

# Manual de Analizador Lexico

Manuel Suárez-Veronica Pozo-Alvaro Ortiz

3 de septiembre de 2013

# Índice general

1. Introducción	2
2. El Analizador Lexico	3
3. Historia	4
4. Aplicación	5
5. Funcionalidad	6
6. Colaboraciones	8
7. Conclusiones	10

# Capítulo 1

## Introducción

El libro a continuación es creado como parte de la documentacion del proyecto de Lenguajes de Programacion, un analizador lexico para C. En el cual se utilizara el lenguaje de programacion Haskell (lenguaje funcional) para programarlo.

## Capítulo 2

# El Analizador Lexico

Un analizador léxico o analizador lexicográfico (en inglés scanner) es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (en inglés parser).

Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o rutina del analizador sintáctico. Recibida la orden “Dame el siguiente componente léxico” del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico. Estos componentes léxicos representan:

- Palabras reservadas if, while, do, . . .
- identificadores: asociados a variables, nombres de funciones, tipos definidos por el usuario, etiquetas,...
- operadores: = \* + - / == >< ! = . . .
- símbolos especiales: ; ( ) [ ] ...
- constantes numéricas: literales que representan valores enteros, en coma flotante, etc, 982, 0xF678, -83.2E+2,...
- constantes de caracteres: literales que representan cadenas concretas de caracteres, ...

# Capítulo 3

## Historia

En 1958, Strong y otros proponen una solución al problema de que un compilador fuera portable, y esta era dividir al compilador en dos fases “front end” (analiza el programa fuente) y “back end” (genera código objeto para la máquina objeto).

En 1959, Rabin y Scott proponen el empleo de AFD y AFN para el reconocimiento lexicográfico de los lenguajes.

Aparece BNF (Backus-1960, Naur-1963, Knuth-1964) como una guía para el desarrollo del análisis sintáctico.

1959, Sheridan describe un método de parsing de FORTRAN para introducir paréntesis en una expresión.

En los 60's se desarrollan diversos métodos de parsers ascendentes y descendentes

Floyd más adelante introduce la técnica de precedencia de operadores y uso de funciones de precedencia

1961, se usa por primera vez un parsing descendente recursivo

En los 60's se estudia el paso de parámetros por nombre, valor y referencia y se incluyen los procedimientos recursivos para Algol 60

Se desarrolla la localización dinámica de datos

1968, se estudia y definen las GLC, los parsers predictivos y la eliminación de recursividad izquierda

1975, aparece LEX generador automático de analizadores léxicos a partir de expresiones regulares bajo UNIX

A mitad de los 70's Johnson crea YACC para UNIX (generador de analizadores sintácticos)

Ahora un compilador se divide en varias fases

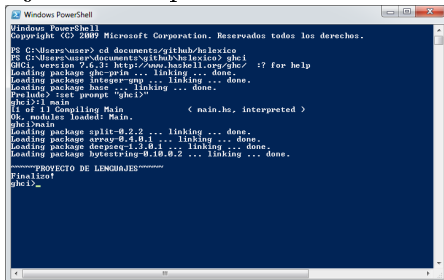
# Capítulo 4

## Aplicación

Nuestra aplicación lee un archivo de un código guardado con el nombre `cod_c`. El archivo debe tener el código separado todas las palabras reservadas, identificadores, operadores y símbolos especiales separados con un espacio. Cuando leemos el archivo, guardamos una lista con todas las palabras (separadas por espacios) y verificamos si se encuentran en el listado de palabras reservadas. Si se encuentra en el listado de palabras reservadas, las guarda en un archivo de texto de esta manera:

```
int, Palabra reservada"
```

Luego se verifica si se trata de operadores o símbolos especiales y se guarda de manera similar. Si no es ninguno de los anteriores se asume que es un identificador. Ejecutando por línea de comando, nuestra aplicación se ve de esta manera:



```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

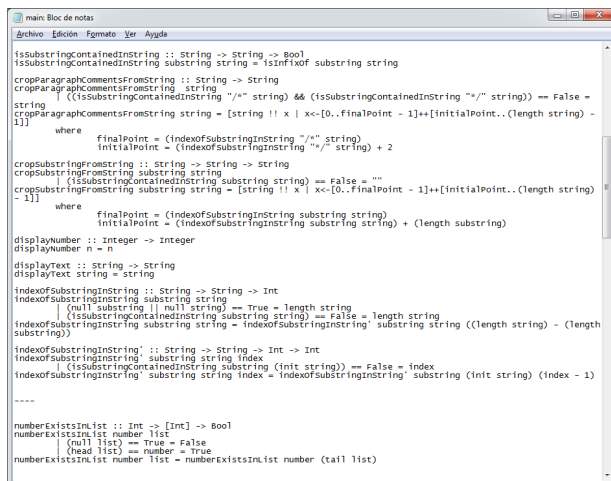
PS C:\Users\user> cd documents/github/helixico
PS C:\Users\user\documents\github\helixico> ghe1
Ghe1 version 7.0.0 http://www.helixico.org/ ? for help
Loading package ghe-print ... linking ... done.
Loading package ghe-arg ... linking ... done.
Loading package ghe ... linking ... done.
Preload>:rat prompt "ghe1"
[1 of 1] Compiling Main. (main.hs, interpreted)
Ok, modules loaded: Main.
ghe1>main
Loading package split-0.2.2 ... linking ... done.
Loading package array-0.4.0.1 ... linking ... done.
Loading package deepseq-1.3.0.1 ... linking ... done.
Loading package bytestring-0.10.0.2 ... linking ... done.
=====PROYECTO DE LENGUAJE=====
Finaliza?
ghe1>
```

# Capítulo 5

## Funcionalidad

- Eliminar comentarios

Antes de separar el archivo en palabras, se eliminan todos los comentarios que esten entre `/* */`



```
main: Bloc de notes
Archivo Edición Formato Ver Ayuda

isSubStringContainedInString :: String -> String -> Bool
isSubStringContainedInString substring string = isPrefixOf substring string

cropParagraphCommentsFromString :: String -> String
cropParagraphCommentsFromString string = if (isSubStringContainedInString "*/" string) == False then
  string
  else
    cropParagraphCommentsFromString string = [string !! x | x <= [0..FinalPoint - 1] ++ [InitialPoint..(length string) - 1]]
    where
      finalPoint = (indexOfSubStringInString "*/" string)
      initialPoint = (indexOfSubStringInString "*/" string) + 2

cropSubStringFromString :: String -> String -> String
cropSubStringFromString substring string = if (isSubStringContainedInString substring string) == False then ""
  else
    cropSubStringFromString substring string = [string !! x | x <= [0..FinalPoint - 1] ++ [InitialPoint..(length string) - 1]]
    where
      finalPoint = (indexOfSubStringInString substring string)
      initialPoint = (indexOfSubStringInString substring string) + (length substring)

displayNumber :: Integer -> Integer
displayNumber n = n

displayText :: String -> String
displayText string = string

indexOfSubStringInString :: String -> String -> Int
indexOfSubStringInString substring string = if (null substring || null string) == True then length string
  else
    if (isSubStringContainedInString substring string) == False then length string
    else
      indexOfSubStringInString' substring string = indexOfSubStringInString' substring string ((length string) - (length substring))

indexOfSubStringInString' :: String -> String -> Int -> Int
indexOfSubStringInString' substring string index = if (isSubStringContainedInString substring (init string)) == False then index
  else
    indexOfSubStringInString' substring string index = indexOfSubStringInString' substring (init string) (index - 1)

----

numberExistsInList :: Int -> [Int] -> Bool
numberExistsInList number list = if (null list) == True then False
  else
    if (head list) == number then True
    else
      numberExistsInList number list = numberExistsInList number (tail list)
```

- Análisis Léxico. Se separa en una lista todas las palabras. Se verifica palabra por palabra de que tipo se trata. Durante la verificación se guarda el archivo de texto tok.txt con los tokens.

```

main: Bloc de notas
Archivo Edición Formato Ver Ayuda

--Main
main = do
  putStr "~~~~~PROYECTO DE LENGUAJES~~~~~\n"
  putStr
  handle <- openFile "cod.c.c" ReadMode
  reser <- openFile "tok.c.txt" ReadMode
  contents <- hGetContents handle
  reservads <- hGetContents reser
  let conten = cropCommentsFromstring contents
  let reservads = splitOn " " reservad
  let espacio = words conten
  escribir espacio reservadas
  hClose reser
  hClose handle

--Escribir escribe en el archivo una lista de tokens
escribir (x:xs) [] = error "asdads"
escribir [] <- putStr "Finalizo!\n"
escribir (x:xs) (c)
  (x == "(") ==> True ==> do(appendFile "tok.txt" (x++", palabra reservada\n"); escribir (xs) c)
  (x == ")") ==> True ==> do(appendFile "tok.txt" (x++", parentesis Abierto\n"); escribir (xs) c)
  (x == "{") ==> True ==> do(appendFile "tok.txt" (x++", parentesis cerrado\n"); escribir (xs) c)
  (x == "[") ==> True ==> do(appendFile "tok.txt" (x++", llave Abierta\n"); escribir (xs) c)
  (x == "]") ==> True ==> do(appendFile "tok.txt" (x++", llave Cerrada\n"); escribir (xs) c)
  (x == ".") ==> True ==> do(appendFile "tok.txt" (x++", Punto y coma\n"); escribir (xs) c)
  (x == ",") ==> True ==> do(appendFile "tok.txt" (x++", Coma\n"); escribir (xs) c)
  (x == ";") ==> True ==> do(appendFile "tok.txt" (x++", Mayor\n"); escribir (xs) c)
  (x == "<") ==> True ==> do(appendFile "tok.txt" (x++", Menor\n"); escribir (xs) c)
  (x == ">") ==> True ==> do(appendFile "tok.txt" (x++", Mayor igual\n"); escribir (xs) c)
  (x == "<=") ==> True ==> do(appendFile "tok.txt" (x++", Menor igual\n"); escribir (xs) c)
  (x == "=") ==> True ==> do(appendFile "tok.txt" (x++", Asignacion\n"); escribir (xs) c)
  (x == "+") ==> True ==> do(appendFile "tok.txt" (x++", Equivalencia\n"); escribir (xs) c)
  (x == "-") ==> True ==> do(appendFile "tok.txt" (x++", Diferenciacion\n"); escribir (xs) c)
  (x == "+=") ==> True ==> do(appendFile "tok.txt" (x++", Aumento unitario\n"); escribir (xs) c)
  otherwise ==> do(appendFile "tok.txt" (x++", Identificador\n"); escribir (xs) c)

isSubStringContainedInString :: String -> String -> Bool
isSubStringContainedInString substring string = isPrefixOf substring string

cropParagraphCommentsFromstring :: String -> String
cropParagraphCommentsFromstring string
  (isSubStringContainedInString "/" string) && (isSubStringContainedInString "*/" string) == False =
  string
  cropParagraphCommentsFromstring string = [string !! x | x <- [0..finalPoint - 1] ++ [initialPoint..(length string) - 1]]
    where
      finalPoint = (indexOfSubStringInString "/" string)
      initialPoint = (indexOfSubStringInString "*/" string) + 2

cropSubStringFromstring :: String -> String -> String
cropSubStringFromstring substring string

```

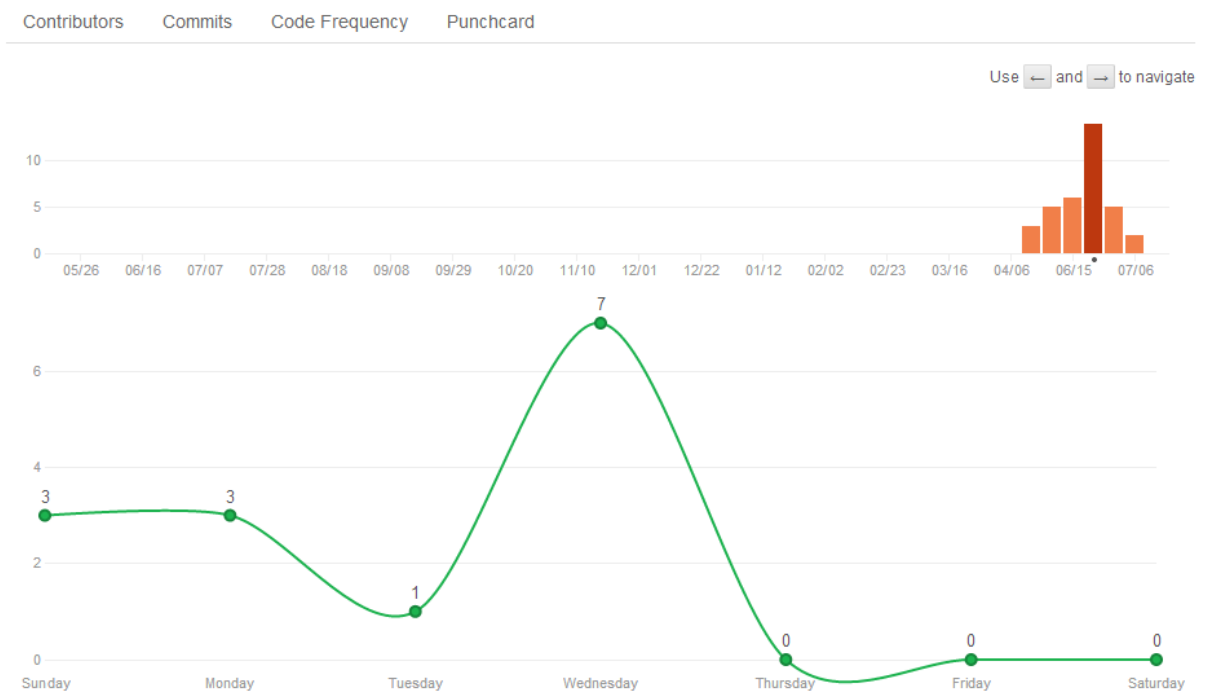


# Capítulo 6

## Colaboraciones

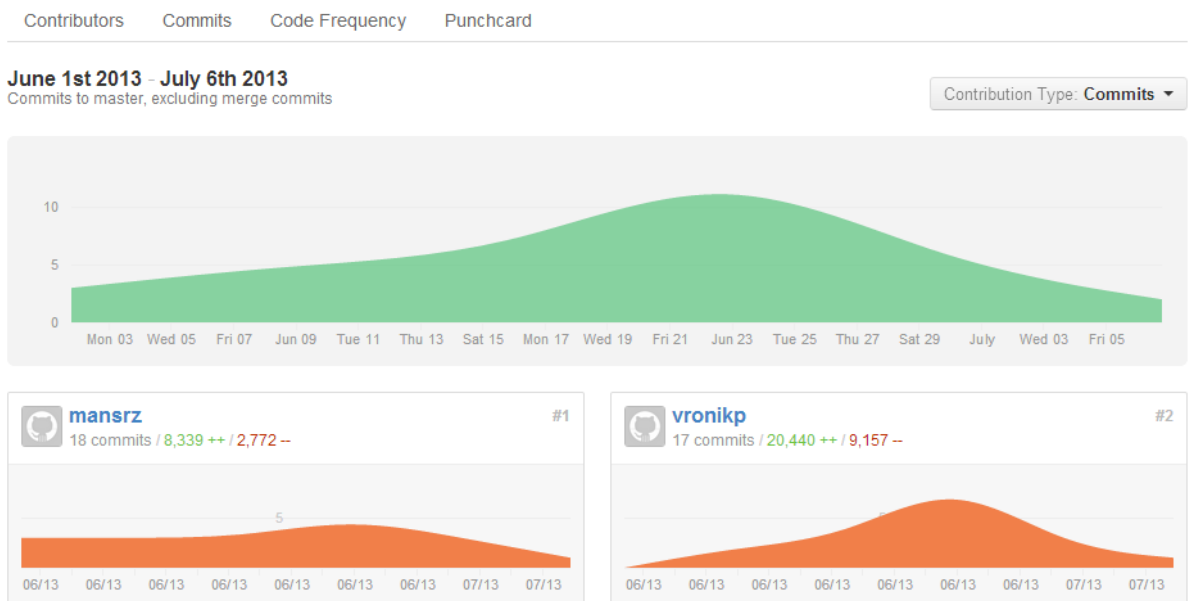
- Commits por integrante.

- Alvaro Ortiz: 2
- Veronica Pozo: 2
- Manuel Suárez: 2



- ¿En qué fechas hay picos?

- 4 de Septiembre



## Capítulo 7

### Conclusiones

El proyecto se realizo con exito logrando implementar la mayoria de caracteristicas en nuestra aplicacion que se pedian. Nuestra aplicación permite archivos con comentarios, los cuales son eliminados antes de hacer el analisis. El analisis está limitado por que todas las palabras estén separadas por espacios a excepción de los punteros y numerales.