**Tecnológico de Costa Rica**

*Escuela de Ingeniería en Computación*

Texto

Descripción generada automáticamente

# Proyecto III “Generación código destino”



*Compiladores e intérpretes*

Profesor:

Rodríguez Dávila, Allan

Presentado por:

Bonilla Espinoza, Alina [2016248502]

Lara Oses, Froylan [2018276191]

**Noviembre, 2023**

# Manual de usuario

**Compilación y ejecución**

* Como primer paso el usuario debe crear un archivo en formato txt con un ejemplo de código en lenguaje C. El archivo puede ser nombrado bajo cualquier nombre.
* Posteriormente, se debe abrir el proyecto en NetBeans y dar click en ejecutar.

A screenshot of a computer

Description automatically generated

* Una vez ejecutado el código este leerá el archivo txt ingresado y mostrará en pantalla el código MIPS y el código de 3 Direcciones correspondiente a este.
* En el caso de encontrar caracteres o símbolos que el analizador no reconozca en el archivo, se maneja el error mostrando en pantalla un mensaje de que ha habido un error. Y posterior a esto se siguen analizando el resto del código.
* Si se encuentran errores semánticos el analizador semántico se encargará de manejar estos errores imprimiendo un mensaje en consola de cuál es el error semántico en el código y se continuara analizando el resto de las líneas de código.
* Errores semánticos el analizador semántico se encargará de manejar estos errores imprimiendo un mensaje en consola de cuál es el error semántico en el código y en qué línea y columna específica sucede y posteriormente proseguirá a seguir analizando el resto de las líneas de código.

# Pruebas de funcionalidad

Una vez ejecutado el programa, este se encargará de hacer el análisis semántico y de realizar la generación de código intermedio 3 direcciones del archivo proporcionado por el usuario. Cómo caso de demostración se tiene un archivo que contiene el siguiente código de ejemplo:

* **Creación, asignación de variables y operaciones aritmeticas**

Código de prueba

A computer code with black text

Description automatically generated

Resultado 3 direcciones

A screenshot of a computer

Description automatically generated

Resultado de código MIPS

A screenshot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A white screen with black text

Description automatically generated

* **Estructuras de control**

**A computer code with black text

Description automatically generated**

Resultado de código de 3 direcciones

A white paper with black text

Description automatically generated

Resultado de código MIPS

A close up of a paper

Description automatically generated

A white paper with black text

Description automatically generated

# Descripción del problema

Se 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.

Este proyecto comprende la fase de generación de código destino, el cuál será MIPS, para la gramática descrita en el Proyecto I y el front-end de compilador generado en los Proyectos I y II. Se debe desarrollar el generador de código MIPS a partir del código tres direcciones generado por el proyecto II.

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

# Diseño del programa

A continuación, se detalla el proceso que se realizó para la creación del proyecto:

decisiones de diseño, algoritmos usados.

1. *Generación del Analizador Léxico:*

El archivo lexemas.cup se utiliza como entrada para Java Cup, que genera un analizador léxico Java basado en las reglas y patrones definidos en lexemas.cup. El analizador léxico generado tiene la capacidad de escanear y reconocer tokens en el código fuente.

1. *Análisis del Código Fuente:*

El programa AnalizadorLexico.java se encarga de abrir un archivo de código fuente o de recibir texto de entrada y lo pasa al analizador léxico generado. El analizador léxico escanea el código fuente identificando y clasificando cada token según las reglas definidas en lexemas.cup.

1. *Etiquetado de Tokens:*

Para cada token reconocido, el analizador léxico asigna un tipo de token de acuerdo con la enumeración Tokens.java. Esto permite que se etiquete cada parte del código fuente con un tipo específico (identificador, palabra clave, operador, etc.).

1. *Generación de Resultados:*

Se generar una lista con todos los tokens encontrados y sus tipos.

1. *Manejo de Errores:*

Si el analizador léxico encuentra caracteres o secuencias que no coinciden con ninguno de los patrones definidos en lexemas.cup, se generará un token de tipo ERROR para señalar el error léxico.

1. *Análisis semántico*

Se crearon todas las comprobaciones pertinentes dentro de las producciones del archivo CUP, por lo cual cada que una de las líneas de código se asocia a una producción está hace las verificaciones necesarias y en el caso de encontrar un error se reporta como un error semántico y se continúa con el análisis (recuperación modo pánico).

1. *Codigo intermedio*

En cada una de las producciones después de que se hace el análisis semántico se hace la generación de código en 3 direcciones mediante código Java incrustado en cada una de las producciones y cada generación de código 3D se adjunta a un Stringbuilder para posteriormente una vez finalizado todo el análisis del archivo txt.

1. *Codigo Mips*

Para la elaboración de este se realizo un proceso muy parecido al del código de 3 direcciones, ya que este se creó de forma simultánea. En cada producción se hace la creación de código MIPS mediante código de Java.

# Librerías usadas

Para la elaboración de este proyecto se hizo uso de las siguientes librería:

* File: Se utiliza para representar rutas de archivos.
* FileReader: Se utiliza para leer archivos.
* FileWriter: Se utiliza para la escritura de archivos.
* java-cup-11b.jar

# Análisis de resultados

## **Objetivos alcanzados**

* Generación de código intermedio: 60%
* Generación de código MIPS: 60%
* Análisis semántico: 75%

**Objetivos no logrados**

* No se genera archivo de .asm

# Bitácora

Seguidamente, se muestra la bitácora recaudada del archivo compartido en la plataforma de GitHub de los integrantes: Froylan Lara (Udue11) y Alina Bonilla (Alina-bonilla).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated