Instituto Tecnológico de Costa Rica
Ingeniería en Computación
Compiladores e Intérpretes
Semestre II, 2025
Profesor: Allan Rodríguez Dávila

Proyecto #1
Análisis Léxico y Sintáctico

Introducción

Un grupo de desarrolladores desea crear un nuevo lenguaje imperativo, ligero, que le permita realizar operaciones básicas para la configuración de chips, ya que esta es una industria que sigue creciendo constantemente, y cada vez estos chips necesitan ser configurados por lenguajes más ligeros y potentes. Es por esto por lo que este grupo de desarrolladores requiere desarrollar su propio lenguaje para el desarrollo de sistemas empotrados.

Proyecto a desarrollar

Este proyecto comprende la fase de análisis léxico y análisis sintáctico para la gramática descrita en el Anexo I (descrita también en Tarea I). Se debe desarrollar un scanner utilizando la herramienta JFlex y se debe desarrollar un parser utilizando la herramienta Cup.

Un programa escrito para este lenguaje está compuesto por una secuencia de declaraciones de procedimientos, que contienen diferentes expresiones y asignaciones de expresiones; todo programa debe contener un único método main.

Para comprobar el desarrollo de ambas fases el programa a presentar deberá tomar un archivo fuente y realizar lo siguiente:

a) El sistema debe leer un archivo fuente.
b) Se debe escribir en un archivo todos los tokens encontrados, identificador asociado con el lexema.

c) Por cada token deberán indicar en cuál tabla de símbolos va y cual información se almacenará.
d) Indicar si el archivo fuente puede o no ser generado por la gramática.
e) Reportar y manejar los errores léxicos y sintácticos encontrados. Debe utilizar la técnica de Recuperación en Modo Pánico (error en línea y continúa con la siguiente). **En el reporte de errores es fundamental indicar la línea en que ocurre.**

Gramática
La gramática BNF que reconocerá el lenguaje será la descrita en el Anexo I que fue desarrollada en la tarea I. La gramática desarrollada en la tarea puede ser sujeta a cambios según se avance en la implementación de las dos primeras fases del compilador. La gramática puede tener ideas de gramáticas de referencia, pero en su estructura debe ser original.

Scanner
El desarrollo del Analizador Léxico se debe realizar utilizando la herramienta JFLex. Incluye el reconocimiento de los tokens y recuperación de errores sobre los archivos fuentes analizados. Debe reportar los errores y seguir procesando el código fuente. Tiene un valor de 40 puntos.

Parser
El desarrollo del Analizador Sintáctico se debe realizar utilizando la herramienta Cup. Incluye la comprobación sintáctica y recuperación de errores. Debe reportar los errores y seguir procesando el código fuente. Tiene un valor de 40 puntos

Puntos Extra

Se darán 2.5 puntos extra al implementar técnicas de recuperación de errores del tipo Recuperación a nivel de frase, Producción de Errores o Corrección Global.

Se darán **2.5 puntos adicionales** al entregar a más tardar el miércoles 24 de septiembre a las 11:55:55 PM el Documento de Requerimientos Detallado, ver plantilla suministrada en el Tec Digital. Debe subirse en la documentación llamada “Proyecto I (archivos adicionales)” debajo de la carpeta de “Proyectos”.
Requerimientos.PrimerNombreMiembro1.PrimerNombreMiembro2.zip

Aspectos técnicos

El proyecto deberá correr en java y utilizar las herramientas JFlex y Cup. En caso de requerir librerías adicionales para compilar y ejecutar el programa, deberán especificarlo en la documentación, ya que de lo contrario se descontarán puntos en la evaluación. Se debe incluir los archivos .flex y .cup, y el proyecto en Java que utilice los archivos generados por JFlex y Cup.

Deberán utilizar el sistema de control de versiones GitHub, el repositorio deberá ser público o incluir al profesor en el control de acceso de este.

Se valorará el aporte generado por cada estudiante, considerando, entre otras cosas, los commit generados por cada uno. Por lo que el puntaje obtenido por cada uno de los estudiantes puede ser diferenciado.

Documentación

La documentación es un aspecto de gran importancia en el desarrollo de programas, especialmente en tareas relacionadas con el mantenimiento de estos.

Para la documentación interna, deberán incluir comentarios descriptivos para cada parte, con sus entradas, salidas, restricciones y **objetivo**.

La documentación externa deberá incluir:

1. Portada.
2. Manual de usuario: **instrucciones de compilación, ejecución y uso bien detalladas**.
3. Pruebas de funcionalidad: incluir *screenshots*.
4. Descripción del problema.
5. Diseño del programa: decisiones de diseño, algoritmos usados.
6. Librerías usadas: creación de archivos, etc.
7. Análisis de resultados: objetivos alcanzados, objetivos no alcanzados, y razones por las cuales no se alcanzaron los objetivos (en caso de haberlos).
8. Justificación de toma de decisiones: por qué se generaron las producciones en expresiones y la precedencia.
9. Bitácora (autogenerada en git, commit por usuario incluyendo comentario).

Evaluación

La evaluación se va a centrar en dos elementos: programación y documentación.

El proyecto programado tiene un valor de **12.5%** de la nota final, en el rubro de Proyectos.

Desglose de la evaluación del proyecto programado:

1. Documentación interna 2 ptos.
2. Documentación externa 8 ptos.
3. Funcionalidad 80 ptos (ver detalle en Proyecto a Desarrollar)
4. Revisión del proyecto (estado del sistema y correcta gestión del tiempo) 5 ptos.
5. Hora de Entrega 5 ptos.

Forma de trabajo

El trabajo se debe realizar en parejas.

Aspectos administrativos

Debe crear un archivo **.zip** ("PP1_Integrante1_Integrante2.zip") que contenga únicamente un archivo **info.txt** y 2 carpetas llamadas **documentacion** y **programa**, en la primera deberá incluir el documento de *word* o pdf solicitado y en la segunda los archivos y/o carpetas necesarias para la implementación de este proyecto programado. El archivo **info.txt** debe contener la siguiente información (cualidades):

a. Nombre del curso
b. Número de semestre y año lectivo
c. Nombre de los Estudiantes
d. Número de carnet de los estudiantes
e. Número de proyecto programado
f. Fecha de entrega

g. Estatus de la entrega (debe ser **CONGRUENTE** con la solución entregada): [Deplorable|Regular|Buena|MuyBuena|Excelente|Superior]

Entrega

Deberá subir el archivo antes mencionado al TEC Digital en el curso de COMPILADORES E INTERPRETES GR 60, en la asignación llamada "P1" debajo del rubro de "Proyectos". En la evaluación del Proyecto el rubro de "Hora de Entrega" valdrá por 5 puntos de la nota total del proyecto, según la siguiente escala:

a. Si se entrega antes de las 11:55:55 **PM** del lunes 06 de octubre de 2025, 5 puntos.
b. Si se entrega antes de las 11:55:55 **AM** del martes 07 de octubre de 2025, 2.5 puntos.
c. Si se entrega antes de las 11:55:55 **PM** del martes 07 de octubre de 2025, 0 puntos. Después de este punto, **NO SE ACEPTARÁN** más trabajos.

Los archivos .flex y .cup pueden ser revisados en el sistema de Control de Plagio del TEC Digital.

Todo el contenido de cada proyecto debe ser 100% original y en caso de conducta fraudulenta se procederá según lo indicado en el artículo 75 del RREA. Todos los miembros del grupo deberán participar en el desarrollo del proyecto y en la revisión, ya que de lo contrario se les penalizará con puntos.

ANEXO I

GRAMÁTICA

La Gramática BNF (considerando únicamente **aspectos léxicos y sintácticos**) para un lenguaje imperativo con las siguientes características:

a. Permitir la creación de funciones, y dentro de ellas, estructuras de control, bloques de código (¿ y ?) y sentencias de código.
b. Se permite crear variables globales.
c. Manejar los tipos de variables enteras, flotantes, booleanas, caracteres, cadenas de caracteres (string) y arreglo estático unidimensional.
d. Se permite crear arreglos de tipo char y entero. Además, se permite obtener y modificar sus elementos, y ser utilizados en expresiones. Se permite creación con asignación (¿ y ?).

e. Permitir sentencias para creación de variables (locales y globales), creación y asignación de expresiones y asignación de expresiones a variables, y algunos casos, sólo expresiones sin asignación. En el caso de creaciones utilizar la palabra reservada `let`.
f. Las expresiones permiten combinar literales, variables, arreglos y/o funciones, de los tipos reconocidos en la gramática.
g. Debe permitir operadores y operandos, respetando precedencia (usual matemática) y permitiendo el uso de paréntesis (ϵ y ϶).
h. Permitir expresiones aritméticas binarias de suma (+), resta (-), división (// y /) –entera o decimal según el tipo--, multiplicación (*), módulo (%) y potencia (^). Para enteros o reales.
i. Permitir expresiones aritméticas unarias de negativo (-), ++ y -- después del operando; el negativo se puede aplicar a literales enteros y flotantes, el ++ y -- se aplica a variables enteres y flotantes.
j. Permitir expresiones relacionales (sobre enteros y flotantes) de menor, menor o igual, mayor, mayor o igual, igual y diferente. Los operadores igual y diferente permiten adicionalmente tipo booleano.
k. Permitir expresiones lógicas de conjunción (@), disyunción (~) y negación (ésta debe ser de tipo caracter (Σ)).
l. Debe permitir sentencias de código para las diferentes expresiones mencionadas anteriormente y su combinación, el delimitador de final de expresión será el carácter dólar ($). Además, dichas expresiones pueden usarse en las condicionales y bloques de las siguientes estructuras de control.
m. Debe permitir el uso de tipos y la combinación de expresiones aritméticas (binarias y unarias), relacionales y lógicas, según las reglas gramaticales, aritméticas, relacionales y lógicas del Paradigma Imperativo, por ejemplo, tomando como referencia el lenguaje C. Cualquier duda con respecto al comportamiento de la funcionalidad debe validarlo con el profesor.
n. La gramática genera un lenguaje con tipado explícito y fuerte.
o. Debe permitir las estructuras de control:
    a. **decide of**, de la forma:
    ```
    decide of
    (condicion -> bloque)*
    (else -> bloque)?
    end decide$
    ```
    b. **loop** (tipo Ada, no usa llaves de bloque)
    ```
    loop
    instrucciones…
    exit when condicion$
    end loop$
    ```
    c. **for** (variación de Pascal con to y downto, utiliza llaves después del do para definir un bloque)
    ```
    for asignacion step valor (to o downto) expresion do
    bloque
    ```

d. Además, permitir **return** y **break**.
   Las expresiones de las condiciones deberán ser valores booleanos combinando expresiones aritméticas, lógicas y relacionales.

p. Debe permitir las funciones de leer (enteros y flotantes) y escribir en la salida estándar (cadena carácter, enteros, boolean y flotantes), se pueden escribir literales o variables. Utilizar palabras reservadas **output** e **input**.
q. Debe permitir la creación y utilización de funciones, estos deben retornar valores (entero, flotantes, char y booleanos) y recibir parámetros (con tipo).
r. Debe definir un único procedimiento inicial main (llamado `principal`), por medio de la cual se inicia la ejecución de los programas, este es de tipo `void` y no recibe parámetros.
s. Además, debe permitir comentarios de una línea (|) o múltiples líneas (¡ y !).