MANUAL TECNICO

INDICE

Contents

PRESENTACION

El siguiente maual describe los pasos necesarios para cualquier persona que tenga cierta base de programación en el framework de express y typescript, manejo de archivos y entender el funcionamiento del programa, a su vez guiara a los usuarios que haran soporte al sistema, el cual les dará a conocer los requerimientos y la estructura para la construcción del sistema.

OBJETIVOS

General:

- Brindar información necesaria para el uso y manejo del programa, con el fin de que se pueda hacer soporte y modificaciones en caso de ser necesarias.

Especificos:

- Describir las clases abstractas creadas en el programa
- Describir los métodos creados en el programa.

PROCESOS

Procesos de entrada:

Ingresar al programa

Seleccionar el archivo .tw

Analizar el codigo del editor

Procesos de salida:

Reporte de tabla de simbolos

Datos en consola

DESCRIPCION DE LOS ARCHIVOS CREADOS

Archivos para el funcionamiento

El programa usa los siguientes archivos para su funcionamiento:

- App.ts
- Analizador.ts
- Reporte.ts

App.ts

Este archivo contiene las rutas que utiliza la aplicación, en este caso solo hacemos uso de la ruta **"/ejectuar"** el cual usa el método post, en este se envia el código del área de texto del editor. Luego verifica si se selecciono la casilla de tabla simbolos para generar la tabla de simbolos, caso contrario solo genera las salidas del código. Para esta archivo se hizo uso de las siguientes importaciones:

```
import express  from "express";
import path from "path";
import bodyParser from "body-parser";

import { Analizador } from "./Analizador/Analizador";
import { Reporte } from "./Analizador/Reporte";
```

El **"Analizador"** es el que se encarga de analizar el código enviado.

El **"Reporte"** se encarga de generar la tabla de simbolos.

Analizador.ts

Este archivo es una clase **"Analizador"** el cual se encarga de analizar el código enviado desde **"app.ts"** este hace uso del parser generado por jison para generar una salida, el cual es una lista de instrucciones, luego procede a ejecutar cada una de las instrucciones, de ultimo returna una lista con la información que será mostrada en el área de texto de la consola. Este archivo se hizo uso de las siguientes importaciones:

```
import { printlist } from "../Reportes/PrintList";
import { Ambito } from "../Entorno/Ambito";
import { Declarar } from "../Instrucciones/Declarar";
import { Funcion } from "../Instrucciones/Funcion";
import { Metodo } from "../Instrucciones/Metodo";
import { Main } from "../Instrucciones/Main";
```

El **“printlist”** es una lista donde esta toda la información que se mostrara en el área de texto de la consola, antes de retornarla en el método **“Analizar()”** se debe vaciar con un splice.

El **“Ambito”** sirve para crear un nuevo ámbito, el cual se envia a cada instrucción que se ejecuta.

El **“Declarar”** sirve para que al momento de ejecutar las instrucciones se pueda validar si es de tipo Declarar.

El **“Funcion”** sirve para lo mismo que la importación **“Declarar”**

El **“Metodo”** sirve para lo mismo que la importación **“Declarar”**

Reporte.ts

Este archivo es una clase **“Reporte”** el cual tiene la misma lógica que la clase **“Analizar”,** la única diferencia es que este retorna la tabla de símbolos el cual es un Array. Las importaciones son similares al archivo **“Analizador.ts”,** los cuales son los siguientes:

```
import { Ambito } from "../Entorno/Ambito";
import { Declarar } from "../Instrucciones/Declarar";
import { Funcion } from "../Instrucciones/Funcion";
import { Metodo } from "../Instrucciones/Metodo";
import { Main } from "../Instrucciones/Main";
import { ListaTabla } from "../Reportes/Tabla_simbolos";
```

La única diferencia es el import **“ListaTabla”** el cual es el array con los simbolos de las variables declaradas, antes de retornarlo se debe vaicar con un splice.

Clases abstractas

El programa hace uso de dos clases abstractas las cuales son las siguientes:

- Expresion
- Instrucción

Expresion.ts

Este archivo exporta la clase abstracta **“Expesion”** el cual recibe como parámetro la línea y columna. Y tiene el método Get() el cual sirve para obtener el valor de una expresion.

Instrucción.ts

Este archivo exporta la clase abstracta **“Instrucción”** el cual recibe como parámetro la línea y columna, y tiene el método Ejecutar*(, el cual sirve para ejectuar las instrucciones del código.

Ambito.ts

Este archivo no exporta una clase abstracta, pero sirve para manejar los ámbitos en el código. Este se encarga de guardar las variables, listas, vectores, funciones y métodos. Tambien se puede obtener la variable, lista, vector, función o método por su id.

Carpeta Simbolos

Esta carpeta contiene los archivos para las variables, vectores y listas. Asi como también contiene un archivo en el cual se almacenan los tipos que maneja nuestro interprete.

Lista.ts

Este archivo exporta la clase **“Lista”** el cual recibe elementos que serán almacenados, el id y tipo de la lista. Cuenta con las funciones de declarar(), getAtributo(), setAtributo(), addAtributo() y printall().

Return.ts

Este exporta un objeto de tipo Return, el cual almacena el valor y tipo. Este sirve para retornar el valor de una expresión.

TipoPrimitivo.ts

Este archivo exporta un enum que almacena los tipos de datos que maneja el interprete.

Variable.ts

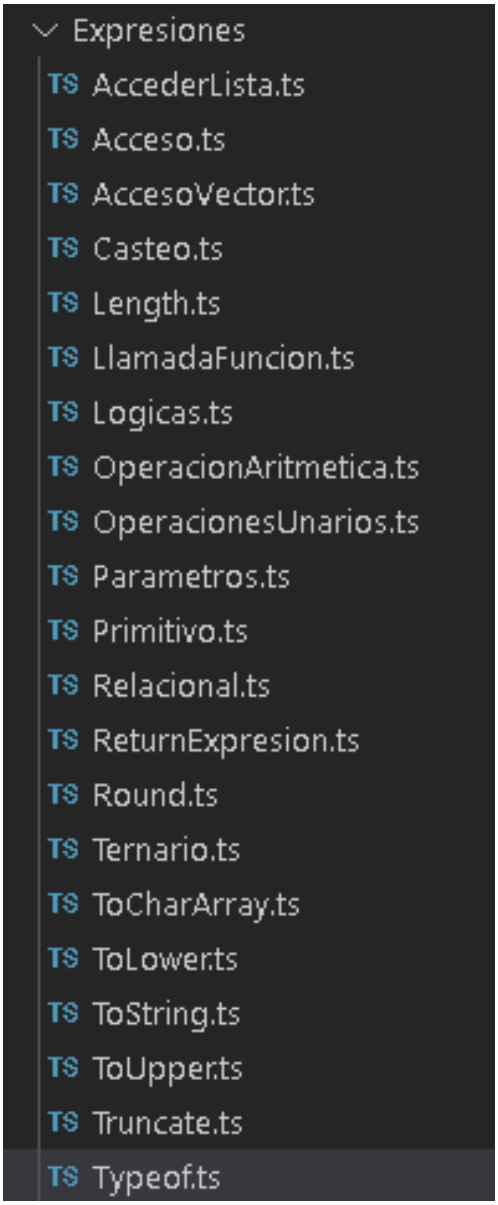
Este archivo exporta la clase **“variable”** el cual recibe el valor, id y tipo de la variable.

Vector.ts

Este archivo exporta la clase **“vector”** el cual recibe los elementos que se guardaran, id, tipo y tamaño del vector. Tiene los métodos declarar_vector, printAll(), getAtributo(), setAtributo().

Carpeta Expresiones

Esta carpeta contiene los archivos de las expresiones que maneja el interprete los cuales son los siguientes:



A continuación se explica mas a detalle cada archivo.

AccederLista.ts

Este archivo exporta la clase "**AccederLista"** el cual extiende la clase abstracta **"Expresion".** Esta clase lo que hace es obtener la lista en el ámbito actual por su id, si lo encuentra se obtiene el valor de la lista en la posición enviada, y de ultimo se retorna el valor y tipo.

Acceso.ts

Este archivo exporta la clase “**Acceso”** el cual extiende la clase abstracta **“Expresion”.** Esta clase obtiene el valor de una variable, lista o vector cuando se asigna a una variable.

AccesoVector.ts

Este archivo exporta la clase “**AccesoVector”** el cual extiende la clase abstracta **“Expresion”.** Esta clase verifica si existe el vector por su id. Si existe obtiene la posición, y de ultimo se obtiene el valor que esta en esa posición y se retorna el valor y tipo.

Casteo.ts

Este archivo exporta la clase “**Casteo”** el cual extiende la clase abstracta **“Expresion”.** Esta clase sirve para hacer los casteos del interprete, verifica que tipo es al que se quiere castear, después hace el casteo y retorna el valor y el tipo.

Lengts.ts

Este archivo exporta la clase “**Lengts”** el cual extiende la clase abstracta **“Expresion”.** Esta clase verifica si la expresión es de tipo string, lista o vector, para retornar su tamaño y tipo, que en este caso seria entero.

LLamadaFuncion.ts

Este archivo exporta la clase “**LlamadaFuncion”** el cual extiende la clase abstracta **“Expresion”.** Esta clase verifica si existe la función declarada. Si existe se verifica que sus parámetros enviados coincidan en cantidad y tipo con la de la función, estos parámetros se guardan. Luego se procede a ejecutar todas las instrucciones que contenga.

Logica.ts

Este archivo exporta la clase “**Logica”** el cual extiende la clase abstracta **“Expresion”.** Esta clase verifica el operador si es “||” o es “&&” para efectuar un or o un and entre ambas expresiones.

OperacionAritemtica.ts

Este archivo exporta la clase “**OperacionAritmetica”** el cual extiende la clase abstracta **“Expresion”.** Esta clase hace las operaciones básicas (suma, resta, multiplicación, división, modulo) y verifica el operador que se esta enviando, luego por medio de una tabla se verifica cual es el tipo del resultado dominante, luego se hace la operación y se retorna su valor y tipo.

OperacionUnario.ts

Este archivo exporta la clase "**OperacionUnario"** el cual extiende la clase abstracta **"Expresion".** Esta clase primero verifica si el operador es "++", si es asi actualiza el valor de la expresión sumado uno. Si el operador es "- -" se actualiza el valor de la expresión restado uno. En ambos casos se retorna su valor y tipo.

Parametros.ts

Este archivo exporta la clase "**Parametros"** el cual extiende la clase abstracta **"Expresion".** Esta clase retorna el valor y tipo de un parámetro.

Primitivo.ts

Este archivo exporta la clase "**Primitivo"** el cual extiende la clase abstracta **"Expresion".** Esta clase retorna el valor y tipo, dependiendo si es int, char, string o boolean.

Relacional.ts

Este archivo exporta la clase "**Relacional"** el cual extiende la clase abstracta **"Expresion".** Esta clase hace la operaciones relaciones entre "==", "!=", "<", ">", "<=" y ">=", verifica cual es el operador, realiza la operación y retorna el valor y el tipo de dato como booleano.

ReturnExpresion.ts

Este archivo exporta la clase "**ReturnExpresion"** el cual extiende la clase abstracta **"Expresion".** Esta clase retorna el valor y tipo que se esté enviando.

Round.ts

Este archivo exporta la clase "**Round"** el cual extiende la clase abstracta **"Expresion".** Esta clae aplica el método Math.round() al valor que se esta enviando, siempre y cuando el tipo sea double, y retorna el resultado y el tipo de dato como entero.

Ternario.ts

Este archivo exporta la clase "**Ternario"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe la condición, la expresión 1 y la expresión 2, aplica una operación ternaria, y retorna el valor obtenido y su tipo.

ToCharArray.ts

Este archivo exporta la clase "**ToCharArray"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expression, y verifica si es de tipo string, si lo es convierte el string en una lista y retorna esa lista y el tipo de dato como Lista.

ToLower.ts

Este archivo exporta la clase "**ToLower"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expresión, verifica si es de tipo string, si lo es aplica un tolowercase y retorna el resultado y el tipo de dato como String.

ToString.ts

Este archivo exporta la clase "**ToString"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expresión y verifica si es de tipo booleano, entero o double, si es asi lo convierte a string y retorna el resultado y el tipo de dato como string.

ToUpper.ts

Este archivo exporta la clase "**ToUpper"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expresión y verifica si es de tipo string, si es asi se le aplica un toUpperCase y retorna el resultado y el tipo de dato como string.

Truncate.ts

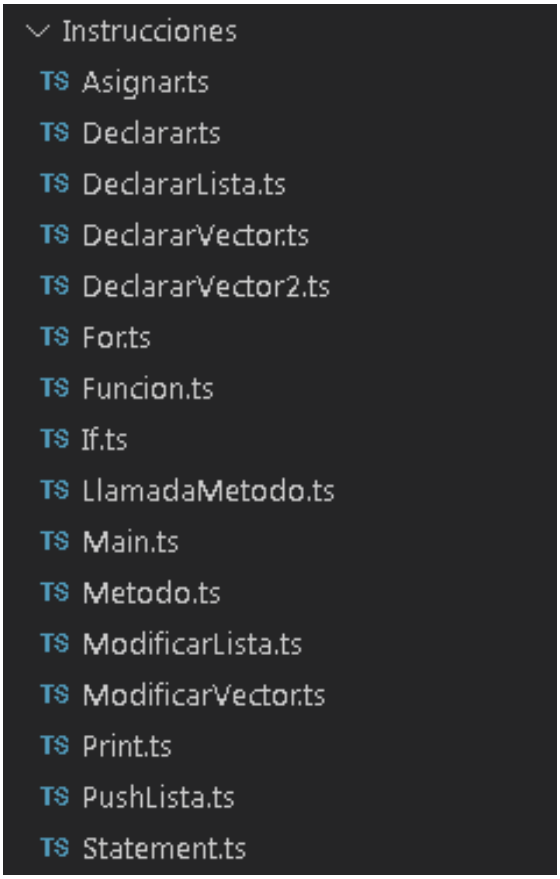
Este archivo exporta la clase "**Truncate"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expresión y verifica si es de tipo double, o de tipo entero, si es asi le aplica el método Math.trunc y retorna su resultado y el tipo de dato como entero.

Typeof.ts

Este archivo exporta la clase "**Typeof"** el cual extiende la clase abstracta **"Expresion".** Esta clase recibe una expresión y verifica que tipo es, dependiendo de que tipo sea va retornar su tipo, por ejemplo si es de tipo Booleano va retornar "booleano" y de tipo de dato retornara string.

Carpeta Instruccines

Esta carpeta contiene los archivos de las instrucciones que tiene el intérprete, las cuales son las siguientes:



A continuación, se explica con más detalle cada archivo:

Asignar.ts

Este archivo exporta la clase **“Asignar”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id y un valor, asigna un valor a una variable y la actualiza.

Declarar.ts

Este archivo exporta la clase **“Declarar”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id, un tipo y un valor, obtiene el valor de la expresión y si no es null guarda la variable.

DeclararLista.ts

Este archivo exporta la clase **“DeclararLista”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id, un tipo y una cadena. Primero verifica que la cadena no sea null, luego se guarda la lista, después se obtiene esa lista guardada para declararla de la segunda manera.

DeclararVector.ts

Este archivo exporta la clase **“DeclararVector”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id, tipo y tamaño. Primero se obtiene el tamaño del vector, luego si el tamaño es de tipo entero se guarda el vector, después se obtiene el vector guardado y si existe se declara el vector.

For.ts

Este archivo exporta la clase **“For”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe una variable de tipo instrucción, una condición de tipo expresión, incremento de tipo instrucción y un código de tipo instrucción. Primero se declara la variable. Luego ejecuta un while, primero se obtiene el valor de la condición. Si la condición es diferente a null y si la condición es false ejecuta la variable y termina, caso contrario ejecuta las instrucciones del código.Tambien aumenta el incremento.

Funcion.ts

Este archivo exporta la clase **“Funcion”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un tipo, id y parametros. Luego se guarda la función en el ámbito actual.

If.ts

Este archivo exporta la clase **“If”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase reicbe una condición, statementIf y statementElse, Primero obtengo la condición, si la condición es de tipo booleano, luego si la condición es true ejecuta las instrucciones del if. Caso contrario mira si hay instrucciones en el statementElse y si las tiene ejecuta las instrucciones.

LlamadaMetodo.ts

Este archivo exporta la clase **“LlamadaMetodo”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id y argumentos de tipo Array de expresiones, y usa la misma lógica de la llamadaFuncion.

Main.ts

Este archivo exporta la clase **“Main”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id, y verifica si es un método o una función, luego aplica la misma lógica de LLamadaMetodo y LlamadaFuncion.

Metodo.ts

Este archivo exporta la clase **“Metodo”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id y un Array de parámetros. Y guarda el método en el ámbito actual.

ModificarLista.ts

Este archivo exporta la clase **“ModificarLista”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id, una expresión y una nueva expresión. Primero se obtiene la lista, luego se obtiene la posición de la lista, si es de tipo entero, le asigna el valor de la nueva expresión a la lista en la posición obtenida.

ModificarVector.ts

Este archivo exporta la clase **“ModificarVector”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un body de tipo Array de instrucciones. Usa la misma lógica de ModificarLista.

Print.ts

Este archivo exporta la clase **“Print”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe una expresión. Se obtiene el valor de la expresión y se almacena en el Printlist.

PushLista.ts

Este archivo exporta la clase **“PushList”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un id y una expresión. Se verifica que la expresión no sea null, luego se obtiene la lista con el id. Despues si existe se agrega el valor a la lista.

Statement.ts

Este archivo exporta la clase **“Asignar”** el cual extiende de la clase abstracta **“Instrucción”.** Esta clase recibe un body de tipo Array de instrucciones. Primero crea un nuevo ámbito, y ejecuta todas las instrucciones del body, si encuentra un return obtiene un valor y se retorna.