MANUAL TECNICO

Nombre: Josué Nabí Hurtarte Pinto ---------- Carné: 202202481

El proyecto CompInterpreter se desarrolló utilizando una combinación de tecnologías modernas para crear un intérprete eficiente y robusto para un nuevo lenguaje de programación. A continuación, se detallan las herramientas y lenguajes utilizados en el desarrollo del proyecto:

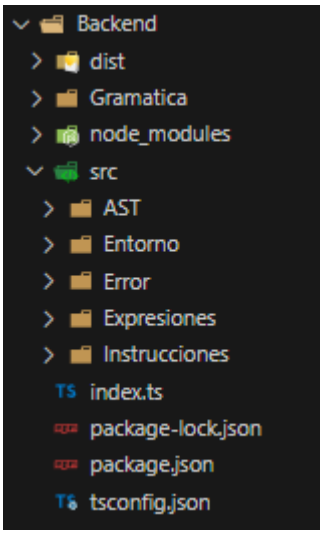
- **Lenguajes de Programación:** El núcleo del intérprete se implementó en TypeScript y JavaScript, aprovechando las ventajas de ambos lenguajes para el desarrollo de aplicaciones robustas y escalables.
- **Frontend:** La interfaz de usuario se desarrolló utilizando React, una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario interactivas y dinámicas.
- **Backend:** El servidor backend se construyó con Express, un framework minimalista y flexible para aplicaciones web en Node.js, que facilita la creación de APIs y el manejo de solicitudes HTTP.
- **Análisis Léxico y Sintáctico:** Para el análisis léxico y sintáctico del lenguaje, se utilizó JISON, una herramienta poderosa que permite generar analizadores léxicos y sintácticos a partir de una gramática definida.

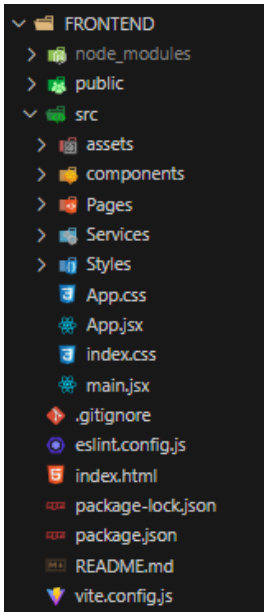
El objetivo principal del proyecto es desarrollar un intérprete capaz de analizar y ejecutar código de manera eficiente, reconociendo correctamente las estructuras del lenguaje, identificando posibles errores y generando reportes útiles. Para lograrlo, se aplicaron técnicas avanzadas de análisis léxico, sintáctico y semántico, aprendidas en el curso de Organización de Lenguajes y Compiladores 1. Este manual técnico está diseñado para proporcionar una visión detallada de la arquitectura del sistema, las decisiones de diseño, y las tecnologías utilizadas, con el fin de facilitar la comprensión y el mantenimiento del proyecto por parte de otros desarrolladores y profesionales del área.

Requerimientos mínimos del entorno de desarrollo

- Node js
- Jison
- React
- Graphviz
- IDE (Entorno de Desarrollo Integrado) en nuestro caso VS Code.
- Navegador Web

Estructura del proyecto:





Analizadores:

Analizador Léxico

Para este analizador se implementó por medio de Jison que nos ayuda a realizar dicha acción por la cual se tuvo que definir expresiones regulares que nos ayuden a poder generar cada palabra reservada. El listado de dichas expresiones regulares, tokens, palabras reservadas:

```
%lex // Inicia parte léxica

%options case-insensitive

%%

// Expresiones regulares

\s+                                     /* IGNORAR ESPACIOS */

"//".*                                  /* IGNORAR COMENTARIOS DE LINEA*/
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]     /* IGNORAR COMENTARIOS DE BLOQUE */

```

```
// tipos de datos
[0-9]+("."[0-9]+)\b                 return 'DOUBLE';
[0-9]+\b                            return 'NUMBER';

// agregando secuencias de escape
\"([^\"\\]|\\[btnr\"\'\\])*\" {
    var texto = yytext.substr(1, yyleng - 2);
    texto = texto.replace(/\\n/g, "\n");
    texto = texto.replace(/\\\\/g, "\\");
    texto = texto.replace(/\\"/g, "\"");
    texto = texto.replace(/\\t/g, "\t");
    texto = texto.replace(/\\r/g, "\r");
    texto = texto.replace(/\\'/g, "'");
    texto = texto.replace(/\\b/g, "\b");
    yytext = texto;
    return 'STRING';
}

\'([^\'\\]|\\[btnr\"\'\\])\' {
    var texto = yytext.substr(1, yyleng - 2);
    texto = texto.replace(/\\n/g, "\n");
    texto = texto.replace(/\\\\/g, "\\");
    texto = texto.replace(/\\'/g, "'");
    texto = texto.replace(/\\t/g, "\t");
    texto = texto.replace(/\\r/g, "\r");
    texto = texto.replace(/\\b/g, "\b");
    yytext = texto;
    return 'CHAR';
}

"true"                              return 'TRUE';
"false"                             return 'FALSE';
"int"|"double"|"bool"|"char"|"string"|NULL  {return 'TIPO';}

([a-zA-Z_][a-zA-Z0-9_]*)\b              return 'ID';
```

Analizador Sintáctico

Para este analizador se implementó también con Jison que nos ayuda a que el programa encuentre todo el orden que debe de seguir el programa para que el texto sea aceptado. Nos ayuda con los errores sintácticos y se inició con la declaración de los terminales los cuales fueron llamados del Lexico

```
// Inicio de gramática
%start ini

// Parte sintáctica  - Definición de la gramática
%%

// La expresión return solamente indica que el análisis terminará allí
ini : instrucciones  EOF { return  new AST($1,errores);}
;
instrucciones : instrucciones instruccion         {  $1.push($2); $$ = $1;}
              | instruccion                       { $$ = [$1];}
;
instruccion
            : funciones                       {$$ = $1;}
            | Variables PYC                   {$$ = $1;}
            | incremento_y_decremento PYC     {$$ = $1;}
            | vectores PYC                    {$$ = $1;}
            | inst_break PYC                  {$$ = $1;}
            | inst_continue PYC               {$$ = $1;}
            | inst_return PYC                 {$$ = $1;}
            | subrutinas                      {$$ = $1;}
            | ejecutar                        {$$ = $1;}
            | error PYC       {errores.push(new Error_(yytext, @1.first_line, @1.first_column, TipoError.SINTACTICO));}
;

// ================ Ejecutar ==================

ejecutar
    : EJECUTAR llamada1 PYC               { $$ = new Ejecutar($2, @1.first_line, @1.first_column); }
;

// ================ Variables ==================

Variables
    : declaracion                     { $$ = $1; }
    | asignacion_var                  { $$ = $1; }
;

declaracion
    : LET lista_var DOSPUNTOS TIPO var_exp     { $$ = new Declaracion($4,$2,$5,false,@1.first_line,@1.first_column);}
    | CONST lista_var DOSPUNTOS TIPO var_exp     { $$ = new Declaracion($4,$2,$5,true,@1.first_line,@1.first_column);}
;

var_exp
    :                                                 { $$ = null; }
    | IGUAL expresion                                 { $$ = $2; }
    ;

lista_var
    : lista_var COMA ID             { $$ = $1; $$.push($3);}
    | ID                            { $$ = [$1];}
;

asignacion_var
    : ID IGUAL expresion           {$$ = new Asignacion($1,$3,@1.first_line,@1.first_column);}
```

AST

Es una parte crucial del proyecto **CompInterpreter**. Este archivo define la clase AST (Abstract Syntax Tree o Árbol de Sintaxis Abstracta), que se encarga de interpretar el código fuente, generar el AST, y producir reportes de errores y símbolos. A continuación, se proporciona una breve explicación del archivo y de las funciones más importantes.

La clase AST se encarga de interpretar las instrucciones del código fuente, generar el AST, y producir reportes de errores y símbolos. Utiliza varias clases auxiliares y estructuras de datos para llevar a cabo estas tareas.

Funciones:

Interpretar(): Esta función interpreta las instrucciones del código fuente. Realiza dos pasadas: la primera para declarar funciones y variables, y la segunda para ejecutar las instrucciones. También genera el AST y agrega los errores léxicos a la lista de errores. Finalmente, genera los reportes de errores, AST y símbolos.

generarReporteAST() : Esta función genera un archivo PDF del AST utilizando Graphviz. Primero escribe el contenido del AST en un archivo .dot, y luego ejecuta el comando dot de Graphviz para generar el archivo PDF.

generarReporteSimbolos(): Esta función genera un reporte HTML de los símbolos. Crea una tabla con las columnas #, ID, Tipo, Tipo2, Línea, y Columna, y agrega una fila por cada símbolo en la lista de símbolos.

generarReporteErrores(): Esta función genera un reporte HTML de los errores. Crea una tabla con las columnas #, Tipo, Descripción, Línea, y Columna, y agrega una fila por cada error en la lista de errores.

Funciones Auxiliares:

Imprimir(): Esta función agrega un valor a la consola, seguido de un salto de línea.

agregarError(): Esta función agrega un error a la lista de errores.

Expresiones:

Define una clase abstracta llamada Expresion, que sirve como base para todas las expresiones en el intérprete. Esta clase contiene propiedades para la línea y columna donde se encuentra la expresión en el código fuente, y define dos métodos abstractos:

calcular: Este método toma un entorno como parámetro y devuelve un resultado. Es utilizado para evaluar la expresión en un contexto determinado.

getAST: Este método toma un identificador de nodo anterior como parámetro y devuelve una cadena que representa el AST (Árbol de Sintaxis Abstracta) de la expresión.

Las clases que hereden de Expresion deberán implementar estos métodos para proporcionar la funcionalidad específica de cada tipo de expresión.

Acceso: Permite acceder al valor de una variable o vector en el entorno. Implementa el método calcular para obtener el valor y getAST para generar el AST de la expresión de acceso.

AccesoVector: Permite acceder a elementos específicos dentro de un vector unidimensional o bidimensional. Implementa el método calcular para obtener el valor del vector y getAST para generar el AST de la expresión de acceso al vector.

Aritmética: Realiza operaciones aritméticas (suma, resta, multiplicación, división, etc.) entre dos expresiones. Implementa el método calcular para evaluar la operación y getAST para generar el AST de la expresión aritmética.

Básico: Representa valores básicos como enteros, decimales, booleanos, caracteres y cadenas. Implementa el método calcular para devolver el valor básico y getAST para generar el AST de la expresión básica.

Casteo: Realiza conversiones de tipo (casteo) entre diferentes tipos de datos. Implementa el método calcular para realizar la conversión y getAST para generar el AST de la expresión de casteo.

Lógicos: Realiza operaciones lógicas (AND, OR, NOT) entre expresiones booleanas. Implementa el método calcular para evaluar la operación lógica y getAST para generar el AST de la expresión lógica.

Relacional: Realiza comparaciones relacionales (igualdad, desigualdad, mayor, menor, etc.) entre dos expresiones. Implementa el método calcular para evaluar la comparación y getAST para generar el AST de la expresión relacional.

Ternario: Implementa el operador ternario que evalúa una condición y devuelve uno de dos valores según el resultado. Implementa el método calcular para evaluar la condición y getAST para generar el AST de la expresión ternaria.

Is: Verifica si el tipo de una expresión coincide con un tipo especificado.

Len: Calcula la longitud de una cadena o un vector.

LowerUpper: Convierte una cadena a minúsculas (LOWER) o mayúsculas (UPPER).

Round: Redondea un número decimal al entero más cercano.

ToCharArray: Convierte una cadena en un arreglo de caracteres.

ToString: Convierte un valor numérico o booleano a una cadena.

Truncate: Trunca un número decimal, eliminando su parte fraccionaria.

Instrucciones

es una clase abstracta que sirve como base para todas las instrucciones en el intérprete. Esta clase define las propiedades y métodos que deben implementar todas las instrucciones específicas.

ejecutar: Este método debe ser implementado por las clases derivadas y se encarga de ejecutar la instrucción en un entorno dado. Puede devolver cualquier valor o null.

getAST: Este método debe ser implementado por las clases derivadas y se encarga de generar una representación del Árbol de Sintaxis Abstracta (AST) de la instrucción. Toma como parámetro un identificador de nodo anterior y devuelve una cadena que representa el AST.

Ciclos

while: Ejecuta un bloque de instrucciones repetidamente mientras una condición booleana sea verdadera. Verifica que la condición sea booleana y maneja instrucciones especiales como break, continue y return.

DoUntil: Ejecuta un bloque de instrucciones al menos una vez y luego repetidamente mientras una condición booleana sea falsa. Verifica que la condición sea booleana y maneja instrucciones especiales como break, continue y return.

For: Ejecuta un bloque de instrucciones un número determinado de veces, basado en una declaración inicial, una condición y un incremento. Verifica que la condición sea booleana y maneja instrucciones especiales como break, continue y return.

Loop: Ejecuta un bloque de instrucciones indefinidamente hasta que se encuentre una instrucción break. Maneja instrucciones especiales como break, continue y return.

Control

If: Evalúa una condición booleana y ejecuta un bloque de instrucciones si la condición es verdadera, con soporte para else y else if. Maneja instrucciones especiales como break), continue) y return).

Switch: Evalúa una expresión y ejecuta el bloque de instrucciones correspondiente a un caso coincidente, con soporte para un caso default. Maneja instrucciones especiales como break), continue) y return).

Break: Representa una instrucción de interrupción de bucle, retornando la propia instrucción para manejar la salida del bucle.

Continue: Representa una instrucción de continuación de bucle, retornando la propia instrucción para manejar la continuación del bucle.

Asignacion: Asigna un nuevo valor a una variable o vector, verificando tipos y manejando errores semánticos.

Bloque: Ejecuta un bloque de instrucciones en un nuevo entorno, manejando instrucciones especiales como break, continue y return.

Declaracion: Declara una o más variables con un tipo específico, opcionalmente asignando un valor inicial.

Echo: Imprime el valor de una expresión en la consola.

Ejecutar: Ejecuta una función previamente definida, verificando su existencia y manejando el retorno de valores.

Función: Declara una función con un tipo de retorno, identificador, parámetros y un bloque de instrucciones.

Incremento_Decremento: Incrementa o decrementa el valor de una variable numérica, verificando tipos y manejando errores semánticos.

Llamada: Llama a una función, pasando los parámetros necesarios y manejando el retorno de valores.

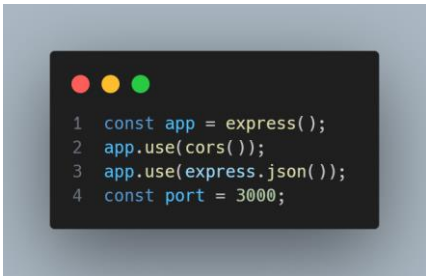
ModificadorVector: Modifica el valor en una posición específica de un vector, verificando tipos y dimensiones.

Return: Representa una instrucción de retorno en una función, devolviendo un valor opcional.

Vector1: Declara un vector unidimensional o bidimensional, inicializándolo con valores predeterminados.

Vectorr2: Declara un vector con valores iniciales, verificando tipos y dimensiones.

Configuración del servidor



```
const app = express();
app.use(cors());
app.use(express.json());
const port = 3000;
```

Express: Se crea una instancia de la aplicación Express.

CORS: Se habilita CORS para permitir solicitudes desde otros dominios.

JSON: Se configura Express para que pueda manejar solicitudes con cuerpos en formato JSON.

Puerto: Se define el puerto en el que el servidor escuchará las solicitudes.

Endpoint para interpretar

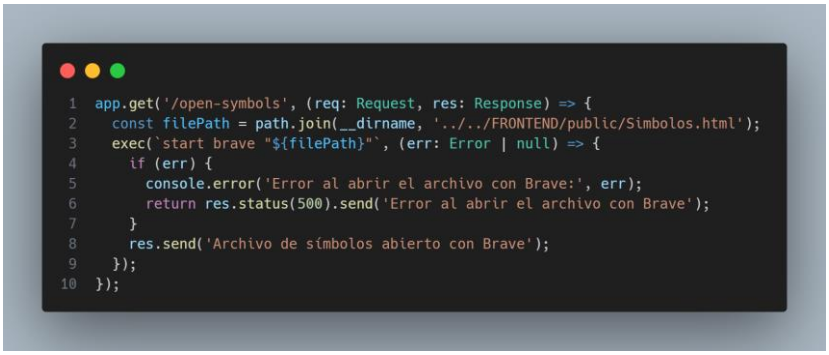
```
app.post('/interpretar', (req: Request, res: Response) => {
  const parametro = req.body["value"];
  const ast: AST = gramatica.parse(parametro); // Retorno de init
  console.log(ast);
  const retorno = ast.interpretar();
  res.json({"mensaje":retorno});
});
```

POST /interpretar: Recibe código en el cuerpo de la solicitud, lo analiza para generar un AST, lo interpreta y devuelve el resultado.

Endpoint para generar y abrir el reporte del AST

```
app.get('/open-ast', (req: Request, res: Response) => {
  const ast = new AST([],[]); // Aquí deberías pasar las instrucciones necesarias
  ast.generarReporteAST();
  const filePath = path.join(__dirname, '../../FRONTEND/public/AST.pdf');
  exec(`start brave "${filePath}"`, (err: Error | null) => {
    if (err) {
      console.error('Error al abrir el archivo con Brave:', err);
      return res.status(500).send('Error al abrir el archivo con Brave');
    }
    res.send('Archivo AST abierto con Brave');
  });
});
```

Endpoint para Abrir el reporte de Símbolos



```
app.get('/open-symbols', (req: Request, res: Response) => {
  const filePath = path.join(__dirname, '../../FRONTEND/public/Simbolos.html');
  exec(`start brave "${filePath}"`, (err: Error | null) => {
    if (err) {
      console.error('Error al abrir el archivo con Brave:', err);
      return res.status(500).send('Error al abrir el archivo con Brave');
    }
    res.send('Archivo de símbolos abierto con Brave');
  });
});
```

Endpoint para Abrir el reporte de Errores

```
app.get('/open-report', (req, res) => {
  const filePath = path.join(__dirname, '../../FRONTEND/public/Errores.html');
  exec(`start brave "${filePath}"`, (err: Error | null) => {
    if (err) {
      console.error('Error al abrir el archivo con Edge:', err);
      return res.status(500).send('Error al abrir el archivo con Edge');
    }
    res.send('Archivo abierto con Edge');
  });
});
```

Inicio del servidor

```
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```