Github Repository: <https://github.com/DragosMoldovanu/PLCD-Lab2>

Documentation:

UML Diagram:



Scanner Methods

scan()

Scans the source file, tokenizes it and distributes the tokens into PIF and ST

Input:-

Output:-

Pre: Input file must exist

Post: Tokens will be distributed into PIF and ST

separate(token, line)

Recursive function that separates a composed token into its base tokens

Input: token -> String

line -> int

Output:-

Pre: token contains no empty spaces

Post: Base tokens will be distributed into PIF and ST

identify(token, line)

Takes a token and identifies whether it is a keyword, an identifier or a constant

Input: token -> String

line -> int

Output:-

Pre: token contains no empty spaces and no delimiters

Post: the corresponding token of the correct time is put into PIF and ST

processString(token)

Adds the string delimiters to PIF and dumps the entire content of the string as a constant

Input: token -> String

Output:-

Pre:-

Post: the string is placed as a single constant into PIF and ST

processChar(token)

Adds the char delimiters to PIF and checks char vailidity before adding it to PIF and ST

Input: token -> String

Output:-

Pre:-

Post: if the string contains one single character, it will be added to PIF and ST as constant

scanInteger(token)

Checks if the string represents a constant integer and adds it to PIF and ST if it is

Input: token -> String

Output: true/false

Pre: token contains no empty spaces and no delimiters

Post: if the string fulfills the requirements, it is added to PIF and ST as a constant

scanBoolean(token)

Checks if the string represents a constant boolean and adds it to PIF and ST if it is

Input: token -> String

Output: true/false

Pre: token contains no empty spaces and no delimiters

Post: if the string fulfills the requirements, it is added to PIF and ST as a constant

scanIdentifier(token)

Checks if the string represents an identifier and adds it to PIF and ST if it is

Input: token -> String

Output: true/false

Pre: token contains no empty spaces and no delimiters

Post: if the string fulfills the requirements, it is added to PIF and ST as an identifier

SymbolTable methods:

addNode(identifier, id)

Adds a new Identifier as a node in the Symbol Table BST

Input: identifier - String

id - Integer

Output:-

Pre: identifier does not exist already

id does not yet exist

Post: Symbol Table will have the new node, correctly placed in the BST

findPosition(node, identifier, id)

Recursive function for adding a node, parses the tree looking for the correct position

Input: node - IdentifierNode, a node from the BST

identifier - String

id - Integer

Output:-

Pre: identifier does not exist already

id does not yet exist

Post: Symbol Table will have the new node, correctly placed in the BST

getFromName(identifier)

Returns the identifier-id pair from the BST, looking for the given identifier

Input: identifier - String

Output: Identifier - the pair containing the given string

Pre: the identifier exists in the BST

Post:-

findNodeByName(identifierNode, identifier)

Recursive function for looking for a pair by name

Input: node: IdentifierNode - a node from the BST

identifier: String

Output: Identifier - the pair containing the given string

Pre: the identifier exists in the BST

Post:-

getFromId(id)

Returns the identifier-id pair from the BST, looking for the given id

Input: id - Integer

Output: Identifier - the pair containing the given integer

Pre: the id exists in the BST

Post:-

findNodeById

Recursive function for looking for a pair by id

Input: node: IdentifierNode - a node from the BST

id: Integer

Output: Identifier - the pair containing the given integer

Pre: the id exists in the BST

Post:-

token.in content:

\+ 1  
- 2  
\\* 3  
/ 4  
<= 5  
== 6  
>= 7  
> 8  
= 9  
< 10  
\( 11  
\) 12  
\[ 13  
\] 14  
\{ 15  
\} 16  
; 17  
\" 18  
' 19  
  
int 20  
string 21  
char 22  
bool 23  
if 24  
else 25  
read 26  
write 27  
while 28  
true 29  
false 30  
  
INT 31  
CHAR 32  
STRING 33  
BOOL 34  
IDENTIFIER 35

EXAMPLES:

Example 1:

p1.in content:

int a;  
int b;  
int c;  
int d;  
a=read;  
b=read;  
c=read;  
d=read;  
int sum;  
sum=a+b+c;  
if(sum==d){  
 write="yes";  
}  
else{  
 write="no";  
}

pif.out content:

21 -1  
0 1  
18 -1  
21 -1  
0 2  
18 -1  
21 -1  
0 3  
18 -1  
21 -1  
0 4  
18 -1  
0 5  
10 -1  
27 -1  
18 -1  
0 6  
10 -1  
27 -1  
18 -1  
0 7  
10 -1  
27 -1  
18 -1  
0 8  
10 -1  
27 -1  
18 -1  
21 -1  
0 9  
18 -1  
0 10  
10 -1  
0 11  
2 -1  
0 12  
2 -1  
0 13  
18 -1  
25 -1  
12 -1  
0 14  
7 -1  
0 15  
13 -1  
16 -1  
28 -1  
10 -1  
19 -1  
1 -1  
19 -1  
18 -1  
17 -1  
26 -1  
16 -1  
28 -1  
10 -1  
19 -1  
1 -1  
19 -1  
18 -1  
17 -1

st.out content:

a 1  
b 2  
c 3  
d 4  
sum 9

Example 2:

perr.in content:

int a;  
int b;  
int 5c;  
int d;  
a=<read>;  
b=<read>;  
c=<read>;  
d=<read>;  
int sum;  
sum=a+b+c;  
if(sum==d){  
 <write>='yes';  
}  
else{  
 <write>="no";  
}

output:

Lexical error on line 3: Invalid token - 5c

Lexical error on line 12: Invalid character constant - 'yes'