MyScanner Class Documentation - Updated

Class Description

The MyScanner class is responsible for reading a source file, tokenizing it, and storing the tokens in the ProgramInternalForm and SymbolTable. This class is designed to parse a simple programming language by identifying tokens such as operators, separators, constants, identifiers, and reserved words.

Function: __init__

Initializes the MyScanner class with the specified file path, a SymbolTable, and a ProgramInternalForm.

- Parameters:
  - file_path (str): The path of the file to be scanned.
- Initializes:
  - symbol_table: An instance of SymbolTable to store identifiers and constants.
  - pif: An instance of ProgramInternalForm to store token information.
- Additionally defines:
  - operators: List of supported operators.
  - separators: List of separators like braces, brackets, quotes, and whitespace.
  - reserved_words: List of reserved words in the language.

Function: read_file

Reads the content of the file located at self.file_path and removes any tab characters.

- Returns:
  - str: The file content as a single string without tabs.
  - None: If the file is not found, returns None and prints a message.

MyScanner Class Documentation - Updated

Function: create_list_of_program_elements

Splits the program content into tokens, identifiers, constants, and separators.

- Calls:
  - read_file() to get the content of the file.
  - re.split() to split the content based on the defined separators.
- Filters out empty tokens except for newline and space characters.
- Returns:
  - list of tokens, where each token is a string or None if file content is not available.

Function: tokenize

Processes each token, tracks line and column positions, and returns a list of tokens with their positions.

- Parameters:
  - tokens_to_be (list): The list of tokens to be processed.
- Uses:
  - is_string_constant and is_char_constant to identify tokens within string or character literals.
  - created_string to build multi-character tokens.
  - number_line and number_column for tracking token positions.
- Returns:
  - list of tuples containing token and its Pair position.

Function: scan

Main function that classifies each token and detects lexical errors.

- Calls:

MyScanner Class Documentation - Updated

  - create_list_of_program_elements() to obtain tokens.

- Classifies tokens as:

  - Reserved words, operators, separators, constants, or identifiers.

- Uses regex patterns to identify constants and identifiers.

- Adds each token to ProgramInternalForm and SymbolTable.

- Outputs:

  - Prints an error message for any invalid token with its line and column position.

  - Flags unexpected identifiers after closing blocks or misplaced tokens.

Function: get_pif

Accessor method to return the ProgramInternalForm instance.

- Returns:

  - ProgramInternalForm: The PIF instance containing classified tokens.

Function: get_symbol_table

Accessor method to return the SymbolTable instance.

- Returns:

  - SymbolTable: The SymbolTable instance containing identifiers and constants.

Input File Examples

File: p1.txt

MyScanner Class Documentation - Updated

This file contains a lexically correct example to check if a number is prime.

Example content:

Declaration:

n -> integer

i -> integer

flag -> integer

start

read(n),

flag => 1,

for(i => 2; i * i <= n; i => i + 1) {

    if (n % i == 0) {

        flag => 0,

        break,

    }:

}

if (flag == 1) {

    write("Number is prime");

}:

else {

    write("Number is not prime");

```
}:


end
```

File: p2.txt

This file contains a lexically correct example that calculates the sum of integers up to n.

Example content:

Declaration:

n -> integer

sum -> integer

i -> integer

```
start


read(n),

sum => 0,


for(i => 1; i <= n; i => i + 1) {

    sum => sum + i,

}:


write(sum),
```

end

File: p3.txt

This file contains a lexically correct example that finds the maximum of three numbers.

Example content:

Declaration:

a -> integer

b -> integer

c -> integer

max -> integer

start

read(a),

read(b),

read(c),

if (a > b) {

    max => a;

}:

else {

```
    max => b;
}:

if (c > max) {
    max => c;
}:

write("Maximum value is "),
write(max),

end
```

File: p1err.txt

This file contains a lexically incorrect example to test the scanner's error detection.

Example content:

Declaration:

a -> integer

b -> integer

max -> integer

```
start
```

MyScanner Class Documentation - Updated

```
read(a),

readd(b),  # Error: Typo in `read`



if (a > b) {

    maxx = a;  # Error: `maxx` is an undeclared variable

}

else

{

    max => b,

}:



if (c > max) {  # Error: `c` is undeclared

    max => c;

}:



write('a),

write(max),



end
```

Errors in this example:

1. Typo in the `read` keyword: `readd(b)`.

2. Undeclared variable `maxx`.

3. Usage of undeclared variable `c`.

4. Syntax error with missing single quote in `write('a)`.