UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1

SECCIÓN “C”

PRIMER SEMESTRE 2023

MANUAL TÉCNICO

Diego Andrés Huite Alvarez

202003585

28/04/2023

Índice

INTRODUCCIÓN

Bienvenido al manual técnico de "TypeWise", un software diseñado analizar un toy-language con sintaxis similar a la de c++.

REQUERIMIENTOS

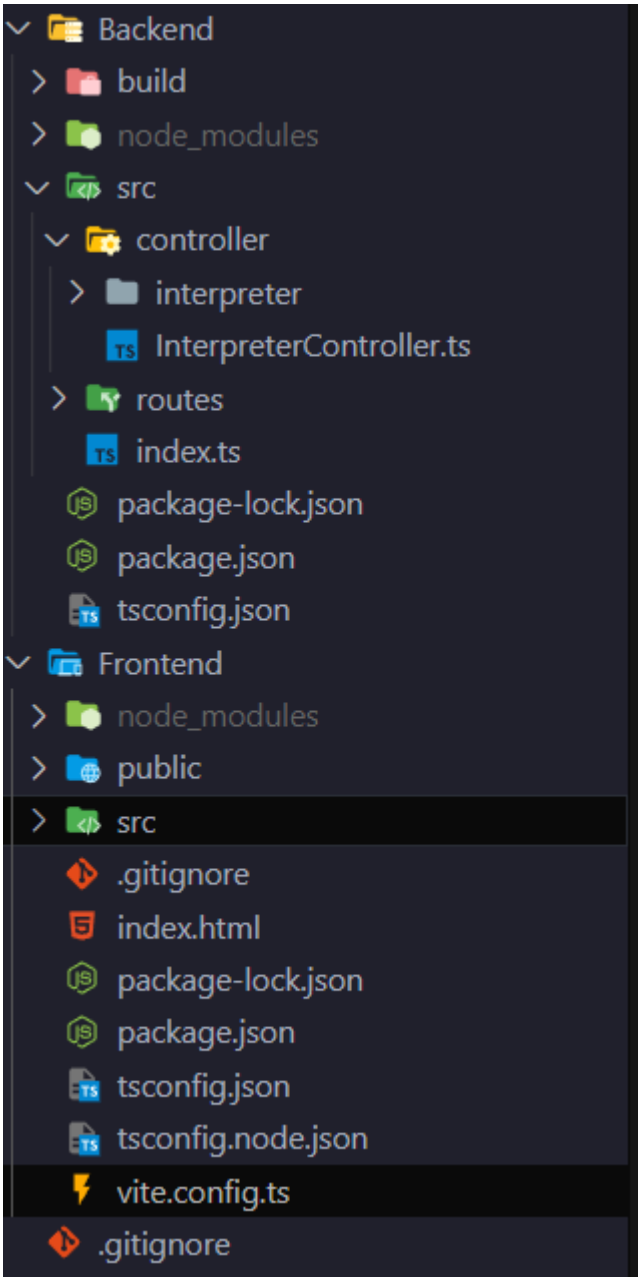
Software:

- Sistema operativo windows, linux o macOS
- Node js y cualquier instalador de paquetes
- Navegador web moderno que soporte react

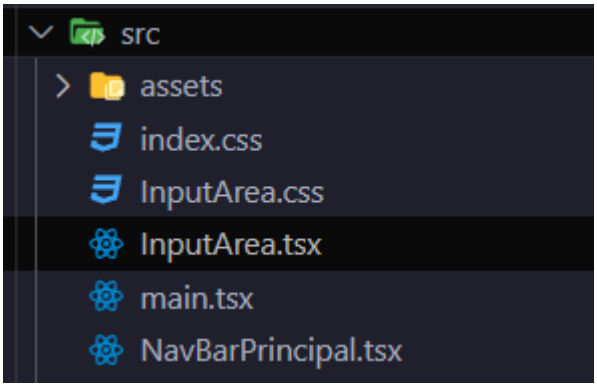
Hardware:

- Mouse
- Teclado
- Monitor
- 2gb de ram

ESTRUCTURA DEL FRONTEND

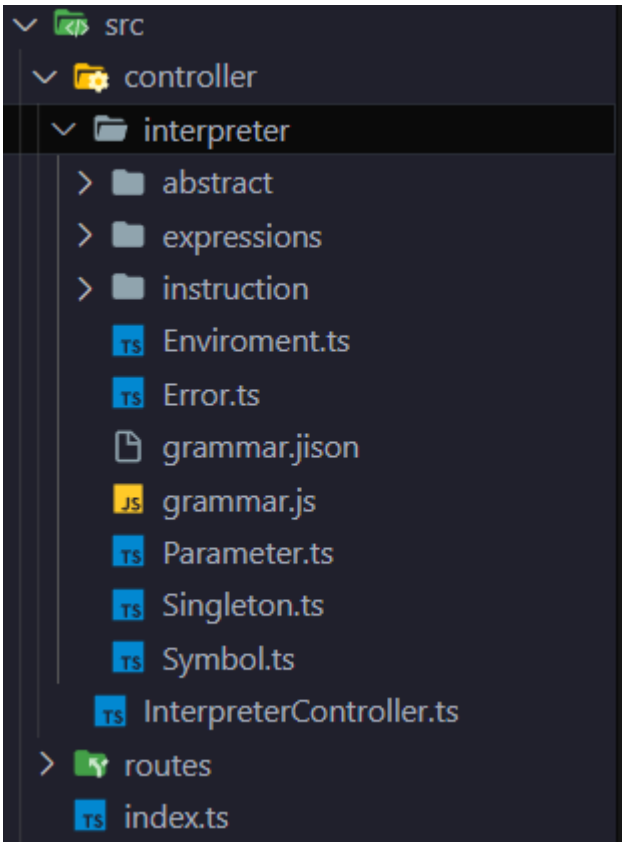


El software está conformado por dos carpetas, siendo una de estas para Frontend y la otra para el Backend. Para el Frontend se hizo uso de react + vite



Y cuenta con los componentes inputArea y NavBarPrincipal. El InputArea es el componente que se encarga de mostrar la consola en el frontend, y el NavBarPrincipal es una navbar hasta arriba de la página.

ESTRUCTURA DEL BACKEND



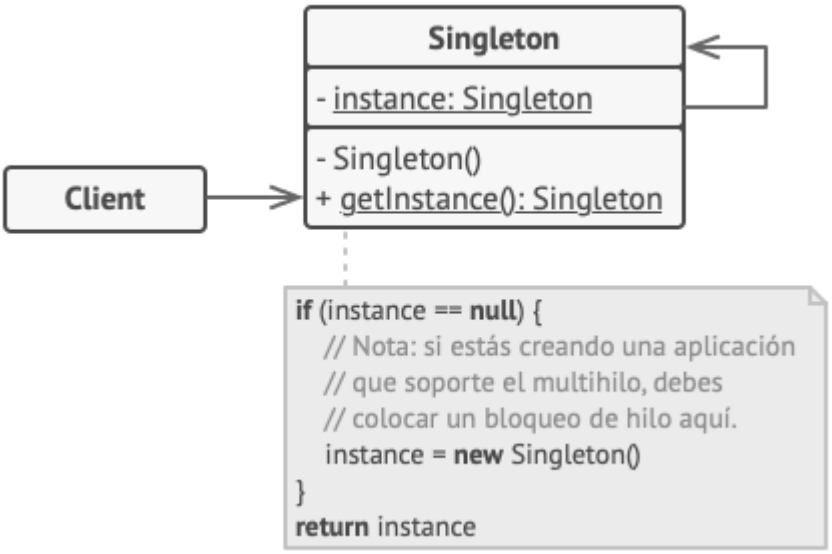
Primeramente, el IntrepreterController.ts es la clase encargada de recibir las peticiones al api y realizar acciones en base a ellas, también está la carpeta de rutas, que se encarga de enrutar hacia el interpretercontroller las rutas, esto principalmente hecho para tener un enrutamiento limpio.

Enviroment: Se encarga de representar un entorno en el lenguaje en desarrollo

Error: Una clase para guardar los errores que se encuentre en el flujo del programa

Parameter: Una clase para llevar el control del tipo de parámetros definidos en una función o método

Singleton: Tal como su nombre lo indica, se hizo uso del patrón singleton para llevar el control de los mensajes que se imprimirán en consola



Symbol: Esta clase se encarga de guardar la información necesaria para el almacenamiento de variables en la clase Enviroment.

ANALISIS LEXICO Y SINTACTICO

Para realizar estos dos tipos de análisis en el lenguaje se hizo uso de la herramienta “Jison”, una librería de javascript para generar un scanner y un parser en base a las expresiones regulares que establezcamos y en baste también al conjunto de producciones que establezcamos.

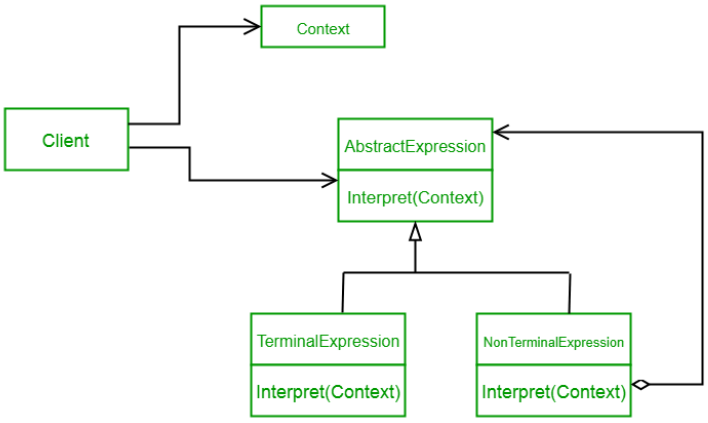
Vista previa del analizador léxico:

```
/* Definición Léxica */
%lex

%options case-insensitive

%x string

%%

/* Whitespaces */
[\r\f\s\t\n]          {}                        // white spaces
\/\/.*                {}                        // oneLineComment
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]   {}        // multilineComment

// reserved words
"int"                 {console.log("Se encontró token con valor: " + yytext); return 'reserved_int';}
"true"                {console.log("Se encontró token true con valor: " + yytext); return 'reserved_true';}
"false"               {console.log("Se encontró token false con valor: " + yytext); return 'reserved_false';}
"double"              {console.log("Se encontró token con valor: " + yytext); return 'reserved_double';}
"boolean"             {console.log("Se encontró token con valor: " + yytext); return 'reserved_boolean';}
"char"                {console.log("Se encontró token con valor: " + yytext); return 'reserved_char';}
"string"              {console.log("Se encontró token con valor: " + yytext); return 'reserved_string';}
"list"                {console.log("Se encontró token con valor: " + yytext); return 'reserved_list';}
"add"                 {console.log("Se encontró token con valor: " + yytext); return 'reserved_add';}
"if"                  {console.log("Se encontró token con valor: " + yytext); return 'reserved_if';}
"else"                {console.log("Se encontró token con valor: " + yytext); return 'reserved_else';}
"print"               {console.log("Se encontró token print con valor: " + yytext); return 'reserved_print';}
"switch"              {console.log("Se encontró token con valor: " + yytext); return 'reserved_switch';}
"case"                {console.log("Se encontró token con valor: " + yytext); return 'reserved_case';}
"default"             {console.log("Se encontró token con valor: " + yytext); return 'reserved_default';}
"break"               {console.log("Se encontró token con valor: " + yytext); return 'reserved_break';}
"while"               {console.log("Se encontró token con valor: " + yytext); return 'reserved_while';}
"for"                 {console.log("Se encontró token con valor: " + yytext); return 'reserved_for';}
"do"                  {console.log("Se encontró token con valor: " + yytext); return 'reserved_do';}
```

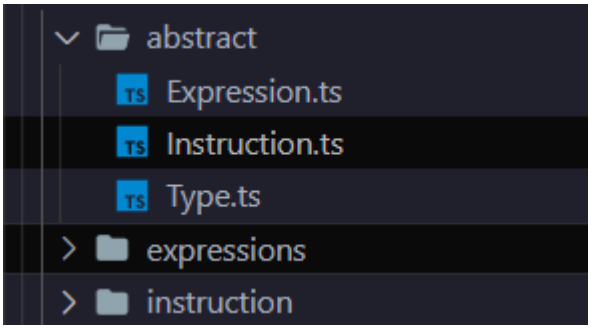
Vista previa analizador sintáctico:

```
%start INICIO

%% /* Definición de la gramática */

INICIO
    : INSTRUCTIONS EOF {console.log('ya entre'); return $1;}
;


// reserved words for type of variables
TYPE:
reserved_char      {$$ = Type.CHAR }
|reserved_boolean  {$$ = Type.BOOLEAN}
|reserved_int      {$$ = Type.INT}
|reserved_double   {$$ = Type.DOUBLE}
|reserved_string   {$$ = Type.STRING}
;


//=============================================================================================================================

// primitive values
OPERAND:  integerNum {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.INT);}
        | decimalNum {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.DOUBLE);}
        | charValue  {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.CHAR);}
        | stringValue {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.STRING);}
        | reserved_false {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.BOOLEAN);}
        | reserved_true  {$$ = new Primitive(@1.first_line,@1.first_column ,$1, Type.BOOLEAN);}
        ;
```

PATRÓN INTERPRETE

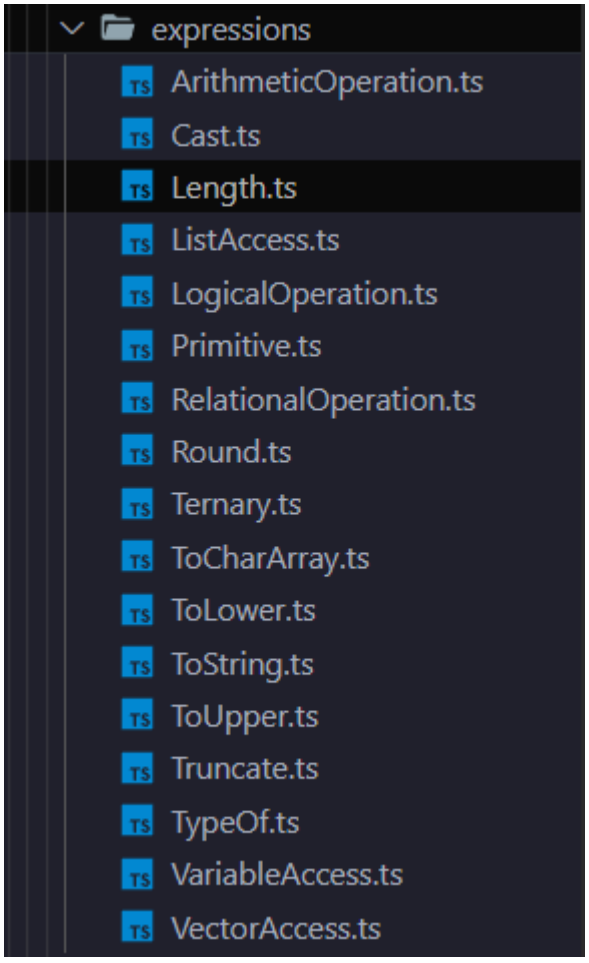


Se hizo uso del patrón interprete para poder llevar el control y flujo de cada uno de las sentencias y expresiones del lenguaje.

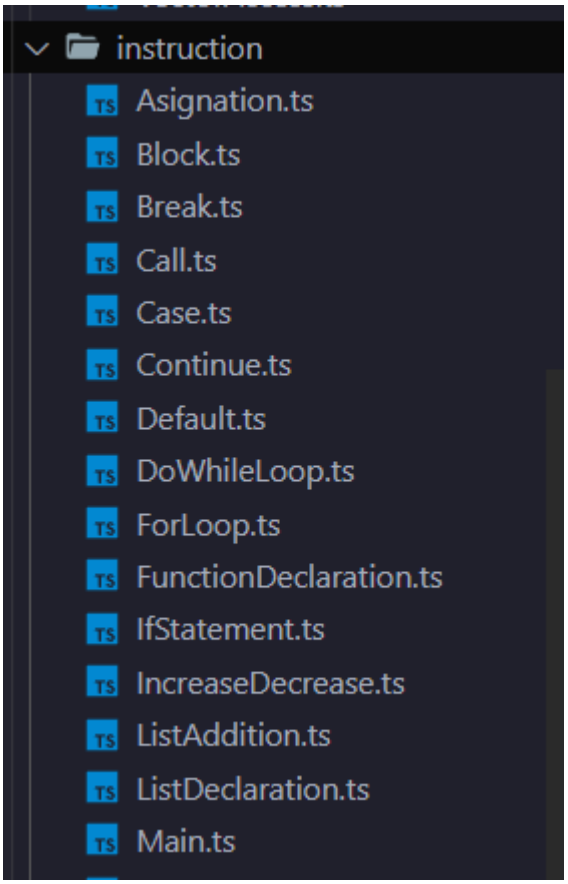


En este caso tenemos dos clases abstractas, una para instrucción y otra para las expresiones.

En la carpeta Expresiones tenemos cada una de las expresiones posibles del lenguaje



Cada expresión hereda de la clase abstracta Expression.ts, y lo mismo sucede con cada una de las instrucciones:



Cada expresión e instrucción poseen un método “execute” que les permite ejecutar la lógica de la expresión o instrucción

```
export class ToString extends Expression {

    private expression: Expression
    constructor(value: Expression, line: number, column: number) {
        super(line, column)
        this.expression = value
    }


    public execute(env: Environment): Return {
        const node = this.expression.execute(env)
        if (node.type === Type.INT || node.type === Type.BOOLEAN){
            return {value:node.value.toString(), type:Type.STRING}
        }


        return { value: "NULL", type: Type.NULL }

    }
```