Tecnológico Nacional de México en Celaya
Departamento de ingeniería en sistemas computacionales.
Lenguajes y autómatas II.
I.S.C Ricardo González González.

COMPILADOR

EQUIPO 3

Integrantes:

- Luis David Garcia Ramirez
- Jose Eduardo Perez Cabrera
- Marco Isaias Ramirez Garcia

ÍNDICE

OBJETIVO.

El trabajo presentado a continuación tiene como objetivo implementar los conocimientos adquiridos en la asignatura de lenguajes y autómatas II, para desarrollar un compilador, el cual pueda ser capaz de determinar si el programa está escrito correctamente desde el punto de vista léxico, sintáctico y semántico.

Todo esto atendiendo las definiciones del lenguaje de programación, como el vocabulario, sintaxis o reglas del mismo.

ALCANCES.

Este compilador será capaz de leer un código fuente en texto plano y evaluar con base en los alfabetos, gramáticas y palabras reservadas para procesar los tres tipos de análisis necesarios para compilar el bloque de código y ejecutar las instrucciones que hayan sido dadas y mostrar el resultado, de lo contrario, si se detecta un error, mostrar los mensajes pertinentes con las descripciones y direcciones de los errores.

JUSTIFICACIÓN.

Para la creación del compilador se hará uso de dos tipos de datos, el primero de ellos, "número" que podrá contener cantidades de números reales y el segundo es a "texto" que podrá contener cadenas de texto a partir de la combinación de símbolos que haya en nuestro alfabeto, por otra parte, se podrán ejecutar instrucciones condicionales y ciclos repetitivos, en las condiciones tendremos la palabra reservada "para" que funciona como un ciclo for y la palabra reservada "mientras" que simula un ciclo while, cada una de las construcciones de los elementos de nuestro compilador contará con reglas gramaticales que nos validaron el lenguaje que evalúa un fragmento de código para ser ejecutado.

Para realizar dichas tareas de análisis a bajo nivel utilizaremos herramientas del lenguaje para las validaciones como lo son Lex para el análisis léxico y Yacc para el análisis sintáctico y semántico declarando nuestra reglas de producción construidos en las gramáticas.

MARCO TEÓRICO.

1. **Análisis léxico**
    a. **¿Qué es?** Es la primera fase que se realiza en un compilador, la cual consta de leer los caracteres de una entrada y elaborar una salida hecha de componentes léxicos.
    b. **¿Para qué?** Esta fase se realiza para entregar una secuencia de

componentes léxicos para que el análisis sintáctico haga su

respectivo análisis, eliminando los datos innecesarios de una cadena recibida como entrada.

c. **¿Cómo aplicarlo?** Este tipo de análisis se puede aplicar por medio de la función llamada charAt, el cual nos retorna el char de una cadena especificando el lugar en dicha cadena, después de esto se tendrá que hacer una validación si el componente es válido en el alfabeto y en caso de que sea válido o no, hacer una acción correspondiente para cada caso, así hasta leer todos los caracteres de una cadena y al final retornar un resultado, ya sea como una cadena para la siguiente fase de análisis o un mensaje de error.

2. **Análisis sintáctico**
    a. **¿Qué es?** Es la segunda fase de análisis del compilador, empieza después de haber terminado la fase de análisis léxico y su función es a partir del resultado de componentes léxico obtenido del análisis léxico, comprobar si la gramática del programa es capaz de generar la cadena recibida de componentes léxicos.
    b. **¿Para qué?** Este análisis se realiza para en dicho caso de que la cadena de componentes léxicos sea correcta entonces elaborar un árbol de análisis sintáctico para que sea pasado a la siguiente fase de análisis para la verificación de la creación de las cadenas, en caso de que la cadena de componentes léxicos sea incorrecta, entonces el análisis deberá mandar un mensaje de error para que el programa pare y evite más errores en las siguientes fases de análisis.
    c. **¿Cómo aplicarlo?** Se puede aplicar mediante operaciones de validación y estructuras de datos, las operaciones de validación para verificar que la cadena pueda ser generada por la gramática, mientras que las estructuras de datos para generar el árbol de análisis sintáctico que resulta de este análisis.

3. **Análisis semántico**
    a. **¿Qué es?** Es la tercera fase de análisis de un compilador y trata de verificar que las declaraciones de variables, nombre de identificadores y que las estructuras de control sean definidas correctamente.
    b. **¿Para qué?** Se realiza este análisis para evitar que los tipos de datos sean declarados incorrectamente y se realicen operación indebidas con estos tipos de datos, también se realiza para que se escriban correctamente el nombre de los identificadores, evitar escribir identificadores que sean inexistentes y también para que en las estructuras de control no contengan código afuera de su

estructura,

c. **¿Cómo aplicarlo?** Este análisis puede ser aplicado en un

compilador a base de verificacion de codigo por medio de un árbol de derivación para comprobar cómo es que fueron creado los tipos de datos y comprobar su sintaxis, también haciendo uso de una tabla de símbolos para verificar los identificadores y poder realizar verificaciones de los nombres de los identificadores y generar mensajes de error en caso de que un nombre de un identificador no exista en el código fuente.

MATERIALES Y HERRAMIENTAS A UTILIZAR.

- **Lenguaje de programación:** Java
- **Entorno de desarrollo:** Visual Studio Code, Git
- **Librerías:** Swing

DESARROLLO PASO A PASO.

Alfabeto contemplado: { a, b, c, d, e, f, g, h, i, j, k, l, m, n, ñ, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, =, <, >, !, (,), |, &, ., ,, :, ;, “, %, λ }

Gramática contempladas:

gramática para el tipo de dato NÚMERO:

Gdn = { (R, U, D, X), (0-9, ., E, +, -), p, s }

p {

< R > -> < U > < D > < X >

< U > -> < U > < U > | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

< D > -> **.** < U > | λ

< X > -> E < I > | E + < I > | E - < I >

}

gramática para el tipo de dato TEXTO:

Gdt = { (I, S, F), (Σ), p, s }

p {

< I > -> “ < S >

< S > -> < S > < S > | < F > | Σ

< F > -> **“**

}

gramática para la condición SI:

Gcs = { (inicio, A,E,L,C,F,G,N,S), (a-z, A-Z, 0-9,), p, s }

p {

<inicio> -> <si> <A>

<A> -> <(> <E>

<E> -> <cierto|falso> (<L>|<G>)

<E> -> <número|variable> <C>

<C> -> <operador comparativo> <F>

<F> -> <número|variable> (<L>|<G>)

<L> -> <operador lógico> (<F>|<E>)

<G> -> <)><{><N>

<N> -> <instrucciones> <}> <S>

<S> -> <sino> <{> <instrucciones> <}>

<S> -> Λ

}

gramática para declarar una variable ENTERO:

Gve = { (inicio, B, N,M,A,U, F), (a-z, A-Z, 0-9,), p, s }

p {

<inicio> -> <entero> <B>

<B> -> < > <N>

<N> -> <[a-z] | [A-Z]> (<M>|<A>|<F>)

<M> -> <[0-9] | [a-z] | [A-Z]> (<M>|<A>|<F>)

<A> -> <=> <U>

<U> -> <NÚMERO ENTERO> (<,><N>|<F>)

<F> -> <;>

}

gramática para la condición MIENTRAS:

Gcm= { (inicio, N, V, O, F), (a-z, A-Z, 0-9), p, s

} p {

<inicio> -> <hacer> <(> <M> <)> <{> <N>

<N> -> <instrucciones> <}>

<M> -> <E>|<F>

<E> -> <cierto|falso> (<L>|<G>)

<E> -> <número|variable> <C>

<C> -> <operador comparativo> <F>

<F> -> <número|variable> (<L>|<G>)

<L> -> <operador lógico> (<F>|<E>)

<G> -> Λ

}

gramática para la condición MIENTRAS:

Gcm= { (inicio, N, V, O, F), (a-z, A-Z, 0-9), p, s
} p {
<inicio> -> <hacer> <{> <N>
<N> -> <instrucciones> <}> <M>
<M> -> <mientras> <(> (<E>|<F>)
<E> -> <cierto|falso> (<L>|<G>)
<E> -> <número|variable> <C>
<C> -> <operador comparativo> <F>
<F> -> <número|variable> (<L>|<G>)
<L> -> <operador lógico> (<F>|<E>)
<G> -> <)><)><;>
}

CASOS DE USO

código de ejemplo para un programa

```
// Bloque de codigo principal
principal {

// Declaracion de variables numericas
entero a = 1 ;
entero b = 10 ;

// Declaracion de variables de texto
texto c = "Hola" ;
texto d = "Mundo" ;

// Declaracion de variables booleanas
booleano ban = verdad ;
booleano opc = falso ;

// Imprimir datos en la consola

imprimir "> Todas las variables declaradas <" ;
```

```
imprimir "Variables -> ["+a+", "+b+", "+c+", "+d+", "+ban+", "+opc+"]";
imprimir "a: "+a ;
imprimir "b: "+b ;
imprimir "c: "+c ;
imprimir "d: "+d ;
imprimir "ban: "+ban ;
imprimir "opc: "+opc ;

// Validacion y reasignacion de expresiones enteras

imprimir "> Actualizacion de las variables <" ;
a = b*(23+6)-1 ;
b = 2*23+a-1 ;
imprimir "a = "+a ;
imprimir "b = "+b ;

// Declaracion de estructura de decision

imprimir "> Estructuras de decision <" ;
si (a<b||b>=a){
imprimir "Condicion 1" ;
}
si (ban&&opc){
imprimir "Condicion 2" ;
}
si (a <= b || b > a) {
imprimir "Condicion 3" ;
}
si (ban && opc && a<b){
imprimir "Condicion 4" ;
}

// Declaracion de estructura de iteracion

imprimir "> Estructuras de iteracion <" ;
entero cont = 0 ;
entero fin = 5 ;
mientras (fin > cont){
cont = cont+1 ;
```

```
imprimir "Cont: "+cont ;
}

// Llave de cierre del bloque principal
}
```

IMPLEMENTACIÓN

La interfaz cuenta con un navbar en la parte superior, en el cual hay 4 secciones con diferentes opciones cada una de estas, que despliega un menú desplegable al pasar el mouse encima de ellas.

La primera sección llamada “Archivo” arroja 5 opciones.

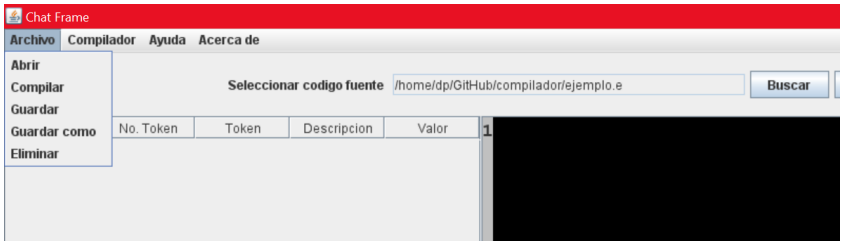
La primera opción “Abrir” permite abrir un documento de texto plano para así desplegarlos en el editor de texto de nuestro compilador para poder visualizarlo y editarlo.

La segunda opción “Compilar” permite compilar el código el cual se encuentra en nuestro editor de texto, para así empezar con los análisis,

La tercer opción “Guardar” permite guardar el código que se encuentra en nuestro editor de texto, aquí hay 2 comportamientos, en caso de que el código que se encuentra en nuestro editor de texto se abrió de un archivo, entonces se guardará en ese mismo archivo que se abrió anteriormente, pero si en caso de que no se haya abierto ningún archivo, entonces desplegará una ventana para guardar un nuevo archivo.

La cuarta opción “Guardar como” permite guardar el código que se encuentra en nuestro editor de texto como un nuevo archivo o seleccionar uno ya existente.

La quinta opción “Eliminar” solo funciona cuando se ha abierto un archivo de texto plano, y su función es eliminar el archivo si el usuario así lo desea.



La segunda sección llamada “Compilador” desplegará 4 opciones.

La primera opción “Limpiar editor de texto” permite limpiar todo lo que se encuentre en el editor de texto.

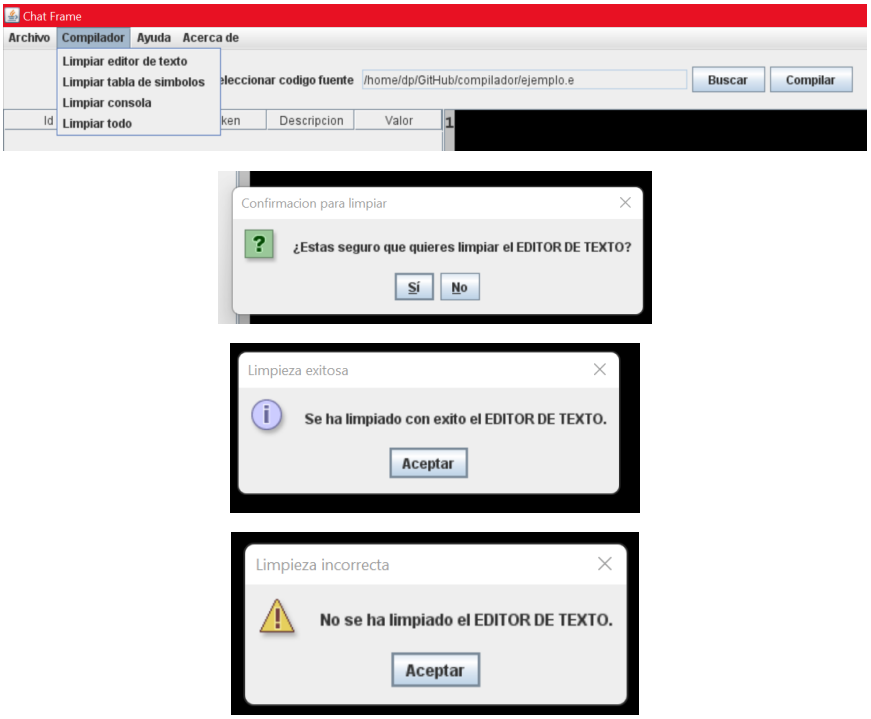
La segunda opción “Limpiar tabla de símbolos” permite limpiar todo lo que se encuentre en la tabla de símbolos, borrando toda información almacenada de anteriores compilaciones.

La tercera opción “Limpiar consola” permite limpiar todo lo que se encuentre en la consola, esto para borrar información como errores de anteriores compilaciones.

La cuarta opción “Limpiar todo” permite limpiar todas las partes de la interfaz,

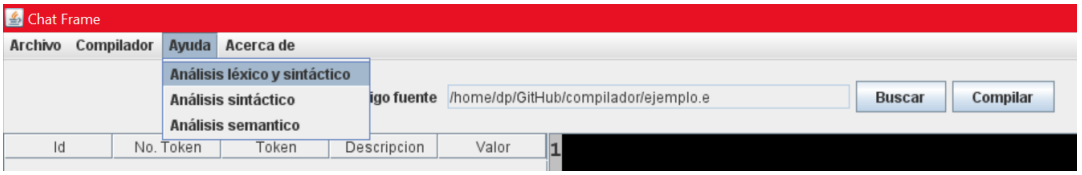
como lo es el editor de texto, la tabla de símbolos y la consola.

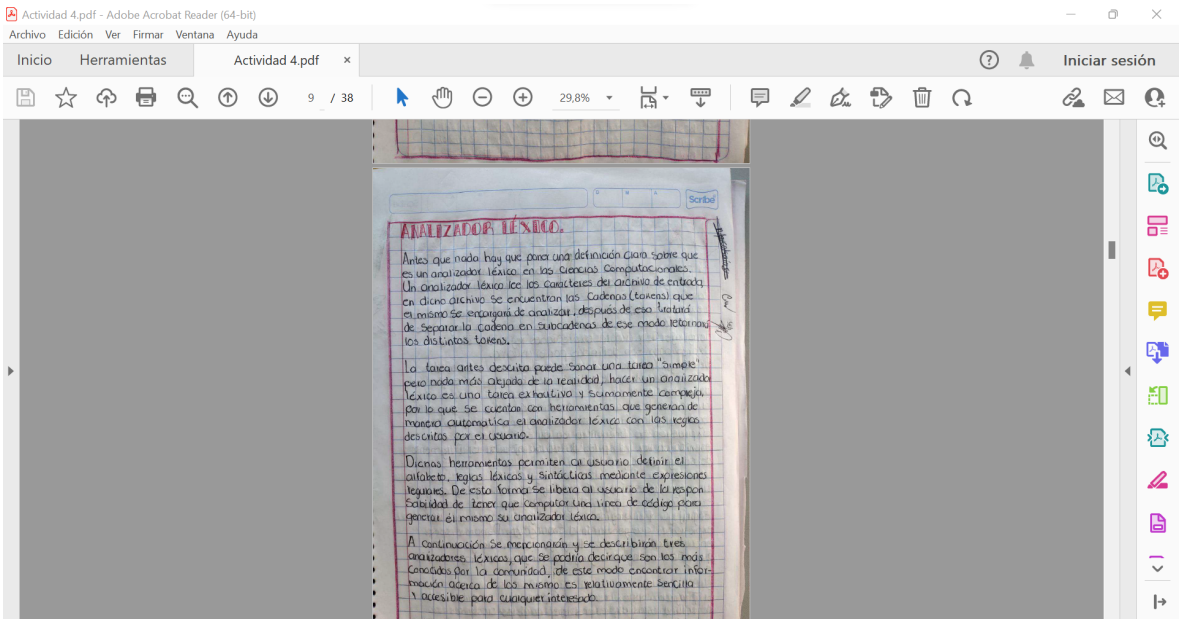
Cabe recalcar que todas estas opciones muestran un mensaje de confirmación para así saber que el usuario está seguro que desea realizar esa acción y sea con su consentimiento.



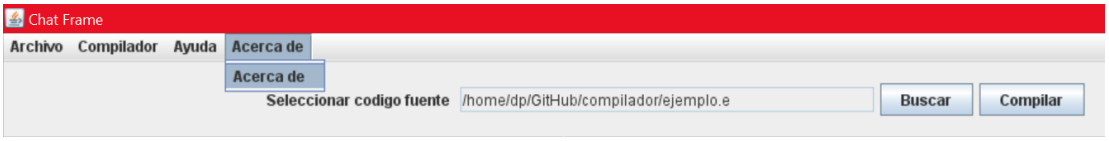
La sección llamada “Ayuda” arroja 3 opciones.

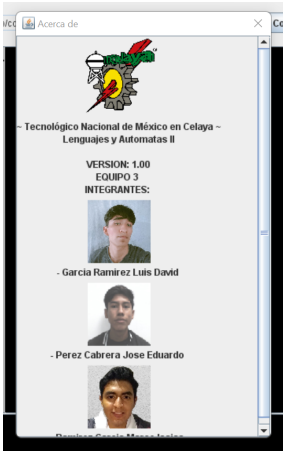
Estas opciones no son más que nada mostrar los PDF correspondientes para que sirvan de ayuda para entender las fases de análisis que realiza el compilador, se muestran PDF del análisis lexico, sintáctico y semántico.



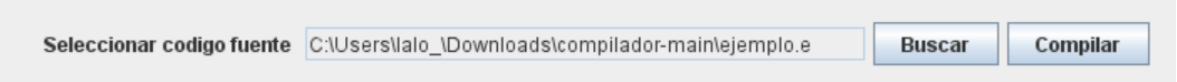


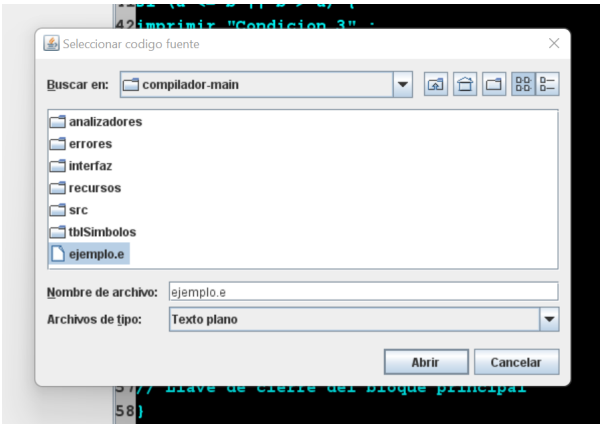
La última sección llamada “Acerca de” solo arroja una opción, la cual muestra una nueva ventana la cual contiene información del compilador y de sus creadores.

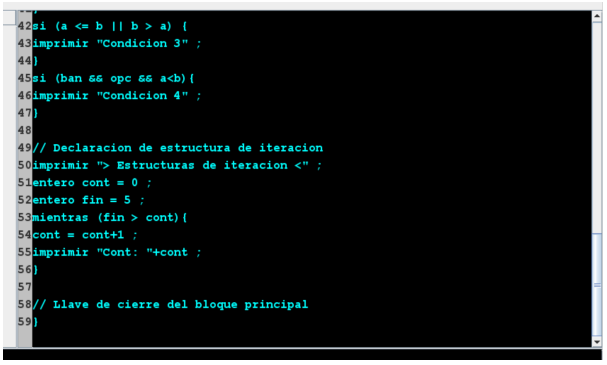




Luego se encuentra la sección para seleccionar un código fuente, en otras palabras para abrir archivos de texto plano, esta sección cuenta con 2 botones, los cuales el primero sirve para abrir una ventana para seleccionar el archivo a abrir y inmediatamente desplegar su contenido en el editor de texto y el segundo sirve para compilar el código que se encuentra en el editor de texto.







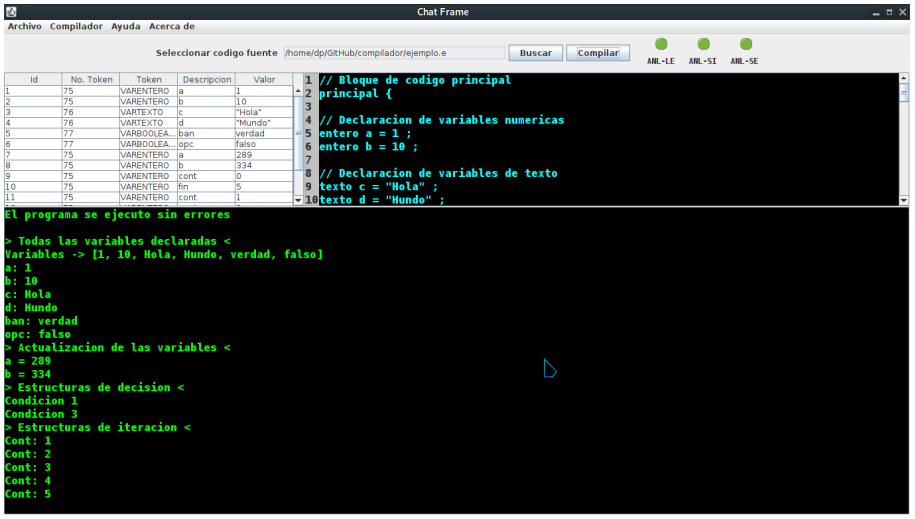
```
si (a <= b || b > a) {
imprimir "Condicion 3" ;
}
si (ban && opc && a<b){
imprimir "Condicion 4" ;
}

// Declaracion de estructura de iteracion
imprimir "> Estructuras de iteracion <" ;
entero cont = 0 ;
entero fin = 5 ;
mientras (fin > cont){
cont = cont+1 ;
imprimir "Cont: "+cont ;
}

// Llave de cierre del bloque principal
}
```

Al presionar el botón “Compilar” se empezaran a realizar las fases de análisis, lo que cambia al presionar el botón es la tabla de símbolos, la consola y el semáforo.

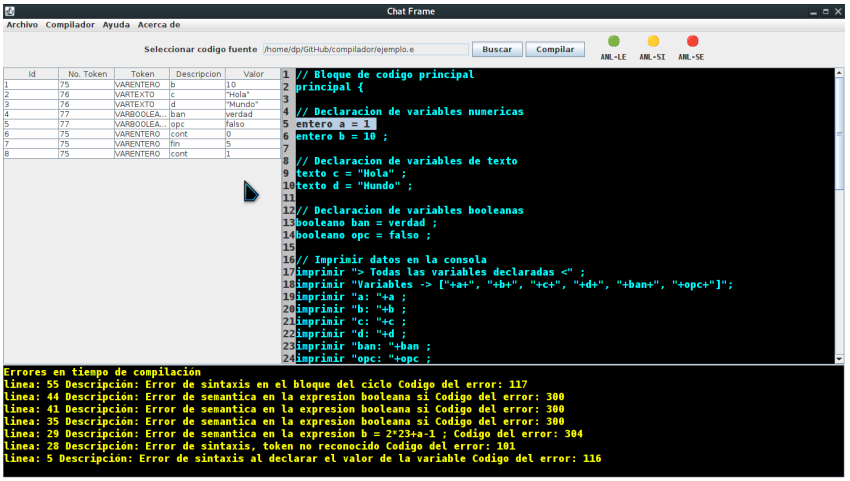
La tabla de símbolos será la encargada de guardar los tokens para así tener una descripción y el valor actual de cada uno de ellos, para poder visualizar qué variables hay en el código.

La consola muestra la información del código, si se ejecutó sin errores, en caso de que haya errores mostrar información sobre ellos y también los métodos para mostrar información de pantalla si es que el código cuenta con alguna instrucción para ello. Cabe aclarar que cuando no hay errores el color de la fuente en la consola es de color verde, mientras que cuando hay errores el color de la fuente se vuelve amarillo.

El semaforo lo unico que hace es mostrarnos el estado de cada una de las fases de analisis, cuenta con 3 estados, si esta en color verde es decir que se realizo correctamente, si esta en color amarillo es que se detuvo ahi porque hubo algun error y si esta en rojo es que todavia no se ha realizado.

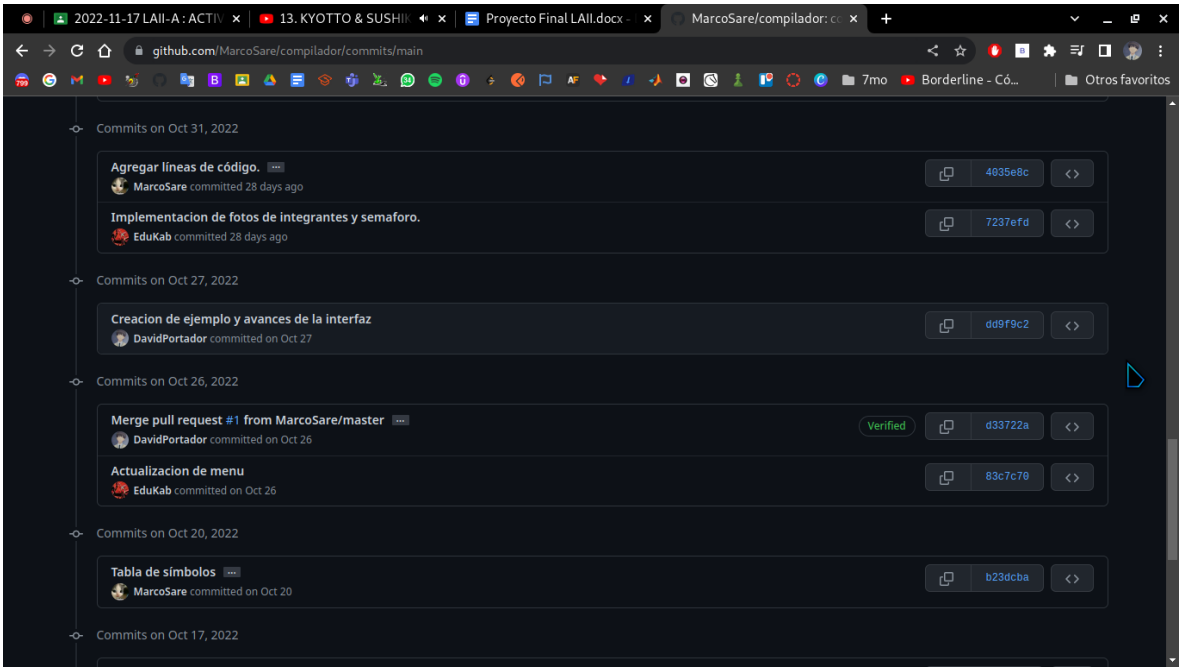


Un ejemplo de como es que se comporta el compilador cuando el codigo es compilado con errores, la consola se muestra con un color de fuente amarillo, especificando en donde se encuentra el error con una corta descripcion y el semaforo muestra los estados de cada una de las fases de analisis para identificar en que fase fallo el compilador.



Control de versiones

Repositorio administrado en github en el enlace:
https://github.com/MarcoSare/compilador/commits/main



MINUTAS.

Martes 11 de octubre de 2022

El profesor nos dijo que tuviéramos en cuenta la hora de entrada de las revisiones, pues sí bien no tenemos clase aún así debemos llegar a las 7:00.

Complemento nuestras gramáticas, haciendo que nuestro proyecto carece de gramáticas para la entrada y salida de datos.

Así como definir bien los delimitadores de inicio y fin de las instruccione

Jueves 13 de octubre de 2022.

El profesor nos retroalimenta sobre cómo podríamos usar la tabla de símbolo, así como orientarnos sobre cómo podríamos clasificar los tokens con valores numéricos, para posteriormente hacer más práctico el uso de los mismos.

Lunes 17 de octubre de 2022

Agregar columna de valor a la tabla de símbolos (Cuando es numérico, cadena o referencia a otra variable)

Interfaz con 3 areas (Edición, Tabla de símbolos y errores)

Barra de menú, abrir el archivo, editar el archivo, limpiar el archivo, ayuda (Acerca de(nombre, integrantes, versión, asignatura), ayuda(Abrir actividades de los PDF (TIPOS DE ANÁLISIS))

Trabajar en casos de uso (Que se hace en caso de algo).

Empezar documentación (Para cada autómata).

Ligar ligas a la interfaz

Jueves 20 de octubre de 2022

El profesor nos dijo que ya quería ver un pequeño programa en nuestro compilador. Nos guío un poco para empezar con el analizador sintáctico.

Jueves 27 de octubre de 2022

En la documentación poner que estamos utilizando un repositorio
Poner fotos de integrantes
Caso de uso
a = 5
b = 10
c = 5+10
Caso de uso
for con un 1 o 0 en condicion
Documentar(Comentarios también)/Implementar casos de uso
Implementar numero de linea
Implementar botones de fases de
análisis(Sintactico, etc)
Implementar la documentación en Github

Martes 01 de noviembre de 2022

Poner comentarios en cada líneas de casos de uso
Abarcar mucho más casos de usos
Llegar más temprano
Asistencia

Martes 08 de noviembre de 2022

1. Se observó que a nuestro proyecto estaría bien implementar la opción de comentar líneas de código y explicar qué hace cada programa que se vaya a presentar.
2. Implementación de errores y que se muestre en qué línea ocurrió el error.
3. Se mencionó del portafolios de evidencias digital.

Martes 15 de noviembre de 2022

Documentar casos de uso
Letra de la consola en amarillo
ÚLTIMA SEMANA DE NOVIEMBRE REVISION FINAL
Adjuntar documento de la gramática definida
.exe (Solo comentado)

Jueves 17 de noviembre de 2022

Quitar comentarios
Descripción de los errores al final de la compilación

Martes 22 de noviembre de 2022

Implementar imprimir valores o texto.

Actualizar PED

Escoger pc y laptop para el compilador (Requisitos mínimos), en comentario

Jueves 24 de noviembre de 2022

Video mínimo de 5 minutos, explicando cómo es que funciona el compilador, agregarlo al PED, explicar todos los ámbitos (Video documental)

OBSERVACIONES.

CONCLUSIONES.

Fue un proyecto bastante extenso ya que la cantidad de funcionalidades que ofrece un compilador es muy amplia lo cual hace muy complejo el proceso conforme se añaden nuevas funcionalidades que puede interpretar el compilador, estamos bastante conformes con nuestro desempeño ya que conseguimos crear una versión funcional de un compilador que funciona con el idioma español creado desde cero en en lenguaje de programación java con las librerías de interfaces de swing.

Link.

https://youtu.be/rUYUo80EmFE

BIBLIOGRAFÍA.

- Cidecame. (NA). 2.1. Función del Analizador Léxico. Recuperado el 10 de octubre de 2022. De: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/21_funcin_del_analizador_lxico.html

- Cidecame. (NA). 3.1 Función del Analizador Sintáctico. Recuperado el 10 de octubre de 2022. De: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/31_funcion_del_analizador_sintctico.html

- Cidecame. (NA). Unidad 4 Análisis Semántico. Recuperado el 10 de octubre de 2022. De: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/unidad_4_anlisis_semntico.html