UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

ORGANIZACION DE LENGUAJES Y COMPILADORES 1

Manual Técnico CompiScript +

MARÍA PATRICIA SERRANO RAMÍREZ

202201989

25 DE ABRIL DEL 2024

Manual Técnico

Proyecto de Interpretación de Lenguaje de Programación

Introducción

Este manual técnico proporciona una visión general de un proyecto de interpretación de lenguaje de programación que utiliza TypeScript, Jison y React. El proyecto tiene como objetivo implementar un intérprete para un lenguaje de programación simple que permita a los usuarios escribir código, ejecutarlo y ver los resultados en tiempo real a través de una interfaz de usuario web.

Tecnologías Utilizadas

- **TypeScript**: Un lenguaje de programación de código abierto desarrollado por Microsoft. Se utiliza en este proyecto para implementar el intérprete del lenguaje de programación. TypeScript proporciona tipos estáticos opcionales que permiten a los desarrolladores agregar tipos a sus variables y funciones.
- **Jison**: Un generador de analizadores de JavaScript basado en las herramientas Bison y Yacc de Unix. Se utiliza para generar los analizadores léxico y sintáctico del lenguaje de programación. Jison toma una gramática en formato Bison o Yacc y genera un analizador JavaScript.
- **React**: Una biblioteca de JavaScript para construir interfaces de usuario. Se utiliza para construir la interfaz de usuario del intérprete, lo que permite a los usuarios interactuar con el intérprete de forma intuitiva y visual.

Estructura del Proyecto

El proyecto se divide en las siguientes partes:

1. **Analizadores**: Generados con Jison, estos archivos contienen el analizador léxico y sintáctico que se utilizan para analizar el código fuente del lenguaje de programación. Por ejemplo, **analizador.lex** contiene las reglas para el analizador léxico y **analizador.y** contiene las reglas para el analizador sintáctico.
2. **Interpretador**: Escrito en TypeScript, este código toma el árbol de sintaxis abstracta generado por los analizadores y lo ejecuta, produciendo un resultado. Por ejemplo, **interpretador.ts** contiene la lógica para interpretar las expresiones del lenguaje.
3. **Interfaz de Usuario**: Construida con React, esta parte del proyecto permite a los usuarios interactuar con el intérprete. Por ejemplo, **App.tsx** contiene la estructura de la interfaz de usuario y **Editor.tsx** contiene el editor de código donde los usuarios pueden escribir su código.

Clases Principales

- **Expresion**: Clase abstracta que sirve como base para todas las clases que representan expresiones en el lenguaje de programación. Define métodos como **interpretar** y **crearGrafico**. Por ejemplo:

typescript

Copy code

abstract class Expresion { constructor(public linea: number, public columna: number) {} abstract interpretar(entorno: Environment): Resultado; abstract crearGrafico(parent: any): any; }

- **Environment**: Clase que representa el entorno de ejecución para el código. Contiene métodos como **getFuncion** para buscar funciones y **getGlobal** para obtener el entorno global. Por ejemplo:

typescript

Copy code

class Environment { private variables: Map<string, any> = new Map(); private funciones: Map<string, Funcion> = new Map(); constructor(private padre: Environment | null = null) {} getFuncion(nombre: string): Funcion | undefined { let funcion = this.funciones.get(nombre); if (!funcion && this.padre) { funcion = this.padre.getFuncion(nombre); } return funcion; } getGlobal(): Environment { if (this.padre) { return this.padre.getGlobal(); } return this; } }

- **Resultado**: Clase que representa el resultado de interpretar una expresión. Por ejemplo:

typescript

Copy code

class Resultado { constructor(public valor: any, public tipo: Tipo) {} }

Funciones Relevantes

- **createNode(label)**: Crea un nodo en un grafo con una etiqueta dada. Por ejemplo:

typescript

Copy code

function createNode(label: string): any { const node = { id: Math.random().toString(36).substring(7), label: label, shape: "box" }; return node; }

- **createEdge(parent, child)**: Crea una arista entre dos nodos en un grafo. Por ejemplo:req: Request: Este es un objeto que contiene toda la información sobre la solicitud HTTP, como la URL, las cabeceras HTTP y cualquier dato enviado con la solicitud.

res: Response: Este es un objeto que el servidor usa para enviar la respuesta HTTP de vuelta al cliente. Contiene métodos para enviar la respuesta y establecer el código de estado y las cabeceras de la respuesta.

Dentro de la función, se llama a res.send("Esta Funcionando"). El método send es parte del objeto Response, y se utiliza para enviar una respuesta de vuelta al cliente. En este caso, está enviando la cadena "Esta Funcionando" como respuesta. Esta cadena se mostraría en el navegador del cliente o se devolvería a la solicitud HTTP del cliente.

En una aplicación del mundo real, esta función probablemente se usaría como un middleware o controlador de ruta en una aplicación Express.js. Por ejemplo, podría usarse para manejar una solicitud GET en una URL específica.

Conclusiones

Este manual técnico proporciona una descripción detallada de las tecnologías utilizadas, la estructura del proyecto y las clases y funciones principales. Proporciona una base sólida para comprender y trabajar en el proyecto de interpretación de lenguaje de programación.