Moldovan Vasilica-Andreea

Group 935/1

Git link: https://github.com/VasilicaMoldovan/FLCD

```
'''
Preconditions: None
Postconditions: This function reads token.in and stores
the the tokens and their associated code
'''
def generate_codes(self):
```

```
'''
Preconditions: identifier - a string
Postconditions: returns True if the given string respects the
                criterias for a valid identifier, or False
                otherwise
'''
def is_identifier(self, identifier):
```

```
'''
Preconditions: constant - a string
Postconditions: returns True if the given string respects the
                criterias for a valid constant, or False
                otherwise
'''
def is_constant(self, constant):
```

```
'''
Preconditions: word - a string
Postconditions: returns True if the given string is a
                reserved word, or False otherwise
'''
def is_reserverd_word(self, word):
```

```
'''
Preconditions: None
Postconditions: Function which detects and classifies the tokens,
reading them from a given file. It also constructs the Symbol Table
and checks if there are lexical errors in the program.
'''
def tokenize(self):
```

```
'''
Preconditions: None
Postconditions: Function which constructs PIF for a given program.
The PIF is stored as a dictionary, which has as key the token, and
as value its positions in the symbol table if it's an identifier or
constant,
or -1 otherwise.
'''
def construct_pif(self):
```

For a correct program, such as p1:

```
individual a, b, gcd;
come a;
come b;
parsing (a != b) {
situation (a > b)
a = a - b;
other
b = b - a;
}
gcd = a;
leave gcd;
```

The Symbol Table is(ST.out):

```
['a', 0]['b', 0]['gcd', 0]
```

And the PIF is(PIF.out):

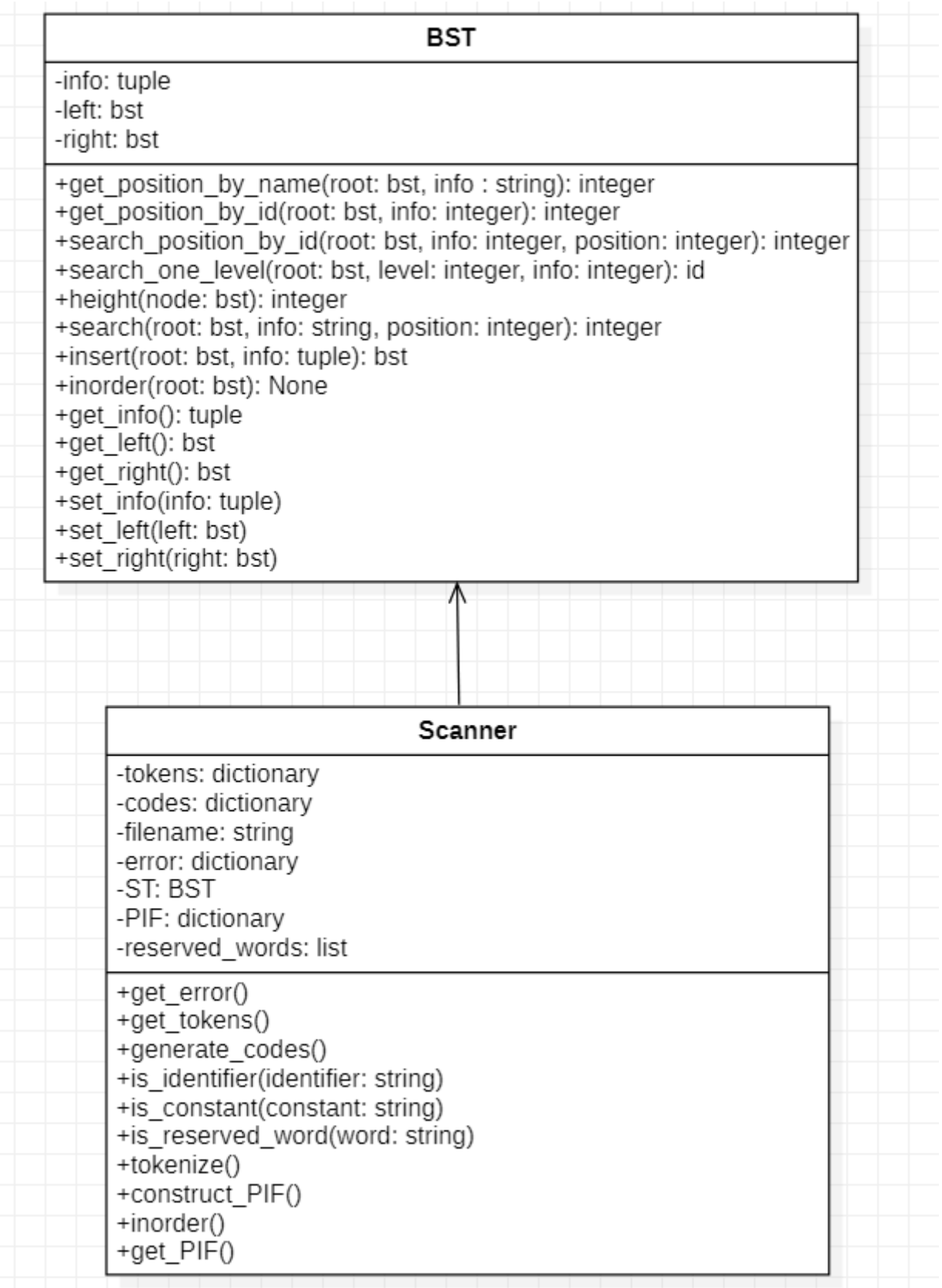
```
'individual': -1, 'a': 0, 'b': 1, 'gcd': 2, 'come': -1, 'parsing': -1,
'situation': -1, 'other': -1, 'leave': -1}
```

And for a program with a lexical error, such as p3(where arr[2i] is the error):

```
individual arr[100];
decision is_smaller = true;
individual n, i;
individual max_number;
come n;
come m;
i = 0;
parsing ( i < n ) {
come arr[i];
situation (i == 0)
max_number = arr[2i];
other {
situation ( max_number < arr[i] )
max_number = arr[i];
}
i = i + 1;
}
situation ( max_number >= m )
is_smaller = false;
leave "The maximum number is";
leave max_number;
leave " and is smaller than ";
leave m;
leave " - ";
leave is_smaller;
```

The Symbol Table(ST.out) and the PIF (PIF.out) are empty.

And the error is print on the console.
{'Invalid identifier at line ': 11}


BST
-info: tuple
-left: bst
-right: bst
+get_position_by_name(root: bst, info : string): integer
+get_position_by_id(root: bst, info: integer): integer
+search_position_by_id(root: bst, info: integer, position: integer): integer
+search_one_level(root: bst, level: integer, info: integer): id
+height(node: bst): integer
+search(root: bst, info: string, position: integer): integer
+insert(root: bst, info: tuple): bst
+inorder(root: bst): None
+get_info(): tuple
+get_left(): bst
+get_right(): bst
+set_info(info: tuple)
+set_left(left: bst)
+set_right(right: bst)
Scanner
-tokens: dictionary
-codes: dictionary
-filename: string
-error: dictionary
-ST: BST
-PIF: dictionary
-reserved_words: list
+get_error()
+get_tokens()
+generate_codes()
+is_identifier(identifier: string)
+is_constant(constant: string)
+is_reserved_word(word: string)
+tokenize()
+construct_PIF()
+inorder()
+get_PIF()