Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragon

-Compiladores-

Profesor: Miguel Angel SanchezHernandez

Proeycto Final

Equipo: 10

Contreras Emigidio Ernesto Misael

Grupo: 2609

Indice

1.-Objetivos

- Realización de un compilador utilizando cup
- Crear un lenguaje de programación para que procese el compilador
- Crear una interfaz visual en la cual se pueda observar el comportamiento del compilador y el lenguaje

2.-Introduccion

En el mundo, compiladores juegan un papel fundamental al traducir el código escrito por los humanos en lenguaje de programación a instrucciones comprensibles por las máquinas, este proceso implica una serie de etapas clave, desde el análisis léxico hasta la generación de código y la optimización.

A lo largo de la materia de compiladores, exploraremos los objetivos principales necesarios para comprender y construir un compilador eficiente, desde la comprensión del funcionamiento básico de un compilador hasta la implementación de técnicas avanzadas de optimización.

Para añadir un plus a esta materia, también abordaremos la fascinante tarea de crear un nuevo lenguaje de programación, con la finalidad de crear un programa completo, el cual se pueda utilizar para enseñar programación incluyendo las bases del como programar.

2.1.-Parte de un compilador

Como bien sabemos, un compilador está conformado por distintas partes que permiten su correcto funcionamiento, las principales partes de un compilador son:

- Análisis léxico: Esta etapa se encarga de leer el código fuente y dividirlo en componentes léxicos o tokens, como palabras clave, identificadores, operadores, etc.
- Análisis sintáctico: Aquí se verifica la estructura gramatical del código fuente mediante el uso de gramáticas formales. Se construye un árbol sintáctico para representar la estructura del programa.
- Análisis semántico: En esta fase, se realizan verificaciones adicionales sobre el significado del código fuente, como la comprobación de tipos y la resolución de referencias.
- Generación de código intermedio: Se produce un código intermedio que es más fácil de analizar y optimizar que el código fuente original, pero aún no es código de máquina.
- Optimización de código: Se aplican diversas técnicas para mejorar el rendimiento del código intermedio, como la eliminación de código muerto, la propagación de constantes y la optimización de bucles.
- Generación de código: En esta etapa, se produce el código de máquina específico para la plataforma de destino a partir del código intermedio optimizado.

3.-Desarrollo

A lo largo de la materia de compiladores comprendimos el funcionamiento de un compilador tanto como la creación desde cero de uno y las partes que lo compone, en este caso se presenta la creación de un programa con interfaz, en el cual se pretende el uso para enseñan básica, del como funciona programar orientado principalmente para niños pequeños.

La base principal para crear este programa era utilizar la tecnología de Cup y JFlex, pero en este caso se utilizó JFlex y CompilerTools

- JFlex: Un generador de analizadores léxicos escrito en Java. Utilizamos JFlex para identificar y dividir el texto de entrada en tokens, que luego serán procesados por el analizador sintáctico. JFlex permite especificar reglas léxicas mediante expresiones regulares, facilitando la creación de analizadores léxicos eficientes.

- CompilerTools: Es una herramienta creada para el diseño de un compilador básico en Java. Aunque no es la herramienta ideal para hacer un compilador con una gran cantidad de gramáticas y palabras reservadas, es lo suficientemente intuitivo y funcional para introducirnos en el mundo de los compiladores.

En el desarrollo del proyecto, surgieron ciertas problemáticas en el desarrollo de la parte de Cup, pero ante esta problemática se optó por empezar un desarrollo con alguna alternativa, lo cual buscando en los rincones de internet y ciertos foros encontramos, la herramienta de CompilerTools las cuales posee las siguientes características:

- **Autocompletado de código:** CompilerTools puede sugerir automáticamente el código mientras escribes, lo que puede ahorrar tiempo y reducir los errores de escritura.
- **Extracción de bloques de código:** CompilerTools puede identificar y extraer bloques de código para su posterior ejecución.
- **Manipulación de archivos:** CompilerTools puede manejar operaciones de archivos como Nuevo, Abrir, Guardar y Guardar Como.
- **Creación de errores:** CompilerTools puede generar errores léxicos, sintácticos, semánticos o lógicos para ayudar a los programadores a depurar su código.
- **Creación de gramáticas:** CompilerTools permite a los programadores definir gramáticas para su lenguaje de programación mediante la agrupación de tokens a través de producciones.
- **Almacenamiento de tokens:** CompilerTools puede almacenar tokens y sus atributos básicos, como lexema, componente léxico, línea y columna.

Esta es una alternativa que ayuda a el aprendizaje básico de la creación de un compilador, lo cual nos ayudo al desarrollo de este programa.

3.1.-Gramatica

Gramática de funcionamiento

El programa presenta una gramática sencilla de entender, esta se basa en las siguientes instrucciones:

- inicio();
- Mover – Aquí se coloca la dirección a mover ya sea arriba, abajo, izquierda, derecha y luego los espacios a mover
- Fin();

Ejemplo:

inicio();

mover derecha 4

mover abajo 2

mover abajo 2

mover izquierda 1

fin();

En este ejemplo se puede observar una serie de repeticiones del personaje inicial y hacia donde se dirigirá.

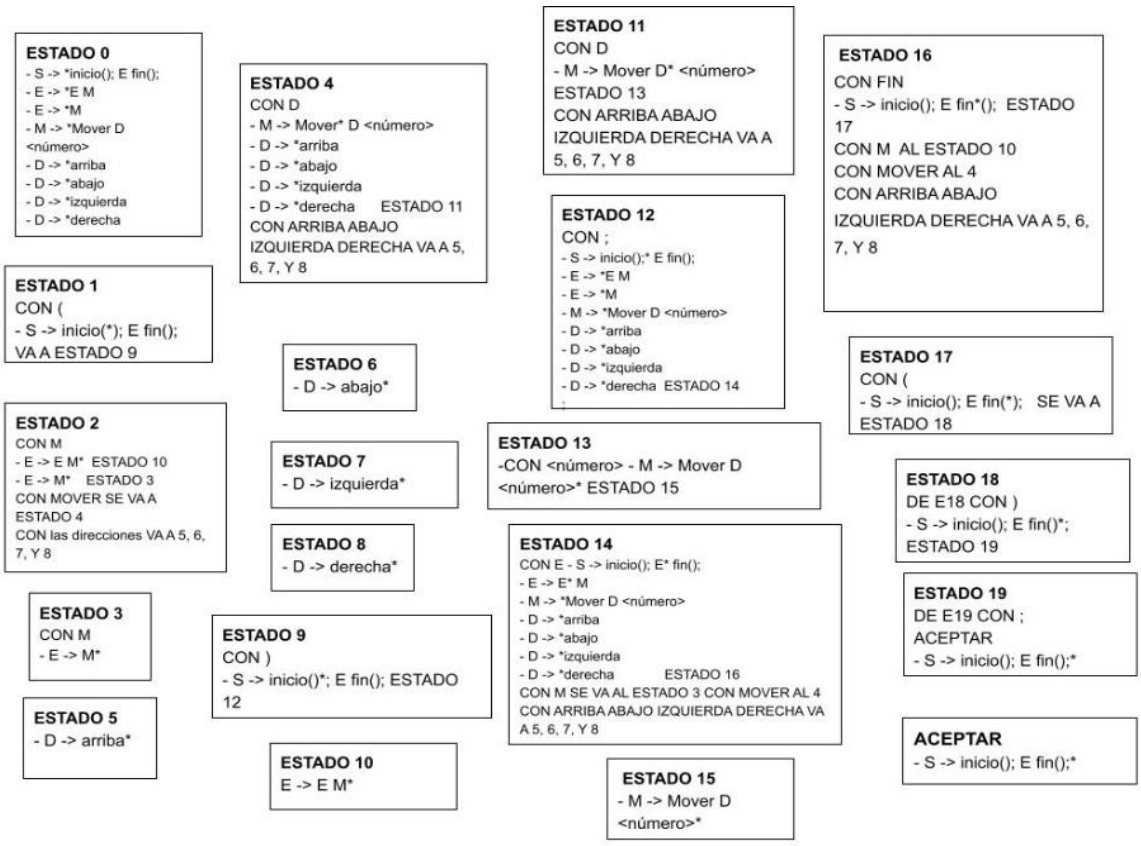
El programa se ejecuta primeramente se escribe el código se compila y finalmente se pasa a la parte de interfaz gráfica para su ejecución visual, tiene características dinámicas a la hora de escribir el código con el fin de llegar al objeto final.

La principal primicia es lograr que el enemigo que somos nosotros alcance al personaje, cada partica cambia de lugar obstáculos y personaje.

Tabla canónica

	MOVER	IZQUIERDA	DERECHA	ARRIBA	ABAJO	<NUMERO>	(	)	;	INICIO	FIN	E	M	D
0	E4	R6	R7	R4	R5					E1		2	3	
1							E9							
2	E4	R6	R7	R4	R5								10|R2	
3						R2				R3	R3			
4	R4					R4				R4	R4			E11
5	R5					R5				R5	R5			
6	R6					R6				R6	R6			
7	R7					R7				R7	R7			
8	R8					R8				R8	R8			
9								E12						
10	R1									R1	R1			
11		R6	R7	R4	R5							13		
12									E14					
13						R2								
14	E4	R6	R7	R4	R5							16	3	
15										R3	R3			
16	E4	R6	R7	R4	R5						E17		10	
17							E18							
18								19						
19									ACEPTAR					

Diagrama en texto de transiciones

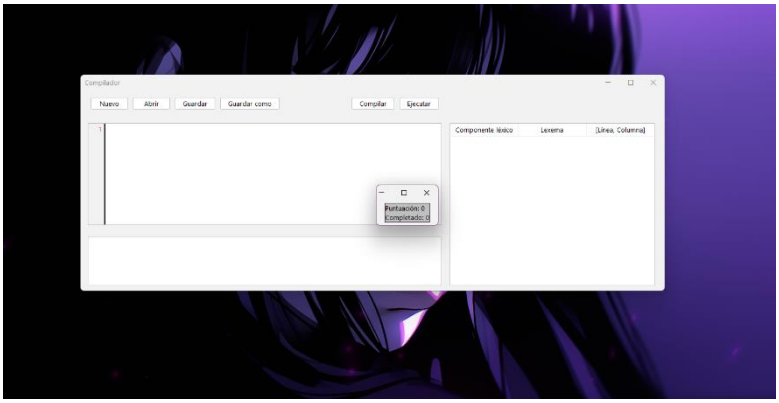


3.1.1.-Funcionamiento

El programa a la hora de cargar el programa en le IDE de su preferencia le aparecerán dos packet uno con el nombre de “pruebas” el cual era el desarrollo incompleto con Cup y el otro con el nombre de “fes.aragon.compilador” el cual es el que contiene el proyecto funcional.

Para ejecutar el programa en el packet buscar la clase llamada “Compilador” y darle a ejecutar, se abritan 2 ventanas emergentes las cuales son: (NOTA: a la hora de ejecutar le puede decir que hay errores esto es por lo de cup, únicamente darle a proceder y funcionara perfectamente)

Apertura de ventanas

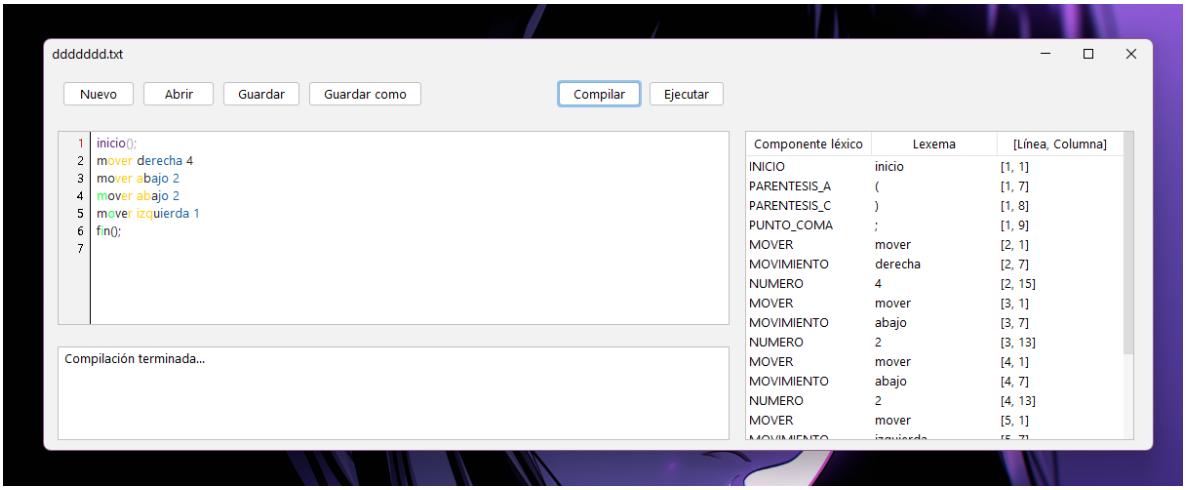


En esta parte aparecen las dos ventanas que conforman al programa, la primera es donde se introduce el código este y una minimizada la cual es la parte visual del programa.

Cada vez que se ejecute el código se gane o se pierda el programa esta hecho para cambiar de posición tanto los obstáculos como el fin de la partida.

3.1.2.-Ventana de compilación

En esta parte se presenta la ventana donde escribiremos el código según nuestra gramática ya presentada anteriormente, el programa tiene la capacidad de compilar previamente el código para saber si hay errores a lo largo del código en este caso también se presenta la capacidad de abrir y guardar cada código introducido y observar el procesamiento y validación del código.



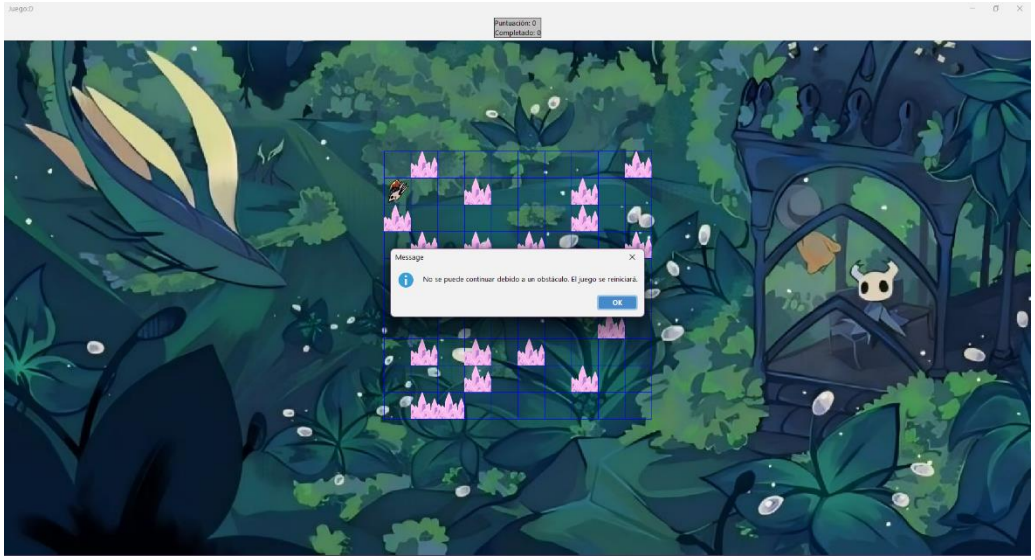
3.1.3.-Ventana de ejecución visual

En esta ventana ya cuando se haya compilado de forma correcta, aquí podremos observar el comportamiento de lo escrito tanto ver si se programó de forma correcta el recorrido, o en su defecto el lograr de forma íntegra el fin del recorrido

NOTA: para hacer funcionar esta parte se necesita una vez maximizar la ventana que aparece pequeña dar click encima de esta en cualquier parte y posterior mente apretar la letra “C2 en le teclado para cargar el programa ya compilado y luego la letra “E” para ejecutar.

Ventana sin código ejecutado maximizada

Ventaja con código ejecutado



En este caso obviamente el código de ejemplo ya presentado anteriormente no cumple de forma correcta para el recorrido asique aparece que no se completó y el porqué.

4.-Conclucion

Aunque no se realizó el programa con la tecnología planteada el cual es CUP, se buscaron alternativas para completar el desarrollo total del proyecto, se presenta un programa de forma 100% funcional y apta para el aprendizaje con una interfaz vistosa y música de fondo a su vez efectos de ejecución y animación de inicio, así concluyendo el desarrollo aplicando los conocimientos obtenidos a lo largo de la materia de compiladores.

5.-Biografia

Vilca, O., Yancachajlla, U., Apaza, J., Quispe, L. B. A., Laura, R., Torres, F., & Araújo, R. (2023). NUEVA GUIA PARA EXPRESIONES REGULARES. En Editora Científica Digital eBooks (pp. 372-381). https://doi.org/10.37885/230813995

Maier, D. (1980). Review of «Introduction to automata theory, languages and computation»
by John E. Hopcroft and Jeffrey D. Ullman. Addison-Wesley 1979. SIGACT News, 12(3), 13-14. https://doi.org/10.1145/1008861.1008863

Klein, G. (s. f.). JFlex - JFlex The Fast Scanner Generator for Java. https://www.jflex.de/

YisusTecFBI. (2023). CompilerTools 2.3.7 - Herramienta de compilador estándar. GitHub - YisusTecFBI/CompilerTools-2.3.7: Herramienta para la creación de un compilador estándar en Java