Pseudocode

Main

**Function Declaration**: int main(int argc, const char \* argv[])

Start Scanner

Token list gets memory space allocated

Continue loop while token string is not a dot “.” <- Sentinel Value

Quit scanner

Return 0

**Function Declaration**: void add\_token\_to\_list(Token \*list, Token \*new\_token)

Adds a new token list to the list knowing that list is a linked list

**Function Declaration**: void quit\_scanner(FILE \*src\_file, Token \*list)

Check list of tokens

Free/deallocate all token list memory

Close Source File

**Function Declaration**: FILE \*init\_lister(const char \*name, char source\_file\_name[], char dte[])

Get time and date

Open File

Print

**Function Declaration**: void print\_line(char line[], char source\_name\_to\_print[], char date\_to\_print[])

New page resets count

New line when reaching “Max\_Print\_Line\_Length”

**Function Declaration**: static void print\_page\_header(char source\_name[], char date[])

Page Header gets formatted

**Function Declaration**: void print\_token(Token \*token)

Determine what function is in scanner and format according to type

Scanner

**Function Declaration**: void init\_scanner(FILE \*source\_file, char source\_name[], char date[]

Initialize Character table

Initialize Array to UNUSED

[48,57]=Digit

[65,90]= Letter

[95,122] = Letter

Set Special Characters

**Function Declaration**: BOOLEAN get\_source\_line(char source\_buffer[])

Create a character buffer

Set line\_number to zero

Get a line from the filestream

If a line is received set Boolean to true

Else return false

**Function Declaration**: Token\* get\_token()

Set current character being examined

Store token

Point to the beginning of token\_string

Return token

Skip blanks

Type of case (letter, digit, quote, eof, or special)

**Function Declaration**: static char\* get\_char(char\* arg\_charPtr)

Check if at the end of current line

Call source line

When in the EOF -> return null character

**Function Declaration**: static char\* skip\_blanks(char\* str)

Skip past the blanks

Return pointer to the first non blank character

**Function Declaration**: static char\* skip\_comment(char\* str)

Skip past the comments

Return pointer to the first non-blank character

**Function Declaration**: static BOOLEAN get\_word(char buffer[], char\* ptr)

Store work in buffer

Extract the word

Downshift the word to make it lower case

If not a reserved word then it is an identifier

**Function Declaration**: static LiteralValue get\_number(char\* str)

Extract number

Convert number to a literal number

Set the token type to number

**Function Declaration**: static LiteralValue get\_string(char buffer[], char\* ptr)

Extract String

Convert String to a String\_lit

Set the token type to String

**Function Declaration**: static void get\_special(char buffer[], char\* str)

Extract Special

Set the token type to Special

**Function Declaration**: static void downshift\_word(char\* str)

Take pointer to a string

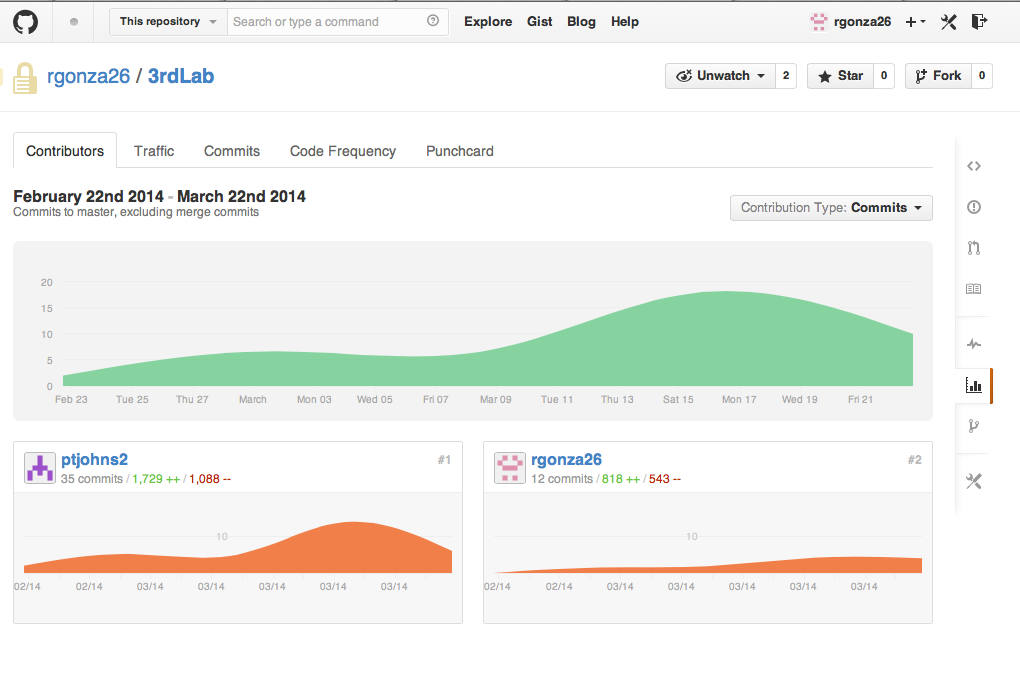
Make all characters lowercase

**Function Declaration**: static BOOLEAN is\_reserved\_word(char\* str)

Examine the reserved work table and check for a reserved word

If length > [2,9] return false

Set token type as reserved



|  |  |  |
| --- | --- | --- |
| Name | Login | Score (0= No contribution 2= good contribution) |
| Peter Johnson | ptjohns2 | 2 |
| Roberto Gonzalez | rgonza26 |  |
|  |  |  |
| URL to Repository: |  | https://github.com/rgonza26/3rdLab |

Test Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Reason | Input | Result (Expected) | Result (Actual) | P/F |
| 1 | Testing number Token | 1.6; | <NUMBER> 1.6 | >> <NUMBER> 1.6 | PASS |
| 2 | Testing string Token | END 'string'; | <STRING> string | >> <STRING> string | PASS |
| 3 | Testing reserved\_word Token | PROGRAM; | PROGRAM program | >> PROGRAM program | PASS |
| 4 | Testing skip\_blanks | " "; | ; ; | >> ; ; | PASS |
| 5 | Testing skip\_comment | this is a /\*comment\*/; | <IDENTIFIER> this  <IDENTIFIER> is  <IDENTIFIER> a  / /  \* \*  <IDENTIFIER> comment  \* \*  / /  ; ; | >> <IDENTIFIER> this  >> <IDENTIFIER> is  >> <IDENTIFIER> a  >> / /  >> \* \*  >> <IDENTIFIER> comment  >> \* \*  >> / /  >> ; ; | PASS |
| 6 | Testing Spaces | this has spaces;  so does this; | <IDENTIFIER> this  <IDENTIFIER> has  <IDENTIFIER> spaces    <IDENTIFIER> so  <IDENTIFIER> does  <IDENTIFIER> this  ; ; | >> <IDENTIFIER> this  >> <IDENTIFIER> has  >> <IDENTIFIER> spaces  >> ; ;    >> <IDENTIFIER> so  >> <IDENTIFIER> does  >> <IDENTIFIER> this  >> ; ; | PASS |
| 7 | Test EOF | END. | END end  . . | >> END end  >> . . | PASS |