UNIVERSIDAD AUTONOMA DE TAMAULIPAS

Materia: Programación Base 2

Profesor: Muñoz Quintero Dante Adolfo

Manual de Usuario

Integrantes del Equipo:

García Azua Jorge Roberto

Cruz Bonifacio Luis Fernando

Gonzalez Cavazos Erick Alan

Introducción

El Compilador de Calculadora Científica es una herramienta de software diseñada para procesar expresiones matemáticas complejas mediante una arquitectura de compilador completa. A diferencia de una calculadora tradicional que interpreta y evalúa al vuelo, este sistema realiza un análisis léxico y sintáctico formal, genera un Código Intermedio de Tres Direcciones y posteriormente ejecuta dicho código.

Propósito

El objetivo principal es demostrar la implementación de las fases críticas de un compilador

Instalación y Configuración

Requisitos Previos

- **Java JDK:** Versión 11 o superior instalada y configurada en el PATH del sistema.
- **Sistema Operativo:** Compatible con Windows, Linux o macOS.
- **Terminal:** CMD, PowerShell o Bash.

Instalación

El proyecto no requiere instalación de software de terceros, ya que la librería antlr-4.13.1-complete.jar se incluye dentro del repositorio.

1. Descargue o clone el repositorio del proyecto:

   Bash

   git clone https://github.com/TU_USUARIO/Calculadora-Compilador.git
2. Acceda al directorio del proyecto:

   Bash

   cd proyecto-compiladores2

Idealmente puede descargar el zip con terminación .7z que teniene solo lo necesario. Y usar los comandos anteriores.

Guía de Uso

El compilador incluye scripts de automatización para facilitar la compilación y ejecución.

Paso 1: Ejecución

En Windows:

Haga doble clic en el archivo build.bat o ejecútelo desde la terminal:

DOS

build.bat
En Linux / macOS:

Otorgue permisos de ejecución y corra el script:

Bash

chmod +x build.sh
./build.sh
Paso 2: Interacción

Una vez iniciado el programa, verá la consola de entrada. Escriba cualquier expresión matemática y presione Enter.

- **Comando de salida:** Escriba salir para cerrar el programa.

Ejemplos textual Paso a Paso

Ejemplo A: Operación Básica

Entrada:

Plaintext

10 + 5 * 2
El sistema respetará la jerarquía (multiplicación antes que suma).

Ejemplo B: Uso de Funciones y Constantes

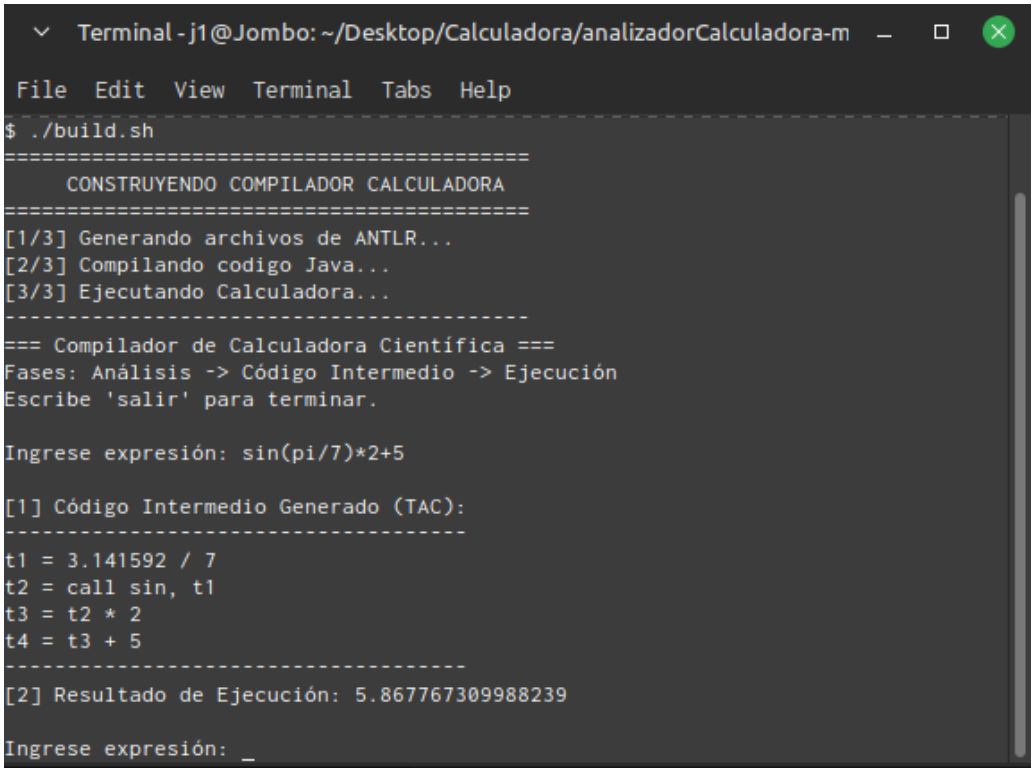
Entrada:

Plaintext

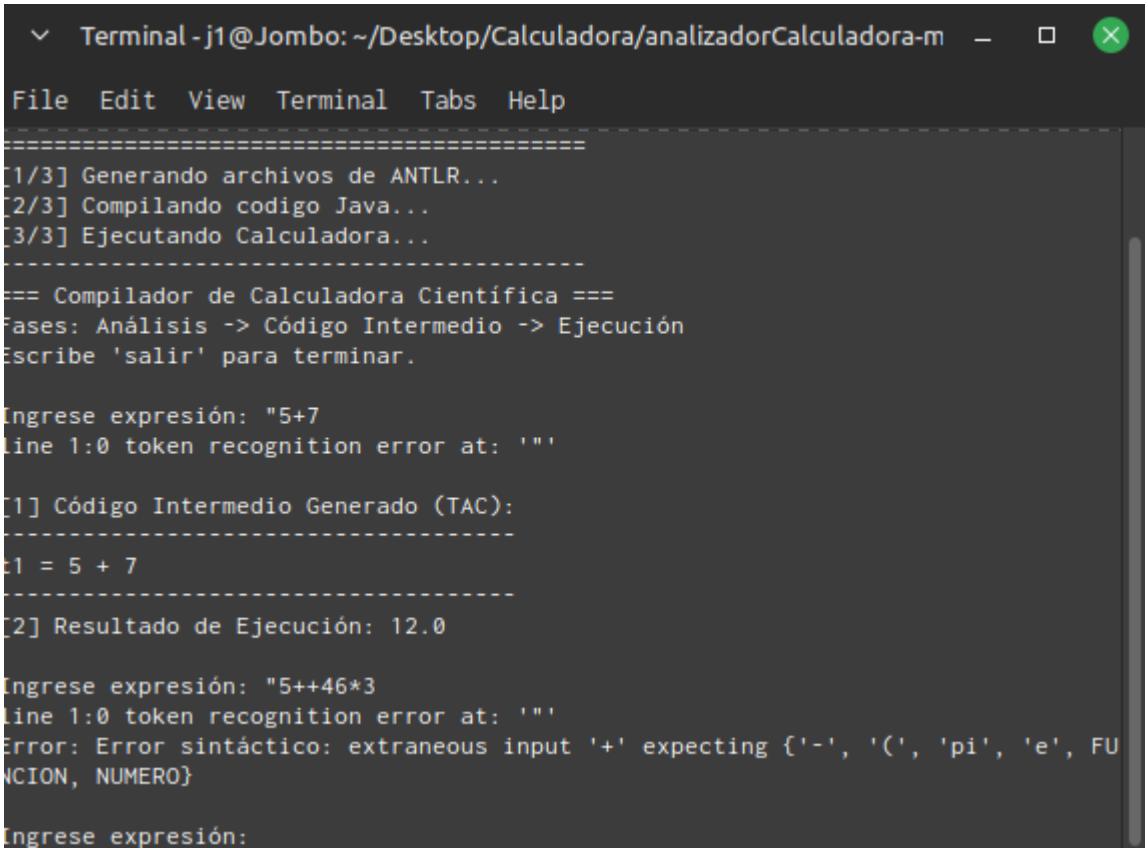
sin(pi/2) + 2^3
El sistema reconocerá pi como 3.1416 y resolverá el seno antes de sumar la potencia.

Capturas de Pantalla

5.1 Ejecución Exitosa (Windows/Linux)



5.2 Manejo de Errores Sintácticos



Referencia Técnica del Lenguaje

El lenguaje aceptado por el compilador se define mediante la gramática calc.g4.

Tipos de Datos

El sistema trabaja nativamente con números de punto flotante de doble precisión (double). Soporta notación científica (ej. 1.5e-10).

Operadores Soportados (por orden de precedencia)

1. **Paréntesis:** ( )
2. **Funciones:** sin, cos, sqrt, etc.
3. **Potencia:** ^
4. **Negativo Unario:** -
5. **Multiplicación/División:** *, /
6. **Suma/Resta:** +, -

Generación de Código Intermedio (TAC)

El compilador transforma el Árbol de Sintaxis Abstracta (AST) en instrucciones lineales.

- **Formato:** t[N] = operando1 operador operando2
- **Instrucción CALL:** t[N] = call funcion, argumento

Arquitectura del Código Fuente

El sistema se divide en tres componentes Java principales ubicados en la carpeta src/.

A. CompilerVisitor.java (Generador de Código)

Clase que implementa el patrón Visitor. Recorre el árbol generado por ANTLR utilizando un algoritmo **Post-Orden** (primero hijos, luego raíz).

- **Función:** No calcula resultados. En su lugar, genera nuevas variables temporales (t1, t2...) y construye la lista de instrucciones.

Java

```
// Ejemplo de lógica de generación
public String visitAddSubExpr(calcParser.AddSubExprContext ctx) {
    String left = visit(ctx.expr(0));
    String right = visit(ctx.expr(1));
    String temp = newTemp();
    emit(temp + " = " + left + " " + ctx.op.getText() + " " + right);
    return temp; // Retorna "tX"
}
```

B. Ejecutor.java (Máquina Virtual)

Es el backend del compilador. Simula un procesador que lee instrucciones de texto.

- **Memoria:** Utiliza un HashMap<String, Double> para simular la memoria RAM donde se guardan los valores de las variables temporales.
- **Ciclo de instrucción:** Lee línea por línea, decodifica la operación y actualiza la memoria.

C. Calculadora.java (Driver)

El punto de entrada (main). Orquesta la lectura del input, la llamada al Lexer/Parser, la invocación del CompilerVisitor y finalmente pasa el código generado al Ejecutor.

Troubleshooting (Solución de Problemas)

P: El comando java o javac no se reconoce.

R: Asegúrese de tener el JDK instalado y la variable de entorno PATH configurada correctamente hacia la carpeta bin de su instalación de Java.

P: Error "Cannot find symbol calcLexer" al compilar.

R: Esto ocurre si no se generaron los archivos de ANTLR primero. Asegúrese de ejecutar el script build.bat o build.sh completo, no solo el comando javac manualmente.

Referencias y Bibliografía

Este proyecto se fundamenta en herramientas estándar de la industria y en teoría clásica de diseño de compiladores.

Herramientas y Librerías

- **ANTLR (ANother Tool for Language Recognition):** Herramienta utilizada para el frontend del compilador (Lexer y Parser).
    - Sitio oficial: https://www.antlr.org/
    - Repositorio oficial: https://github.com/antlr/antlr4

Proyectos de Referencia

Para la arquitectura del compilador, se tomaron como referencia implementaciones de código abierto que separan la fase de análisis de la fase de ejecución:

Compilador con Generación de Código y Máquina Virtual:

- *Repositorio:* https://github.com/AmirHMousavi/compiler-antlr4

  Este proyecto sirvió de inspiración conceptual para la separación entre la fase de "CodeGen" (nuestro `CompilerVisitor`) y la ejecución en una máquina de pila o virtual (nuestro `Ejecutor`).

- **Calculadora con Patrón Visitor en ANTLR4:**
- *Repositorio:* https://github.com/shmatov/antlr4-calculator

Referencia para la implementación base del patrón Visitor y la gestión de precedencia de operadores en la gramática `.g4`.