TEC | Tecnológico de Costa Rica

20 DE ABRIL DE 2023

PROYECTO 1

ANALISIS LÉXICO Y SINTÁCTICO

JEAN HUNT, PRISCILLA CHACÓN

COMPILADORES E INTERPRETES

Ingeniería en computación

TEC | Tecnológico de Costa Rica

Tabla de contenidos

1. Manual de usuario

A continuación, se detallan los pasos para la compilación y ejecución del proyecto además de pruebas de su funcionalidad.

1) Descargue o clone el repositorio del proyecto a la maquina en donde desea ejecutarlo. Esto se puede hacer directamente desde la pagina de github o utilizando el comando “git clone” con el siguiente enlace: https://github.com/JPHuntV/P1-Compiladores-IS-2023
2) Abra una terminal en la carpeta src del proyecto y ejecute los siguientes comandos para la compilación del lexer y del parser:
    a. Lexer: java -jar "../lib/jflex-full-1.9.0.jar" scanner.jflex
    b. Parser: java -jar "../lib/java-cup-11b.jar" -interface parser.cup
3) El paso anterior generara los archivos “Analizador.java”, “parser.java” y “sym.java”.
4) En el archivo “Main.java” se hace referencia a del archivo que se desea analizar, asegúrese que sea la ruta correspondiente a su equipo:

```java
package src;
public class main {
    Run | Debug
    public static void main(String[] args) {
        App app = new App();
        app.analizar(archivoFuente:"programa/src/archivoFuente.txt");//lexer
        app.parsear(archivoFuente:"programa/src/archivoFuente.txt");//parser
    }
}

```

Ilustración 1:Main.java

5) Ejecute el archivo main.java haciendo uso de java. Esto lo puede hacer directamente desde su editor de preferencia.
6) La ejecución del programa generara 2 archivos:
    a. Lexemas.txt: contiene todos los lexemas encontrados en el archivo fuente.

```
Token: 45   valor: int
Token: 43   valor: residuo
Token: 6    valor: (
Token: 45   valor: int
Token: 43   valor: num
Token: 15   valor: ,
Token: 48   valor: char
Token: 43   valor: letra
Token: 15   valor: ,
Token: 47   valor: string
Token: 43   valor: a
Token: 7    valor: )
Token: 13   valor: {
Token: 45   valor: int
Token: 43   valor: numw
Token: 10   valor: =
Token: 38   valor: 2
Token: 44   valor: $
Token: 43   valor: numa
Token: 10   valor: =
Token: 38   valor: 2
Token: 44   valor: $
```

Ilustración 2:Resultado Lexer

b. Tablasimbolos.txt: Contiene las tablas de símbolos generadas durante el parseo.

```
Tabla: residuo  Tipo de retorno: int
    valor: a    tipo: string
    valor: numw    tipo: int
    valor: num    tipo: int
    valor: letra    tipo: char


Tabla: main  Tipo de retorno: int
    valor: miBool    tipo: bool
    valor: temp    tipo: int
    valor: numero    tipo: int
    valor: miString    tipo: string


Tabla: residuo2  Tipo de retorno: int

```

Ilustración 3:Tablas de símbolos

7) Si el programa encuentra algun error este será reportado en consola y no generará los archivos anteriores.

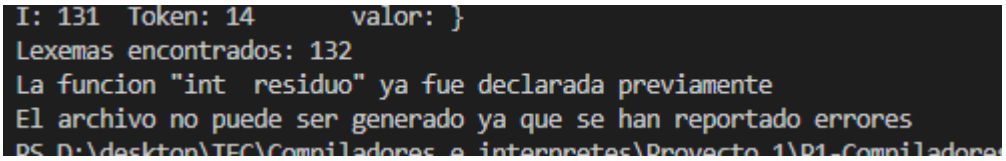

Ilustración 4: Parseo fallido

2. Descripción del problema

Existe la necesidad de desarrollar un programa informático que permita configurar chips de forma eficiente y eficaz, en una industria que se encuentra en constante crecimiento. Para lograr esto, se requiere de un programa imperativo, lo que significa que el flujo de control del programa se basa en una serie de instrucciones específicas que se ejecutan en orden secuencial.

Además, el programa debe ser ligero y potente, lo que implica que debe tener un tamaño reducido para poder ser ejecutado en sistemas con limitaciones de recursos, como los chips que se van a configurar y debe ser capaz de realizar operaciones básicas de configuración de chips de manera efectiva y eficiente.

En la industria de los chips, la configuración es una tarea crítica, ya que afecta directamente el funcionamiento del dispositivo en el que se utilizará el chip. Por lo tanto, es necesario contar con un programa que permita configurar los chips de manera precisa y confiable.

Para lograr esto, el programa debe contar con una serie de funcionalidades específicas, como la capacidad de leer y escribir datos en el chip, así como de realizar operaciones aritméticas y lógicas básicas. También es importante que el programa cuente con un sistema de gestión de errores para poder detectar y corregir cualquier problema que pueda surgir durante la configuración del chip.

En resumen, el problema consiste en desarrollar un programa informático imperativo, ligero y potente que permita configurar chips de manera eficiente y confiable en una industria en constante crecimiento.

3. Diseño del programa

El programa se puede visualizar en 3 grandes partes según el lenguaje o herramienta utilizada para su desarrollo:

1) Lexer: Escrito haciendo uso de la herramienta jflex. Este contiene la definición explicita de las terminales que serán utilizadas en el parser. Este documento se puede visualizar como un catalogo de todos los posibles símbolos que serán permitidos en el archivo fuente que se desea analizar, es decir, todos los lexemas.
2) Parser: Escrito haciendo uso de cup. Este contiene la gramática BNF que define el lenguaje y hace uso de las terminales definidas en el lexer, además de instrucciones que se deberán ejecutar al encontrar estas producciones.
3) Aplicación: Todos aquellos archivos escritos en lenguaje java que ayudan a la ejecución del programa. Es decir, clases, como “función”, “parámetros”, “listaparametros” , app.java y el main
4) Resultados: Aquellos archivos generados por el programa, tales como “lexemas.txt” y “tablasimbolos.txt”
5) Además, se adjunta un archivo “archivoFuente.txt” el cual fue utilizado durante el desarrollo del proyecto para la comprobación del funcionamiento del programa, el codigo que contiene este archivo fue extraído de la especificación del proyecto.

Para la creación de las tablas de símbolos se utiliza una estructura de tipo stact para almacenar cada una de estas tablas que pertenecen a las funciones identificadas en el archivo. Cada tabla de símbolos contiene los parámetros y variables(identificadores) encontrados dentro de la función. Para el reconocimiento de todos estos identificadores se utilizan estructuras de array para asegurarse que estos

no se repitan dentro de las funciones las cuales son representadas por una clase “Función”. Cada tabla de símbolos contiene la siguiente información:

a) Nombre de la función
b) Tipo de retorno de la función
c) Lista de identificadores
    a. Nombre
    b. Tipo

No podrán existir dos funciones con el mismo nombre y tipo de retorno, así como identificadores iguales dentro de estas.

4. librerías utilizadas

Para el desarrollo del programa se ha hecho uso de las siguientes librerías:

a) Java-cup-11b-runtime.jar, Java-cup-11b.jar : Escritura del parser en Cup
b) Jflex-full-1.9.0.jar : Escritura del lexer en jflex
c) Java.util.list, Java.util.arraylist: Manipulación de las listas de elementos que iran en las tablas de símbolos.
d) Java.util.Map, Java.util.HashMap: Mapeo de los símbolos que van dentro de dichas listas
e) Java.util.Stack: Manipulación del stack que almacena todas las listas de símbolos en relación a su función
f) BufferedReader, FileReader: Lectura del archivo fuente
g) BufferedWriter, FileWriter: Escritura de los archivos de resultado

5. Análisis de resultados

En la siguiente tabla se muestra una lista de objetivos para los cuales se evaluará si fueron cumplidos o no, en caso de no cumplirse también se indicará la razón del porqué:

Objetivo	***Alcanzado***	***Razón***
Desarrollo de la gramática BNF del lenguaje	Si	
Implementar un analizador léxico utilizando Jflex	Si	
Implementar un analizador sintáctico utilizando cup	Si	
Lectura de archivo fuente	Si	
Identificación correcta de tokens	Si	
Validar el archivo fuente con la gramática	Si	
Manejo de errores		
Extra: Derivación	si	
Extra: Entregar documento de requerimientos	si	
Extra: Implementar distintos métodos de recuperación de errores	No	Priorización de otras funcionalidades requeridas

En el siguiente enlace se puede encontrar el repositorio en GitHub que contiene el proyecto: https://github.com/JPHuntV/P1-Compiladores-IS-2023