be taken for each token types
def factor(self):
    token = self.current_token

    #Checks the token type of the current token
    if token.type == TokenType.WORD:
        #Translate word tokens using translate_word_to_pig_latin() function
        translated_word = self.translate_word_to_pig_latin(token.value)
        self.current_token = self.lexer.get_next_token()
        return translated_word
    elif token.type == TokenType.NUMBER:
        # For numbers, just return the number itself
        num = token.value
        self.current_token = self.lexer.get_next_token()
        return num
    elif token.type == TokenType.PUNCTUATION:
        self.current_token = self.lexer.get_next_token()
        return token.value
```

One of the most important functions in the parser class is the factor() function, which checks for the tokens from the input text using the get_next_token() function and processes it according to each token type.

```
#Function to translate words into pig latin using formula
def translate_word_to_pig_latin(self, word):
    vowels = ['a', 'e', 'i', 'o', 'u']
    if len(word) > 0 and word[0].lower() in vowels:
        return word + 'way'
    elif len(word) > 0:
        i = 0
        while i < len(word) and word[i].lower() not in vowels:
            i += 1
        return word[i:] + word[:i] + 'ay'
    else:
        return word  # Return the word as-is if it's empty
```

Another important function is the translate_word_to_pig_latin() function, which contains the formula to process the word tokens into Pig Latin in the factor() function.

```
#Defining what constitutes as a term
def term(self):
    result = self.factor()

    while self.current_token.type == TokenType.PUNCTUATION:
        result += self.factor()

    return result

#Define what constitues an expression
def expr(self):
    result = self.term()

    while self.current_token.type == TokenType.WORD or self.current_token.type == TokenType.NUMBER:
        result += ' ' + self.term()
```

Lastly, the function that outputs the fully translated text is the expr() function, which concatenates the terms found in the input text based on the definition provided in the term() function.

```
#Main loop
Comment Code
def main():
    while True:
        try:
            #Ask user input
            text = input('Enter English text: ')
        except EOFError:
            print("No input detected")
            break

        #Set conditional to exit on the input "exit"
        if not text:
            continue
        elif text.lower() == "exit":
            break

        #Implement Previously mentioned classes
        lexer = Lexer(text)
        parser = Parser(lexer)
        translated_text = parser.expr()
        print('Pig Latin Translation:', translated_text)

if __name__ == '__main__':
    main()
```

Finally, the main function loop asks the user to input a text in English and processes it through the lexer and parser classes, and based on the result of the parser expr() function, the translated text is printed to the console.

Input and Output

Here is the flow of the Pig Latin translator program, considering user inputs, the process inside the code, and the output it produces by displaying the results:

- Input

- When the user runs the program, there will be a prompt in the console asking users to input a text in English; this is taken as the input that will be processed through the program and translated into Pig Latin. If, for any reason, the user did not input a valid English text, then the loop will terminate and it will throw an invalid syntax error.

- Output
  - After processing the text using the functions in all four classes, the user will be presented with the translated text in the format of "Pig Latin Translation: (translated text)".

- Example

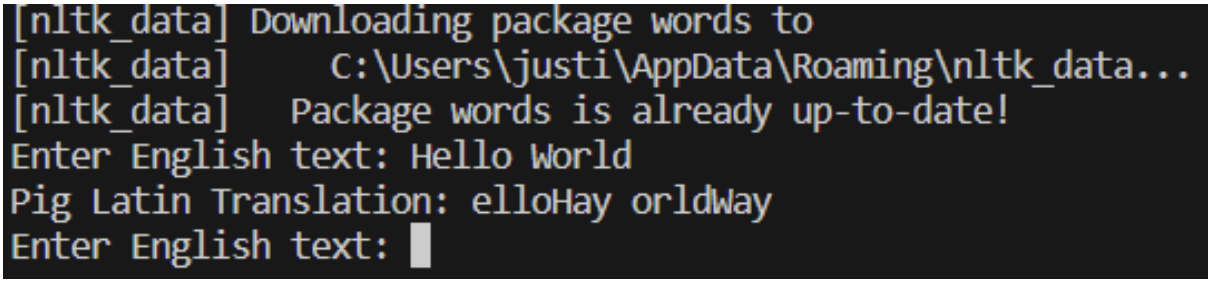

Image 1: Input Valid Text with only words

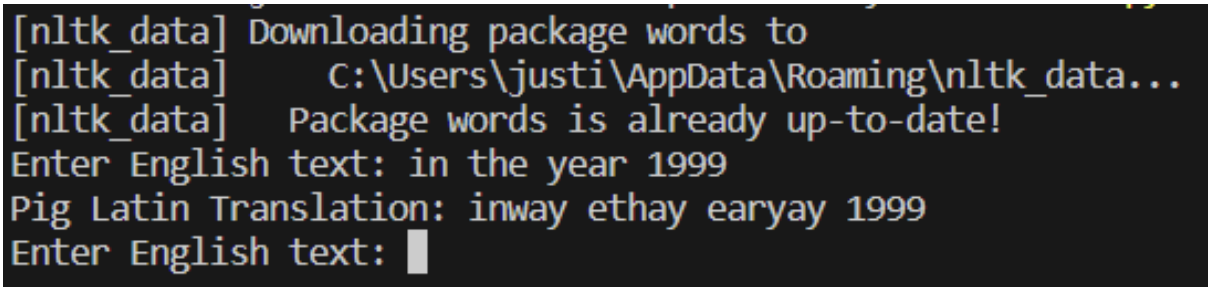

Image 2: Input Valid Text With Combination of Words and Numbers

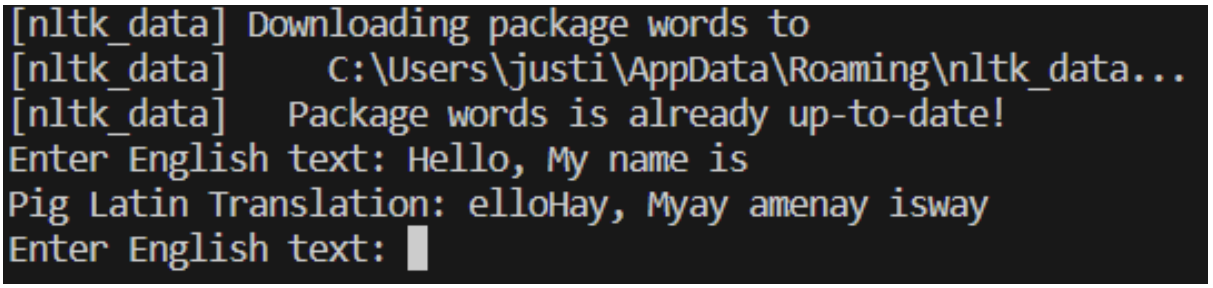

Image 3: Input Valid Text With Combination of Words and Punctuations

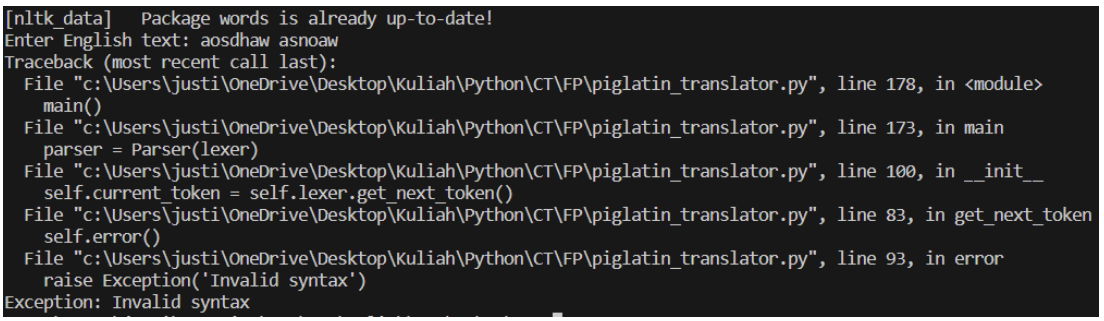

Image 4: Invalid Input Text

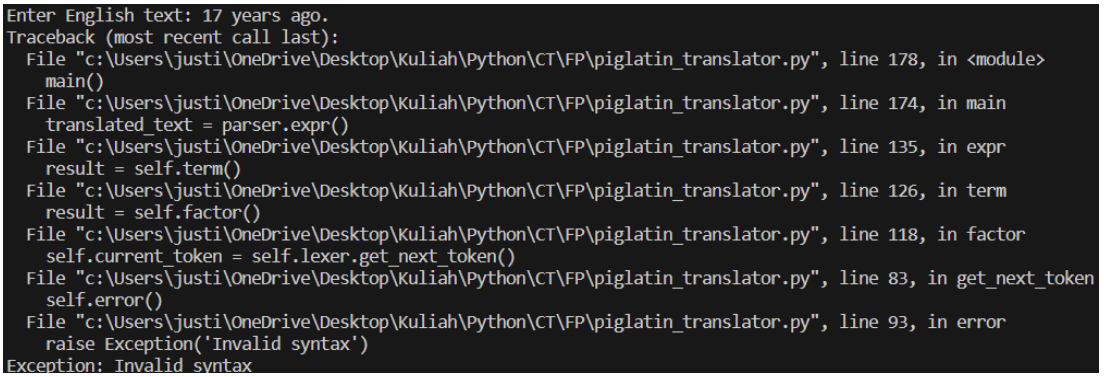

Image 5: Limitation Case 1 (numbers at the beginning or at the middle of the input text return an error)

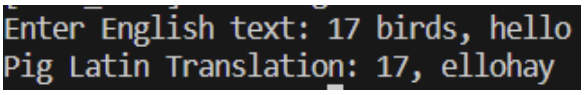

Image 6: Limitation Case 2 (numbers at the beginning followed by a word followed by a punctuation returns incorrect output)

- Explanation
    - As can be seen from the snippets above, the English to Pig Latin Translator program is able to translate most English texts into Pig Latin format, taking into consideration the types of tokens that exist in the input text, such as words, numbers, and punctuation.
    - However, there are certain limitations that the current iteration has. This can be seen in fifth and sixth images, where sentences beginning with numbers followed by words return a weird output or an error in different cases. This is thought to have been caused by the way the numbers are processed through the lexer and parser, which makes it return these buggy outputs.
    - Other than that, there have been no observed limitations when translating English to Pig Latin through the use of this program.

CONCLUSION AND RECOMMENDATION

In conclusion, the English to Pig Latin translator program demonstrates a thoughtful and structured approach to implementing a language translation tool. The program uses a combination of regular expressions, context-free grammar, and lexer techniques to tokenize and parse input English text, ultimately producing a Pig Latin translation.

The program's strengths lie in its systematic design, leveraging the NLTK library for tokenization and incorporating well-defined regular expressions and context-free grammar rules. The use of classes such as TokenType, Token, Lexer, and Parser contributes to a modular and readable codebase, enhancing maintainability and extensibility.

The translator successfully handles typical English sentences and words, following the Pig Latin rules of moving consonants or consonant clusters to the end and adding "ay" or "way" based on vowel presence. The inclusion of error handling mechanisms ensures robustness in syntax validation during the scanning process.

However, there are identified limitations in the handling of sentences starting with numbers followed by words, leading to unexpected outputs or errors. This issue suggests a need for refinement in the parsing logic to address such specific cases.

The English to Pig Latin translator program serves as a solid foundation, and the team recommends the following steps for improvement:

- Address Limitations:

  Refine the parsing logic to handle sentences starting with numbers more effectively, ensuring consistent and expected translations.

- Vocabulary Expansion:

  Enhance the program by expanding the vocabulary database, allowing for a broader range of English words to be translated into Pig Latin.

- Refinement of Translation Logic:

  Continuously refine the translation logic to improve accuracy and handle additional edge cases that may arise in diverse English inputs.

- User-Friendly Features:

  Consider implementing user-friendly features, such as a graphical interface, to enhance accessibility and usability for a wider audience.

By addressing these recommendations, the program can evolve into a more robust and versatile English to Pig Latin translator, offering an improved user experience and handling a broader range of input scenarios.

REFERENCES

Dictionary.com. (2021). What exactly is Pig Latin? In Dictionary.com. https://www.dictionary.com/e/pig-latin/

Bird, S., & Klein, E. (2006, January 29). Regular Expressions for Natural Language Processing (Version 0.6.2, Revision 1.13). Copyright © 2001-2006 University of Pennsylvania. Licensed under the Creative Commons Attribution-ShareAlike License. Retrieved from https://courses.ischool.berkeley.edu/i256/f06/papers/regexps_tutorial.pdf

NLTK :: Natural Language Toolkit. (n.d.). https://www.nltk.org/

Python 3 Notes: String Methods 2. (n.d.). https://sites.pitt.edu/~naraehan/python3/string_methods2.html

Tirosh, O. (2023, March 30). Pig Latin—What is it and how do you speak it? Tomedes. https://www.tomedes.com/translator-hub/pig-latin

Tubagus Rifky Fajar Kusumah Sakti, Moch Arif Bijaksana, Anisa Herdiani. (2020). Sistem Pengubah Kalimat Bahasa Indonesia ke dalam Bentuk Parse Tree Menggunakan Metode Bottom-Up Parsing. e-Proceeding of Engineering, 7(3), 10021. Retrieved from https://openlibrary.telkomuniversity.ac.id/pustaka/files/163038/jurnal_eproc/sistem-pengubah-kalimat-bahasa-indonesia-ke-dalam-bentuk-parse-tree-menggunakan-metode-bottom-up-parsing.pdf

APPENDICES

Program Manual

1. Run the program.
2. After the program runs, this will be displayed in the terminal.

```
PS C:\Users\justi> python -u "c:\Users\justi\OneDrive\Desktop\
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\justi\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
Enter English text:
```

3. Input English words or sentences.

```
Enter English text: Hello World
```

4. Press Enter.
5. The program will display the translated output.

```
Enter English text: Hello World
Pig Latin Translation: elloHay orldWay
```

6. Type the word “exit” to exit the program.

```
Enter English text: exit
PS C:\Users\justi>
```

Demo Video

https://drive.google.com/file/d/1SnMiE2Pqupx4_Jptxd6tlnaJBn0lqQb-/view?usp=sharing

GitHub Link

https://github.com/Theotrgl/Compilation-Techniques_FP_PigLatin_Translator

Canva Link

https://www.canva.com/design/DAF2B3deru4/qE2SHrgDm-b0wxQnthwy6w/edit?utm_content=DAF2B3deru4&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton