INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO
LABORATORIO de SISTEMAS OPERATIVOS

Intérprete de Comandos

Proyecto final: Intérprete de comandos

Grupo
Aficionados

Integrantes
Leslie Pamela Brenes Valencia – 143851
Stephanie Lizeth Malvaes Diaz – 135515
Pablo A. Ruz Salmones – 151218

Fecha de entrega
04/06/2020

Documentación proyecto final

Introducción

El propósito del presente proyecto fue crear un intérprete de comandos que fuera capaz de leer correctamente y ejecutar ciertas instrucciones y, en su caso, guardar el resultado de las mismas en archivos .txt.

Para ello, fue necesario analizar desde un inicio los pasos adecuados para la creación de un “shell” (*ver bibliografía 1*), su implementación y la relación existente entre los diversos archivos que se generan con el intérprete. Aunque la implementación evidentemente difiere en cierta medida de la teoría expuesta en los documentos de la bibliografía, la estructura general es la misma, la cual será descrita con mayor detalle en las siguientes secciones.

Marco teórico

A continuación se describen brevemente las características de cada uno de los archivos y la conexión que existe entre los mismos:

- shell.l → describe los tokens usando expresiones regulares, de modo que se procesa cuando el programa *lex* genera el analizador léxico. Posteriormente, el Parser procesa los tokens de acuerdo con la gramática y la tabla de comandos.
- interprete.y → es el archivo “principal”, mismo en el que se describen a detalle los comandos y la lógica seguida por el intérprete en cada uno de los casos. De igual forma, se inicializan las matrices de comandos y argumentos, con una lógica como la que se muestra en la *Fig. 1.*

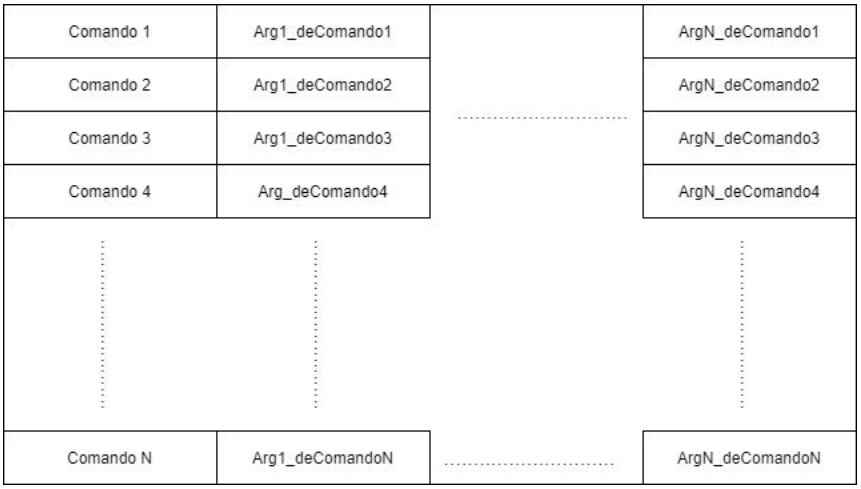

Fig.1

Muestra de la matriz de comandos y argumentos. En un inicio, todos los valores están en NULL, y conforme se agregan comandos y argumentos, la matriz se va llenando con los mismos.

- `lex.yy.c` → archivo en C que compila el intérprete. Se genera a través del shell.l, e implementa el scanner que el parser utiliza para traducir los caracteres a tokens.
- `y.tab.c` → archivo generado por Yacc que ayuda a compilar el intérprete
- `y.tab.h` → archivo header generado por Yacc
- `a.out` → archivo del intérprete compilado para ejecutarse

Como puede deducirse de los bullets anteriores, los únicos archivos que se tienen al inicio son el *shell.l* y el *interprete.y*. A partir de ellos se generan todos los demás, ya sea con diversos comandos en la terminal o bien a partir de la ejecución del .l y el .y.

Para entender cómo funciona la compilación se detallará el funcionamiento de Yacc y Lex y cómo los archivos *shell.l* y el *interprete.y*. son utilizados por dichos programas.

YACC (Yet Another Compiler Compiler) es un programa generador de compiladores; genera el código para el analizador sintáctico partiendo de un compilador que verifica que el código fuente cumple con las especificaciones sintácticas del lenguaje. Su extensión es .y, que crea una rutina yyparse en un archivo denominado y_tab.c

Lex genera un código C para un analizador léxico. La extensión de un archivo para Lex es `.l`; crea una rutina yylex en un archivo `lex.yy.c`. Las rutinas deben ser compiladas, y las librerías Lex y Yacc deben cargarse durante la compilación.

Tanto el analizador léxico como el sintáctico pueden ser escritos en cualquier lenguaje de programación, son más flexibles y más fáciles de usar que los mismos lenguajes de propósito general.

La siguiente figura muestra el desarrollo de la compilación usando los elementos mencionados:

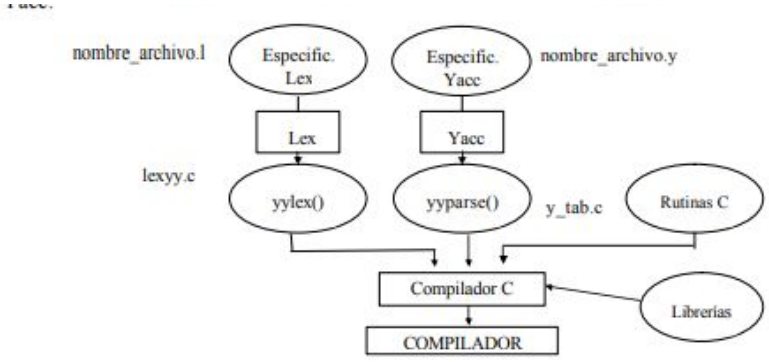

Fig. 2

Yacc y Lex permiten a un lenguaje automatizar tareas repetitivas, que de otro modo tendrían que hacerse manualmente. En el archivo de entrada Yacc se pondrá un encabezado donde se definen los tipos de datos y se declaran los tokens y se especifican las reglas . La estructura del archivo Lex es una sección de declaraciones, una sección de

reglas y una sección de código en C, lex toma un fichero .l y produce un fichero en C llamado lex.yy.c que contiene el analizador léxico. Yacc generará un archivo y.tab.c que contiene el analizador sintáctico y una función que realizada lo especificado en el .y.

Funcionalidades

El intérprete es capaz de analizar y ejecutar comandos que se escriban directamente en la consola, y de liberar los resultados de dichos comandos a un archivo de salida. No obstante, NO cuenta con la funcionalidad de recibir comandos por medio de archivos de entrada.

Ejemplo: `ls -1 | grep y > salida.txt` guarda en un archivo la salida del comando `ls -l`

Complicaciones

Una de las principales dificultades encontradas fue la ejecución de los archivos. Aunque se siguió la metodología que a continuación se detalla, numerosos errores se presentaron, algunos de los cuales versaban sobre la no existencia de algunas de las funciones declaradas internamente. Los pasos básicos para la ejecución fueron los que se muestran en la *Fig. 3.*

For Compiling YACC Program:

1. *Write lex program in a file file.l and yacc in a file file.y*
2. *Open Terminal and Navigate to the Directory where you have saved the files.*
3. *type lex file.l*
4. *type yacc file.y*
5. *type cc lex.yy.c y.tab.h -ll*
6. *type ./a.out*

Fig. 3
Serie de pasos generales para poder ejecutar el archivo "interprete.y"

Aunque los pasos generales son correctos, la ejecución adecuada no es con el y.tab.h, como se indica en el paso 5, sino con y.tab.c. Una vez resuelto ese problema, se procedió a corregir la serie de errores sintácticos que se detectaron durante la elaboración del archivo.

En el código, la principal complicación que se presentó fue el lograr entender la estructura general que el mismo debía de tener. Esto es, sabíamos por lo visto durante la teoría la manera en la que las matrices debían estar estructuradas, pero la conexión que se muestra en la *Fig. 2* de las secciones anteriores no fue sencilla de entender, y menos aún su implementación. Aunque evidentemente encontramos errores de sintaxis de ";" y similares, que se fueron corrigiendo durante el avance, por mucho la conexión y la relación que tenía cada uno de los archivos fue lo más complicado.

Conclusiones

- Generar el intérprete permitió entender con mayor profundidad los procesos por los cuales la computadora ejecuta los comandos que se escriben en la terminal.
- Se mejoró el entendimiento del uso de los pipes así como su propósito.
- También se aprendió a utilizar Lex y Yacc, bibliotecas que fueron de ayuda al momento de generar los archivos .l y .y.

Referencias

- Gustavo A. Junipero Rodriguez-Rivera and Justin Ennen, "*Book Online: Introduction to Systems Programming: a Hands-on Approach*", online distribution, CH.5
- *Tutorial: Write text parsers with yacc and lex*, IBM, consultado el 2 de junio de 2020, https://developer.ibm.com/technologies/systems/tutorials/au-lexyacc/#listing1
- *Introduction to YACC*, Geeks for Geeks, consultado el 2 de junio de 2020, https://www.geeksforgeeks.org/introduction-to-yacc/
- *Using LEX*, GitHub, consultado el 2 de junio de 2020, https://silcnitc.github.io/lex.html
- *yylval undefined with flex and bison*, stackoverflow, consultado el 2 de junio de 2020, https://stackoverflow.com/questions/6298260/yylval-undefined-with-flex-and-bison
- dup() and dup2() Linux system call, Geeks for Geeks, consultado el 2 de junio de 2020, https://www.geeksforgeeks.org/dup-dup2-linux-system-call/
- yylval undefined with lex and yacc, stackoverflow, consultado el 2 de junio de 2020, https://stackoverflow.com/questions/32 427264/yylval-undefined-with-lex-and-yacc
- yyparse() y yylex(), IBM, consultado el 2 de junio de 2020, https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.bpxa600/bpxa688.htm
- y.tab.h not found, github, consultado el 2 de junio de 2020, https://github.com/gregrahn/tpcds-kit/issues/27
- Diseño de compiladores I, unicen, consultado el 2 de junio de 2020,
- http://www.exa.unicen.edu.ar/catedras/compila1/index_archivos/Herramientas/Yacc