Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Vacaciones de Junio de 2022
Catedrático: Ing. Mario Bautista
Auxiliar: Moises Gonzalez Fuentes

LFScript
Fase 2

Objetivos

Objetivo general

Implementar una aplicación donde se apliquen los conceptos adquiridos en el curso de Organización de Lenguajes y Compiladores 1.

Objetivos específicos

- Aplicar la fase de análisis léxico, sintáctico y semántico para la implementación de un intérprete.
- Construir un árbol de sintaxis abstracta (AST) para la ejecución de código de alto nivel.
- Conocer el patrón de diseño, intérprete.
- Detectar y reportar errores léxicos, sintácticos, semánticos y de ejecución.

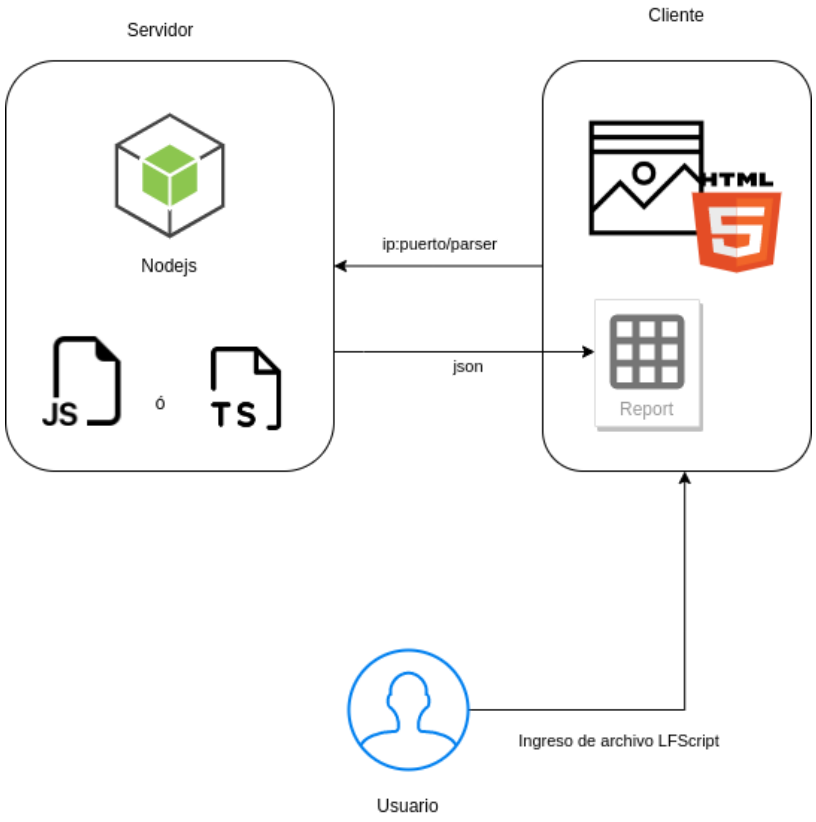
Descripción de la solución

El estudiante tiene que implementar un *intérprete* que ejecute instrucciones de alto nivel definidas en el lenguaje **LFScript**, el cual se definirá más adelante. Para la implementación se tiene que crear un *AST*, el cual servirá para la ejecución de las instrucciones del lenguaje.

Por medio de un servidor, se procesarán las solicitudes hechas al intérprete. Y las salidas serán visibles en una página web.

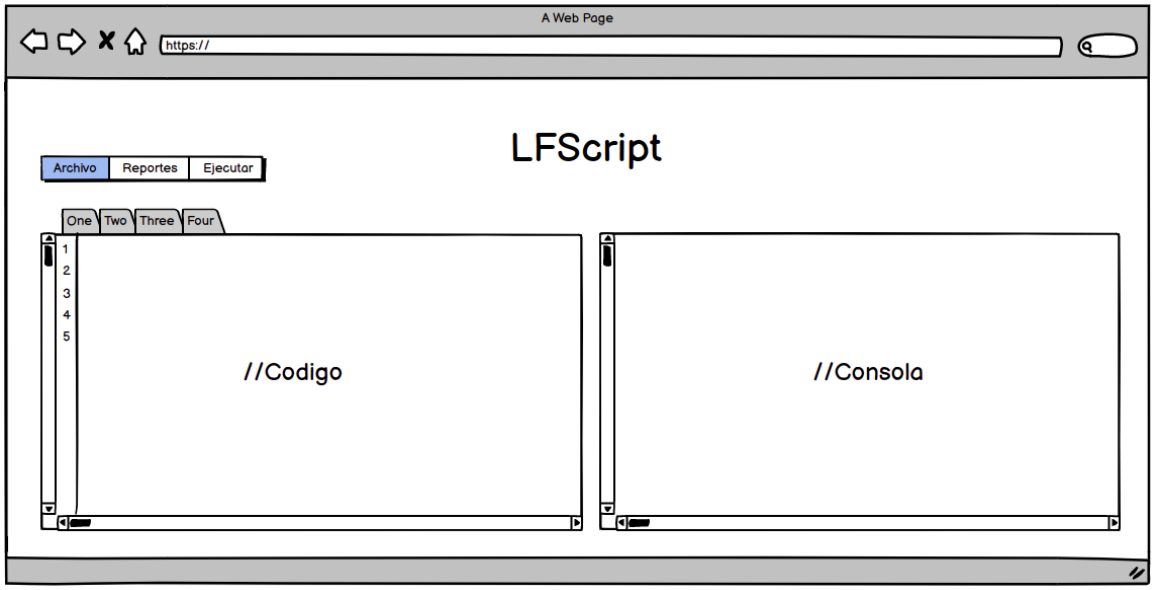
Flujo de la aplicación

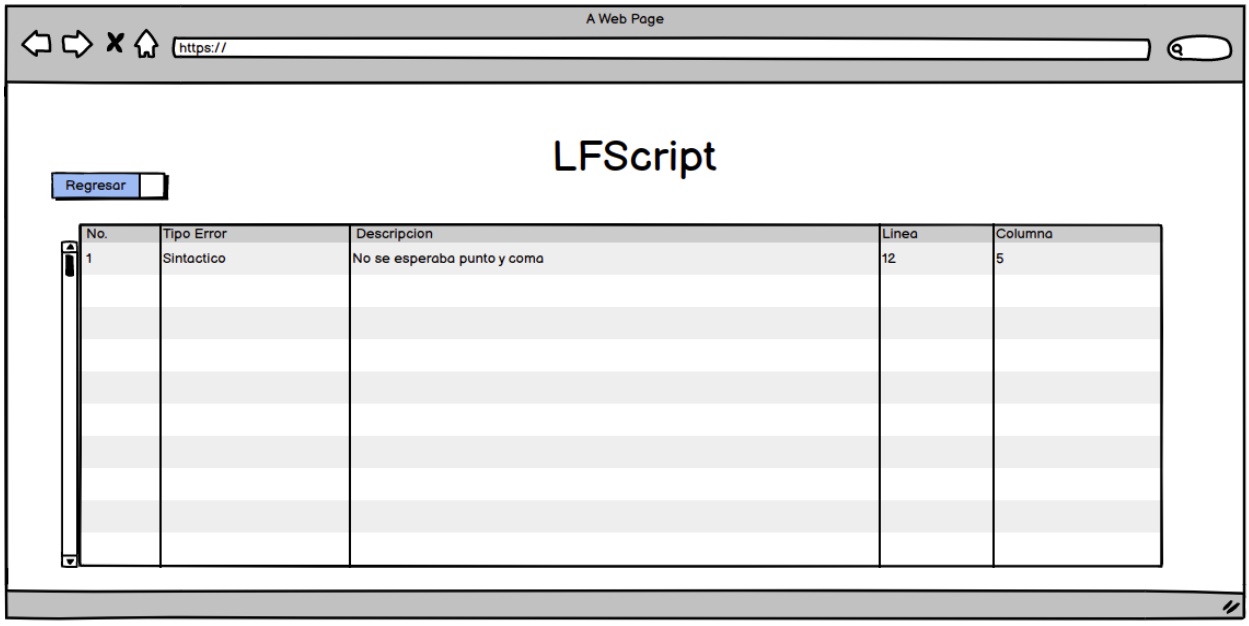
- El usuario abre la página web e ingresa el código fuente. O puede cargar el archivo con extensión **[.LF]**
- La página web hace la solicitud al servidor.
- El servidor analiza la solicitud y realiza el análisis léxico, sintáctico y semántico.
- El servidor retorna la información por medio de un JSON.
- El usuario observa las salidas de una forma amigable.



Interfaz gráfica

La aplicación deberá de contar con una interfaz gráfica amigable, la cual permitirá interactuar con el intérprete, manipular archivos de entrada y generar reportes, cada uno de sus elementos se describe a continuación:





Características del editor

- Tendrá múltiples pestañas.
- Debe de mostrar la fila y columna actual del cursor dentro de una pestaña.

Menú de archivos

- Abrir: permitirá cargar el contenido de un archivo a una pestaña dentro del editor.
- Guardar: guardará en un archivo la cadena de texto que se encuentre en la pestaña actual.
- Guardar como: guardará en un nuevo archivo la cadena de texto que se encuentre en la pestaña actual.
- Crear: creará una nueva pestaña dentro del editor, la pestaña no tendrá contenido.

Menú de ejecutar

- Ejecutar: se tomará la cadena que se encuentre en la pestaña actual y se hará la solicitud al servidor para que realice el proceso de interpretación.

Menú de reportes

- Reporte AST: en este reporte se visualiza una imagen con el AST que se genera al analizar la cadena de entrada de la pestaña actual con el código **LFScript**.
- Reporte de errores: en este reporte se mostrarán todos los errores que ocurran durante la interpretación. El reporte debe de ser generado en un archivo HTML en forma de tabla.
- Reporte TS: en este reporte se mostrarán todas las tablas de símbolos que se generaron con el código **LFScript.**

Área de salida

El editor debe de contar con una consola en la cual se mostrará la salida del código **LFScript**.

Definición del lenguaje

Archivos LFScript

Los archivos con código LFScript tendrá extensión **[.LF]**, el contenido de cada archivo se define a continuación:

- Puede venir la definición de una o más clases dentro del archivo.
- El código **LFScript** no será sensible a las mayúsculas.
- Los identificadores aceptaran números, pero siempre debe iniciar con una letra.

Comentarios

Comentario de una línea

Los comentarios de una línea iniciarán con "//" y terminarán con un salto de línea.

```
1
2
3   //este es un comentario de una linea
4
```

Comentarios d e varias líneas

Los comentarios de varias líneas iniciarán con "/*" y terminarán con "*/".

```
3   /*******************
4   ******** olc1 ******
5   *******************/
6
```

Tipos de datos

El lenguaje LFScript es un lenguaje con tipado, esto quiere decir que se conoce el tipo de dato de una variable desde el momento en que fue declarada. También tiene un tipado estático, lo que indica que una variable o expresión puede tener solo un tipo durante toda la ejecución.

Tipos de datos primitivos

Tipo de dato	Ejemplo
int	100, 024, 1, - 450, -5
double	0.2455, -6.245, -4.0
char	'a', 'f', ' ', '%'
boolean	true, false
String	"OLC1", "", "20", "- 4.45", "%%%#", "luisa", "{cadena}"
null	null

Caracteres de escape

Los caracteres de escape son caracteres que se definen de forma especial dentro de una

cadena de texto (String). A continuación, se definen los caracteres que se tienen que manejar.

Carácter de escape	Descripción
\”	Comillas dobles
****	Barra invertida
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal

Expresiones

Signos de agrupación

Los signos de agrupación se utilizarán para dar orden y cierta precedencia a las operaciones. Se utilizarán los paréntesis : (10 + 1) * 3

Literales

Los literales hacen referencia a las expresiones donde su tipo de dato es primitivo.

Literal	Ejemplo
Entero	100, - 42, 6, ...
Decimal	-0.5482, 5.1247, - 0.0, …
Booleano	true, false
Carácter	‘’, ‘(‘, “$’, …
Cadena	“Compiladores 1”, “”, “100”, “,#,”, …

Operador ternario

Consta de tres partes importantes: una expresión, y dos instrucciones o dos expresiones. Este operador puede ser de tipo instrucción o de tipo expresión. Se usará el símbolo ‘?’ después de la expresión a evaluar y el símbolo ‘:’ para separar las expresiones o instrucciones.

Nota: Cuando es una instrucción únicamente serán Instrucciones tipo Print, Asignación, Incremento, Decremento, Llamada a función o método.
A continuación un ejemplo de su implementación;

```
(5+5+9) ?  console.log("condicion verdadera"): console.log("condicion falsa");


int numero= (VarTime == 12)? 15 : 16 ;
String  nombre= ("compi" == "vacas")? "verdadera" : "falso" ;
```

Sentencia GraficarTs

Esta sentencia permitirá al usuario generar una foto del manejo de la tabla de símbolos en cierta parte del código, dentro del archivo de entrada se puede encontrar muchas sentencias de este tipo. Para lo cual se tiene que almacenar todas las gráficas y después mostrar al usuario por medio de la interfaz gráfica. A continuación un ejemplo de su implementación.

```
//declaraciones TS0
graficar_ts();
{
    //declaraciones TS1
    {
        //declaraciones TS2
        graficar_ts();
        graficar_ts();

    }
    graficar_ts();

}
graficar_ts();
```

Expresiones

1.ToLower

Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas.

```
string cad_1 = toLower("hOla MunDo"); // cad_1 = "hola mundo"
string cad_2 = toLower("RESULTADO = " + 100); // cad_2 = "resultado = 100"
```

2.ToUpper

Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras mayúsculas.

```
string cad_1 = toUpper("hOla MunDo"); // cad_1 = "HOLA MUNDO"
string cad_2 = toUpper("resultado = " + 100); // cad_2 = "RESULTADO = 100"
```

3.Round

Esta función recibe como parámetro únicamente tipo decimales. Permite redondear los números decimales según las siguientes reglas:
- Si el decimal es mayor o igual que 0.5, se aproxima al número superior
- Si el decimal es menor que 0.5, se aproxima al número inferior.

Nota: retorna una expresión de tipo int.

```
Double valor = round(5.8); //valor = 6
Double valor2 = round(5.4); //valor2 = 5
```

Vectores

Declaración

Los vectores son una estructura de datos de **tamaño variable** que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string. El lenguaje permitirá únicamente el uso de arreglos de una o dos dimensiones.

Observaciones:
• La posición de cada vector será N-1. Por ejemplo, si deseo acceder al primer valor de un vector debe acceder como miVector[0].

```
Int vector1[] = new int[4]; //se crea un vector de 4 posiciones,
                            //con 0 en cada posición, usar valores default

Char vectorDosd[][] = new char [4] [4] ; // se crea un vector de dos dimensiones de 4x4  [x]*[y]
```

Al momento de declarar un vector, tenemos dos tipos que son:

• Declaración tipo 1: En esta declaración, se indica por medio de una expresión numérica del tamaño que se desea el vector, además toma los valores por default para cada tipo.

• Declaración tipo 2: En esta declaración, se indica por medio de una lista de valores separados por coma, los valores que tendrá el vector, en este caso el tamaño del vector será el de la misma cantidad de valores de la lista.

```
char vectordosd2 [ ][ ] = [[ 0 ,0],[0 , 0]];
```

vector de dos dimensiones con valores de 0 en cada posición

Acceso a vectores

Para acceder al valor de una posición de un vector, se colocará el nombre del vector seguido de : [EXPRESION].

```
string vector2[ ] = ["hola", "Mundo"]; //creamos un vector de 2 posiciones de tipo string

string valorPosicion = vector2[0]; //posición 0, valorPosicion = "hola"

Char vectorDosd[ ][ ] = new char [4] [4] ; // creamos vector de 4x4

Char valor = vectorDosd[0][0]; // posición 0,0
```

Modificación de Vectores

Para modificar el valor de una posición de un vector, se debe colocar el nombre del vector seguido de '[' EXPRESION ']' = EXPRESION;

Observaciones:
A una posición de un vector se le puede asignar el valor de otra posición de otro vector o del mismo vector.

```
string vector2[ ] = ["hola", "Mundo"]; //vector de 2 posiciones, con "Hola" y "Mundo"

int vectorNumero[ ] = [2020,2021,2022];

vector2[0] = "OLC1 ";

vector2[1] = "1er Semestre "+vectorNumero[1];
```

Funciones nativas

1. Length

Esta función recibe como parámetro una cadena y devuelve el tamaño de dicha cadena como una expresión. También es posible enviar un vector y retorna el tamaño del vector, únicamente será para vectores de una dimensión.

```
string  vector2[] = ["hola", "Mundo"];

int tam_vector = length(vector2); // tam_vector = 2

int tam_hola = length(vector2[0]); // hola = 4
```

2. toCharArray

Esta función permite convertir una cadena en un vector de caracteres. A continuación un ejemplo de su implementación.

```
Char caracteres[] = toCharArray("Hola");

/*
caracteres [0] = "H"
caracteres [1] = "o"
caracteres [2] = "l"
caracteres [3] = "a"
*/
```

3. indexof

Esta función permite buscar un elemento dentro de un vector (1 dimensiones) y retornar la posición donde fue encontrado. El valor que retorna es de tipo numérico. En caso de no encontrarlo se retorna el valor -1

```
string  vector2[] = ["hola", "Mundo"];

console.log(vector2.indexof("hola")) // 0
console.log(vector2.indexof("Hola_")) // -1
console.log(vector2.indexof("Mundo")) // 1
```

Push

Esta función realiza una inserción rápida dentro de un vector de una dimensión. A continuación un ejemplo de su implementación. Esta funcionalidad es de tipo instrucción y expresión . Cuando se usa como una expresión retorna un tipo de dato booleano, que indica si realizó correctamente la operación.

Nota: La inserción se hará al final del vector.

```
string  vector2[] = ["hola", "Mundo"];

console.log(vector2.push("hola")) //true
console.log(vector2.push(3)) //false por el tipo de dato

vector2.push("olc");


/*
vector2[0]="hola"
vector2[1]="Mundo"
vector2[2]="olc"
*/
```

Pop

Esta operación elimina el último elemento de un vector. Únicamente se usará como una instrucción. A continuación un ejemplo de su implementación.

```
string  vector2[] = ["hola", "Mundo", "olc"];

vector2.pop();

/*
vector2[0]="hola"
vector2[1]="Mundo"
*/
```

Splice

Esta operación permite insertar un elemento dentro de un vector de una dimensión, dentro de una posición específica. El primer parámetro es una expresión que indique la posición a insertar y el segundo parámetro será una expresión con el valor a insertar.

```
string  vector2[] = ["hola", "Mundo", "olc"];

vector2.splice(1,"hi");

print(vector2);
/*
["hola","hi","Mundo", "olc"]
*/
```

Otras indicaciones:

Cuando se imprime un vector de una o dos dimensiones se tiene que mostrar visualmente todos los elementos de dicho vector, con el formato siguiente:

```
string  vector2[] = ["hola", "Mundo", "olc"];

print(vector2);

/*
["hola","Mundo"]
*/
```

Restricciones del proyecto

- La aplicación deberá de desarrollarse utilizando el lenguaje **JavaScript o TS**.
- Se debe respetar la arquitectura del proyecto (Flujo del proyecto).
- Las herramientas para realizar los analizadores será **Jison**.
- El proyecto se realizará de manera **individual** si se detecta algún tipo de copia el laboratorio quedará anulado.
- La calificación será sobre la interfaz gráfica.
- Utilizar controlador de versiones Git.
- Es una continuación de la fase 1, por lo tanto es necesario tener las instrucciones de la fase anterior.

Forma de entrega del proyecto

Entregables UEDI

- Código fuente de la aplicación.
- Archivo con la gramática utilizada en el proyecto.
- Link de repositorio remoto **privado** GitHub (El mismo de la fase 1)

Fecha de entrega

Jueves 30 de junio de 2022, hasta las 23:59