Universidad Nacional Autónoma de México

Facultad de Ingeniería

Compiladores

Proyecto 2

Integrantes:
Díaz González Rivas Ángel Iñaqui
Gayosso Rosillo Sebastian Emiliano
Perez Delgado Erandy Estefanya
Siliano Haller Rodrigo

Profesora:
Sáenz García Elba Karen

Grupo: 1

Semestre: 2025-1

Fecha de entrega: 04/Noviembre/2024

ÍNDICE

1. ÍNDICE
2. OBJETIVO
3. DESCRIPCIÓN DEL PROBLEMA
4. CONJUNTO SELECCIÓN DE CADA PRODUCCIÓN
5. INDICACIONES
6. CONCLUSIONES
7. REFERENCIAS

Objetivo

Desarrollar un analizador léxico y un analizador sintáctico descendente recursivo en un solo programa que pueda interpretar y validar programas escritos en un lenguaje específico, siguiendo las reglas definidas por la gramática LL(1) proporcionada.

Descripción del problema

El proyecto consiste en la implementación de un analizador léxico-sintáctico en un solo programa, cuyo objetivo es analizar y verificar la sintaxis de programas escritos en un lenguaje definido por una gramática específica.

El analizador léxico, su principal función es leer el código fuente y dividirlo en sus componentes más básicos, conocidos como *tokens*. Estos tokens representan las unidades mínimas de significado en el código, como palabras reservadas, operadores, identificadores, constantes numéricas, cadenas de texto, y caracteres especiales. Este analizador léxico lee el código fuente carácter por carácter desde el archivo de entrada, buscando patrones que coincidan con los componentes léxicos definidos en la gramática.

Identificación de Tokens: Clasifica los patrones encontrados en categorías léxicas, como palabras clave, operadores, identificadores y constantes numéricas. Con la obtención de los Tokens, genera la cadena de átomos que servirá de entrada para el siguiente paso en el análisis, el cual corresponde al análisis sintáctico.

El analizador sintáctico recibe la secuencia de tokens generada por el analizador léxico y utiliza un enfoque de análisis descendente recursivo basado en las reglas gramaticales del lenguaje para verificar la validez sintáctica del código fuente.

El manejo de errores implica la identificación y gestión adecuada de errores léxicos y sintácticos.

El conjunto de selección de cada producción

1: <Programa>→ <ListaD>[<SerieF>]	First(<ListaD>)	gndh[aiwfb
2 <ListaD>→ D<ListaD>	First(D)	gndh
3 <ListaD>→ ξ	Follow(<ListaD>) = {[┤} U First(S)	[┤ aiwfb
4 <SerieF>→ <Func><otraF>	First(<Func>)	gndhv
5 <otraF>→ <Func><otraF>	First(<Func>)	gndhv
6 <otraF>→ ξ	Follow(<otraF>) = Follow(<SerieF>)	]
7 D→ <Tipo>L:	First(<Tipo>)	gndh
8 <Tipo>→ g		g
9 <Tipo>→ n		n
10 <Tipo>→ d		d
11 <Tipo>→ h		h
12 L→ a<Valor>l’		a
13 l’→ ,a<Valor>l’		,
14 l’ → ξ	Follow(l’) = Follow(L)	:
15 <Valor>→ =V		=
16 <Valor>→ ξ	Follow(<Valor>) = First(l’)	,:
17 V→ c		c
18 V→ s		s
19 V→ z		z
20 V→ r		r

21 A→ aA’:		a
22 A’→ =E		=
23 A’→ mE		m
24 A’→ kE		k
25 A’→ pE		p
26 A’→ tE		t

27 A'→ uE		u
29 I→ i[R]F'		i
30 F'→ (S)B		(
31 B→ e(S)		e
32 B→ξ	Follow(B)= Follow(F')=Follow(I)=Follow(S') =First (<otraS>)	aiwfb[)
33		
34 W→ w[R](S)		w
35 <For>→ f[E](S)		f
36 <Return>→ bZ		b
37 Z→ [E]:		[
38 Z→ :		:
39 E→ TE'	First(T)	(azr[
40 E'→ +TE'		+

41 E'→ -TE'		-
42 E'→ ξ	Follow(E')= Follow(E) = Follow(A') U {]} U {)} U First(R') U Follow(R') = {:} U {]} U {)} U {><?y|¿} U Follow(R) ={:} U {]} U {)} U {><?y|¿] U {]}	:])><?y|¿
43 T→ FT'	First(F)	(azr[
44 T'→ *FT'		*
45 T'→ /FT'		/
46 T'→ $FT'		$
47 T' → ξ	Follow(T') = Follow(T) = First(E')	+-:])><?y|¿
48 F→ (E)		(
49 F→ a		a
50 F→ z		z
51 F→ r		r
52 F→ [a]		[
53 R→ ER'	First(E)	(azr[
54 R'→ >E		>

55 R'→ <E		<
56 R'→ ?E		?
57 R'→ yE		y
58 R'→ |E		|
59 R'→ ¿E		¿
60 S→ S'<otraS>	First(S')	aiwfb[

61 S'→ A	First(A)	a
62 S'→ I	First(I)	i
63 S'→ W	First(W)	w
64 S'→ <For>	First(<For>)	f
65 S'→ <Return>	First(<Return>)	b
66 S'→ [a]:		[
67 <otraS>→ S'<otraS>	First(S')	aiwfb[
68 <otraS> → ξ	Follow(<otraS>) = Follow(S)	)
69 <Func>→ <TipoFun>a(<listaD>S)	First(<TipoFun>)	gndhv
70 <TipoFun>→ <Tipo>	First(<Tipo>)	gndh
71 <TipoFun>→ v		v

Indicaciones

Para inicializar el programa, primero tenemos que crear un archivo de texto que queremos analizar.

Ya creado el texto a analizar, abrimos nuestro analizador de extensión '.l' desde la terminal.

```
milo@milo-Nobilis:~/Imágenes$ flex sint.l
```

Una vez abierto el programa, se procede a compilar, lo que nos generará un archivo ejecutable.

```
milo@milo-Nobilis:~/Imágenes$ gcc lex.yy.c -lfl
```

Por último, para analizar el texto deseado, ejecutamos el comando: ‘./a.out prueba.txt ‘ en la terminal.

```
milo@milo-Nobilis:~/Imágenes$ ./a.out prueba.txt
```

Automáticamente se nos generarán los 6 archivos.txt que son: tokens, errores_lexicos, tabla_simbolos, tabla_literales, atomos y errores_sintacticos.

Conclusiones

Díaz González Rivas Ángel Iñaqui: Durante el desarrollo del proyecto, utilizamos el analizador léxico creado previamente para identificar los tokens y construir la cadena de átomos necesaria para el análisis sintáctico. Posteriormente, calculamos los conjuntos de selección para cada producción de la gramática, realizando los ajustes necesarios para que fuera compatible con un analizador LL(1). Finalmente, implementamos el analizador sintáctico descendente recursivo, el cual, mediante funciones auxiliares, detectó errores en las cadenas de átomos. Con esta estructura, se logró cumplir el objetivo de crear un sistema de análisis léxico-sintáctico completo y preciso.

Gayosso Rosillo Sebastian Emiliano: En conclusión, el proyecto alcanzó satisfactoriamente sus objetivos, permitiéndonos desarrollar un sistema de análisis léxico-sintáctico completo y eficiente. Utilizamos el analizador léxico previamente creado para identificar tokens y construir una cadena de átomos que sirvió como base para el análisis sintáctico. Posteriormente, implementamos un analizador sintáctico descendente recursivo, ajustando la gramática y los conjuntos de selección para asegurar su compatibilidad con un analizador LL(1). Este proceso nos permitió identificar errores en las cadenas de átomos de manera precisa.

A diferencia del proyecto léxico inicial, este proyecto implicó mayores retos, como la integración de reglas gramaticales y el manejo de la recursividad. Además, la colaboración en equipo fue fundamental en todas las etapas del desarrollo, desde la planificación hasta la implementación. La organización y el trabajo conjunto garantizaron una ejecución fluida y eficaz, contribuyendo significativamente a la calidad y precisión del sistema final. Este enfoque colaborativo consolidó nuestros conocimientos y habilidades en el desarrollo de un analizador léxico-sintáctico robusto y funcional.

Perez Delgado Erandy Estefanya:

Este proyecto me pareció muy interesante, ya que al utilizar conceptos previos como el manejo de errores y la identificación de tokens, pudimos aplicarlos ahora en el contexto de un analizador sintáctico. Este proceso me demostró la importancia de comprender las bases del análisis léxico y sintáctico para lograr una verificación completa de los programas. Además, trabajar con la gramática LL(1) me ayudó a apreciar cómo las reglas gramaticales pueden guiar y estructurar un análisis preciso de la sintaxis de un lenguaje.

Otro aspecto fundamental fue la organización y la colaboración en equipo. Este proyecto, siendo algo complejo, requirió un alto nivel de comunicación y coordinación, lo cual fue esencial para resolver problemas rápidamente y compartir conocimientos entre todos los integrantes.

Siliano Haller Rodrigo: El desarrollo de este proyecto fue algo complejo debido a que teníamos que enlazar el proyecto anterior con el nuevo analizador sintáctico, lo cual conllevó mucho análisis en su planteamiento y fue un reto bastante interesante.
Se concluyó el proyecto haciendo uso de la teoría vista en clase y algunos conceptos de flex que se investigaron por fuera, lo cual facilitó su desarrollo.
Comprendí de mejor manera como es la estructura de los codigos en flex, así como su sintaxis, lo cual me facilitará su manejo en proyectos futuros.

Referencias

- Aaby, A. (2003). Compiler construction using flex and bison. Walla Walla College.
- Catalán, J. R. (2010). COMPILADORES. Teoría e implementación. RC Libros.
- Dick Grune, Kees van Reeuwijk, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen. Modern compiler design. Disponible únicamente Libro electrónico Springer, 2012 1 recurso en línea Base de datos: LIBRUNAM
- Hernández, A. (n.d.). FACULTAD DE INGENIERIA UNIVERSIDAD NACIONAL AUTOMOMA DE MEXICO DIVISION DE INGENIERIA MECANICA Y ELECTRICA DEPARTAMENTO DE INGENIERIA EN COMPUTACION. http://www.ptolomeo.unam.mx:8080/jspui/bitstream/132.248.52.100/10230/1/7J%20APUNTES%20DE%20COMPILADORES_OCR.pdf

- Resendiz Chavez, G. (2015). Herramienta Flex para el analisis lexicográfico de lenguajes formales.