**Lab 4 – INTEGRATION**

**REQUIREMENTS**

Final evaluation of scanner (integrate L1, L2, L3)

**DOCUMENTATION**

For both symbol tables of identifiers and constants I used the ST implemented in the previous lab using HashTable.

For the program information file, I am using a list of pairs: code + position in ST (both unique). If the element is not in the ST (operators, separators, reserved words) position is (-1, -1).

The codes are provided in the tokens.in file, from where they are collected at the beginning of the program (identifiers: 0, constants: 1).

The program starts reading line by line the file it was provided, and tokenizes it on the spot – it divides the line in a list of each element on the line: identifiers, constants, operators, separators. At this point, the program starts identifying each element from the tokenized line: if the codes list contains the token, it only adds it to the PIF as it is a reserved word, separator or operator. If not, then it adds it to the corresponding symbol table (STI – identifiers or STC – constants) and then to the PIF. If there is any token that cannot pe identified, an error is returned along with the line it occurs on.

**CLASS SCANNER**

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **DESCRIPTION** | **PRE** | **POST** |
| run | returns the list of errors which occurred while scanning the file provided in the path | *path: Path* – the path of the program to be run | *errors:* *String[ ]* – the list of lexical error found in the file run |
| tokenizeLine | tokenizes the line provided and returns the list of obtained tokens | *line: String[ ]* – list of all tokens (until now) from the line  *separator: String* – the new separator the line will pe split by | *newTokens: String[ ]* – the list of new tokens found after the split by separator |
| identify | categorizes the token as separator/operator/reserved word, identifier or constant | *token: String* – the token to be identified | *void* – calls the right function according to what the token was identified by or throws an error if not identifiable |
| addToPIF | adds the corresponding pair of (code – position) to the PIF | *token: String* – token that should be added to PIF | *void* – (code- position) of the token is added to PIF |
| addToST | adds the token to its position in the corresponding symbol table | *token: String* – token that should be added to ST  *ST: SymbolTable* – symbol table the token should be added to | *void* ­– token is added to ST |
| getCodes | retrieves the codes from the file tokens.in | *tokens.in* – input text file | *void* – codes from tokens.in are added to a String[ ] |

**LANGUAGE SPECIFICATION**

**Alphabet:**

Upper (A-Z) and lowercase letters (a-z) of the English alphabet

Decimal digits (0-9)

**Lexic:**

a. Special symbols, representing:

- operators

- arithmetic: + - \* / %

- relational: > >= < <= = !=

- logical: and or not

- assignment: is

- separators ( ) [ ] { } ; space

- reserved words: inty booly ify elseify elsy loopy to begin\_appy end\_appy sparkle arry

b. Identifiers - a sequence of letters and digits, such that the first character is a letter

identifier = letter{letter | digit}

letter = “A” | ‘B” | ... | “Z” | “a” | “b” | ... | “z”

digit = “0” | “1” | ... | “9”

nz\_digit = “1” | ... | “9”

c. Constants

- integer = [“ – “] ( “0” | nz\_digit { “0” | nz\_digit } )

- bool = “true” | “false”

***P1.in – STs and PIF***

begin\_appy

inty a, b, c, max;

a is -10;

b is 20;

c is 12;

ify ( a>=b and a >= c ) {

max is a;

} elseify ( b >= a and b >= c) {

max is b;

} elsy {

max is c;

}

end\_appy

--- STI ---

97 [a]

98 [b]

99 [c]

70 [max]

--- STC ---

98 [20]

99 [12]

142 [-10]

***P1.out***

Lexically correct

--- PIF ---

(2, (-1, -1))

(5, (-1, -1))

(1, (97, 0))

(24, (-1, -1))

(1, (98, 0))

(24, (-1, -1))

(1, (99, 0))

(24, (-1, -1))

(1, (70, 0))

(23, (-1, -1))

(1, (97, 0))

(13, (-1, -1))

(0, (142, 0))

(23, (-1, -1))

(1, (98, 0))

(13, (-1, -1))

(0, (98, 0))

(23, (-1, -1))

(1, (99, 0))

(13, (-1, -1))

(0, (99, 0))

(23, (-1, -1))

(8, (-1, -1))

(17, (-1, -1))

(1, (97, 0))

(25, (-1, -1))

(1, (98, 0))

(14, (-1, -1))

(1, (97, 0))

(25, (-1, -1))

(1, (99, 0))

(18, (-1, -1))

(21, (-1, -1))

(1, (70, 0))

(13, (-1, -1))

(1, (97, 0))

(23, (-1, -1))

(22, (-1, -1))

(9, (-1, -1))

(17, (-1, -1))

(1, (98, 0))

(25, (-1, -1))

(1, (97, 0))

(14, (-1, -1))

(1, (98, 0))

(25, (-1, -1))

(1, (99, 0))

(18, (-1, -1))

(21, (-1, -1))

(1, (70, 0))

(13, (-1, -1))

(1, (98, 0))

(23, (-1, -1))

(22, (-1, -1))

(10, (-1, -1))

(21, (-1, -1))

(1, (70, 0))

(13, (-1, -1))

(1, (99, 0))

(23, (-1, -1))

(22, (-1, -1))

(3, (-1, -1)

***Perr.in – ST’s and PIF***

begin\_appy

inty a;

a is 2#3;

booly 3rror;

end\_appy

***Perr.out***

Line 3: Illegal token "2#3".

Line 4: Illegal token "3rror".

--- STI ---

97 [a]

--- STC ---

--- PIF ---

(2, (-1, -1))

(5, (-1, -1))

(1, (97, 0))

(23, (-1, -1))

(1, (97, 0))

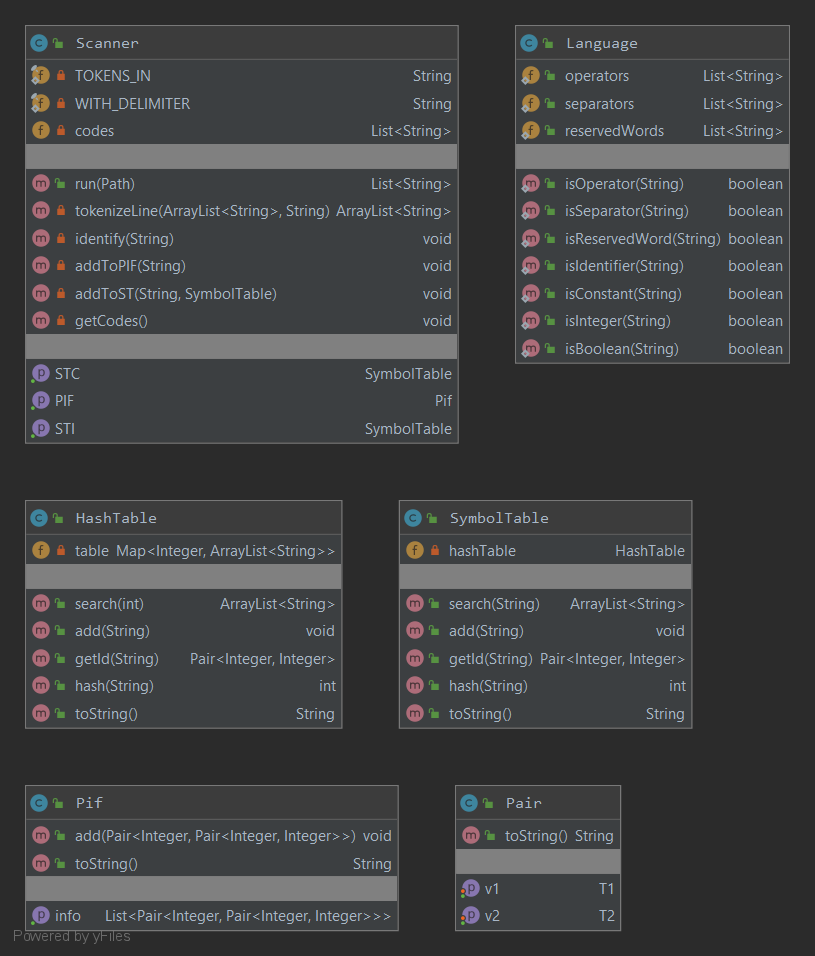
(13, (-1, -1))

(23, (-1, -1))

(6, (-1, -1))

(23, (-1, -1))

(3, (-1, -1))



1

1

2