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 II

Tabla de contenido

1. Objetivo General 3

2. Objetivos Específicos 3

3. Descripción General 3

4. Entorno de Trabajo 4

Editor 4

Flujo de la aplicación 5

Funcionalidades 5

Reportes 6

5. Descripción del lenguaje 7

Vectores 7

Procedimientos: 10

Funciones Nativas: 12

Funcion Seno: 12

Funcion Coseno: 12

Funcion inversión: 12

6. Restricciones 14

7. Fecha de entrega 15

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.
- Que el estudiante sea capaz de realizar acciones gramaticales utilizando el lenguaje de programación Python.
- Que el estudiante comprenda como se usa la memoria a nivel de compilador.

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, sintácticos y semánticos. 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).

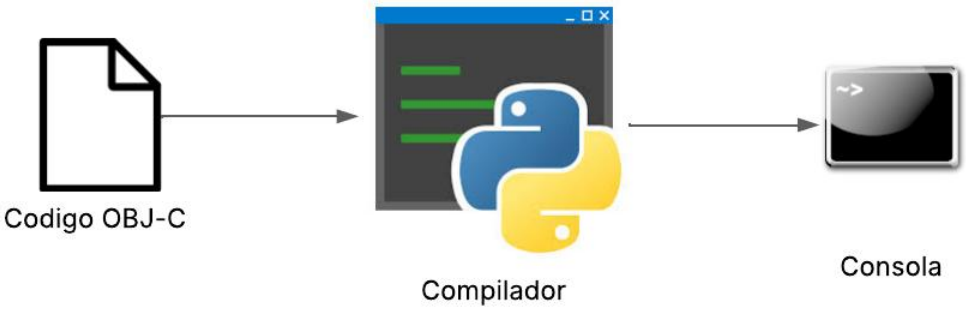
El enfoque estará centrado en construir una gramática robusta y clara, capaz de interpretar correctamente estructuras propias del lenguaje, permitiendo definir variables, controlar flujo, realizar operaciones aritméticas y lógicas, entre otras funcionalidades típicas de lenguajes de alto nivel. Adicionalmente, 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



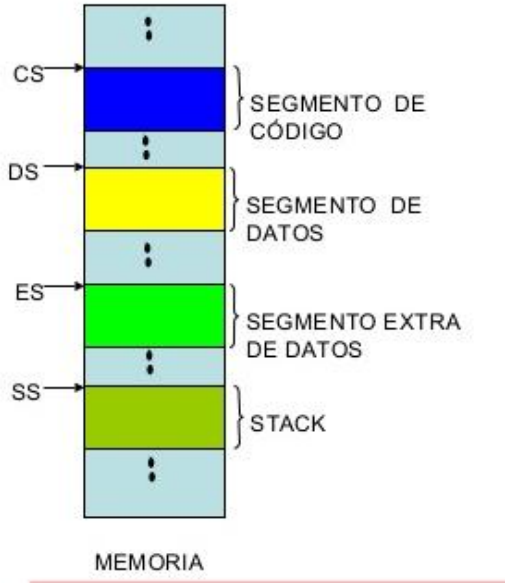
Funcionalidades

- **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.
- **Reporte de Vectores**: Se mostrarán todos los vectores declarados, cada vector será almacenado de forma lineal en una estructura de datos en memoria, el reporte permitirá visualizar la estrategia utilizada para almacenar cada vector, los vectores pueden tener n-dimensiones, y cada uno debe tener una representación lineal.
- **Reporte de memoria**: Dependiendo de las estructuras utilizadas se debe generar un reporte para cada entidad que necesite de almacenar datos de manera dinámica durante la ejecución. Por ejemplo, un procedimiento necesita parámetros, instrucciones y un lugar de vuelta por cada llamada este.



En este reporte no es necesario reportar vectores ni variables, debido a que cada uno de estos tiene una estructura de reportes propia.

- **Reporte de advertencias:** Durante la declaración de los procedimientos se deberá de validar que se usen las variables dentro de este mismo, de lo contrario se generara una advertencia, pero no se considerara como un error.

5. Descripción del lenguaje

Al sistema realizado durante la fase 1, se le deberán agregar las siguientes funcionalidades:

Vectores

Los vectores serán una estructura de datos capaz de contener distintos elementos de un mismo tipo, la sintaxis será la siguiente:

- **Declaración:**

Vector[<TIPO>] <ID>(<DIMEMSIONES>);

Vector[<TIPO>]<ID>(<DIMEMSIONES>) = [<EXP>, <EXP>, ….],[<EXP>,….],…

Las dimensiones tendrán la forma num, num, ….

Donde num será únicamente un numero entero, no puede ser una expresión (2+1 es incorrecto) u otro tipo de dato (True, False es incorrecto), tampoco podrá ser un numero negativo o 0. Cada numero agregado a esta lista agregara una dimensión, el tamaño de este número será la escala de la dimensión, por ejemplo: (5) representa una única dimensión, la cual será capaz de almacenar cinco elementos.

Ejemplo:

Vector[int] vec_ejemplo (2,2) = [1,2],[1,5];

Es un vector de 2x2, donde

vec_ejemplo[0][0] = 1

vec_ejemplo[1][0] = 2

vec_ejemplo[0][1] = 1

vec_ejemplo[1][1] = 5

- **Asignación:**

Se podrá hacer asignación únicamente dando los parámetros para una dirección de variable dentro del vector, se debe considerar que los parámetros para hacer la asignación deben de estar en el rango definido dentro de la declaración.

Ejemplo:

Vector:

ID: vec_ejemplo

Valores:

1	2
1	5

vec_ejemplo[0][0] = 2;

vec_ejemplo[1][0] = 3;

vec_ejemplo[0][1] = 4;

vec_ejemplo[1][1] = 1;

Valores actualizados:

2	3
4	1

- **Acceso:**

Se permitirá el acceso a los valores dentro del vector únicamente en donde las variables eran accesibles en la Fase 1, y demás casos que se planteen durante este enunciado.

Ejemplo:

```
Int a = vec_ejemplo[0][0]; // Esto es valido
vec_ejemplo[0][1] = vec_ejemplo[1][0]; // Esto es valido
IF ( vec_ejemplo[0][1] == 1){
    Println(“Esto también es valido”);
}
```

Procedimientos:

En el contexto de compiladores, los procedimientos son bloques de código que encapsulan una secuencia de instrucciones y que pueden ser llamados desde otras partes del programa. Su propósito principal es modularizar el código, facilitar la reutilización y mejorar la organización del programa.

Los procedimientos dentro del lenguaje Object-C deben obedecer la siguiente forma:

- **Declaración**:

Durante la declaración se deberán de colocar de forma explícita los parámetros de la declaración y las instrucciones que ejecutara. Además, estas instrucciones no pueden ser anidadas, por lo que no se puede declarar un procedimiento dentro de otro y cada procedimiento debe generar su propio entorno (scope), todos los procedimientos deben ser capaces de acceder a las variables en el entorno global. También, cada procedimiento generara su propia tabla de símbolos en el apartado de reportes.

PROC <ID> (<PARAMETROS>) { <INSTRUCCIONES> }

Donde <PARAMETROS> estará definido de la siguiente forma:

<TIPO> : <ID>, <TIPO> : <ID> , ……

Nota: No se podrán incluir vectores como parámetros. Pero si direcciones dentro de un vector.

En caso de repetirse algún ID, se deberá retornar un error semántico, y no se almacenara el procedimiento.

- **Llamada:**

Durante la llamada de un procedimiento se deberán colocar todos los parámetros, de lo contrario se considerará como un error semántico. Los parámetros únicamente pueden ser datos nativos o IDs, no pueden ser operaciones o llamadas de funciones.

Durante la llamada de la se deberán incluir todos los parámetros definidos en la declaración, de lo contrario se considerará como un error semántico.

EXEC <ID> (<PARAMETROS>);

Donde <PARAMETROS> estará definido de la siguiente forma:

<ID>, <INT>, <CHAR>, ….

Nota: En caso de que un parámetro no coincida con el tipo de dato requerido durante la declaración se considerara como error semántico.

La sentencia EXEC si deberá poder ser llamada dentro de un procedimiento. Pero no dentro del procedimiento al que esta invocando. No se permitirá este tipo de recursividad.

Ejemplo:

```
// Declaración
Int a = 1;
PROC actualizar_a(int: actualización, char: mensaje){
    Println(“Mensaje”);
    A = a+actualización;
    // Aquí también puede ir EXEC
```

}

EXEC Actualizar_A(3,”Hola mundo”);

Funciones Nativas:

Una función nativa es una función predefinida, generalmente implementada directamente por el lenguaje de programación o su entorno de ejecución, y cuya lógica ya está optimizada o compilada, por lo que no requiere que el programador la defina desde cero.

El lenguaje Object-C tendrá las siguientes funciones nativas:

Funcion Seno:

Calcula el seno de un número en radianes, únicamente aceptara números, IDs o direcciones de un vector.

Ejemplo:

Float sin_1 = Seno(0); // Res: 0

Funcion Coseno:

Calcula el coseno de un número en radianes, únicamente aceptara números, IDs o direcciones de un vector.

Ejemplo:

Float cos_1 = Coseno(0); // Res: 1

Funcion inversión:

Reordena los dígitos de un numero, únicamente aceptara enteros.

Int x_inv = Inv(123456789); // Res: 987654321

Funcion shuffle:

Reordena los elementos de un vector, cambiando la estrategia de ordenamiento a column major.

/*

Originalmente como row major

1 2

1 5

*/

Vector[int] vec_ejemplo (2,2) = [1,2],[1,5];

/*

Cambiando de estrategia a columna major

1 1

2 5

*/

Vector[int] vec_shuffled (2,2) = shuffle(vec_ejemplo);

Nota: El vector destino debe ser del mismo tamaño que el vector del parámetro, de lo contrario se considerara como error semántico.

Funcion Sort:

Ordena los elementos de forma ascendente, los vectores utilizados como parámetros para esta función únicamente deberán ser numéricos y de una dimensión (una lista), de lo contrario se considerará como un error semántico.

Vector[int] vec_original (5) = [1,4,3,2,5];

Vector[int] vec_shuffled (5) = sort(vec_original); // Res: 1,2,3,4,5

Nota: El vector destino debe ser del mismo tamaño que el vector del parámetro, de lo contrario se considerara como error semántico.

Nota: Para todas las funciones nativas, únicamente se asignaran a una variable, pero no se pueden operar directamente.

6. 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 número 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.
- Es requerido un manual técnico y de usuario.
- En el manual técnico debe tener la gramática en BNF.

7. Fecha de entrega

Viernes 27 de Junio hasta las 11:59 horas.