“Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de las heroicas batallas de Junín y Ayacucho”

CARRERA: INGENIERÍA DE SOFTWARE
WAYRASIMI: IMPLEMENTACIÓN DE SU ANALIZADOR LÉXICO
COMPILADORES

ESTUDIANTES:

Ortiz Castañeda Jorge Luis
Huamani Huamani Jhordan Steven Octavio
Flores Leon Miguel Angel

DOCENTE:
Vicente Enrique Machaca Arceda

Arequipa, Perú
23 de marzo de 2025

Índice

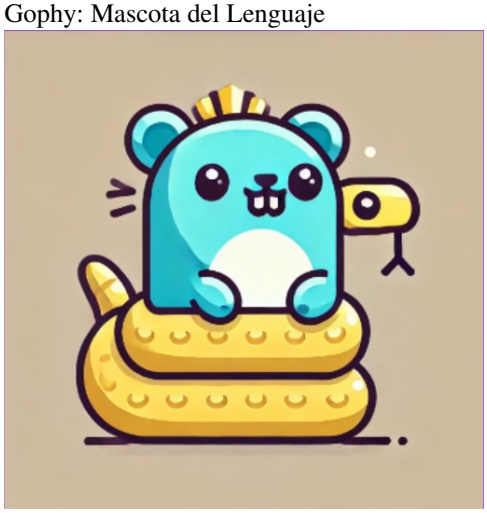
1. Introducción

En el presente informe, se expondrá el desarrollo de nuestro lenguaje en quechua, denominado "WayraSimi", durante su fase de análisis y pruebas con el analizador léxico. Para llevar a cabo esta implementación, se ha utilizado el lenguaje de programación Python, junto con la librería "ply.lex", con el objetivo de construir un compilador básico para nuestro lenguaje propuesto.

2. WayraSimi

Presentamos WayraSimi, un lenguaje de programación compilado que fusiona la claridad de Python con la eficiencia de Go. Diseñado para aplicaciones de alto rendimiento y sistemas concurrentes, WayraSimi busca ser una herramienta poderosa y accesible para desarrolladores.

- Ejemplo de documento: WayraSimi.ws
- Composición: Wayra (Viento, Aire) + Simi (Palabra, Lenguaje)
- Significado: Lenguaje del Viento o Lenguaje Veloz, implicando rapidez y eficiencia.
- Gophy: Mascota del Lenguaje



3. Justificación y Descripción del Lenguaje

3.1. Sintaxis Python-inspirada

Utiliza la indentación para definir bloques de código, buscando la legibilidad y simplicidad sintáctica de Python.

3.2. Tipado estático e inferencia de tipos

Similar a Go, WayraSimi será un lenguaje de tipado estático para garantizar la seguridad y el rendimiento.

3.3. Concurrencia integrada

Inspirado en Go, WayraSimi tendrá soporte nativo para concurrencia ligera (gorutinas) y comunicación entre procesos (canales).

4. Tipos de Datos en WayraSimi

- **Yupay (entero)**: Números enteros, sin parte decimal (int, int32, int64).
- **Chiqi_kay (flotante)**: Números con parte decimal (punto flotante, float32, float64).
- **Qillga (texto)**: Cadenas de caracteres, texto (string).

WayraSimi soporta varios tipos de datos, incluyendo enteros, flotantes, texto, booleanos, listas y mapas, cada uno con su propia representación y uso.

5. Ejemplos

```
ruray hatunRuray() {
    imprimiy(" Allin  punchaw, Pachamama!");
}
```

Listing 1: Hola Mundo

```
ruray hatunRuray() {
    para i := 0; i < 5; i++ {
        para j := 0; j < 5; j++ {
            imprimiy(i, j);
        }
    }
}
```

Listing 2: Bucles Anidados

```
ruray factorial(n yupay) yupay {
    sichus n == 0 {
        kutipay 1;
    }
    kutipay n * factorial(n-1);
}

ruray hatunRuray() {
    imprimiy(factorial(5));
}
```

Listing 3: Recursividad

6. Código en Python

6.1. pruebita.py

```

import ply.lex as lex
from prettytable import PrettyTable


tokens = [
    'YUPAY_TOKEN',
    'CHIQI_KAY_TOKEN',
    'QILLQA_TOKEN',
    'CHIQAP_TOKEN',
    'IDENTIFICADOR_TOKEN',
    'OPERADOR_MAS',
    'OPERADOR_MENOS',
    'OPERADOR_PACHA',
    'OPERADOR_RAKI',
    'OPERADOR_MODULO',
    'OPERADOR_ASIGNACION',
    'OPERADOR_IGUALDAD',
    'OPERADOR_MANA_IGUAL',
    'OPERADOR_MENOR',
    'OPERADOR_MAYOR',
    'OPERADOR_MENOR_IGUAL',
    'OPERADOR_MAYOR_IGUAL',
    'OPERADOR_LOGICO_WAN',
    'OPERADOR_LOGICO_UTAQ',
    'OPERADOR_LOGICO_MANA',
    'PARENTESIS_ABRE',
    'PARENTESIS_CIERRA',
    'LLAVE_ABRE',
    'LLAVE_CIERRA',
    'CORCHETE_ABRE',
    'CORCHETE_CIERRA',
    'COMA_TOKEN',
```

```
    'PUNTO_TOKEN',
    'DOS_PUNTOS_TOKEN',
    'PUNTO_Y_COMA_TOKEN',
    'COMENTARIO_TOKEN_LINEA',
    'COMENTARIO_TOKEN_BLOQUE_ABRE',
    'COMENTARIO_TOKEN_BLOQUE_CIERRA',
    'PALABRA_RESERVADA_SICHUS',
    'PALABRA_RESERVADA_MANA_SICHUS',
    'PALABRA_RESERVADA_MANA',
    'PALABRA_RESERVADA_PARA',
    'PALABRA_RESERVADA_RURAY',
    'PALABRA_RESERVADA_KUTIPAY',
    'PALABRA_RESERVADA_IMPRIMIY',
    'PALABRA_RESERVADA_AYLLU',
    'PALABRA_RESERVADA_VAR',
    'PALABRA_RESERVADA_TAKYAQ',
    'PALABRA_RESERVADA_UYWA',
    'PALABRA_RESERVADA_UYA',
    'HATUN_RURAY_TOKEN',
]

reserved_words = {
    'sichus': 'PALABRA_RESERVADA_SICHUS',
    'mana sichus': 'PALABRA_RESERVADA_MANA_SICHUS',
    'mana': 'PALABRA_RESERVADA_MANA',
    'para': 'PALABRA_RESERVADA_PARA',
    'ruray': 'PALABRA_RESERVADA_RURAY',
    'kutipay': 'PALABRA_RESERVADA_KUTIPAY',
    'imprimiy': 'PALABRA_RESERVADA_IMPRIMIY',
    'ayllu': 'PALABRA_RESERVADA_AYLLU',
    'variable': 'PALABRA_RESERVADA_VAR',
    'takyaq': 'PALABRA_RESERVADA_TAKYAQ',
    'uywa': 'PALABRA_RESERVADA_UYWA',
    'uya': 'PALABRA_RESERVADA_UYA',
    'chiqap': 'CHIQAP_TOKEN',
    'mana_chiqap': 'CHIQAP_TOKEN',
    'wan': 'OPERADOR_LOGICO_WAN',
    'utaq': 'OPERADOR_LOGICO_UTAQ'
}

t_OPERADOR_MAS    = r'\+'
t_OPERADOR_MENOS   = r'-'
t_OPERADOR_PACHA   = r'\*'
t_OPERADOR_RAKI    = r'/'
t_OPERADOR_MODULO  = r'%'
t_OPERADOR_ASIGNACION = r'='
t_OPERADOR_IGUALDAD = r'=='
t_OPERADOR_MANA_IGUAL = r'!='
t_OPERADOR_MENOR   = r'<'
t_OPERADOR_MAYOR   = r'>'
t_OPERADOR_MENOR_IGUAL = r'<='
t_OPERADOR_MAYOR_IGUAL = r'>='
t_PARENTESIS_ABRE  = r'\('
t_PARENTESIS_CIERRA = r'\)'
t_LLAVE_ABRE       = r'\{'
t_LLAVE_CIERRA      = r'\}'
t_CORCHETE_ABRE    = r'\['
t_CORCHETE_CIERRA   = r'\]'
t_COMA_TOKEN       = r','
t_PUNTO_TOKEN      = r'\.'
t_DOS_PUNTOS_TOKEN = r':'
t_PUNTO_Y_COMA_TOKEN = r';'

def t_CHIQI_KAY_TOKEN(t):
    r'\d+\.\d+'
    t.value = float(t.value)
    return t

def t_YUPAY_TOKEN(t):
    r'\d+'
    t.value = int(t.value)
    return t

```

```
107
108 def t_QILLQA_TOKEN(t):
109     r'("([^"]*)")|(\'([^\']*)\')'
110     t.value = t.value[1:-1]
111     return t
112
113 def t_HATUN_RURAY_TOKEN(t):
114     r'hatun_ruray'
115     return t
116
117 def t_IDENTIFICADOR_TOKEN(t):
118     r'[a-zA-Z_][a-zA-Z0-9_]*'
119     if t.value in reserved_words:
120         t.type = reserved_words[t.value]
121     return t
122
123 def t_COMENTARIO_TOKEN_LINEA(t):
124     r'\#.*'
125     pass
126
127 def t_COMENTARIO_TOKEN_BLOQUE_ABRE(t):
128     r'/\*'
129     print(">> COMENTARIO_TOKEN_BLOQUE_ABRE encontrado")
130     t.lexer.comment_start = t.lexer.lexpos
131     t.lexer.level = 1
132     t.lexer.begin('comment')
133
134 def t_comment_COMENTARIO_TOKEN_BLOQUE_CIERRA(t):
135     r'\*/'
136     print(">> COMENTARIO_TOKEN_BLOQUE_CIERRA encontrado")
137     t.lexer.level -= 1
138     if t.lexer.level == 0:
139         t.lexer.begin('INITIAL')
140
141 def t_comment_error(t):
142     print(f">> ERROR en comentario de bloque en pos {t.lexer.comment_start}:
        ↪ '{t.value[0]}'")
143     print(f"Estado Actual: {t.lexer.current_state()}")
144     t.lexer.skip(1)
145
146 def t_comment_newline(t):
147     r'\n+'
148     t.lexer.lineno += len(t.value)
149
150 t_comment_ignore = r'.'
151
152
153 states = (
154    ('comment','exclusive'),
155 )
156
157 t_ignore           = ' \t\r'
158
159 def t_newline(t):
160     r'\n+'
161     t.lexer.lineno += len(t.value)
162
163 def t_ANY_error(t):
164     print(f"Carácter alterno '{t.value[0]}' en la línea {t.lineno}, posición
        ↪ {t.lexpos}, estado: {t.lexer.current_state()}")
165     t.lexer.skip(1)
166
167 lexer = lex.lex()
168
169 def analyze_file(filepath):
170     try:
171         with open(filepath, 'r', encoding='utf-8') as file:
172             data = file.read()
173     except FileNotFoundError:
174         print(f"Error: No se encontró el archivo '{filepath}'.")
175         return
176
177     lexer.input(data)
```

```
    tokens_list = []

    table = PrettyTable(["Tipo", "Valor", "Línea", "Posición"])

    while True:
        tok = lexer.token()
        if not tok:
            break
        tokens_list.append(tok)
        table.add_row([tok.type, tok.value, tok.lineno, tok.lexpos])
    print(table)
    return tokens_list

if __name__ == '__main__':
    example_files = [
        "ejemplito1.txt",
        "ejemplito2.txt",
        "ejemplito3.txt",
        "ejemplito4.txt",
        "ejemplito5.txt"
    ]

    for filepath in example_files:
        print(f"\n--- Analizando el archivo: {filepath} ---")
        tokens = analyze_file(filepath)
        if tokens:
            print(f"Se analizaron {len(tokens)} tokens en el archivo '{filepath}'.")
```

Listing 4: Lexema en lenguaje Python

6.2. Explicación pruebita.py

6.2.1. Importación de la librería

Se importa la librería **ply.lex**, que es una herramienta para construir analizadores léxicos en Python.

6.2.2. Definición de los tokens

Se define una lista de tokens (**'YUPAY_TOKEN'**, **'CHIQAP_TOKEN'**, etc), que son los componentes léxicos que el analizador reconocerá. Cada token representa un elemento del lenguaje, como números, operadores, palabras reservadas, etc.

6.2.3. Expresiones regulares para tokens simples

Aquí se definen las expresiones regulares para los tokens simples, como operadores (+, -, *, etc.), símbolos de puntuación (",", ";", ":", etc.) y palabras reservadas (sichus, mana, ruray, etc.).

6.2.4. Funciones para tokens complejos

Algunos tokens requieren lógica adicional para ser reconocidos. Por ejemplo:

- **Números (YUPAY_TOKEN):** Reconoce números enteros o decimales y los convierte a int o float.
- **Booleanos (CHIQAP_TOKEN):** Reconoce los valores booleanos chiqap (verdadero) y mana_chiqap (falso).
- **Cadenas de texto (QILLQA_TOKEN):**Reconoce cadenas de texto entre comillas simples o dobles.
- **Identificadores (IDENTIFICADOR_TOKEN):**Reconoce nombres de variables o funciones.

6.2.5. Manejo de comentarios y espacios

- Comentarios de línea
- Espacios y tabulaciones
- Saltos de línea

6.2.6. Manejo de errores

Si se encuentra un carácter no reconocido, se imprime un mensaje de error y se ignora el carácter.

6.2.7. Creación del lexer

Se crea una instancia del analizador léxico.

6.2.8. Pruebas y análisis de archivos

- **Función test_lexer:** Prueba el lexer con una cadena de entrada.
- **Función analyze_file:** Lee un archivo y analiza su contenido léxicamente.

6.3. Ejecución principal

Se analizan varios archivos de ejemplo y se imprimen los tokens encontrados.

6.4. Output del Pruebita

```
--- Analizando el archivo: ejemplito1.txt ---
+---------------------------+---------------------------+-------+----------+
|           Tipo            |           Valor           | Línea | Posición |
+---------------------------+---------------------------+-------+----------+
|  PALABRA_RESERVADA_RURAY  |           ruray           |   1   |    0     |
|     HATUN_RURAY_TOKEN     |        hatun_ruray        |   1   |    6     |
|      PARENTESIS_ABRE      |             (             |   1   |    17    |
|     PARENTESIS_CIERRA     |             )             |   1   |    18    |
|         LLAVE_ABRE        |             {             |   1   |    20    |
| PALABRA_RESERVADA_IMPRIMIY |          imprimiy         |   2   |    26    |
|      PARENTESIS_ABRE      |             (             |   2   |    34    |
|        QILLQA_TOKEN       | ¡Allin punchaw, Pachamama! |   2   |    35    |
|     PARENTESIS_CIERRA     |             )             |   2   |    63    |
|     PUNTO_Y_COMA_TOKEN    |             ;             |   2   |    64    |
|        LLAVE_CIERRA       |             }             |   3   |    66    |
+---------------------------+---------------------------+-------+----------+
Se analizaron 11 tokens en el archivo 'ejemplito1.txt'.
```