UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA, ESCUELA DE CIENCIAS Y SISTEMAS

OLC1

Proyecto 2 - Manual Técnico

Diego Josue Guevara Abaj

Fecha: 22 de marzo de 2024

Introducción

Bienvenido al manual Técnico de CompiScrip+, una guía esencial para comprender y poder aprovechar al máximo el lenguaje de programación que se estará creando y manejando en este proyecto, un proyecto bastante innovador y ambicioso que surge como parte del desarrollo del curso de Organización de Lenguajes y Compiladores 1; CompiScrip+ está diseñado para crear un apoyo a los estudiantes de IPC1 de la prestigiosa Escuela de Ciencias y Sistemas de la Facultad de Ingeniería, perteneciente a la Universidad de San Carlos de Guatemala.

Objetivo General

Utilizando los principios y técnicas fundamentales de análisis léxico y sintáctico de un compilador, se propone la creación de un intérprete sencillo. Este intérprete estará diseñado con las funcionalidades principales necesarias para garantizar su funcionalidad óptima. El objetivo es aplica estos conocimientos para desarrollar un sistema que pueda interpretar y ejecutar instrucciones de manera efectiva y eficiente, proporcionando así una herramienta útil para la ejecución de programas y scripts.

Objetivos Específicos

1. Reforzar los conocimientos de análisis léxico, sintáctico y semántico para la creación de un lenguaje de programación.
2. Aplicar los conceptos de compiladores para implementar un proceso de interpretación de código de alto nivel.
3. Aplicar los conceptos de compiladores para analizar un lenguaje de programación y producir las salidas esperadas.
4. Aplicar la teoría de compiladores para la creación de soluciones de software.
5. Generar aplicaciones utilizando la arquitectura cliente-servidor.

Descripción General

El curso de Organización de Lenguajes y Compiladores 1 ha iniciado un nuevo proyecto a solicitud de la Escuela de Ciencias y Sistemas de la Facultad de Ingeniería. Este proyecto consiste en desarrollar un lenguaje de programación destinado a los estudiantes del curso de Introducción a la Programación y Computación 1. El objetivo es que los estudiantes aprendan a programar y adquieran conocimientos sobre las características generales de un lenguaje de programación. Es importante destacar que este lenguaje será utilizado para las primeras prácticas de laboratorio del curso mencionado.

Para comenzar a utilizar este sistema, los usuarios deben familiarizarse con el entorno de trabajo proporcionado, que incluye un editor integrado para la creación y edición de archivos de código fuente.

Requisitos del Sistema

Requisitos Mínimos:

- Sistema Operativo: Ubuntu 22 (u otra distribución Linux compatible) o Windows 10/11.
- Memoria RAM: 4 GB (Se recomiendan 8 GB o más para un rendimiento óptimo).
- Procesador: Procesador de doble núcleo a 2.0 GHz o superior.
- Espacio en Disco Duro: Al menos 500 MB de espacio disponible.
- Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

Requisitos Recomendados:

- Sistema Operativo: Ubuntu 22 o Windows 10/11.
- Memoria RAM: 8 GB o más.
- Procesador: Procesador de cuatro núcleos a 2.5 GHz o superior.
- Espacio en Disco Duro: 1 GB de espacio disponible.

◆ Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

Dado que el tamaño del proyecto es relativamente pequeño, no debería tener un impacto significativo en los requisitos del sistema. La cantidad de RAM y el tipo de procesador serán los principales factores por considerar para un rendimiento óptimo, junto con la disponibilidad de espacio en disco y la instalación adecuada del JDK.

Instalación

La instalación del programa en un entorno Ubuntu 22 (u otra distribución Linux compatible) y en Windows 10/11 será bastante similar, dado que el programa está desarrollado en Java y es compatible con ambos sistemas operativos. Aquí tienes una guía básica para la instalación en ambos sistemas:

En Ubuntu 22 (o distribución Linux compatible):

Instalación del JDK 17:

1. Abre una terminal.
2. Ejecuta el siguiente comando para instalar el JDK 17:

   sudo apt-get install openjdk-17-jdk

- **Descarga del programa:**
  - Descarga el archivo del programa desde la fuente proporcionada.
  - Puedes guardar el archivo en tu carpeta de inicio o en cualquier otro directorio de tu elección.

- ¿**Extracción del archivo (si es necesario):**
  - Si el archivo se descargó en formato comprimido (por ejemplo, .zip), deberás extraer su contenido.

- Puedes hacerlo haciendo clic derecho sobre el archivo y seleccionando "Extraer aquí" o utilizando el comando `unzip` en la terminal.

- **Ejecución del programa:**
  - • Abre una terminal en la ubicación donde se encuentra el archivo ejecutable del programa.
  - Ejecuta el programa utilizando el comando `java -jar nombre_del_archivo.jar`.
  - Por ejemplo: java -jar programa.jar

En Windows 10/11:

Instalación del JDK 17:

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

Descarga del programa:

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

Ejecución del programa:

1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión .jar) para ejecutar el programa.

Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

En Windows 10/11:

Instalación del JDK 17:

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

Descarga del programa:

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

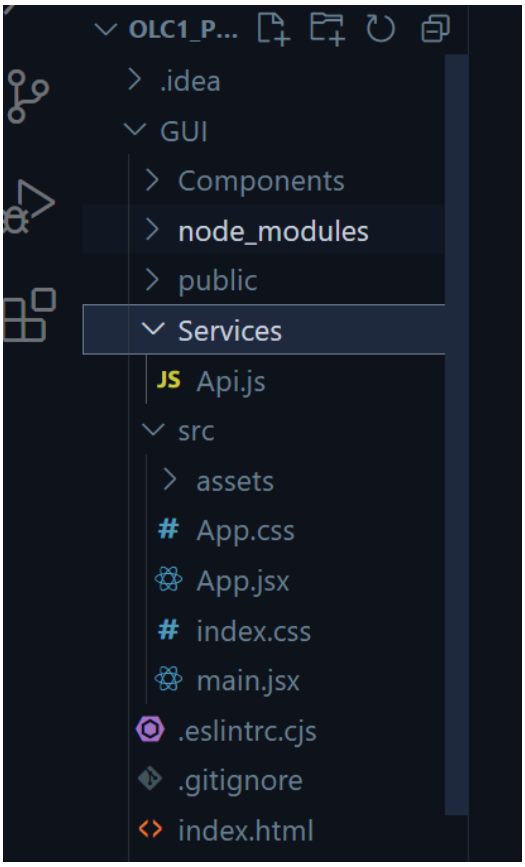
Ejecución del programa:

1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión .jar) para ejecutar el programa.

Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

Arquitectura del código

El proyecto está organizado en los siguientes directorios, donde se incluye información necesaria para comprender el proyecto.



GUI:

En este apartado se puede visualizar lo siguiente:

Editor.jsx:

1. **Importaciones y Hooks:**
    - Se importan los módulos necesarios, como React, useRef y useState.
    - Se declara un componente funcional llamado EditorT.
    - Se inicializan los estados con useState para result, AST, ERRORS y Simbols.
    - Se crea una referencia (editorRef) para el editor de código.
2. **Funciones del Componente:**
    - handleEditorDidMount: Esta función se ejecuta cuando el editor se monta. Guarda la referencia al editor en editorRef.
    - showValue: Realiza una solicitud POST a “http://localhost:3000/interpreter” con el contenido actual del editor. Luego, actualiza los estados con los resultados.
    - getAst: Realiza una solicitud GET a “http://localhost:3000/ast” para obtener el árbol de sintaxis abstracta (AST).
    - getErrors: Realiza una solicitud GET a “http://localhost:3000/errors” para obtener los errores del código.

- getSimbols: Realiza una solicitud GET a “http://localhost:3000/simbols” para obtener los símbolos del código.
- newArchive: Restablece los estados y borra el contenido del editor.
- saveFile: Pide al usuario un nombre de archivo, crea un archivo Blob con el contenido del editor y lo guarda.

3. **Uso de Graphviz y TransformWrapper:**
   - El código también importa Graphviz y utiliza TransformWrapper para manejar el zoom y el desplazamiento.

Public, Api.js:

1. **Función** POST:
   - La función POST acepta dos parámetros: path (la URL a la que se enviará la solicitud) y content (los datos que se enviarán).
   - Dentro de la función:
     - Se muestra el contenido de content en la consola mediante console.log(content).
     - Se realiza una solicitud **POST** a la URL especificada en path utilizando el método fetch.
     - Se envía el contenido como un objeto JSON en el cuerpo de la solicitud.
     - Se establecen las cabeceras para permitir el acceso desde cualquier origen ("Access-Control-Allow-Origin": "*").
     - Se especifica que el tipo de contenido es JSON ("Content-Type": "application/json").
     - Finalmente, se convierte la respuesta en formato JSON y se devuelve.

2. **Función** GET:
    - La función GET acepta un solo parámetro: path (la URL desde la que se obtendrán los datos).
    - Dentro de la función:
        - Se realiza una solicitud **GET** a la URL especificada en path utilizando el método fetch.
        - Se establecen las mismas cabeceras que en la función POST.
        - La respuesta también se convierte en formato JSON y se devuelve.

Estas funciones son útiles para interactuar con servicios web o APIs desde una aplicación de React. El uso de fetch permite realizar solicitudes HTTP de manera sencilla y manejar las respuestas de manera eficiente.

App.css:

1. **Selector** #root**:**
    - Este selector apunta a un elemento con el ID “root”.
    - Las siguientes propiedades se aplican al elemento “root”:
        - max-width: Establece el ancho máximo del elemento en 1280 píxeles.
        - margin: Centra el elemento horizontalmente al establecer los márgenes izquierdo y derecho en “auto”.
        - padding: Agrega 2 rems (unidades em raíz) de relleno a todos los lados del elemento.
        - text-align: Alinea el contenido de texto dentro del elemento al centro.
2. **Clase. logo:**
    - Esta clase apunta a elementos con el nombre de clase “logo”.

- Propiedades aplicadas a elementos con esta clase:
  - height: Establece la altura del elemento en 6em (6 veces el tamaño de fuente).
  - padding: Agrega 1.5em de relleno a todos los lados del elemento.
  - will-change: Indica que la propiedad “filter” cambiará durante las animaciones.
  - transition: Especifica que la propiedad “filter” tendrá una transición de 300 milisegundos al cambiar.
- Al pasar el cursor sobre un elemento con la clase “logo”:
  - La propiedad filter aplica una sombra con un color de #646cffaa.
  - Si el elemento también tiene la clase “react”, el color de la sombra cambia a #61dafbaa.

3. **Animación** @keyframes logo-spin**:**
   - Define una animación personalizada llamada “logo-spin”.
   - La animación rota un elemento desde 0 grados hasta 360 grados.
   - Está destinada a usarse con la clase “logo”.
4. **Consulta de medios** @media (prefers-reduced-motion: no-preference)**:**
   - Esta consulta de medios se dirige a dispositivos donde el usuario no ha reducido explícitamente las preferencias de movimiento.
   - Afecta al segundo enlace (a:nth-of-type(2)) con la clase “logo”.
   - La animación “logo-spin” se aplica infinitamente durante 20 segundos con una función de temporización lineal.
5. **Clase. card:**

- Esta clase apunta a elementos con el nombre de clase “card”.
- La propiedad padding agrega 2em de relleno a todos los lados del elemento.

6. **Clase. read**-the-docs**:**
   - Esta clase apunta a elementos con el nombre de clase “read-the-docs”.
   - El color del texto se establece en #888.

App.jsx:

1. **Importación de módulos**:
   - import {useState} from 'react': Aquí estamos importando la función useState desde el módulo ‘react’. useState es un **hook** de React que nos permite agregar una variable de estado a nuestro componente. Más adelante, veremos cómo se utiliza.
   - import './App.css': Esto importa un archivo CSS llamado ‘App.css’. Es común importar archivos CSS para aplicar estilos a los componentes.
2. **Definición del componente** App:
   - function App () {...}: Aquí estamos definiendo un componente funcional llamado App. Los componentes en React son bloques reutilizables de código que encapsulan una parte específica de la interfaz de usuario.
3. **Estado con** useState:
   - const [count, setCount] = useState(0): Esta línea declara una variable de estado llamada count e inicializa su valor en 0. La función useState devuelve un array con dos elementos: el estado

actual (count en este caso) y una función para actualizar ese estado (setCount). Cuando llamamos a setCount, React re-renderiza el componente con el nuevo valor de count.

4. **Renderizado del componente**:
    - return (...): Aquí estamos devolviendo el contenido del componente. El componente App renderiza un encabezado (<h1>Interfaz</h1>) y otro componente llamado EditorT.
5. **Componente** EditorT:
    - EditorT es un componente personalizado que se encuentra en el archivo "…/Components/Editor.jsx". No se proporciona el código completo aquí, pero se espera que EditorT sea un componente que se renderice dentro de App.

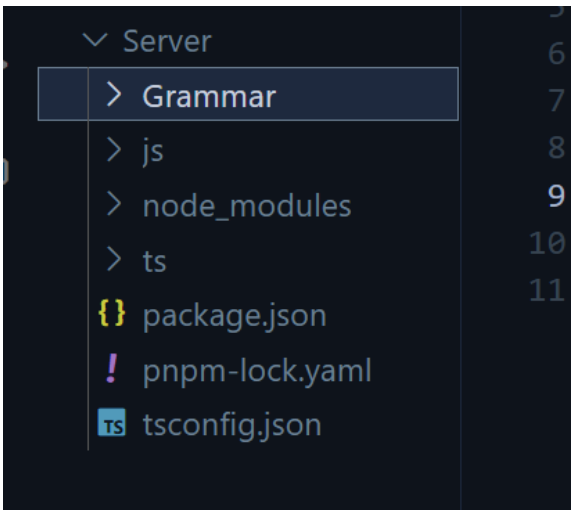
Main:

1. **Importación de módulos:**
    - **import React from 'react'**: Aquí estamos importando el módulo 'react'. React es la biblioteca principal que nos permite crear componentes y manejar el estado de nuestra aplicación.
    - **import ReactDOM from 'react-dom/client':** Estamos importando el módulo 'react-dom/client'. ReactDOM es responsable de renderizar los componentes de React en el DOM (Document Object Model) del navegador.
2. **Importación del componente App:**
    - import App from './App.jsx': Aquí estamos importando el componente App desde el archivo 'App.jsx'. El archivo 'App.jsx' debe contener la definición del componente App.

3. **Estilos:**
    - import './index.css': Esto importa un archivo CSS llamado ‘index.css’. Es común importar archivos CSS para aplicar estilos a los componentes.
4. **Renderizado del componente raíz:**
    - ReactDOM.createRoot(document. getElementById('root')). render(...): Aquí estamos creando un “root” (raíz) de React utilizando createRoot. El elemento con el ID ‘root’ en el HTML será el punto de entrada para nuestra aplicación React.
    - <React.StrictMode> ... </React.StrictMode>: Esto envuelve el componente App en un modo estricto de React. El modo estricto ayuda a detectar problemas potenciales en nuestra aplicación durante el desarrollo.
5. **Renderizado del componente App:**
    - <App />: Aquí estamos renderizando el componente App dentro del root creado anteriormente.

Server:

En este directorio podremos visualizar lo siguiente:



Grammar:

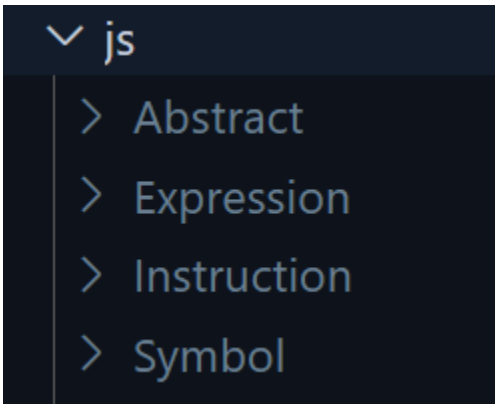
Esta parte de código es generado por **Jison**, una herramienta que nos permite crear analizadores léxicos y sintácticos para lenguajes de programación. Vamos a desglosarlo:

1. **Estructura del analizador**:
    - Parser: Representa un objeto analizador generado por Jison.
    - Parser.prototype: Contiene métodos y propiedades relacionadas con el analizador.
    - yy: Un objeto utilizado para almacenar información adicional relacionada con el análisis.
2. **Analizador léxico (Lexer)**:
    - El analizador léxico se encarga de leer los caracteres de entrada del programa fuente, agruparlos en lexemas y producir una secuencia de tokens.
    - lexer: Representa el analizador léxico generado por Jison.
    - Propiedades y métodos relevantes:
        - EOF: Representa el final del archivo.
        - setInput(input): Establece la entrada para el analizador léxico.
        - input (): Devuelve el siguiente carácter de entrada.
        - lex (): Realiza el análisis léxico y devuelve el siguiente token.
        - rules: Define las reglas para reconocer lexemas y generar tokens.
3. **Analizador sintáctico (Parser)**:
    - El analizador sintáctico verifica que una cadena de tokens pueda generarse con la gramática del lenguaje fuente.

- Parser.prototype.parse(input): Realiza el análisis sintáctico de la entrada.
- productions_: Define las producciones gramaticales.
- performAction(yytext, yyleng, yylineno, yy, yystate, $$, _$): Ejecuta acciones asociadas a las producciones gramaticales.

4. **Opciones y configuraciones**:
   - options: Define configuraciones para el analizador, como el uso de rangos en la información de tokens o el comportamiento del lexer.
   - backtrack_lexer: Si es true, el lexer prueba todas las expresiones regulares para encontrar la coincidencia más larga.
   - ranges: Si es true, la información de ubicación de los tokens incluirá un atributo. range[].
5. **Información de ubicación de tokens**:
   - token location info: Describe la posición de los tokens en el código fuente (líneas, columnas y rangos).

JS:



Asbtract:

Esta es una parte fundamental de un sistema de compilación o interpretación de código. Veamos como se desglosa:

1. Clase Expression e Instruction:
    - Ambas clases representan entidades básicas en el código fuente.
    - Tienen propiedades para registrar la línea y la columna en las que aparecen en el código, lo que es útil para informar errores o realizar seguimientos durante la ejecución.
2. Enumeraciones:
    - dataType: Enumera los tipos de datos posibles que pueden encontrarse en el código. Cada tipo está representado por un valor numérico.
    - ArithmeticOp, RelationalOp, LogicalOp: Enumeran los diferentes tipos de operadores aritméticos, relacionales y lógicos respectivamente, también representados por valores numéricos.
3. Funciones de utilidad:
    - Las funciones getDataTypeName, getArithmeticOpName, getRelationalOpName y getLogicalOpName permiten obtener el nombre legible de los tipos de datos y operadores a partir de sus valores numéricos correspondientes en las enumeraciones. Esto facilita la generación de mensajes de error o la depuración del código

Expression, Arithmetic:

1. Uso de 'use strict':
    - Esta declaración indica que el código debe ejecutarse en modo estricto, lo que significa que se aplicarán ciertas restricciones adicionales y buenas prácticas de programación.
2. Importaciones y definición de la clase Arithmetic:
    - La clase Arithmetic extiende la clase Expression, que forma parte de un conjunto de clases relacionadas con el análisis semántico del código.
    - La clase Arithmetic tiene un constructor que toma como parámetros la expresión izquierda, la expresión derecha, el operador aritmético, la línea y la columna en las que aparece la operación en el código.
    - Esta clase implementa un método interpreter(environment) que realiza la evaluación de la operación aritmética en un entorno específico.
3. Función convertType:
    - Esta función convierte el tipo de datos de los resultados de las expresiones a un formato compatible con las operaciones aritméticas. Por ejemplo, convierte valores booleanos a 0 o 1, y caracteres a su código ASCII.
4. Matrices SUM, RES, MUL, DIV, POW y MOD:
    - Estas matrices determinan los tipos de datos resultantes de las operaciones aritméticas entre diferentes tipos de operandos. Por ejemplo, la matriz SUM indica que la suma entre un número y una cadena resultará en una cadena.
5. Enumeraciones y constantes:

- El código hace uso de enumeraciones y constantes definidas en otros archivos para representar tipos de datos y operadores.

Symbol:

Error.js

1. **Uso de 'use strict':**
    - **Esta declaración indica que el código debe ejecutarse en modo estricto, aplicando ciertas restricciones adicionales y buenas prácticas de programación.**
2. **Definición de la clase Error_:**
    - **La clase Error_ tiene un constructor que toma como parámetros un identificador (id), un tipo (type), un mensaje de error (message), así como la línea (line) y la columna (column) en las que ocurrió el error.**
    - **La clase tiene también un método toString() que devuelve una representación en formato de cadena del error, incluyendo el tipo de error, el mensaje, la línea y la columna donde ocurrió.**
3. **Exportación de la clase:**
    - **La clase Error_ es exportada para que esté disponible para ser utilizada en otros módulos o archivos del programa.**

TablaSimbolos.js

1. **Uso de 'use strict':**

- **Esta declaración indica que el código debe ejecutarse en modo estricto, lo que ayuda a evitar errores comunes y promueve buenas prácticas de codificación.**

2. **Definición de la clase tablaSimbolos:**

   - **La clase tablaSimbolos tiene un constructor que toma como parámetros un número (num), un identificador (id), un tipo (tipo), un tipo 2 (ticpo2), así como la línea (linea) y la columna (columna) donde se encuentra esta entrada en la tabla de símbolos.**

   - **Dentro del constructor, estos parámetros se asignan a las propiedades correspondientes de la instancia de tablaSimbolos.**

3. **Exportación de la clase:**

   - **La clase tablaSimbolos es exportada para que esté disponible para ser utilizada en otros módulos o archivos del programa.**

TS:

Abstract

1. **Importaciones:**

   - **El código importa las clases Result y Environment desde otros archivos de su proyecto. Esto sugiere que estas clases contienen funcionalidades relevantes para el proceso de interpretación o compilación.**

2. **Definición de la clase abstracta Expression:**

   - **Esta clase tiene dos propiedades públicas line y column, que representan la línea y la columna en la que aparece la expresión en el código fuente.**

- **El constructor de la clase toma como parámetros la línea y la columna y las asigna a las propiedades correspondientes.**
- **La clase tiene un método abstracto interpreter(environment: Environment): Result;. Este método debe ser implementado por las clases que extienden Expression y se encarga de interpretar la expresión en un entorno de ejecución dado (Environment). Retorna un objeto de tipo Result, que probablemente contenga el resultado de la interpretación.**
- **La clase también tiene un método abstracto getAst(last: string): string;. Este método también debe ser implementado por las clases hijas. Genera el árbol de sintaxis abstracta (AST) de la expresión, con el parámetro last como el nodo padre del árbol. Retorna una representación en forma de cadena del AST.**

Expression

Veamos los elementos principales:

- **Imports: Importa varias dependencias necesarias para el funcionamiento de la clase, incluyendo clases como Result, Environment, y Expression, entre otros.**
- **Definición de la clase Arithmetic:**
  - **Tiene propiedades públicas left, right, y op, que representan la expresión aritmética izquierda, derecha y el operador respectivamente.**
  - **El constructor inicializa estas propiedades junto con las propiedades line y column, que representan la posición de la expresión en el código fuente.**
  - **Implementa el método interpreter(environment: Environment): Result, que interpreta la expresión aritmética en un entorno dado y devuelve un objeto Result que contiene el resultado de la operación.**

    - **Implementa el método getAst(last: string): string, que genera el Árbol de Sintaxis Abstracta (AST) de la expresión aritmética.**

- **Función convertType(tmpResult: Result): void: Una función auxiliar que convierte el tipo de dato del resultado, principalmente de bool a number, y de char a su correspondiente valor ASCII.**
- **Matrices de Operaciones Aritméticas: Define matrices para determinar el tipo de resultado de las operaciones aritméticas (SUM, RES, MUL, DIV, POW, MOD, UMINUS). Estas matrices ayudan a determinar el tipo de dato resultante de una operación aritmética basándose en los tipos de datos de los operandos involucrados.**

Instruction

- **Imports: Importa las clases y tipos necesarios para su funcionamiento, como Expression, dataType, Instruction, Block, Environment, tError y Error_.**
- **Definición de la clase FN_IF:**
    - **Tiene propiedades condition, blockIf y blockElse, que representan la condición de la instrucción, el bloque de código que se ejecutará si la condición es verdadera, y el bloque de código que se ejecutará si la condición es falsa, respectivamente.**
    - **El constructor inicializa estas propiedades junto con las propiedades line y column, que representan la posición de la instrucción en el código fuente.**
    - **Implementa el método interpreter(environment: Environment): any, que interpreta la instrucción condicional en un entorno dado. Verifica si la condición es booleana y ejecuta el bloque correspondiente (el bloque "if" si la condición es verdadera, o el bloque "else" si la condición es falsa).**

- **Implementa el método getAst(last: string): string, que retorna una cadena vacía porque esta instrucción no contribuye al Árbol de Sintaxis Abstracta (AST).**

Control: