

Psueudocode

Main

`main (int argc, const char *argv[])`

Create a scanner

Create a token

Have scanner start reading the file

Check for <identifiers>

Return 0

Print

`Print (char source_name[], char date[])`

Set the filename to source_name

get current time

set the print date to the current time

set the page number to zero

`printLine (char line[])`

increment the line count

if line_count > page height Print header

print the string argument

`PrintPageHeader ()`

Print header (page number, source file name , current date)

`PrintToken (Token *token)`

```
Increment the line count
Switch (token->getCode()) {
    NUMBER is an integer -> print integer
    Number is a real -> print a real
    String is a string -> print the string
Default -> print token
}
```

Scanner

Scanner (FILE *source_file, char source_name[], char date[], Print printer)

```
Src_file= source_file
Copy (src_name, source name)
Copy (todays_date, date)
Initialize char table to identify what type of char we are looking at
Initialize Line number=0
Source line [0] = '\0'
```

getSourceLine(char source_buffer)

```
create source buffer
create false Boolean
get a line from the filestream
if line received then true
return Boolean
```

getToken()

```
initialize a character code variable
skip past all the blanks
```

examine ch for LETTER, DIGIT, QUOTE, EOF, or SPECIAL
call appropriate function depending on ch
return new_token

`getChar(char souce_buffer[])`

set a temp char to EOF
if at the end of line ->return null character
else return the char at the index

`skipBanks (char source_buffer[])`

skip past the blanks
return pointer to the first non blank character

`skipComments (char source_buffer[])`

skip past the comments
return pointer to the first non blank character
watch for the EOF character

`getWord (char *str, char *token_ptr, Token *tok)`

Extract the word
Downshift the word, to make it lower case
Check if the word is a reserved word
If is not a reserved word its an identifier
Set token to identifier

`getNumber (char *str, char *token_ptr, Token *tok)`

extract number and convert it to a literal number
check if real or float
temp string number

set the token type to NUMBER

getString (char *str, char *token_ptr, Token *tok)

Initialize a temporary string

While char ch is not a '\'

 Read more characters

 Append characters to temp string

Set the setType to STRING_LIT

Set the setCode to STRING

getSpecial (char *str, char *token_ptr, Token *tok)

initialize the temp string

check for character operators (:,<,>,,|)

 read next character

 if (= or .) -> append both to temp string

 else -> append first character to the temp string

downshiftWord (char word[])

make all characters in the incoming word lower case

isReservedWord (char *str, Token *tok)

Scan the token table for reserved words

If it is a reserved word -> set the token code member ->return True

Else -> return False

getLineNumber ()

return line_number

Token

Token ()

Initialize variables for binary search tree (lines, left, right)

setCode (TokenCode newCode)

set newCode

getCode ()

Return token code member

setType (LiteralType newType)

set Type

getType ()

return Token code member

setLiteral (int newInteger)

setLiteral to integer

getIntLiteral ()

return Token code member

setLiteral (int newReal)

setLiteral to real

getRealLiteral ()

return Token code member

setLiteral (string newString)

setLiteral to String

malloc space for string
copy stringLiteral to newString
`getStringLiteral ()`
return Token code member

`setTokenString (string s)`
setTokenString = s
`getTokenString (string s)`
return Token code member

`getLeft ()`
get left "leaf"

`getRight ()`
get right "leaf"

//implement binary tree

`addLineNumber (int lineNumber)`
add line number to node

`addTokenNodeToBinarySearchTree(Token* &headToken, Token* newToken, int lineNumber)`
Add token node to the binary search tree

`getLinesString ()`
return lone to ToString Expand

getBinarySearchTreeLinesStringsInOrder (Token*head)

arrange print of head

get head left or right

LineNumberNode

LineNumberNode ()

Set val to 0

Set next to NULL

LineNumberNode (int val)

Set val to val

Set next to NULL

addNode (LineNumberNode *&headNode, LineNumberNode* newNode)

if headNode= Null

headNode is NewNode

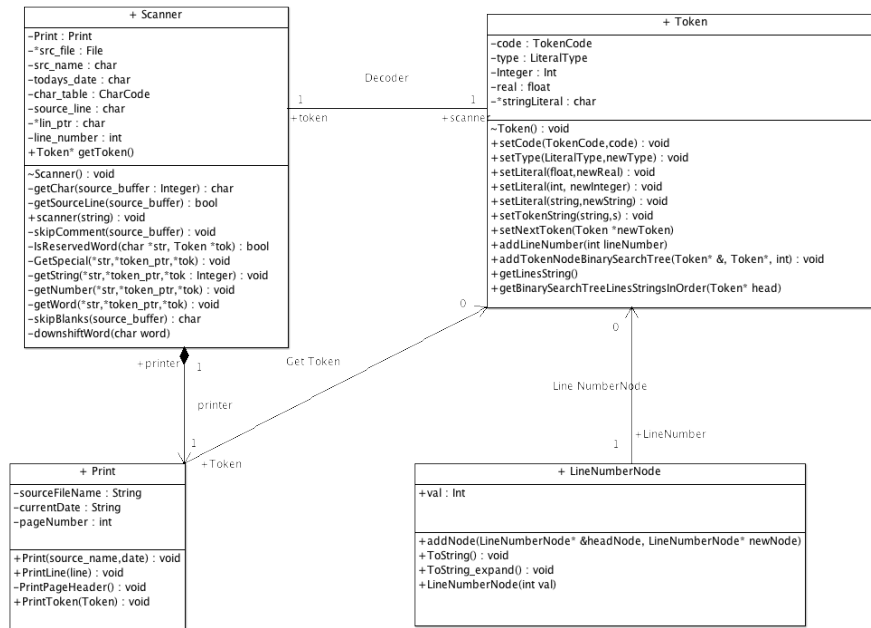
return

else

ptr next= newNode

return

UML Diagram



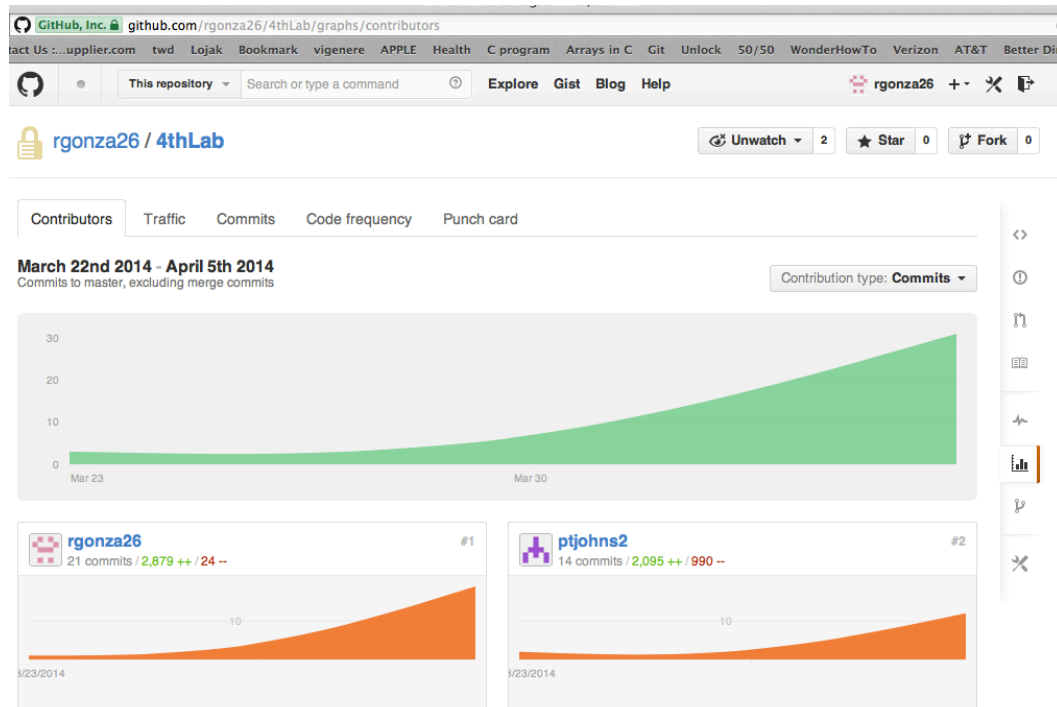
Team Contribution Table

Name	Login	Score (0= No contribution 2= good contribution)
Peter Johnson	ptjohns2	2
Roberto Gonzalez	rgonza26	2

URL to

Repository:

<https://github.com/rgonza26/4thLab>



Function	Parameter (V->Valid, I ->Invalid)	Expected Result	Acutal Result
main	argv		
Test 1	V	Program Produces Expected Results	Expected result
Test 2	I	Application Crashes	Expected result

Function	Parameter (V->Valid, I ->Invalid)	Expected Result	Acutal Result
print	sourceFileName		
Test 3	V	File Name matches valid flie name	Expected result
Test 4	I	File Name does not match valid file name	Expected result
printLine	line		
Test 5	V	prints line	Expected result
Test 6	I	prints nothing	Expected result
printPageHeader			
Test 7		prints header	Expected result
printToken	Token		
Test 8	V (4)	Prints the integer literal	Expected result
Test 9	V (3.1)	Prints the Float literal	Expected result
Test 10	V ("test")	Prints the STRING literal	Expected result
Test 11	V ("PROGRAM")	Prints the Token	Expected result
Test 12	I	Nothing Prints NO_Token type	Expected result

Function	Parameter (V->Valid, I ->Invalid)	Expected Result	Actual Result
Scanner	Tested during Main		
getSourceLine	Tested during Main		
getToken	Tested during Main		
getChar	source_buffer		
Test 13	V ("test)	returns 't'	Expected result
Test 14	I ""	returns"	Expected result
skipBlanks	source_buffer	returns 2 returns 1	Expected result Expected result
Test 15	V (two spaces)		
Test 16	I (spaces here"		
skipComment	source_buffer	Removes Comment from source line Prints the line	Expected result Expected result
Test 17	V "this is a {comment}"		
Test 18	I "this is a comment"		
getWord	ch	Tested during print Empty token string	Expected result Expected result
Test 19	V 'a'		
Test 20	I '{'		
getNumber	ch	Tested during print Empty token string	Expected result Expected result
Test 21	V '5'		
Test 22	I '{'		
getString	ch	Tested during print Empty token string	Expected result Expected result
Test 23	V		
Test 24	I '{'		
getSpecial	str	returns 1 returns false	Expected result Expected result
Test 25	V "program"		
Test 26	I 'nothing"		

downshiftfWord	char		
Test 27	V 'OK'	ok	Expected result
Test 28	I '{'	Empty token string	Expected result
isREservedWord	ch		
Test 29	V "DO"	Tested during print	Expected result
Test 30	I '{'	Empty token string	Expected result

Function	Parameter (V->Valid, I ->Invalid)	Expected Result	Acutal Result
Token	Tested during Main		
setCode	Tested during Main		
setType	Tested during Main		
setLiteral	Tested during Main		
setLiteral	Tested during Main		
setLiteral	Tested during Main		
setTokenString	Tested during Main		
getLeft	ch		
Test 31	V (1)	left=1	Expected result
Test 32	I (")	invalid	Expected result
getRight	ch		
Test 33	V(2)	right=2	Expected result

Test 34	I(~)	invalid	Expected result
addLineNumber	int		
Test 35	V(1)	lineNumber=1	Expected result
Test 36	I(A)	invalid	Expected result
addTokenNodeToBinary- SearchTree	ch, ch, int		
Test 37	V V V	added TokenNode to BST	Expected result
Test 38	I	invalid	Expected result
getBinarySearchTree- LinesStringsInOrder	char		
Test 41	V	return oss.str()	Expected result
Test42	I	invalid	Expected result

Function	Parameter (V->Valid, I ->Invalid)	Expected Result	Acutal Result
LineNumberNode			
addNode	char		
Test 43	V	LineNumberNode* ptr= headNode	Expected result
Test 44	I	invalid	Expected result