Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 1

Vacaciones Primer Semestre 2025

Catedrático:

Ing. Mario Bautista

Tutores académicos:

Cristian Axpuac

OBJ-C

Proyecto Fase I

Tabla de contenido

1. Objetivo General

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador y su importancia dentro del contexto de lenguajes de programación.

2. Objetivos Específicos

- Que el estudiante aprenda a generar analizadores léxicos y sintácticos utilizando la librería PLY.
- Que el estudiante aprenda los conceptos de token, lexema, patrones y expresiones regulares.
- Que el estudiante pueda realizar correctamente el manejo de errores léxicos.
- Que el estudiante sea capaz de realizar acciones gramaticales utilizando el lenguaje de programación Python.

3. Descripción General

Este proyecto tiene como propósito principal guiar al estudiante en el diseño e implementación de un lenguaje de programación propio, partiendo desde sus fundamentos léxicos y sintácticos. A lo largo del desarrollo, el estudiante aplicará conocimientos adquiridos sobre teorías de lenguajes formales y autómatas, así como técnicas de análisis utilizadas en compiladores modernos. Se implementarán componentes esenciales como el analizador léxico y el analizador sintáctico, integrando herramientas de programación en Python, particularmente mediante la utilización de la librería PLY (Python Lex-Yacc).

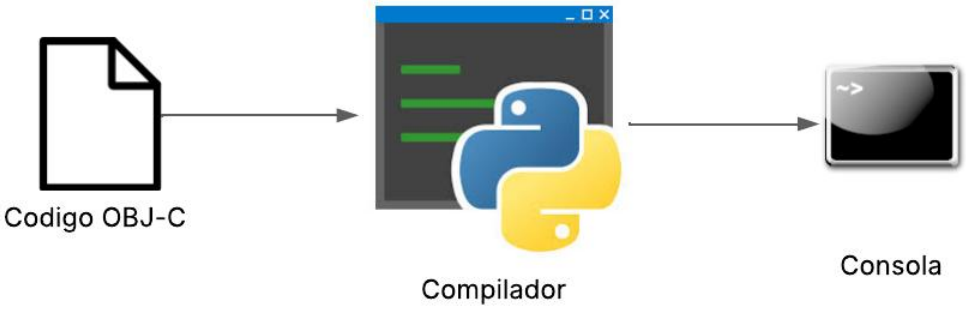
El enfoque estará centrado en construir una gramática robusta y clara, capaz de interpretar correctamente estructuras propias del lenguaje, permitiendo definir variables, controlar flujo, realizar operaciones aritméticas y lógicas, entre otras funcionalidades típicas de lenguajes de alto nivel. Adicionalmente, se abordará el manejo de errores en tiempo de análisis, promoviendo buenas prácticas de detección y recuperación de errores.

4. Entorno de Trabajo

Editor

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso del código fuente que será analizado. Queda a discreción del estudiante el diseño.

Flujo de la aplicación



Funcionalidades

- **Cargar Archivo:** El editor permitirá al usuario cargar un archivo con código de obj-c, la extensión permitida es “.objc”. En caso de que el editor contenga texto se deberá de eliminar, al final del proceso el editor únicamente contendrá el código del archivo cargado.
- **Interpretar:** Se enviará una petición al backend en Python usando como cuerpo de la petición el contenido actual del editor. Cuando se reciba la respuesta del backend se colocará la respuesta en la consola. Ademas se debe incluir un encabezado en la respuesta indicando el numero de errores encontrados durante la interpretación del código.

Reportes

- **Reporte de Errores**: se mostrarán todos los errores encontrados al realizar el análisis léxico, sintáctico y semántico.
- **Generar AST:** se debe generar el AST del código en el editor.
- **Reporte de Tabla de Símbolos**: se mostrarán todas las variables, métodos y funciones que han sido declaradas dentro del flujo del programa.

5. Descripción del lenguaje

Case Insensitive

El lenguaje es case insensitive por lo que no reconoce entre mayúsculas y minúsculas.

```
INT a = 1 + 5 ;

Int A = 1 + 5 ;
```

Nota: Ambos casos son lo mismo.

Comentarios

Comentarios de una línea

Este comentario comenzara con // y deberá terminar con un salto de línea.

```
// Esto es un comentario de una línea

Esto ya no es parte del comentario
```

Comentarios multilinea

Este comentario comenzara con /* y deberá terminar con */.

```
/* Esto es

  Un Comentario Multilinea */

Esto ya no es parte del comentario
```

Tipos de datos

El lenguaje Obj-C es fuertemente tipado, esto quiere decir que se conoce el tipo de una variable desde el momento en que es declarada. También se tiene tipado estático, lo que indica que una variable puede tener solo un valor durante toda la ejecución.

Tipo	Definición	Descripción	Ejemplo	Observaciones	Valor por defecto
Entero	INT	Este tipo de dato aceptará solamente números enteros	1, 50, 100, -120	Del -2,147,483,648 al 2,147,483,647	0
Decimal	FLOAT	Admite valores numéricos con decimales	1.2, 50.23, -0.34	Se maneja cualquier cantidad de decimales	0
Booleano	BOOL	Admite valores que indican verdadero o falso	true, false	Si se asigna un valor booleano a un entero se tomará como 1 (true) o 0 (false)	true
Carácter	CHAR	Tipo de dato que únicamente aceptará un único carácter, delimitado por comillas simples	‘a’, ‘b’, ‘E’, ‘1’, ‘&’, ‘\n’	Para escribir comilla simple: \' — Para \: \\. También se permiten \n, \t, \r, \"	‘\u0000’ (carácter nulo)

Cadena	STR	Conjunto de caracteres delimitado por comillas dobles	“cadena”, “-cad”	Se permite cualquier carácter entre comillas dobles, incluidas secuencias de escape como \n, \". También puede ser una cadena vacía "".	"" (string vacío)

Secuencias de escape

Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto. Las secuencias de escape disponibles son las siguientes:

Secuencia	Descripción	Ejemplo
\n	Salto de línea	"Hola\nMundo"
\\	Barra invertida	"C:\\miCarpeta"
\"	Comilla doble	"\"esto es una cadena\""
\t	Tabulación	"\tEsto es una tabulación"
\'	Comilla simple	"\'Estas son comillas simples\'"

Operadores aritméticos

Suma

Es la operación aritmética que consiste en realizar la suma entre dos omás valores. Para esta se utiliza el signo más (+)

Especificaciones de la operación suma

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

+	Entero	Decimal	Booleano	Carácter	Cadena
Entero	Entero	Decimal	No válido	Entero	Cadena
Decimal	Decimal	Decimal	No válido	Decimal	Cadena
Booleano	No válido	No válido	No válido	No válido	Cadena
Carácter	Entero	Decimal	No válido	Cadena	Cadena
Cadena	Cadena	Cadena	Cadena	Cadena	Cadena

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Resta

Es la operación aritmética que consiste en realizar la resta entre dos o más valores. Para esta se utiliza el signo menos (-).

Especificaciones de la operación resta

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-	Entero	Decimal	Carácter
Entero	Entero	Decimal	Entero
Decimal	Decimal	Decimal	Decimal
Carácter	Entero	Decimal	No válido

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Multiplicación

Es la operación aritmética que consiste en sumar un número(multiplicando) tantas veces como indica otro número (multiplicador). El signo para representar la operación es el asterisco (*).

Especificaciones de la operación multiplicación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

*	Entero	Decimal	Carácter
Entero	Entero	Decimal	Entero
Decimal	Decimal	Decimal	Decimal
Carácter	Entero	Decimal	No válido

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

División

Es la operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal (/).

Especificaciones de la operación división

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

/	Entero	Decimal	Carácter
Entero	Decimal	Decimal	Decimal
Decimal	Decimal	Decimal	Decimal
Carácter	Decimal	Decimal	No válido

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Potencia

Es una operación aritmética de la forma a**b donde a es el valor de la base y b es el valor del exponente que nos indicará cuantas veces queremos multiplicar el mismo número.

Especificaciones de la operación potencia

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

**	Entero	Decimal
Entero	Entero	Decimal
Decimal	Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Módulo

Es una operación aritmética que obtiene el resto de la división de un número entre otro. Para realizar la operación se utilizará el signo (%).

**	Entero	Decimal
Entero	Entero	Decimal
Decimal	Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Especificaciones de la operación módulo

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

%	Entero	Doble
Entero	Decimal	Decimal
Decimal	Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Negación Unaria

Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original. Se utiliza el símbolo menos (-).

Especificaciones de la operación negación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-exp	Resultado
Entero	Entero
Decimal	Decimal

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Operadores Relacionales

Son los símbolos que tienen como finalidad comparar expresiones, dando como resultado valores booleanos. A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

Operador	Descripción	Ejemplo
==	Igualación: compara ambos valores y verifica si son iguales	1 == 125.654 == 54.34
!=	Diferenciación: compara ambos lados y verifica si son distintos	1 != 250 != 30
<	Menor que: verifica si el derecho es mayor que el izquierdo	5.5 < 3050 < 'F'
<=	Menor o igual que: verifica si el derecho es mayor o igual que el izquierdo	5.5 <= 3050 <= 'F'
>	Mayor que: verifica si el izquierdo es mayor que el derecho	5.5 > 3050 > 'F'
>=	Mayor o igual que: verifica si el izquierdo es mayor o igual que el derecho	5.5 >= 3050 >= 'F'

Especificaciones de los operadores relacionales

A continuación, se especifica en una tabla los resultados que se deberán obtener con estas operaciones.

Relacional	Entero	Decimal	Booleano	Carácter	Cadena
Entero	Booleano	Booleano	No válido	Booleano	No válido
Decimal	Booleano	Booleano	No válido	Booleano	No válido
Booleano	No válido	No válido	Booleano	No válido	No válido
Carácter	Booleano	Booleano	No válido	Booleano	No válido
Cadena	No válido	No válido	No válido	No válido	Booleano

Nota: Cualquier otra combinación no especificada en esta tabla es invalida y será considerado un error de tipo semántico.

Operadores Lógicos

Son los símbolos que tienen como finalidad comparar expresiones a nivel lógico (verdadero o falso). A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

Operador	Descripción	Ejemplo
||	**OR**: Compara expresiones lógicas y si al menos una es verdadera, entonces devuelve verdadero, en otro caso retorna falso	true || 5 < 2 (Devuelve true)
&&	**AND**: Compara expresiones lógicas y si ambas son verdaderas, entonces devuelve verdadero, en otro caso retorna falso	true && "hola" == "hola" (Devuelve true)
^	**XOR**: Compara expresiones lógicas y si ambas no son iguales retorna verdadero, en cualquier otro caso retorna falso	true ^ false (Devuelve true)

!	**NOT**: Devuelve el valor inverso de una expresión lógica. Si esta es verdadera, entonces devolverá falso; de lo contrario, retorna verdadero	!true (Devuelve false)

Signos de Agrupación

Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por (y).

```
Int A = ( 5 + 5 ) * 4 ;
```

Precedencia de Operaciones

La precedencia de operadores nos indica la importancia de que una operación debe realizarse por encima del resto. A continuación, se define la misma:

Nivel	Operador	Asociatividad
0	- (unario)	Derecha
1	**	No asociativa
2	*, /	Izquierda
3	+, -	Izquierda
4	==, !=, <, <=, >, >=	Izquierda
5	!	Derecha
6	^	Izquierda
7	&&	Izquierda
8	`	

Nota: el nivel 0 es el nivel de mayor importancia.

Caracteres de finalización y encapsulamiento de sentencias

El lenguaje se verá restringido por dos reglas que ayudan a finalizar una instrucción y encapsular sentencias:

Finalización de instrucciones

Para finalizar una instrucción se utilizará el signo ;

```
Int a = 1 ;   // El signo ; finaliza la declaracion
```

Encapsular sentencias: para encapsular sentencias dadas por ciclos, métodos, funciones, etc, se utilizará los signos { y }.

```
if (true) {                    // Inicio de encapsulamiento “{”
        int edad = 18;
}                              // Fin de encapsulaiento “}”
```

Declaración de variables

Las variables deben ser declaradas antes de su uso, especificando un nombre de identificador, un tipo de dato y, opcionalmente, un valor inicial. Las variables pueden declararse tanto a nivel global como local.

Si no se asigna un valor inicial, la variable tomará el valor por defecto correspondiente a su tipo de dato.

La sintaxis para declarar una variable es la siguiente:

```
<TIPO> <ID>;
<TIPO> <ID> = <EXPRESIÓN>;
Bool flag;
Str saludo = “Hola Mundo!”;
```

Las variables no pueden cambiar de tipo de dato, se deben mantener con el tipo declarado inicialmente, por lo que se debe de validar que el tipo de la variable y el valor sean compatibles

Asignación de variables

Esta instrucción permitirá modificar el valor de la variable que fue previamente declarada.

```
Int a = 5+1;        // Declaracion de variable a
a = 8;              // Asignacion de nuevo valor
```

Incremento y decremento

Los incrementos y decrementos nos ayudan a realizar la suma o resta continua de un valor de uno en uno, es decir si incrementamos una variable, se incrementará de uno en uno, mientras que, si realizamos un decremento, hará la operación contraria. Esta instrucción es válida tanto para Enteros como para Decimales

```
<EXPRESION> ++;
<EXPRESION> --;
int edad = 18;
edad++; // tiene el valor de 19
edad--; // tiene el valor de 18
```

Sentencias de control

Estas sentencias modifican el flujo del programa introduciendo condicionales. Las sentencias de control para el programa son el IF y el SWITCH

Sentencia IF

El lenguaje obj-C posee la sentencia IF similar a otros lenguajes de programación, la cual permite ejecutar bloques de código se ejecuten si la condición a evaluarse es verdadera. Esta sentencia se define por las instrucciones if, else if, else.\

Consideraciones

- Las instrucciones else if y else son opcionales
- La instrucción else if se puede utilizar tantas veces como se desee

```
if ( <EXPRESION> ) {
<INSTRUCCIONES>
}
if ( <EXPRESION> ) {
<INSTRUCCIONES>
} else {
<INSTRUCCIONES>
}
if ( <EXPRESION> ) {
<INSTRUCCIONES>
} else <IF>
/* Son 3 variantes en total
1. IF
2. IF-ELSE
3. IF- ELSE IF
*/
```

Sentencia SWITCH

El lenguaje Obj-C posee la sentencia switch, similar a otros lenguajes de programación, la cual permite ejecutar diferentes bloques de código dependiendo del valor de una expresión. Esta sentencia es útil cuando se desea evaluar múltiples posibles valores de una misma variable o expresión, evitando el uso excesivo de sentencias if.

Consideraciones

- Cada case representa un valor posible que puede tomar la expresión.
- Es obligatorio usar break al final de cada case para evitar la ejecución de los casos siguientes (caída o *fall-through*).
- El bloque default es opcional, pero recomendable como caso por defecto si ninguno de los case coincide.
- La expresión dentro del switch debe evaluar a un valor entero. No se permiten expresiones de tipo float, double o string.

```
switch (expresion) {
    case valor1:
        // Código a ejecutar si expresion == valor1
        break;
    case valor2:
        // Código a ejecutar si expresion == valor2
        break;
    default:
        // Código a ejecutar si ningún caso coincide
        break;
}
```

Sentencias Cíclicas

Los ciclos o bucles son una secuencia de instrucciones de código que se ejecutan una vez tras otra mientras la condición, que se ha asignado para que pueda ejecutarse, sea verdadera. En el lenguaje actual, se podrán realizar 3 sentencias cíclicas que se describen a continuación.

Observaciones:

- Es importante destacar que pueden tener ciclos anidados entre las
- sentencias a ejecutar.
- También, entre las sentencias pueden tener ciclos diferentes anidados

While

El ciclo o bucle While, es una sentencia que ejecuta una secuencia deinstrucciones mientras la condición de ejecución se mantengaverdadera

```
while ( <EXPRESION> ) {
        <INSTRUCCIONES>
}
```

FOR

El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella.

Observaciones:

- Para la actualización de la variable del ciclo for se puede utilizar
  - Incremento |Decremento: i++ | i--
  - Asignación: como i = i+1, i = i-1, etc, es decir, cualquiertipo de asignación
- Dentro pueden venir N instrucciones

```
for ( <ASIGNACIÓN> ;<CONDICIÓN> ;<ACTUALIZACIÓN> ) {
        <INSTRUCCIONES>
}
```

Do-While

El ciclo o bucle Do-While, es una sentencia que ejecuta al menos una vez el conjunto de instrucciones que se encuentran dentro de ella y que se sigue ejecutando mientras la condición sea verdadera.

Observaciones:

- Dentro pueden venir N instrucciones

```
do {
    [<INSTRUCCIONES> ]
} while ( <EXPRESION> ) ;
```

Sentencias de Transferencia

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

Break

La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo.

```
break ;


int i = 0;
for(i = 0; i < 3; i++){
        if (i==2){
                break; // me salgo en i = 2 y ya no se ejecuta lo demás
        }
        println(i);
}
```

Continue

La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error

```
continue ;
int i = 0;
for(i = 0; i <= 2; i++){
        println(i);
        if (i==1){
                continue;
                //omite esta iteracion
        }
        println(i*5);
}
// Salida
/*
0
0
1
2
10
*/
```

6. Reportes

Tabla de errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente.

#	Tipo	Descripción	Línea	Columna
1	Léxico	El carácter “$” no pertenece al lenguaje	5	3
2	Sintáctico	Se encontró Identificador y se esperaba Expresión.	6	3
3	Semántico	No se puede realizar resta entre CADENA y CADENA	8	10

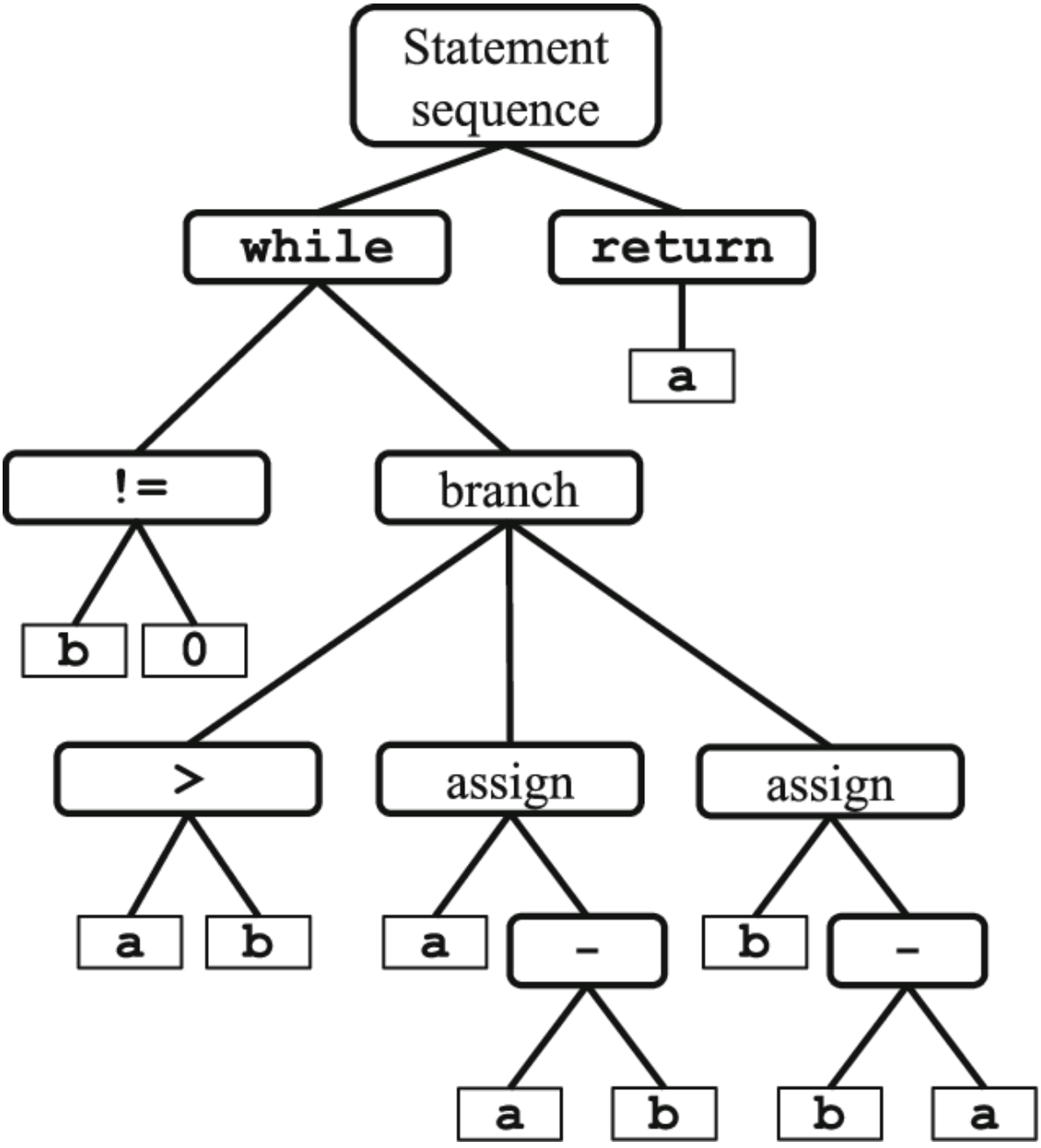
Tabla de Símbolos

Este reporte mostrará la tabla de símbolos después de la ejecución. Deberá mostrar las variables y arreglos declarados, así como su tipo, valor y toda la información que considere necesaria.

#	Id	Tipo	Tipo2	Entorno	Valor	Línea	Columna
1	var	Variable	Entero	Funcion1	1	15	20
2	var2	Variable	Bool	Funcion2	true	20	13

AST

Este reporte muestra el AST producido al analizar los archivos de entrada. Este debe de representarse como un grafo. Se deben mostrar los nodos que el estudiante considere necesarios para describir el flujo realizado para analizar e interpretar sus archivos de entrada (No confundir con CST).



7. Salidas en consola

La consola es el área de salida del intérprete. Por medio de esta herramienta se podrán visualizar las salidas generadas por la instrucción “println”, así como el encabezado con el número de errores después esta sentencia se dejara un salto de linea.

```
Println(“Hola”);
Println(1234);
/* Respuesta:
Hola
1234
*/
```

8. Restricciones

- La aplicación deberá de desarrollarse utilizando el lenguaje Python.
- La herramienta para realizar los analizadores serán PLY.
- El proyecto debe ser realizado en grupos de 4.
- Copias completas/parciales de: código, gramática, etc. Serán merecedoras de una nota de 0 puntos, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas.
- La entrega debe ser realizada mediante UEDI.
- El nombre del repositorio de Github debe ser OLC1_VJ2025_G# (donde # debe ser remplazado con el numero de grupo, ej: G12)
- Se debe agregar al auxiliar como colaborador del repositorio (username: crisAx197).
- Todas las salidas se deben de visualizar en la aplicación.

9. Fecha de entrega

Miércoles 18 de Junio hasta las 11:59 horas.