UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SEMESTRE 2023-1

COMPILADORES-0434

GRUPO: 03

M.I. LAURA SANDOVAL MONTAÑO

PROYECTO COMPILADORES PT1

“Analizador léxico de un compilador”

Alumnos:

- **Castelan Ramos Carlos(317042711)**
- **Rivas Solís Carlos Eduardo(317224445)**

FECHA DE ENTREGA: 02 de Noviembre del 2022

Índice

Analizador léxico de un compilador.

Objetivo.

Elaborar un analizador léxico en lex/flex que reconozca los componentes léxicos pertenecientes a las clases abajo descritas.

Descripción del problema.

Las clases de los componentes léxicos válidos para el analizador léxico son:

Clase	Descripción
0	Palabras reservadas (ver tabla).
1	Identificadores. Iniciar con $ y le sigue al menos una letra minúscula o mayúscula. Ejemplos: $ejemplo, $Varible, $OtraVariable, $XYZ
2	Constantes numéricas enteras. En base 10 (secuencia de dígitos del 0-9 sin 0's a la izquierda, excepto para el número 0), en base 8 (inicien con O u o y le sigan dígitos del 0 al 7).
3	Constantes numéricas reales. Siempre deben llevar parte decimal y es opcional la parte entera. Ejemplos: 73.0, .0, 10.2 No aceptados: . , 12 , 4.
4	Constantes cadenas. Encerrado entre comillas (") cualquier secuencia de más de un carácter que no contenga " ni '. Para cadenas de un solo carácter, encerrarlo entre apóstrofos ('). La cadena de unas comillas debe ser encerrada entre apóstrofos: '"' . La cadena de un apóstrofo debe ser encerrada por comillas: "'" . No se aceptan cadenas vacías. Ejemplos NO válidos: "ejemplo no "valido" , "" , "" , "hola 'mundo"
5	Símbolos especiales [ ] ( ) { } , : ;
6	Operadores aritméticos + - * / % \ ^
7	Operadores relacionales (ver tabla).
8	Operador de asignación =

Es entonces que planteamos las expresiones regulares para su posterior utilización en el programa.

Clase	**Nombre**	**Expresión regular**
0	Palabras reservadas	alternative|big|evaluate|instead|large|loop|make|number|other|real|repeat|select|small|step|stop|symbol|throw

1	Identificadores	[$][A-Za-z]+
2	Constante enteras ● Base 10 ● Base 8	 0|[1-9]+[0-9]* [oO][0-7]+
3	Constantes reales	[0-9]*\.[0-9]+
4	Constantes ● Cadenas ● Carácteres	Revisar “planeación” por favor. [Pp 7] [']['][A-Za-z0-9][A-Za-z0-9]+['] [']['][A-Za-z0-9"][']
5	Símbolos especiales	[\[\](){},:;]
6	Op aritméticos	[\^+\-*/%\\]
7	Op relacionales	<|>|<=|>=|==|!=
8	Op asignación	[=]

Otras expresiones regulares creadas

9	Espacio	[ ]
10	Tabulador	[\n]
11	Salto de línea	[\t]
12	No reconocidos	‘.’

Propuesta de solución y fases del desarrollo del sistema.

Propuesta de solución.

Implementación de un programa escrito en lex y c que identifique y clasifique mediante expresiones regulares los datos que se reciban mediante un archivo externo de entrada con extensión .txt. Para que así después de su clasificación se inserten en tablas dinámicas según se piden en los requerimientos.
La explicación detallada de esta propuesta se encuentra en el *diseño e implementación*.

Planificación.

- Viernes 28 de Octubre del 2022.
    - Reunión presencial para realización de esta planificación.
    - Instalación de paquetes lex y yacc en computadoras personales, así como sus compiladores de Bison.

    - Lectura y análisis de los requerimientos del proyecto así como su rúbrica.
- Sábado 29 de Octubre del 2022.
    - Creación del repositorio para el control de versiones del proyecto.
        - A continuación se comparte con la profesora Laura Sandoval un usuario con permisos para acceder al repositorio.
          https://github.com/CarlosCERS/ProyectoCompiladores
          Correo: carlosscastelan@outlook.com
          Contraseña: cucho010601
    - Creación de los ejemplos vistos en clase para comprender el funcionamiento del paquete lex, así como el uso de comandos para su compilación.
    - Primeras pruebas del correcto funcionamiento del paquete.
- Domingo 30 de Octubre del 2022.
    - Creación del archivo oficial para la creación del sistema.
    - Repartición de trabajo para la creación de expresiones regulares.
    - Implementación de expresiones regulares en código.
    - Primeras pruebas del correcto funcionamiento de expresiones regulares.
- Lunes 31 de Octubre del 2022.
    - Creación de funciones de lectura y escritura de archivos.
    - Análisis de creación de tablas de token generales, literales y símbolos.
    - Creación de tablas estáticas para palabras reservadas, símbolos especiales, operadores aritméticos y operadores relacionales.
    - Creación de función para generación de tabla dinámica para Constantes Reales.
- Martes 01 de Noviembre del 2022.
    - Creación de función para generación de tabla dinámica para Constantes Cadenas.
    - Creación de función para generación de tabla dinámica para Identificadores.
    - Creación formal de la documentación del proyecto.
    - Entrega del proyecto.

Diseño e implementación.

FASE 1:

- Viernes 28 de Octubre del 2022.
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Instalación de paquetes y compiladores de Lex y yacc: Existió una reunión presencial, consultamos fuentes y videos para la instalación de los paquetes ***Referencia [1]***, donde hicimos uso de los siguientes comandos.

1. *sudo apt-get update*
2. *sudo apt-get install flex*
3. *sudo apt-get install bison*
4. *sudo apt-get install byacc*
5. *sudo apt-get install bison++*
6. *sudo apt-get install byacc -j*

Responsable(s): Carlos Castelan, Carlos Rivas.
Además se hizo lectura de los requerimientos esenciales para la creación del proyecto, los cuales fueron clasificados de la siguiente forma:

1. *Creación de archivo de lectura de datos.*
2. *Creación de expresiones regulares a partir de las clases de los componentes léxico iniciales en lex.*
3. *Creación de tabla dinámica para identificadores.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
    c. *Tipo (Por default -1).*
4. *Creación de tabla estática para palabras reservadas.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
5. *Creación de tabla estática para símbolos especiales.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
6. *Creación de tabla estática para operadores aritméticos.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
7. *Creación de tabla estática para operadores relacionales.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
8. *Creación de tabla dinámica para constantes enteras decimales y octales.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
9. *Creación de tabla dinámica para constantes reales.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
10. *Creación de tabla dinámica para constantes cadenas.*
    a. *Índice (Por default en una estructura de datos).*
    b. *Dato.*
11. *Generación de archivo de salida que almacena la tabla de tokens generales de todas las expresiones regulares.*
12. *Generación de archivo de salida que almacena la tabla de identificadores.*

13. *Generación de archivo de salida que almacena la tabla de palabras reservadas.*
14. *Generación de archivo de salida que almacena la tabla de símbolos especiales.*
15. *Generación de archivo de salida que almacena la tabla de operadores aritméticos.*
16. *Generación de archivo de salida que almacena la tabla de operadores relacionales.*
17. *Generación de archivo de salida que almacena la tabla de constantes enteras decimales y octales.*
18. *Generación de archivo de salida que almacena la tabla de constantes reales.*
19. *Generación de archivo de salida que almacena la tabla de constantes cadenas.*
20. *Generación de archivo de salida que almacena la tabla de carácteres no reconocidos.*

FASE 2:

- Sábado 29 de Octubre del 2022.
  Responsable(s): Carlos Rivas.
  Creación del repositorio del proyecto en el sistema de control de versiones Github.
  Se pide al lector acceder al repositorio para comprender mejor las descripciones de este documento.
  https://github.com/CarlosCERS/ProyectoCompiladores
  Correo: carlosscastelan@outlook.com
  Contraseña: cucho010601
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Instalación de comandos nativos de git en linux, así como su estudio y repaso del mismo, para ello se hizo consulta de diferentes fuentes. ***Referencia [2, 3].***
  Responsable(s): Carlos Castelan.
  Creación de ejemplos vistos en clase para comprensión del funcionamiento.
  - Ejemplo de identificadores y constantes en distintas bases.

  Responsable(s): Carlos Castelan, Carlos Rivas.
  Primeras pruebas del correcto funcionamiento del paquete.

FASE 3:

- Domingo 30 de Octubre del 2022.
  Responsable(s): Carlos Castelan, Carlos Rivas.
  1. Creación del archivo oficial del analizador léxico.
  2. Repartición de la creación de las expresiones regulares.
     a. **Responsable(s):** Carlos Castelan, Carlos Rivas.
     Clase 0-Palabras reservadas:

ER:
alternative|big|evaluate|instead|large|loop|make|number|other|real|repeat|select|small|step|stop|symbol|throw

b. **Responsable(s):** Carlos Castelan.
Clase 1-Identificadores:
ER: [$][A-Za-z]+

c. **Responsable(s):** Carlos Rivas.
Clase 2-Constantes enteras:
Decimales - ER: 0|[1-9]+[0-9]*
Octales - ER: [oO][0-7]+

d. **Responsable(s):** Carlos Castelan, Carlos Rivas.
Clase 3-Constantes reales:
ER: [0-9]*\.[0-9]+

e. **Responsable(s):** Carlos Castelan, Carlos Rivas.
Clase 4-Constantes cadenas y carácter:
Cadenas - ER: ['][A-Za-z0-9][A-Za-z0-9]+[']
Carácter - ER: ['][A-Za-z0-9"][']

Es muy importante aclarar la situación de estas ER

La forma en que se definieron las expresiones regulares para cadenas y carácteres fue que toda cadena y carácter deben iniciar con comillas simples ‘ y cerrar con comilla simple ‘. En primera instancia es necesario aclarar que el analizador léxico diferencia a las cadenas de los carácteres porque las cadenas en su expresión regular al menos debe tener dos carácteres o dos elementos, en cambio la expresión regular de los carácteres está definida para albergar sólo un carácter o dato. Es así que se puede hacer la clasificación de clases.

La razón por la que nuestras cadenas y carácteres funcionan forzosamente con comillas simples es porque más adelante tendremos una función que crea una lista ligada en C. Esta lista ligada en C fue declarada como ***addList(char atomo[])***. Donde el argumento que recibe sea cual sea lo convierte en una cadena, es decir si el archivo de carga inicial contiene un 5, entonces pasa a ser argumento de la función como “5”.

Nuestro problema surgió cuando el archivo contenía una carga inicial con una cadena, es decir llega por ejemplo un “Hola a todos”, este pasaba como argumento a la función y se convertía en una cadena, es decir obtenemos “ “Hola a todos” “. Esto se detecta como un error ya que las comillas dobles iniciales eran detectadas como una cadena vacía, es entonces que no logramos almacenar las cadenas en su propia tabla. Por lo que optamos por aplicar la solución del primer párrafo donde ese ‘Hola a todos’ pasa a ser un “ ‘Hola a todos’ “ como argumento de la función.

Finalmente, por simplicidad del analizador léxico y bajo los ejemplos mostrados en las instrucciones decidimos que nuestras cadenas y carácteres solo identificaran letras, mayúsculas, minúsculas, números y espacios. Sin embargo, la implementación de símbolos especiales no se nos complicaba. Recalcando que se hizo así con intención de simplicidad de la ER y el analizador léxico.

f. **Responsable(s):** Carlos Castelan.
   Clase 5-Símbolos especiales.
   ER: [\[\](){},:;]
g. **Responsable(s):** Carlos Rivas.
   Clase 6-Operadores aritméticos.
   ER: [\^+\-*/%\\]
h. **Responsable(s):** Carlos Castelan, Carlos Rivas.
   Clase 7-Operadores relacionales.
   ER: <|>|<=|>=|==|!=
i. **Responsable(s):** Carlos Castelan.
   Clase 8-Operador asignación.
   ER: [=]
j. **Responsable(s):** Carlos Rivas.
   Clase 9-Espacio.
   ER: [ ]
k. **Responsable(s):** Carlos Rivas, Carlos Rivas.
   Clase 10-Salto de línea.
   ER: [\n]
l. **Responsable(s):** Carlos Rivas.
   Clase 11-Tabulador.
   ER: [\t]
m. **Responsable(s):** Carlos Castelan, Carlos Rivas.
   Clase 12-No reconocidos.
   ER: ‘.’

Se realizaron pruebas y se hizo commit en el repositorio.

FASE 4:

- Lunes 31 de Octubre y martes 01 de Noviembre del 2022..
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Creación de funciones de lectura y escritura de archivos.

  Creación de archivo inicial de carga y creación de archivos para albergar las tablas ya descritas en la FASE 1.

  - Se hizo uso de las funciones fopen para la apertura de archivos, fprintf para la escritura de archivos y fclose para la cerradura de archivos.

  Análisis de creación de tablas de token generales, literales y símbolos.

  - Para tablas estáticas como las de las palabras reservadas, símbolos especiales, operadores aritméticos y operadores relacionales, se crearon estructuras estáticas **arreglos** declaradas como char. Se encuentran ya documentadas al inicio del código fuente.

  - Para tablas dinámicas:
    Primero: Con la tabla de identificadores, que es la única que contiene (índice, dato y tipo), se creó su propia estructura de datos lista conocida como **nodeIdentificador** y **listIdentidicador** respectivamente. Para su posterior uso en una función que almacena en esta lista todo lo que reciba como parámetro a partir del ***yytext***, la función es conocida como ***addListIdentificador***. (La creación de estas funciones se encuentra en el repositorio en un archivo de prueba conocido como lista.c)

    Segundo: Con las tablas de constantes cadenas, reales y enteras, se creó una estructura similar a la de los identificadores, solo que esta lista sólo contiene (índice y dato), siendo las estructura conocida como ***node*** y ***list*** respectivamente. Ésta estructura es llenada desde la función ***addList*** que recibe como parámetro ***yytext***. (La creación de estas funciones se encuentra en el repositorio en un archivo de prueba conocido como lista.c)

  Finalmente teniendo todas las tablas estáticas y dinámicas listas decidimos escribir los archivos generados. Para tablas estáticas, dentro de lex, se crearon estructuras de control que imprimen las tablas estáticas a partir de los arreglos generados. Para las tablas dinámicas se generaron dos funciones de escritura de archivos, una para identificadores conocida como ***printTablaIdentificadores*** y otra

para cadenas y reales conocida como ***printTabla***. Además de otras funciones similares para tablas auxiliares.
Además, dentro de las funciones de lex, dentro de las operaciones del token de ***No reconocidos***, {noReconocido} se colocó directamente un fprintf en su propio archivo.

FASE 5:

- Martes 01 de Noviembre del 2022.
  Durante esta fase, se mejoraron funciones, impresión y se quitaron detalles. Además se acabó de documentar completamente el código fuente, se verificaron todos los cambios hechos en el repositorio y se actualizaron todas las ramas de trabajo.
  Es importante aclarar, para la creación exitosa de cada funcionalidad se realiza un commit en el repositorio. Es por eso que recomendamos acceder al repositorio para verificar todos estos cambios.
  Por último en ésta fase se afinaron detalles y se concluyó la escritura de este documento.

Manual de usuario del analizador léxico.

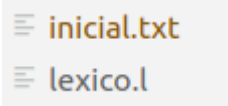
1. Tendremos el archivo ***lexico.l*** que es nuestro archivo del programa fuente.
2. Para ejecutarlo debemos tener un archivo de carga inicial, el cual debe contener diferentes tipos de datos que cumplan con las especificaciones de la funcionalidad del analizador léxico, de preferencia este archivo de entrada deberá ser de alguna extensión .txt para su mejor interpretación.
3. Ambos archivos, el del programa fuente ***lexico.l*** y el de la carga inicial, en nuestro caso ***inicial.txt*** deben estar en la misma carpeta.
4. Proceso de ejecución del programa:
   a. Ejecutar en la terminal el comando ***flex lexico.l***
      Este comando inmediatamente generará un archivo llamado ***lex.yy.c***
   b. Ejecutar en la terminal el comando ***gcc lex.yy.c -lfl***, en caso de que desee que el archivo ejecutable de salida tenga un nombre en especifico colocar el comando ***gcc lex.yy.c -lfl -o <nombre de salida>***
      Es entonces que para el primer comando obtendremos un archivo de ejecución ***a.out*** y en el caso del segundo comando obtendremos un archivo de ejecución ***<nombre>.out***
   c. Ejecutar en la terminal el comando ***./a.out inicial.txt*** o en su caso ***./<nombre.out> <nombre archivo de carga inicial.txt>***
      La ejecución de este comando nos traerá distintas acciones.
      La primera es que en la terminal nos aparecerá la ejecución del programa que va leyendo los datos del archivo inicial.

La segunda es que nos generará los siguientes archivos

- TokensGenerales.txt
- TablaPalabrasReservadas.txt
- TablaIdentificadores.txt
- TablaEnterosOctales.txt
- TablaConsReales.txt
- TablaCadenas.txt
- TablaSimbolosEspeciales.txt
- TablaOperadoresArimeticos.txt
- TablaOperadoresRelacionales.txt
- CaracteresNoReconocidos.txt

Ejemplo de uso.

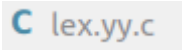
1. Archivos iniciales para su ejecución.

2. Ejecución del comando ***flex lexico.l***

   ```
   nahual@nahualT:~/inge/7moSemestre/Compiladores/ProyectoPT1/ProyectoCompiladores$ flex lexico.l
   ```

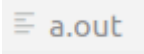
3. Se genera el archivo ***lex.yy.c***

4. Se compila el archivo ***lex.yy.c*** con el comando ***gcc lex.yy.c -lfl***

   ```
   nahual@nahualT:~/inge/7moSemestre/Compiladores/ProyectoPT1/ProyectoCompiladores$ gcc lex.yy.c -lfl
   ```

5. Se genera el archivo de ejecución ***a.out***

6. Se ejecuta el archivo ***a.out*** con el comando ***./a.out inicial.txt***

   ```
   nahual@nahualT:~/inge/7moSemestre/Compiladores/ProyectoPT1/ProyectoCompiladores$ ./a.out inicial.txt
   ```

7. Existe una respuesta en terminal y se generan los siguientes archivos.

   ```
   Respuesta en terminal:
   Probando la compilación

   ? es un dato No reconocido

    Hay un Salto de Línea
   other es una Palabra Rervada
    Hay un Salto de Línea
   10.25 es un Constante real

    Hay un Salto de Línea
   0.5 es un Constante real
   ```

```
 Hay un Salto de Línea
make es una Palabra Rervada
 Hay un Salto de Línea
'5' es un Caracter

 Hay un Salto de Línea
$identidaa es un Identificador

 Hay un Salto de Línea
10.2 es un Constante real

 Hay un Salto de Línea
11.5 es un Constante real
< es un Operador Relacional
.6 es un Constante real
= es un Operador Asignacion
15.4 es un Constante real

 Hay un Salto de Línea
$olaKAse es un Identificador

 Hay un Salto de Línea
( es un Simbolo especial
+ es un Operador Aritmético
- es un Operador Aritmético
) es un Simbolo especial
} es un Simbolo especial

 Hay un Salto de Línea
^ es un Operador Aritmético

 Hay un Salto de Línea
'Hola a todos' es un Consantes cadenas

 Hay un Salto de Línea
61 es un Constante entera

 Hay un Salto de Línea
$SoyYo es un Identificador

 Hay un Salto de Línea
o12 es un Constante octal

 Hay un Salto de Línea
$identidaa es un Identificador

 Hay un Salto de Línea
10 es un Constante entera

 Hay un Salto de Línea
O73 es un Constante octal

 Hay un Salto de Línea
```

```
'Hola a todos' es un Consantes cadenas

 Hay un Salto de Línea
'a' es un Caracter

 Hay un Salto de Línea
# es un dato No reconocido
```

Creación de archivos:

Conclusiones:

- Carlos Castelan Ramos:
  Este proyecto ha sido uno de los más complejos que he realizado a lo largo de la carrera, siendo aún la primera parte del proyecto, me he dado cuenta de la gran complejidad que lleva el análisis de la parte más pura de la computación. Tuvimos muchos retos a lo largo de esta primera entrega, partiendo de la planeación del proyecto, el análisis para el entendimiento de requerimientos del proyecto, e implementación en código, siendo la parte más complicada la creación de listas y escritura de archivos para las tablas de símbolos y literales, ya que teníamos problemas internos de memoria y procesos con el mismo sistema operativo. Sin embargo creo totalmente que esta primera parte del proyecto cumple con las expectativas escritas en los requerimientos. Finalmente, digo que se cumplieron con éxito el objetivo planteado al inicio de éste documento,la rúbrica y nuestros objetivos personales.

- Carlos Eduardo Rivas Solís:
  Cuando iniciamos este proyecto, yo no tenía ni idea de cómo iniciar con este o que rumbo ir siguiendo, al comienzo iniciamos con lo más fácil de crear las expresiones regulares pero, después nos tomó tiempo entender el cómo continuar, cómo hacer que leyera los archivos de entrada, los creara pero lo más complicado fue el crear las listas ligadas que utilizaremos para guardar las tablas, valores e índices y el cómo irlos creando y añadiendo elementos a estas listas.
  Al final del día con mucho esfuerzo logramos crear nuestro analizador léxico que nos hace orgullosos de haberlo logrado con éxito, y viendo todo lo que aprendimos (o reestudiamos) durante el proceso de su creación, podemos concluir que los objetivos de este primer proyecto se cumplieron con éxito.

Referencias.

- [1]tech M. (2020, 19 julio). *Install Lex and Yacc packages on ubuntu*. YouTube. https://www.youtube.com/watch?v=T9-WZZrqOiY
- [2]ChepeCarlos. (2021, 23 febrero). *(☛ Fácil) instalación y Configuración GIT en Linux*. YouTube. https://www.youtube.com/watch?v=5zM5cv1Gz34
- [3]*GitHub Training Kit*. (s. f.). GitHub Cheatsheets. Recuperado 29 de octubre de 2022, de https://training.github.com/
- [4] makigas: aprende a programar. (2015, 8 octubre). *Estructuras de datos – 3. Listas enlazadas en C (COMPLETO)*. YouTube. https://www.youtube.com/watch?v=vldM-3PYAmo