Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas
Organización de Lenguajes y Compiladores 2
Segundo semestre 2020
Catedráticos: Ing. Edgar Saban, Ing. Bayron López, Ing. Erick Navarro e Ing. Luis Espino
Tutores académicos: Pavel Vásquez, Erik Flores, Juan Carlos Maeda, Cristian Alvarado

MatrioshTS

Contenido

1. Competencias

1.1. Competencia general

Que el estudiante aplique la fase de síntesis del compilador para realizar un traductor e intérprete utilizando herramientas.

1.2. Competencia especifica

- Que el estudiante utilice un generador de analizadores léxicos y sintácticos para construir un traductor.
- Que el estudiante implemente la ejecución de dicha traducción utilizando traducción dirigida por la sintaxis haciendo uso de atributos heredados y sintetizados.
- Que el estudiante comprenda los conceptos acerca de traducciones.
- Que el estudiante maneje la pila que proporciona el analizador sintáctico para simular el paso de atributos heredados.

2. Descripción

MatrioshTS es un lenguaje de programación inspirado en Typescript, su característica principal es la inclusión de tipos explícitos.
El sistema de tipos de MatrioshTS realiza una formalización de los tipos de Javascript, mediante una representación estática de sus tipos dinámicos. Esto permite a los desarrolladores definir variables y funciones tipadas sin perder la esencia. Otra inclusión importante de MatrioshTS es la inclusión de funciones anidadas por lo que su traducción genera el mismo lenguaje solo que las funciones estarán desanidadas.

2.1. Componentes de la aplicación

Se requiere la implementación de un entorno de desarrollo que servirá para la creación de aplicaciones en lenguaje MatrioshTS.

2.1.1. Características básicas

- Numero de línea y columna
- Botón para traducir
- Botón para ejecutar
- Reporte de errores
- Reporte de tabla de símbolos
- Reporte de AST

2.1.2. Consola

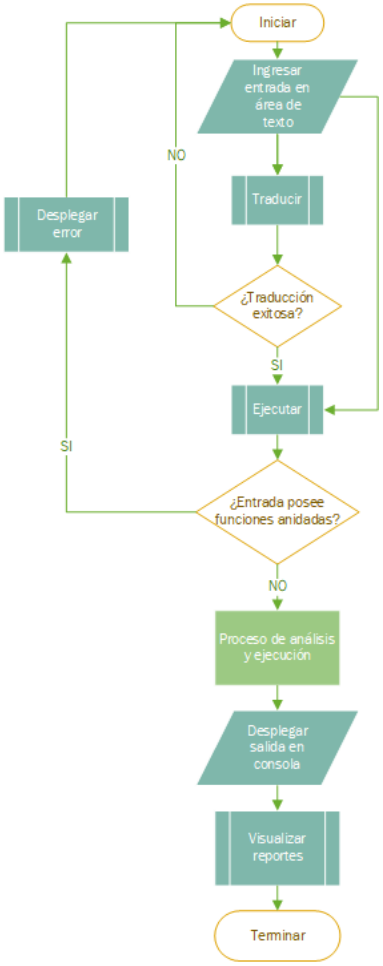
La consola es un área especial dentro del IDE que permite visualizar las notificaciones, errores, y advertencias que se produjeron durante el proceso de análisis de un archivo de entrada.

3. Flujo de la aplicación

A continuación, se explica el flujo de la aplicación.

La aplicación es sencilla por lo que su funcionalidad se basa en traducir, ejecutar y desplegar reportes.

- **Traducir:**
  Esta opción nos va a permitir traducir una entrada, cuando hablamos de traducción solamente nos referiremos a la parte de desanidar funciones, si la entrada no posee funciones anidadas entonces quedara exactamente como se ingresó, pero si tuviera funciones anidadas entonces la salida debería estar con las funciones desanidadas.
- **Ejecución:**
  Esta opción nos va a permitir ejecutar el código traducido, si el código contiene funciones anidadas entonces debería arrojar error por lo que debería pasar por el proceso de traducción.
- **Reportes:**
  Esta opción nos va a permitir visualizar los reportes generados después de traducir o ejecutar una entrada.



En la ejecución se debe implementar los dos métodos de recuperación: uno para los errores léxicos y sintácticos descartando hasta el “;”; y otro para los errores semánticos, se debe descartar la instrucción con error.

3.1. Flujo especifico de la aplicación

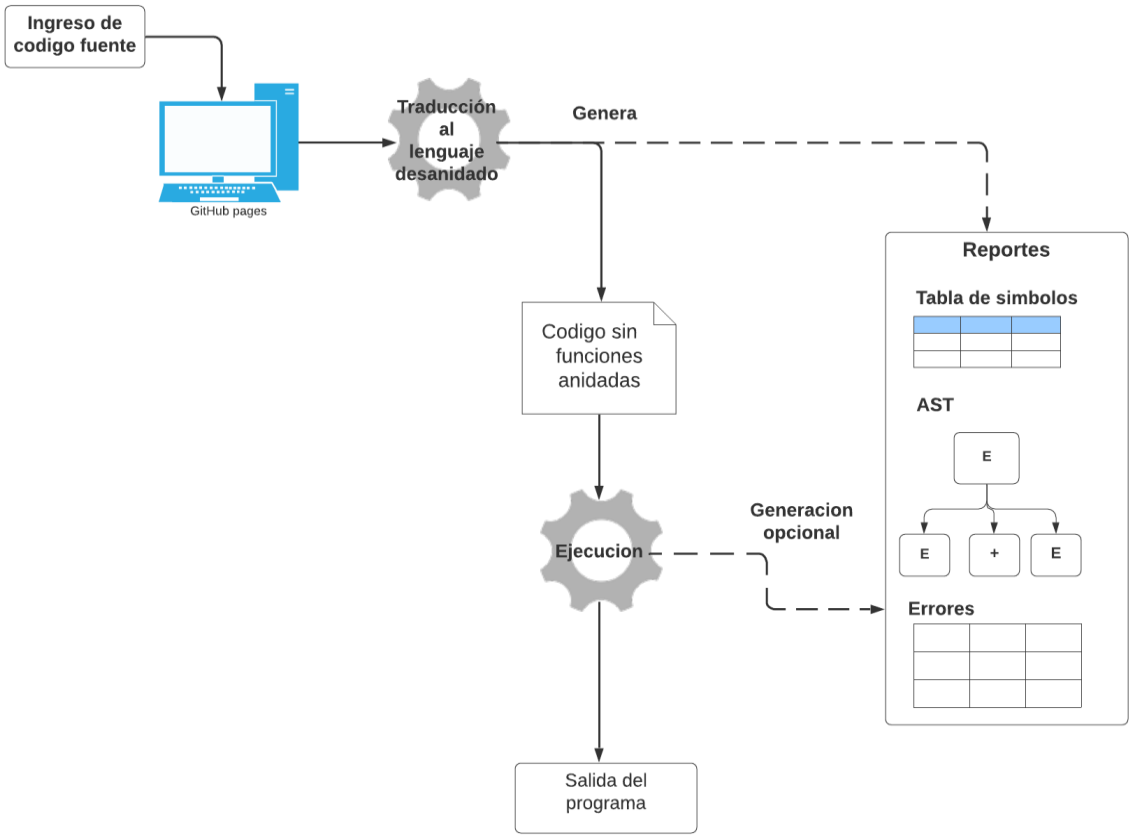


Ilustración 1. Flujo de la aplicación

3.1.1. Ingreso de código fuente

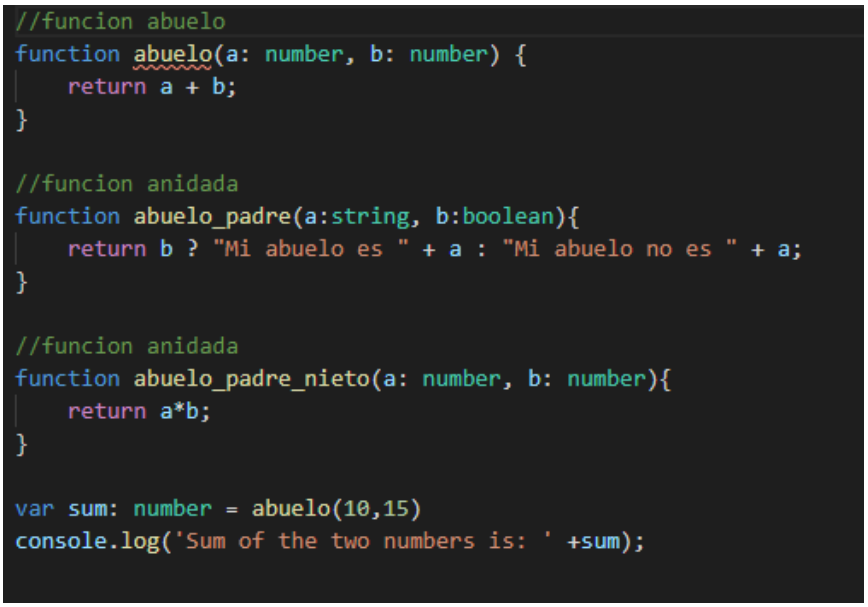
El lenguaje está basado en typescript con instrucciones limitadas además de poder definir funciones dentro de funciones, no hay límite para el nivel de anidamiento, se explica más a detalle en la sección 4.11 para las funciones anidadas y en la sección 4 se especifican las limitaciones del lenguaje.

```
//ejemplo funciones anidadas

//funcion abuelo
function abuelo(a: number, b: number) {

    //funcion anidada
    function padre(a:string, b:boolean){

        //funcion anidada
        function nieto(a: number, b: number){
            return a*b;
        }

        return b ? "Mi abuelo es " + a : "Mi abuelo no es " + a;
    }
    return a + b;
}
var sum: number = addNumbers(10,15)
console.log('Sum of the two numbers is: ' +sum);
```

Ilustración 2. Ejemplo de funciones anidadas

3.1.2. Traducción al lenguaje desanidado

La traducción se realizará sobre la entrada, se realiza el análisis léxico, sintáctico y semántico sobre la entrada y generará el mismo lenguaje basado en typescript, luego al final de la traducción posiblemente mostrara la lista de errores que se encontraron luego de los análisis, mostrara el reporte de la tabla de símbolos para asegurar un correcto análisis de la cadena de entrada, luego generara el lenguaje de salida que es con la misma sintaxis que el lenguaje de entrada, pero con las funciones desanidadas y con nombres diferentes para evitar conflictos en el programa.



```
//funcion abuelo
function abuelo(a: number, b: number) {
    return a + b;
}

//funcion anidada
function abuelo_padre(a:string, b:boolean){
    return b ? "Mi abuelo es " + a : "Mi abuelo no es " + a;
}

//funcion anidada
function abuelo_padre_nieto(a: number, b: number){
    return a*b;
}

var sum: number = abuelo(10,15)
console.log('Sum of the two numbers is: ' +sum);
```

Ilustración 3. Ejemplo de posible solución a las funciones anidadas

- Las restricciones sobre las funciones se describirán en la sección 4.11
- El nombre de las funciones desanidadas queda a discreción del estudiante, se recomienda nombrarlas a manera de que se pueda identificar cuáles son sus funciones padres.
- Se calificará el correcto funcionamiento de cada función anidada.
- Si existen errores en el lenguaje fuente, deberá recuperarse de estos errores y seguir traduciendo, el archivo de salida tendrá que contar con los mismos errores, la recuperación de errores se especifica en la sección.

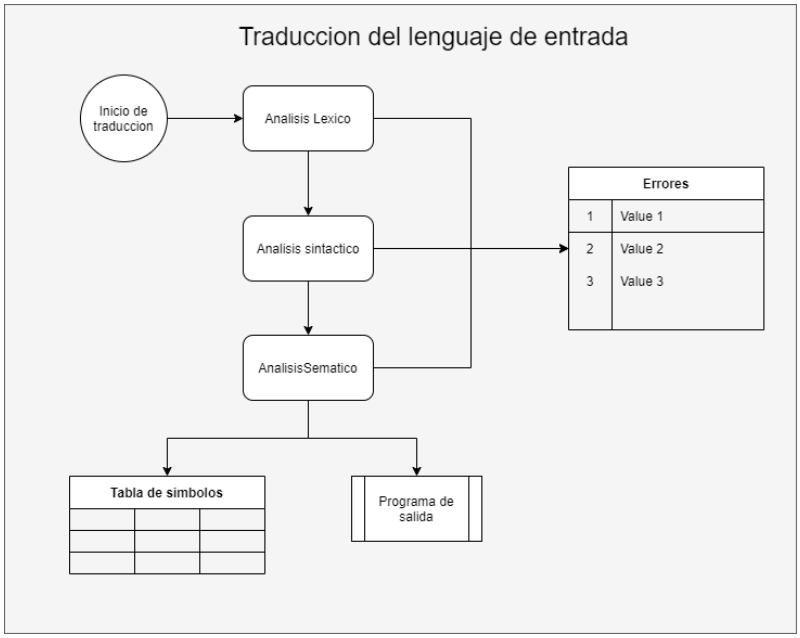


Ilustración 4. Proceso de traducción del lenguaje de entrada

- El reporte de tabla de símbolos será necesario para comprobar que se realizó el análisis léxico, sintáctico y semántico.
- El lenguaje de salida debe ser funcional al probarlo con el intérprete que el estudiante debe realizar, así se verificará que la traducción fue exitosa.

3.1.3. Ejecución

Ya con el lenguaje traducido se procederá a ejecutar el Código, si el código que se ejecuta cuenta con funciones anidadas, entonces se marcará como un error y se reportará, de lo contrario generará la salida en consola.

```
// 1. Code runs as you type: edit message
let msg = 'hola compiladores 2'
function abuelo(a, b) {
    return a + b;
}

//funcion anidada
function abuelo_padre(a, b) {
    return b ? "Mi abuelo es " + a : "Mi abuelo no es " + a;
}

//funcion anidada
function abuelo_padre_nieto(a, b) {
    return a * b;
}

var res = abuelo_padre("Compiladores 2", true);
console.log('Mi resultado es: ' + res);
```

```
CONSOLE
Mi resultado es: Mi abuelo es Compiladores 2
{
  myMessage: "hola compiladores 2"
}
```

Ilustración 5. Salida de código valido

3.1.4. Generación de reportes

Cuando el programa traduce el código fuente o ejecuta el código resultado, es posible generar los reportes de tabla de símbolos, ast y errores para la verificación de como el estudiante usa las estructuras internas para la interpretación del lenguaje

El reporte de errores generará los errores encontrados en el código, se especificará la información mínima que debe llevar el reporte en la sección 5.14

Tipo	Descripción	Línea	Columna
Sintáctico	Se esperaba "==", en lugar se encontró "="	112	15
Semántico	El tipo string no puede multiplicarse con un numeric	80	10
sintáctico	Se esperaba la palabra reservada "if" en su lugar se encontró “else”	1000	5

Tabla 1. Ejemplo de posible reporte de errores

El reporte de la tabla de símbolos mostrara una tabla de todos los registros que ingresaron a la tabla, la información a mostrar queda a discreción del estudiante, se explica más a fondo en la sección 5.12

Nombre	Tipo	Ámbito	Fila	Columna
Temp	string	global	5	8
A	number	local	10	7
B	number	local	12	5
C	boolean	local	13	10

Tabla 2. Ejemplo de posible reporte de tabla de símbolos

El reporte de AST mostrara el árbol de análisis sintáctico del código que se ejecuta o traduce, se especifica su funcionamiento en la sección 5.13

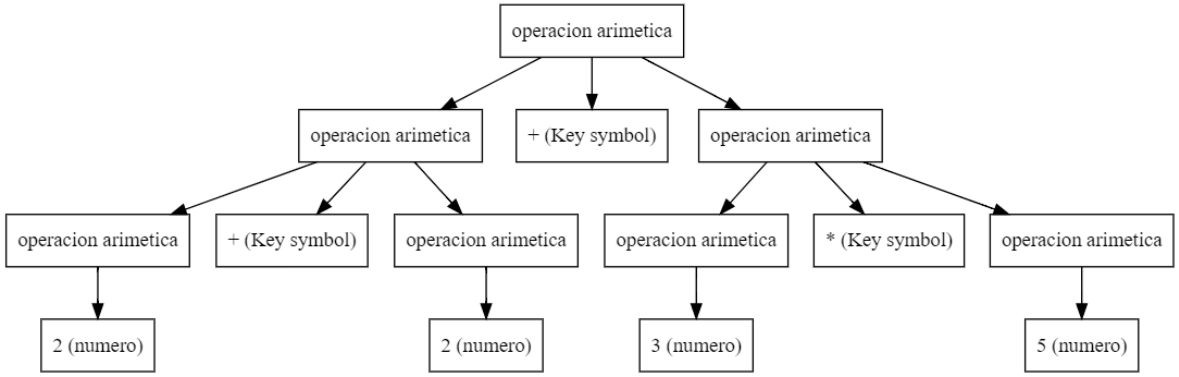


Ilustración 6. Ejemplo de posible salida del reporte AST

4. Sintaxis de MatrioshTS

MatrioshTS nos provee la posibilidad de tipado estático, funciones anidadas y que su traducción sea exactamente la misma a excepción de las funciones, ya que las funciones estarán desanidadas luego de la traducción inicial.

La sintaxis de MatrioshTS es similar a typescript, pero con la salvedad que no se utilizaran todas las funcionalidades que este lenguaje posee, a continuación, se describen las instrucciones válidas.

En el siguiente enlace se encuentra de forma más detallada la sintaxis y ejemplos: https://www.tutorialsteacher.com/typescript/

4.1. Generalidades

- El lenguaje es case sensitive, por lo tanto, un identificador declarado como PRUEBA es diferente a uno declarado como prueba, esto también aplica para las palabras reservadas.
- Los comentarios pueden ser de una línea (//) y de múltiples líneas (/* … */)
- Secuencias de escape

Secuencia	Descripción
\”	Comilla doble
****	Barra invertida
\n	Salto de línea
\r	Retorno de carro
\t	Tabulación

4.2. Tipos de dato validos

4.2.1. String:

Se pueden utilizar comillas simples y dobles, así también la sintaxis de plantillas de cadena de texto.

4.2.2. Number:

Pueden ser números entero o decimales, positivos o negativos.

4.2.3. Boolean:

Valores true y false.

4.2.4. Void:

Para funciones y variables de tipo void (para asignar funciones de tipo void)

4.2.5. Types

Estos pueden contener cualquier tipo de dato en su interior, incluso otros types, arreglos o arreglos de types.

Consideraciones:

- Cuando se inicializa el type es obligatorio inicializar sus atributos.
- Si un atributo del type contiene un tipo de dato definido por el usuario, este podrá tener el valor de null.

4.2.6. Arrays

Los arreglos pueden ser de cualquier tipo de dato valido (Incluso arreglos o types) y son dinámicos, además poseen las siguientes propiedades:

- **Push:** Sirve para agregar un nuevo valor al arreglo.
- **Pop:** Saca el ultimo valor del arreglo y lo devuelve.
- **Length:** Devuelve el tamaño del arreglo.

4.3. Declaraciones de variables

La sintaxis de la declaración puede realizarse utilizando ya sea:

- **let:** Este tipo de declaración proporciona accesibilidad únicamente en el ámbito donde se declaró y en ámbitos hijos, por lo tanto no se podrá utilizar en ningún ámbito externo.
- **const:** Funciona igual que let, pero con la diferencia de que no permite cambiar el valor de la variable en ningún momento.

Consideraciones:

- Las constantes es obligatorio que se declaren con un valor.
- Si el tipo de dato de una variable no se especifica se toma el tipo de dato que se obtenga de la expresión.
- No puede declararse una variable o constante con un identificador que ya haya sido utilizado dentro del mismo ámbito.
- Todas las variables deben estar inicializadas para poder ser operadas caso contrario arrojar error.
- Las variables solo aceptan un tipo de dato, el cual no puede cambiarse en tiempo de ejecución.
- En Typescript se puede especificar uno o más tipos de dato que soporta una variable, MatrioshTS lo limita a un solo tipo de dato.
- Se debe validar que el tipo de dato y el valor sean compatibles.

4.4. Asignación de variables

Las variables no pueden cambiar de tipo de dato, se deben mantener con el tipo declarado inicialmente a menos que se hayan inicializado sin un tipo de dato.

Consideraciones

- No es posible cambiar el tipo de dato de una variable, a menos que el tipo de dato no se haya especificado al declarar la variable.
- Una constante no puede ser asignada.
- Se debe validar que el tipo de la variable y el valor sean compatibles.

4.5. Operaciones aritméticas

Entre las operaciones disponibles vamos a encontrar las siguientes

- Suma
- Resta
- Multiplicación
- División
- Negación
- Exponenciación
- Modulo
- Incremento
- Decremento

4.6. Operaciones relacionales

- Mayor que
- Menor que

- Mayor o igual que
- Menor o igual que
- Igualdad
- Diferenciación

4.7. Operaciones lógicas

Las operaciones lógicas se evalúan de izquierda a derecha, por lo tanto, se manejará corto circuito para no evaluar todas las condiciones si no es necesario.

- AND
- OR
- NOT

4.8. Operador ternario

El operador condicional (ternario) es el único operador que tiene tres operandos. Este operador se usa con frecuencia como atajo para la instrucción if.

4.9. Estructuras de control

Las estructuras de control permiten regular el flujo de la ejecución del programa. Este flujo de ejecución se puede controlar mediante sentencias condicionales que realicen ramificaciones e iteraciones. El lenguaje soporta las siguientes estructuras:

- If ... Else
- Switch – Case
- While
- Do While
- For, For … in y For … of

4.10. Sentencias de transferencia

Las sentencias de transferencia permiten manipular el comportamiento normal de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias:

- Break
- Continue
- Return con expresión
- Return sin expresión

Consideraciones

- Se debe verificar que la sentencia break aparezca dentro de un ciclo o dentro de una sentencia Switch – Case.
- Se debe verificar que la sentencia continue aparezca dentro de un ciclo.
- Se debe verificar que la sentencia return aparezca dentro de una función.

4.11. Funciones

MatrioshTS como cualquier otro lenguaje permite el uso de funciones, estas pueden o no retornar un valor, la característica especial y distintivo con otros lenguajes es que nos permite anidar funciones, por lo que el estudiante deberá aplicar conceptos de atributos heredados para realizar el des anidamiento de las mismas, protegiendo siempre el acceso de las variables en su ámbito correspondiente.

A continuación, se muestra un ejemplo de la traducción de funciones del lenguaje fuente al lenguaje destino:

```
// Ejemplo entrada de una funcion adentro de una funcion
function fun1() : void{
  function fun2() : number{
    return x * 10; //La funcion 2 tiene acceso a la variable x
  }
  let x = 100;
  let y = fun2();
  console.log(y);
}

//Ejemplo traduccion de MatrioshTS

function fun1() : void {
  let x = 100;
  let y = fun1_fun2();
  console.log(y);
}

// Se desanida la función, un buena práctica es agregar el prefijo de la función padre.
function fun1_fun2() : number {
  return x * 100; //La funcion hijo sigue teniendo acceso a la variable x
}
```

Consideraciones:

- ***Es obligatorio que la gramática de las funciones este expresada para un análisis descendente, por lo tanto es obligatorio el uso de atributos heredados o en su defecto utilizar la pila del analizador sintáctico para obtener los valores anteriores.***
- ***La traducción para funciones debe de deshacer las funciones anidadas y colocarlas en otra parte del código, ya que la ejecución no debería permitir tal anidamiento.***
- Las funciones no soportan sobrecarga.
- Las funciones en matrioshTS se pueden anidar cualquier cantidad de veces
- Las funciones anidadas pueden invocar otras funciones GLOBALES y locales siempre y cuando estén en el mismo ámbito.
- Los parámetros deben tener distinto nombre.
- Las funciones pueden retornar cualquier tipo de dato y este debe estar especificado.
- Las funciones tipo flecha no son soportadas por MatrioshTS.

4.12. Funciones nativas

4.12.1. console.log

Esta función nos permite imprimir cualquier expresión que le mandemos como parámetro, y en un formato comprensible.

4.12.2. graficar_ts

Esta función nos va a permitir graficar la tabla de símbolos en cualquier parte del programa donde se encuentre esta instrucción, esto servirá para validar el correcto manejo de los ámbitos.

5. Reportes generales

5.12. Tabla de símbolos

Este reporte mostrará la tabla de símbolos durante (utilizando la función graficar_ts) y después de la ejecución del archivo. Se deberán de mostrar todas las variables, funciones y procedimientos que fueron declarados, así como su tipo y toda la información que el estudiante considere necesaria.

5.13. AST

Este reporte mostrara el árbol de análisis sintáctico que se produjo al analizar el archivo de entrada. Este debe de representarse como un grafo, se recomienda se utilizar Graphviz. El Estudiante deberá mostrar los nodos que considere necesarios y se realizarán preguntas al momento de la calificación para que explique su funcionamiento.

5.14. Reporte de errores

El traductor e intérprete deberá ser capaz de detectar todos los errores que se encuentren durante el proceso de traducción. Todos los errores se deberán de recolectar y se mostrará un reporte de errores en el que, como mínimo, debe mostrarse el tipo de error, su ubicación y una breve descripción de por qué se produjo.

Los tipos de errores se deberán de manejar son los siguientes:

- Errores léxicos.
- Errores sintácticos.
- Errores semánticos

La tabla de errores debe contener la siguiente información:

- Línea: Número de línea donde se encuentra el error.
- Columna: Número de columna donde se encuentra el error.
- Tipo de error: Identifica el tipo de error encontrado. Este puede ser léxico, sintáctico o semántico.
- Descripción del error: Dar una explicación concisa de por qué se generó el error.
- Ámbito: Si fue en una función(decir en cual fue) o en el ámbito global.

Consideraciones

Se deben reportar errores tanto en tiempo de traducción como tiempo de ejecución. Queda a discreción del estudiante si presentarlos en reportes separados o en un mismo reporte especificando en qué fase ocurrió el mismo.

6. Entregables y calificación

Para el desarrollo del proyecto se deberá utilizar un repositorio de github, este repositorio deberá ser privado.

6.12. Entregables

Todo el código fuente se va a manejar en github, por lo tanto, el estudiante es el único responsable de mantener actualizado dicho repositorio hasta la fecha de entrega, ya que, si se agregan más archivos luego de la fecha y hora establecidos, entonces no se tendrá derecho a calificación.

- Código fuente publicado en un repositorio de github.
- Enlace al repositorio y permisos a los auxiliares para poder acceder.
- Aplicación web con la funcionalidad del proyecto publicada en github-pages.

6.13. Restricciones

- La herramienta para generar los analizadores del proyecto será JISON. La documentación se encuentra en el siguiente enlace http://zaa.ch/jison/docs/
- Copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ciencias y Sistemas los involucrados.
- El resultado final del proyecto debe ser una aplicación web funcionando en github pages, no será permitido descargar el repositorio y calificar localmente.
- El desarrollo y entrega del proyecto es individual.

6.14. Consideraciones

- Durante la calificación se realizarán preguntas sobre el código para verificar la autoría de este, de no responder correctamente la mayoría de las preguntas se reportará la copia.
- Es válido el uso de cualquier framework para el desarrollo de aplicaciones con javascript siempre y cuando la aplicación final pueda ser publicada en github pages (Se recomienda react, angular o ionic para facilitar el desarrollo).
- Los auxiliares de cada sección deben explicar el proceso de publicación en github pages.
- El repositorio únicamente debe contener el código fuente empleado para el desarrollo, no deben existir archivos pdf o docx.
- El sistema operativo a utilizar es libre.
- Los lenguajes están basados en Typescript y Javascript, por lo que el estudiante es libre de realizar archivos de prueba en estas herramientas, el funcionamiento debería ser el mismo y limitado a lo descrito en este enunciado.
- Typescript online https://www.typescriptlang.org/play
- Se van a publicar archivos de prueba en el siguiente repositorio: https://github.com/PvasquezF/ArchivosPruebaOLC2

6.15. Calificación

- La calificación del proyecto será mediante la aplicación web publicada en github pages.
- Se tendrá un máximo de 20 minutos por estudiante para calificar el proyecto.
- La hoja de calificación describe cada aspecto a calificar, por lo tanto, si la funcionalidad a calificar falla en la sección indicada se tendrá 0 puntos en esa funcionalidad y esa nota no podrá cambiar si dicha funcionalidad funciona en otra sección.
- Los archivos de entrada podrán ser modificados solamente antes de iniciar la calificación eliminando funcionalidades que el estudiante indique que no desarrollo.

- Los archivos de entrada podrán ser modificados si contienen errores léxicos, sintácticos o semánticos no descritos en el enunciado o provocados para verificar el manejo y recuperación de errores.

6.16. Entrega del proyecto

- La entrega será mediante github, y se va a tomar como entrega el código fuente publicado en el repositorio a la fecha y hora establecidos.
- Cualquier commit luego de la fecha y hora establecidos invalidará el proyecto, por lo que no se tendrá derecho a calificación.
- No habrá prorroga
- Fecha de entrega:

Domingo 27 de septiembre a las 23:59 PM