MANUAL TECNICO OACKLAND

202001814

USAC Lenguajes y compiladores 2

Manual Técnico de OakLand

1. Introducción

OakLand es un intérprete para un lenguaje de programación inspirado en la sintaxis de Java, diseñado para manejar múltiples paradigmas de programación, incluyendo orientación a objetos, programación funcional y programación procedimental. Este manual técnico está dirigido a desarrolladores y mantenedores del proyecto, detallando la arquitectura, componentes, y procesos involucrados en su desarrollo.

2. Arquitectura del Sistema

2.1. Componentes Principales

1. Editor de Código:

   - Implementado en JavaScript vanilla.

   - Permite crear, abrir, editar y guardar archivos `.oak`.

   - Incluye funciones para resaltar la línea actual, manejar múltiples archivos y ejecutar código directamente desde el editor.

2. Interprete:

   - Basado en un árbol de sintaxis abstracta (AST) generado a partir de una gramática definida en PeggyJS.

   - Realiza análisis léxico, sintáctico, y semántico del código fuente.

   - Ejecuta instrucciones en tiempo de ejecución, gestionando variables, estructuras de control de flujo, y funciones.

3. Consola de Ejecución:

- Muestra resultados de la ejecución del código, mensajes de error, y salidas del programa.

- Implementada como un componente visual en el IDE.

4. Reportes:

- Genera reportes de errores y tabla de símbolos.

- Proporciona detalles sobre los entornos, variables, funciones, y cualquier error encontrado durante la ejecución.

2.2. Diagrama de Arquitectura

- Cliente: Interfaz gráfica en un navegador web.

- Interprete: Procesa el código fuente y gestiona la ejecución.

- Editor de Código: Interfaz de usuario para edición y manipulación de archivos `.oak`.

- Consola: Área de salida para resultados de ejecución y mensajes.

---

3. Diseño del Intérprete

3.1. Análisis Léxico y Sintáctico

- Herramienta: PeggyJS.

- Gramática: Define la estructura sintáctica del lenguaje OakLand.

- AST (Árbol de Sintaxis Abstracta): Estructura de datos utilizada para representar el código fuente de manera jerárquica.

3.2. Componentes del Intérprete

1. Analizador Léxico:

- Divide el código fuente en tokens (palabras clave, identificadores, operadores, etc.).
- Proporciona entradas al analizador sintáctico.

2. Analizador Sintáctico:

- Utiliza la gramática definida para construir el AST.
- Detecta y reporta errores de sintaxis.

3. Ejecutor (Visitor Pattern):

- Recorre el AST y ejecuta las instrucciones.
- Gestiona entornos de ejecución, asignación de variables, y control de flujo.

4. Manejador de Errores:

- Captura errores durante el análisis y ejecución.
- Reporta errores léxicos, sintácticos y semánticos a través de la consola.

3.3. Gestión de Entornos

- Entorno Local: Cada bloque de código (función, bucle, etc.) tiene su propio entorno.
- Entorno Global: Variables y funciones definidas en el nivel más alto del código.
- Herencia de Entornos: Los entornos locales heredan del entorno en el que fueron creados, permitiendo la reutilización de variables y funciones.

3.4. Funciones y Llamadas

- Declaración: Funciones declaradas con un tipo de retorno y una lista de parámetros.

- Llamada: Las funciones pueden ser invocadas desde cualquier parte del código, respetando el ámbito de declaración.

- Recursividad: Soporte completo para llamadas recursivas.

3.5. Manejador de Control de Flujo

- If-Else, Switch-Case: Estructuras condicionales soportadas.

- Bucle While, For: Implementación de bucles controlados por condiciones.

- Transferencia de Control: Soporte para `break`, `continue`, y `return`.

---

4. Implementación Técnica

4.1. Estructura del Proyecto

- frontend/

  - Contiene los archivos de la interfaz gráfica (HTML, CSS, JS).

- analyzer/

  - Contiene la lógica del intérprete, incluidas las definiciones de gramática y clases de nodos.

- styles.css:

  - Define el estilo visual del editor y la consola.

- script.js:

  - Contiene la lógica de interacción del usuario con el editor y la consola.

4.2. Dependencias

- PeggyJS: Utilizado para la generación del analizador léxico y sintáctico.

- JavaScript Vanilla: El resto del sistema está implementado sin dependencias externas, utilizando JavaScript puro.

4.3. Ejecución del Código

1. Ingreso del Código:

  - El usuario escribe o carga código OakLand en el editor.

2. Análisis y Ejecución:

  - El intérprete analiza el código, genera el AST, y lo ejecuta.

3. Salida:

  - La consola muestra los resultados de la ejecución y cualquier error encontrado.

---

5. Consideraciones de Desarrollo

- Mantenimiento: El código está modularizado para facilitar su mantenimiento y actualización.

- Extensibilidad: Se pueden agregar nuevas funcionalidades al lenguaje o al IDE sin alterar significativamente la base de código existente.

- Despliegue: El sistema está diseñado para ser desplegado en GitHub Pages, lo que facilita la distribución y el acceso.

---

6. Documentación Complementaria

- Guía de Usuario: Disponible para los usuarios finales, detalla cómo utilizar el IDE y ejecutar código OakLand.

- Documentación de la Gramática: Describe la gramática utilizada en PeggyJS para la construcción del intérprete.

---

7. Ejemplos de Código y Pruebas

El manual incluye ejemplos básicos y avanzados de código OakLand, junto con descripciones de cómo ejecutar y probar cada uno en el IDE. Se recomienda realizar pruebas exhaustivas para garantizar el correcto funcionamiento del intérprete.

---

8. Conclusión

Este manual técnico proporciona una visión detallada de la arquitectura, diseño, y componentes del intérprete OakLand. Está diseñado para facilitar la comprensión y el mantenimiento del sistema, así como para guiar futuras expansiones y mejoras.

Este manual técnico está diseñado para proporcionar a los desarrolladores toda la información necesaria para comprender y mantener el sistema OakLand. Si hay áreas específicas que deseas que se amplíen o detallen más, por favor házmelo saber.