UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SEMESTRE 2023-1

COMPILADORES-0434

GRUPO: 03

M.I. LAURA SANDOVAL MONTAÑO

PROYECTO COMPILADORES PT2

“Análisis Léxico-Sintáctico ”

Alumnos:

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

FECHA DE ENTREGA: 06 de Diciembre del 2022

Índice

Índice 2

Análisis léxico-sintáctico 3

- **Objetivo.** 3
- **Descripción del problema.** 3
- **Propuesta de solución y fases del desarrollo del sistema.** 6
  - **Propuesta de solución.** 6
  - **Planificación.** 6
  - **Diseño e implementación.** 7
  - **Manual de usuario del analizador léxico.** 15
  - **Ejemplo de uso.** 16
  - **Conclusiones.** 17
  - **Referencias.** 18

Análisis léxico-sintáctico

Objetivo.

Construir, en un mismo programa, los analizadores Léxico y Sintáctico Descendente Recursivo que revisen programas escritos en el lenguaje definido por la gramática del Anexo A de este documento.

Descripción del problema.

Dada la construcción previa de la primera parte del compilador con el analizador léxico, logramos diferenciar y clasificar el lenguaje natural con el uso de las *clases* definidas en la primera parte de la documentación aplicando expresiones regulares expresadas en tokens con el uso del paquete *flex/lex* de *Bison* en un programa escrito en lenguaje C.
Es entonces, que hechas las debidas correcciones del analizador léxico (revisar apartado de correcciones), comenzamos con el planteamiento del problema para la implementación del analizador sintáctico.
Debemos tener un analizador sintáctico, que trabaje a partir de los tokens generados con el analizador léxico, para que así mediante el uso de funciones creadas con la aplicación del método de *Análisis Sintáctico Descendente Recursivo* (a partir de ahora lo llamaremos ASDR) identifique los átomos que aparecen en los tokens y los almacene en una cadena. Además por defecto al aplicar el método de ASDR es necesario calcular los conjuntos de selección de las gramáticas declaradas en el Anexo A donde estos deben de cumplir con las características de una Gramática LL(1). Así finalmente con los conjuntos de selección y las funciones creadas en C del ASDR tendremos la aplicación de una correcta sintaxis del orden de los tokens generados del análisis léxico, donde tendremos que mostrar los casos en los que las funciones pueden fallar.

A continuación el Anexo A.

Anexo A

De acuerdo con el ejercicio realizado por todo el grupo, la gramática que define la sintaxis de nuestro lenguaje se muestra a continuación.

Sintaxis de las diferentes estructuras del lenguaje

1) Sentencia declarativa:

Ejemplos:

```
big  $Grande=2, $num;
real $numReal;
symbol  $cad="cadena#1", $cadVacia;
```

Gramática:

D → <Tipo>K;	Q → =NC
<Tipo>→ b	Q → ,K
<Tipo>→ g	N → n
<Tipo>→ #	N → r
<Tipo>→ y	N → s
<Tipo>→ x	C → ξ
K → iQ	C → ,K
Q → ξ	

2) Sentencias de asignación:

Ejemplos:

```
$num=$Grande+15*(o12-93);
$cadVacia= "cadena#2";
$numReal=27.9;
```

Gramática:

A → i=A';
A' → s
A' → E

3) Expresión aritmética

Ejemplos:

34.7*25 (12.6+$num)/[$calcula(63)]

E → T E'	T' → %FT'
E' → + T E'	T' → ^FT'
E' → - T E'	T' → ξ
E' → ξ	F → (E)
T → F T'	F → i
T' → *F T'	F → n
T' → /FT'	F → r
T' → \FT'	F →<Llama>

4) Expresión relacional

Ejemplos:

```
2 <= $num
$cad=="cadena#2"
40.8>$numReal
```

Gramática:

R → iR'V	V → i
R → nR'V'	V → n
R → rR'V''	V → r
R → sR'V'''	V → s
R' → >	V' → n
R' → <	V' → i
R' → l	V'' → r
R' → e	V'' → i
R' → d	V''' → s
R' → u	V''' → i

5) Proposiciones

P → A	P → <Llama>
P → I	P → <Devuelve>
P → H	P → c;
P → W	
P → J	

6) Lista de 0 más proposiciones:

<listaP> → ξ
<listaP> → P<listaP>

7) Sentencia Loop

Ejemplo:

```
loop ($num>6) make
{
    <proposiciones>
}
```

Gramática:

W → w(R)m{<listaP>}

8) Sentencia Evaluate

Ejemplos:

```
evaluate (<expRel>)
    <proposiciones>
instead
    <proposiciones>
:

evaluate (<expRel>)
    <proposiciones>
:
```

Gramática:

I → f(R)<listaP>I':
I' → t<listaP>
I' → ξ

9) Sentencia Repeat

Ejemplo:

```
repeat($n=6; $n<10; $n=$n+1)
{
    <proposiciones>
}
```

Gramática

J → j(YXZ{<listaP>}
Y → i=E;
Y → ;
X → R;
X → ;
Z → i=E)
Z →)

10) Sentencia Select

Ejemplo:

```
select($num){
  case 1: <proposiciones>
          stop
  case 2: <proposiciones>
  other: <proposiciones>
}
```

Gramática:

H→ h(i){C'O'}
C' → an:<listaP>UC'
C' → ξ
O' → o:<listaP>
O' → ξ
U → q
U → ξ

11) Sentencia Throw

Ejemplos:

```
throw($valor);
throw("cadena1");
throw();
```

Gramática:

<Devuelve> → z(<valor>);
<valor> → V
<valor> → ξ

12) Llamada a una función

Ejemplo:

```
[$fun($x,8)]
```

Gramática:

<Llama> → [i(<arg>)]
<arg> → ξ
<arg> → V<otroArg>
<otroArg> → ,V<otroArg>
<otroArg> → ξ

13) Funciones

Ejemplo:

```
number  $fun (real $x, number $n){
    <sentencias>
}
```

Gramática:

<Func> → <Tipo>i(<Param>){<Cuerpo >}
<Param> → <Tipo>i<otroParam>
<Param> → ξ
<otroParam>→,<Tipo>i<otroParam>
<otroParam> → ξ
<Cuerpo> → <Decl><listaP>
<Decl> → ξ
<Decl> → D<Decl>

Estructura del Programa:

<Serie de funciones>

Gramática:

<Program> →<Func><otraFunc>
<otraFunc> →<Func><otraFunc>
<otraFunc> → ξ

Propuesta de solución y fases del desarrollo del sistema.

Propuesta de solución.

Implementación de un programa escrito en C y lex hecho a partir del programa del análisis léxico, consiguiendo así el análisis léxico-sintáctico.
Para esto, es necesario aplicar el método del *Analizador Sintáctico Descendente Recursivo*, el cual deriva del uso y aplicación de la metodología de las Gramáticas LL(1), donde se deben obtener los conjuntos de selección de las producciones mostradas en el Anexo A.
Aplicando el ASDR obtendremos las funciones de cada No Terminal (NT) para su construcción recursiva mediante el uso de llamadas y así obtener la correcta aplicación de sintaxis acompañada de sus excepciones para la identificación de errores.
Enseguida es necesario crear las sentencias adecuadas que identifiquen los átomos de cada clase definidas en el analizador léxico, para su posterior almacenamiento en alguna lista y así finalmente obtener la cadena de átomos del analizador sintáctico.
La explicación detallada de esta propuesta se encuentra en el *diseño e implementación*.

Planificación.

- Viernes 02 de Diciembre del 2022.
    - Lectura y análisis de los requerimientos del proyecto así como su rúbrica.
    - Apertura y descarga de la última versión del proyecto desde el repositorio creado en la primera parte.
        - 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
    - Análisis y estudio de la primera parte del compilador.
    - Lectura de correcciones(3) hechas por la profesora Laura Sandoval.
    - Realizar correcciones.
- Sábado 03 de Diciembre del 2022.
    - Completar correcciones y detalles faltantes de la primera parte.
    - Obtención de conjuntos de selección de producciones.
    - Obtención funciones del ASDR en código directamente.
- Domingo 04 de Diciembre del 2022.
    - Implementación de las funciones del ASDR en el código principal.
    - Creación función que convierte un token a átomo (únicamente aplica para Palabras Reservadas y Operadores relacionales).

- Creación de función que almacena los átomos en una lista.
- Modificación de las funcionalidades que tienen los token en lex para que hagan un casteo a átomos y se almacenen en una lista, o directamente si no es el caso, almacenarlas en la lista.
- Creación de función que convierte la lista de átomos en una cadena.
- Creación de archivo que almacena la cadena de átomos.
- Aplicación de pruebas y manejo de errores.

- Lunes 05 de Diciembre del 2022.
  - Creación formal de la documentación del proyecto.
  - Entrega del proyecto.

Diseño e implementación.

FASE 1:

- Viernes 02 de Diciembre del 2022.
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Lectura y análisis de los requerimientos.
  Apertura y descarga de la última versión del proyecto desde el repositorio creado en la primera parte.
  Análisis y estudio de la primera parte del compilador.
  Lectura de correcciones(3) hechas por la profesora Laura Sandoval
  Responsable(s): Carlos Castelan.
  Realizar correcciones.
  - ➢ Corrección 1: Los enteros se ponen directamente en los tokens, no utilizan tabla.
    Solución:
    Únicamente fue necesario eliminar la llamada a la función addList en el token para que no cree una lista, además se eliminó el archivo que almacenaba la lista de enteros.
  - ➢ Corrección 2: Las cadenas pueden contener cualquier carácter, no sólo letras y números.
    Solución:
    Se modificó la expresión regular de cadenas y carácteres para que acepten cualquier carácter exceptuando comillas simples y dobles respectivamente para que no existan errores.
    ER(Lex/Flex) Cadenas = `\"[^"]{2,40}\"`
    ER(Lex/Flex) Carácter = `\'[^']{1,1}\'`
  - ➢ Corrección 3: Se repiten los identificadores en la tabla de símbolos.
    Solución:
    Se creó una estructura de control while dentro de la función addListIdentificador que es la función que crea la lista de identificadores, esta estructura while se puso enmedio de la primera

inserción y de las posteriores inserciones para la identificación de identificadores repetidos.

```
//Recorre lista para buscar identificador repetido
        int posicion=0;
        NodeIdentificador* aux=
listaIdentificadores.init;
        while(aux != NULL){
            if(!strcmp(aux->dato, atomo)){ //Verifica si
esta
                return posicion; //Regresa la posicion
            }
            else{
                aux = aux->sig; //Pasa al siguiente nodo
                posicion ++;
            }
        }
```

FASE 2:

- Sábado 03 de Diciembre del 2022.
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Obtención de conjuntos de selección de producciones.
  Obtención funciones del ASDR en código directamente.
    - Para ambas actividades, se trabajó en una tabla para la repartición del trabajo.

No.	Función	Conjunto de selección	Responsable	No terminales
1:	<Program> →<Func><otraFunc>	C.S.(1)={bg#yx}	Carlos Rivas	<Program>
2:	<otraFunc> →<Func><otraFunc>	C.S.(2)={bg#yx}	Carlos Rivas	<otraFunc>
3:	<otraFunc> → ξ	C.S.(3)={-|}	Carlos Rivas	<Func>
4:	<Func> → <Tipo>i(<Param>){<Cuerpo>}	C.S.(4)={bg#yx}	Carlos Rivas	<Param>
5:	<Param> → <Tipo>i<otroParam>	C.S.(5)={bg#yx}	Carlos Rivas	<otroParam>
6:	<Param> → ξ	C.S.(6)={)}	Carlos Rivas	<Cuerpo>
7:	<otroParam>→,<Tipo>i<otroParam>	C.S.(7)={,}	Carlos Rivas	<Decl>
8:	<otroParam> → ξ	C.S.(8)={)}	Carlos Rivas	D
9:	<Cuerpo> → <Decl><listaP>	C.S.(9)={i}bg#yxcwfjhz[}	Carlos Rivas	<Tipo>
10:	<Decl> → ξ	C.S.(10)={i}cwfjhz[}	Carlos Rivas	K
11:	<Decl> → D<Decl>	C.S.(11)={bg#yx}	Carlos Rivas	Q
12:	D → <Tipo>K;	C.S.(12)={bg#yx}	Carlos Rivas	N

13:	<Tipo>→ b	C.S.(13)={b}	Carlos Rivas	C
14:	<Tipo>→ g	C.S.(14)={g}	Carlos Rivas	A
15:	<Tipo>→ #	C.S.(15)={#}	Carlos Rivas	A’
16:	<Tipo>→ y	C.S.(16)={y}	Carlos Rivas	E
17:	<Tipo>→ x	C.S.(17)={x}	Carlos Rivas	E’
18:	K → iQ	C.S.(18)={i}	Carlos Rivas	T
19:	Q → ξ	C.S.(19)={;}	Carlos Rivas	T’
20:	Q → =NC	C.S.(20)={=}	Carlos Rivas	F
21:	Q → ,K	C.S.(21)={,}	Carlos Rivas	R
22:	N → n	C.S.(22)={n}	Carlos Castelan	R’
23:	N → r	C.S.(23)={r}	Carlos Castelan	V
24:	N → s	C.S.(24)={s}	Carlos Castelan	V’
25:	C → ξ	C.S.(25)={;}	Carlos Castelan	V’’
26:	C → ,K	C.S.(26)={,}	Carlos Castelan	V’’’
27:	A → i=A’;	C.S.(27)={i}	Carlos Castelan	P
28:	A’ → s	C.S.(28)={s}	Carlos Castelan	<listaP>
29:	A’ → E	C.S.(29)={(inr[}	Carlos Castelan	W
30:	E → T E’	C.S.(30)={(inr[}	Carlos Castelan	I
31:	E’ → + T E’	C.S.(31)={+}	Carlos Castelan	I’
32:	E’ → - T E’	C.S.(32)={-}	Carlos Castelan	J
33:	E’ → ξ	C.S.(33)={;)}	Carlos Castelan	Y
34:	T → F T’	C.S.(34)={(inr[}	Carlos Castelan	X
35:	T’ → *F T’	C.S.(35)={*}	Carlos Castelan	Z
36:	T’ → /FT’	C.S.(36)={/}	Carlos Castelan	H
37:	T’ → \FT’	C.S.(37)={\}	Carlos Castelan	C’

38:	T’ → %FT’	C.S.(38)={%}	Carlos Castelan	O’
39:	T’ → ^FT’	C.S.(39)={^}	Carlos Castelan	U
40:	T’ → ξ	C.S.(40)={;)+-}	Carlos Castelan	<Devuelve>
41:	F → (E)	C.S.(41)={(}	Carlos Castelan	<valor>
42:	F → i	C.S.(42)={i}	Carlos Castelan	<Llama>
43:	F → n	C.S.(43)={n}	Carlos Castelan	<arg>
44:	F → r	C.S.(44)={r}	Carlos Castelan	<otroArg>
45:	F → <Llama>	C.S.(45)={[}	Carlos Castelan	
46:	R → iR’V	C.S.(46)={i}	Carlos Castelan	
47:	R → nR’V’	C.S.(47)={n}	Carlos Castelan	
48:	R → rR’V’’	C.S.(48)={r}	Carlos Castelan	
49:	R → sR’V’’’	C.S.(49)={s}	Carlos Castelan	
50:	R’ → >	C.S.(50)={>}	Carlos Castelan	
51:	R’ → <	C.S.(51)={<}	Carlos Castelan	
52:	R’ → l	C.S.(52)={l}	Carlos Castelan	
53:	R’ → e	C.S.(53)={e}	Carlos Castelan	
54:	R’ → d	C.S.(54)={d}	Carlos Castelan	
55:	R’ → u	C.S.(55)={u}	Carlos Rivas	
56:	V → i	C.S.(56)={i}	Carlos Rivas	
57:	V → n	C.S.(57)={n}	Carlos Rivas	
58:	V → r	C.S.(58)={r}	Carlos Rivas	
59:	V → s	C.S.(59)={s}	Carlos Rivas	
60:	V’ → n	C.S.(60)={n}	Carlos Rivas	
61:	V’ → i	C.S.(61)={i}	Carlos Rivas	

62:	V” → r	C.S.(62)={r}	Carlos Rivas	
63:	V” → i	C.S.(63)={i}	Carlos Rivas	
64:	V”’ → s	C.S.(64)={s}	Carlos Rivas	
65:	V”’ → i	C.S.(65)={i}	Carlos Rivas	
66:	P → A	C.S.(66)={i}	Carlos Rivas	
67:	P → I	C.S.(67)={f}	Carlos Rivas	
68:	P → H	C.S.(68)={h}	Carlos Rivas	
69:	P → W	C.S.(69)={w}	Carlos Rivas	
70:	P → J	C.S.(70)={j}	Carlos Rivas	
71:	P → <Llama>	C.S.(71)={[}	Carlos Rivas	
72:	P → <Devuelve>	C.S.(72)={z}	Carlos Rivas	
73:	P → c;	C.S.(73)={c}	Carlos Rivas	
74:	<listaP> → ξ	C.S.(74)={}:taoq}	Carlos Rivas	
75:	<listaP> → P<listaP>	C.S.(75)={icwfjhz[}	Carlos Rivas	
76:	W → w(R)m{<listaP>}	C.S.(76)={w}	Carlos Rivas	
77:	I → f(R)<listaP>I’:	C.S.(77)={f}	Carlos Rivas	
78:	I’ → t<listaP>	C.S.(78)={t}	Carlos Rivas	
79:	I’ → ξ	C.S.(79)={:}	Carlos Rivas	
80:	J → j(YXZ{<listaP>}	C.S.(80)={j}	Carlos Rivas	
81:	Y → i=E;	C.S.(81)={i}	Carlos Rivas	
82:	Y → ;	C.S.(82)={;}	Carlos Rivas	
83:	X → R;	C.S.(83)={inrs}	Carlos Castelan	
84:	X → ;	C.S.(84)={;}	Carlos Castelan	
85:	Z → i=E)	C.S.(85)={i}	Carlos Castelan	
86:	Z →)	C.S.(86)={)}	Carlos Castelan	
87:	H→ h(i){C’O’}	C.S.(87)={h}	Carlos Castelan	
88:	C’ → an:<listaP>UC’	C.S.(88)={a}	Carlos Castelan	
89:	C’ → ξ	C.S.(89)={}o}	Carlos Castelan	
90:	O’ → o:<listaP>	C.S.(90)={o}	Carlos Castelan	

91:	O' → ξ	C.S.(91)={}	Carlos Castelan	
92:	U → q	C.S.(92)={q}	Carlos Castelan	
93:	U → ξ	C.S.(93)={}ao}	Carlos Castelan	
94:	<Devuelve> → z(<valor>);	C.S.(94)={z}	Carlos Castelan	
95:	<valor> → V	C.S.(95)={inrs}	Carlos Castelan	
96:	<valor> → ξ	C.S.(96)={)}	Carlos Castelan	
97:	<Llama> → [i(<arg>)]	C.S.(97)={[}	Carlos Castelan	
98:	<arg> → ξ	C.S.(98)={)}	Carlos Rivas	
99:	<arg> → V<otroArg>	C.S.(99)={inrs}	Carlos Rivas	
100:	<otroArg> → ,V<otroArg>	C.S.(100)={,}	Carlos Rivas	
101:	<otroArg> → ξ	C.S.(101)={)}	Carlos Rivas	

Responsable(s): Carlos Castelan, Carlos Rivas.
Funciones del ASDR pasadas directamente a código en C (Las funciones fueron repartidas igual que los conjuntos de selección), además sólo se mostrarán el nombre de las funciones.

Funciones en C:

Simbología:

- Carlos Rivas
- Carlos Castelan

Aclaración previa importante:

El símbolo de fin de cadena se representa con un 0.

1. <Program>:
2. <otraFunc>
3. <Func>
4. <Param>:
5. <otroParam>
6. <Cuerpo>
7. <Decl>
8. D
9. <Tipo>
10. K
11. Q

12. N
13. C
14. A
15. A’
16. E
17. E’
18. T
19. T’
20. F
21. R
22. R’
23. V
24. V’
25. V”
26. V””
27. P
28. <listaP>
29. W
30. I
31. I’
32. J
33. Y
34. X
35. Z
36. H
37. C’
38. O’
39. U
40. <Devuelve>
41. <valor>
42. <Llama>
43. <arg>
44. <otroArg>

FASE 3:

- Domingo 04 de Diciembre del 2022.
  Responsable(s): Carlos Castelan, Carlos Rivas.
  Implementación de las funciones del ASDR en el código principal.
  Responsable(s): Carlos Castelan.
  Creación función que convierte un token a átomo (únicamente aplica para Palabras Reservadas y Operadores relacionales).
  - ➢ Esta función fue declarada como:

    ```
    char* atomos(char* id);
    ```

  Responsable(s): Carlos Castelan, Carlos Rivas.

Creación de función que almacena los átomos en una lista.

- ➢ Esta función fue declarada como:

```
addAtomo(char atomo[1]);
```

Responsable(s): Carlos Castelan, Carlos Rivas.

Modificación de las funcionalidades que tienen los token en lex para que hagan un casteo a átomos y se almacenen en una lista, o directamente si no es el caso, almacenarlas en la lista.

- ➢ Para esto únicamente fue necesario modificar los tokens para que trabajen con:

```
addAtomo("atomo en carácter");
addAtomo(yytext);
addAtomo(atomos(yyetext));
```

Responsable(s): Carlos Castelan, Carlos Rivas.

Creación de función que convierte la lista de átomos en una cadena.

- ➢ Se creó una función que obtiene cada elemento de la lista de átomos y las convierte en una cadena, posteriormente se almacena en una variable.

```
char* getCadena(struct list lista)
```

Responsable(s): Carlos Rivas.

Creación de archivo que almacena la cadena de átomos.

- ➢ Se crea el archivo con su identificador para posteriormente enviarlo con su nombre en formato txt.

```
FILE *ArchCadenaAtomos;
CadenaAtomos.txt
```

Responsable(s): Carlos Castelan, Carlos Rivas.

Aplicación de pruebas y manejo de errores.

- ➢ Se revisaron y analizaron errores existentes en las pruebas de compilación.
- ➢ Se solucionaron errores y bugs de memoria, que se crearon en funciones.
- ➢ Se revisaron la correcta declaración de funciones hechas con el ASDR.

FASE 4:

- Lunes 05 de Diciembre del 2022.

Responsable(s): Carlos Castelan, Carlos Rivas.

Creación formal de la documentación del proyecto.

Responsable(s): Carlos Castelan.

Entrega del proyecto.

Manual de usuario del analizador léxico.

1. Tendremos el archivo ***compilador.l*** que es nuestro archivo del programa fuente.
2. Para ejecutarlo debemos tener un archivo de carga inicial, el cual debe contener el código fuente con diferentes tipos de datos que cumplan con las especificaciones de la funcionalidad del analizador léxico y además siga la estructura sintáctica del lenguaje declarada en el análisis sintáctico, de preferencia este archivo de entrada deberá ser de alguna extensión .txt para su mejor interpretación.
3. Ambos archivos, el del programa fuente ***compilador.l*** y el de la carga inicial, en nuestro caso ***pruebaAnalSint.txt*** deben estar en la misma carpeta.
4. Proceso de ejecución del programa:
    a. Ejecutar en la terminal el comando ***flex compilador.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 pruebaAnalSint.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

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

Ejemplo de uso.

1. Archivos iniciales para su ejecución.

   compilador.l
   pruebaAnalSint.txt

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

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

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

   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***

   a.out

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

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

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

   ```
   nahual@nahualT:~/inge/7moSemestre/Compiladores/ProyectoPT1/ProyectoCompiladores$ ./a.out pruebaAnalSint.txt
   Probando la compilación . . .

   Iniciando analizado léxico . . .
   Análisis léxico terminado

   Iniciando analizador sintáctico . . .
   Análisis sintáctico terminado
   El codigo fuente fue analizado sintácticamente por completo.
   ```

   a.out
   CadenaAtomos.txt U
   CaracteresNoReconocidos.txt U
   compilador.l M
   lex.yy.c
   pruebaAnalSint.txt
   TablaCadenas.txt U
   TablaConsReales.txt U
   TablaIdentificadores.txt U
   TablaOperadoresArimeticos.txt U
   TablaOperadoresRelacionales.... U
   TablaPalabrasReservadas.txt U
   TablaSimbolosEspeciales.txt U
   TokensGenerales.txt U

8. Archivo ***CadenasAtomos.txt***: Este nos mostrará la cadena de átomos, para verificar que existe el fin de cadena se ocupó el ***0 (cero)*** como símbolo de fin de cadena.

   ```
   Cadena de átomos:
   yi(){xi=r,i;yi,i=s;bi=n,i,i;f(iur)w(iln)m{i=i+n\n;i=(n+i)
   ^n;f(i>n)c;:}i=s;:j(i=n;iln;i=i+n){[i(i)]}f(iei)z(s);tz(s
   ```

```
);:}#i(bi){yi;#i;h(i){an:an:an:i=s;qan:an:an:i=s;qan:an:a
n:i=s;qo:i=s;}i=i*r%n-i;z(i);}0
```

Conclusiones.

- Carlos Castelan Ramos:
  La realización de la segunda parte de este compilador tomó tiempo, desde la planeación de horarios para coincidir con mi compañero, el análisis práctico y teórico hasta la escritura del código fueron partes indispensables para su correcto desarrollo. Sin embargo, opino que debido a las cualidades que adquirimos respecto a compiladores de la primera entrega, se nos facilitaron distintas partes del proceso, como la creación de listas, escritura de archivos y manipulación de tokens. Por otro lado, el análisis práctico como los conjuntos de selección, fueron desgastantes, por la gran cantidad de conjuntos que habían, así también fue con la creación de funciones del ASDR, ya que eran muchas funciones y era necesario estar pendiente a cada llamada recursiva. Considero que esta entrega cumple con todos los requerimientos sin excepción, por lo que el objetivo se cumplió correctamente.

- Carlos Eduardo Rivas Solís:
  El desarrollo de esta segunda entrega se basó un poco más en la creación y análisis teórico de justamente el análisis sintáctico, poniendo en práctica lo aprendido del tema 3 y 4 pero a mayor escala, para esto hicimos una repartición de trabajo equitativa para la obtención de conjuntos de selección y las funciones de los NT.
  a parte de programación se llevó con éxito casi sin complicaciones, a excepción de detalles de declaración en las funciones del ASDR, ciertos detalles de sintaxis y uso de memoria. Al final del día creó que se cumplieron todos los puntos, ya que con esfuerzo, revisamos minuciosamente cada detalle de este proyecto.

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