**Documentation for Scanner Class**

**Overview**

The Scanner class performs lexical analysis on a mini-language input file. It reads tokens from a source file, categorizes them, and outputs two files:

* **PIF.out** (Program Internal Form): Lists tokens and their positions.
* **ST.out** (Symbol Table): Contains constants and identifiers found in the source code.

The Scanner class:

1. Uses regular expressions (regex) to identify identifiers, constants, and specific tokens.
2. Validates tokens against known types (keywords, operators, separators).
3. Detects lexical errors, including unmatched literals and invalid numbers.

**Class Initialization**

* \_\_init\_\_(self, token\_file): Initializes the Scanner with a list of accepted tokens from the token\_file. It also creates an empty Program Internal Form (pif), error list (errors), and symbol table (symbol\_table) for storing identifiers and constants.

**Attributes:**

* token\_list: List of recognized language-specific keywords, operators, and symbols.
* pif: A list holding tokens and positions for valid tokens.
* errors: A list that tracks all detected lexical errors.
* symbol\_table: Stores unique identifiers and constants, leveraging a hash table structure with list-based buckets for collision handling.

**Methods**

**1. is\_identifier(token)**

Checks if a token is a valid identifier, adhering to the format:

* Begins with a letter or '~'
* May contain letters, digits, or '~'

**Regex pattern**: ^[a-zA-Z~][a-zA-Z0-9~]\*$

**2. is\_constant(token)**

Determines whether a token is a valid constant of type int, char, or string.

**Regex patterns**:

* **Integer**: Matches 0 or any non-zero integer, including negative values. ^0$|^[+-]?[1-9][0-9]\*$
* **Character**: Matches a single alphanumeric or special character between single quotes: ^'[a-zA-Z0-9~!@#$%^&\*()\_+{}|:<>?;',.\\/\[\]-]'$
* **String**: Matches any sequence up to 256 characters enclosed in double quotes: ^".{0,256}"$

Returns "int", "char", or "string" based on the match, or None if no match is found.

**3. scan(program\_file)**

Processes an input source code file line-by-line:

1. Strips whitespace and scans each line for tokens using scan\_line.
2. If errors are detected, they are printed and stored in the errors list.
3. Returns True if no errors were found, otherwise False.

**4. scan\_line(line, line\_num)**

Parses a single line for tokens using a combination of regex patterns:

* **Regex**: \".\*?\"|\'[^\']\*\'|[-+]?\d+|\w+|!=|==|<=|>=|[^\s\w]
  + Matches strings, characters, integers, identifiers, and operators.
* **Token Validation**: Each token is validated in this order:
  + **Invalid numeric literals**: Tokens like 07 are flagged as errors.
  + **Unknown tokens**: If a token doesn’t match identifiers, constants, or keywords, it’s flagged.

Depending on the token:

* **Reserved Keywords**: Added directly to pif with a position of -1.
* **Identifiers**: If new, it is added to the symbol\_table; otherwise, the existing position is referenced.
* **Constants**: Similarly handled by checking if they’re already in symbol\_table.

**5. write\_outputs()**

Writes the output files PIF.out and ST.out.

* **PIF.out**: Contains each token in pif with its symbol table position.
* **ST.out**: Contains the string representation of the symbol\_table, showing constants and identifiers only once, in the order they first appeared.