Nery Oswaldo Jiménez Contreras Carné: 201700381
Organización de Lenguajes y Compiladores 1 Sección: C

MANUAL TÉCNICO TypeWise

Herramientas

Backend: Node TS

Node es un entorno en tiempo de ejecución multiplataforma de código abierto, se utilizó la versión 10.19.0

Librerías

- Dontenv v10
- Morgan v1.1
- uuid v8.3
- Cors v2.8

Frontend: React TSX

React es una biblioteca de JavaScript open source diseñada para crear interfaces de usuarios con el objetivo de facilitar el desarrollo de aplicaciones en una sola página y renderizando componentes. Se utilizó la versión 17

Se han agregado dos scripts para aumentar el manejo de memoria ram dentro de nuestro proyecto, permitiendo levantarlo a 3 Gb.

```
"start": "react-scripts --max_old_space_size=3072 start",
    "build": "react-scripts --max_old_space_size=3072 build",
```

Librerías

- Graphviz-react v1.2
  Renderiza objetos Graphviz en react.
- react-codemirror-ts v0.0.2
  Editor de texto implementado en JS para navegadores. Se especializa en la edición de código.
- react-table v7.7
  Colección de ganchos para crear tablas potentes y experiencias de cuadrícula de datos.

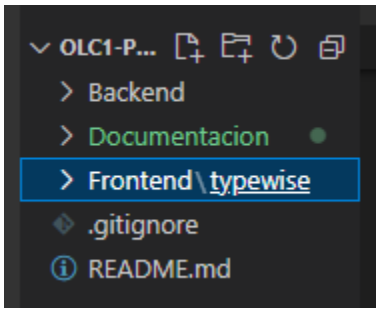
- axios v0.22
  Cliente http ligero basado en el servicio $http.
- react-router-dom v5.4
  Es una colección de componentes de navegación que se componen de forma declarativa con su aplicación.

Entorno de desarrollo

- Visual Studio Code
- Sistema operativo Windows 10
- Procesador core i7
- Memoria ram 8 gb

Estructura del proyecto

El proyecto consta de dos segmentos fundamentales. El Backend, y el Frontend.

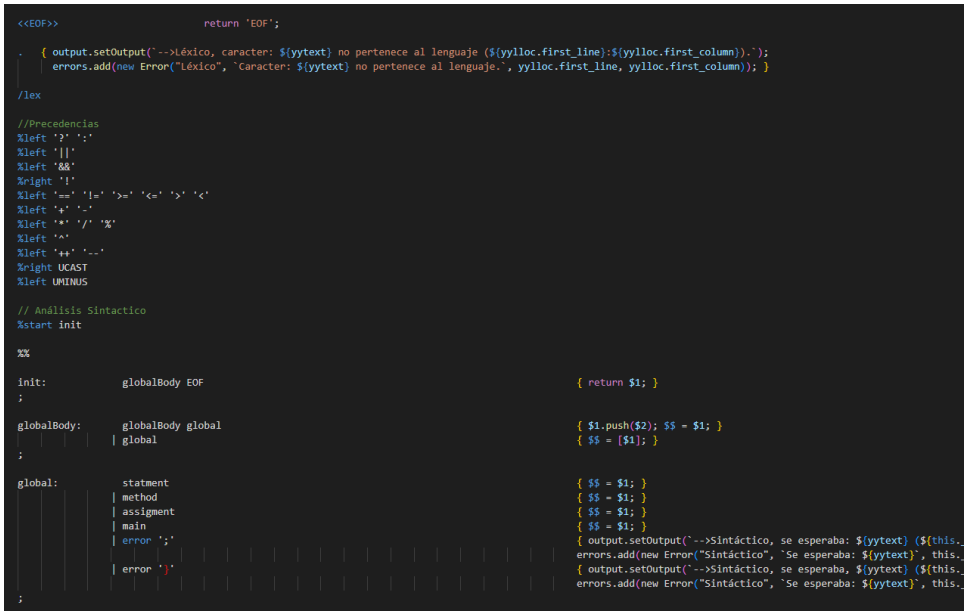


Backend

En este apartado tenemos la gramática de nuestro analizador, las clases abstractas que nos permiten generar el análisis semántico y ejecución del software, y el servidor que levantará la comunicación entre frontend y backend.

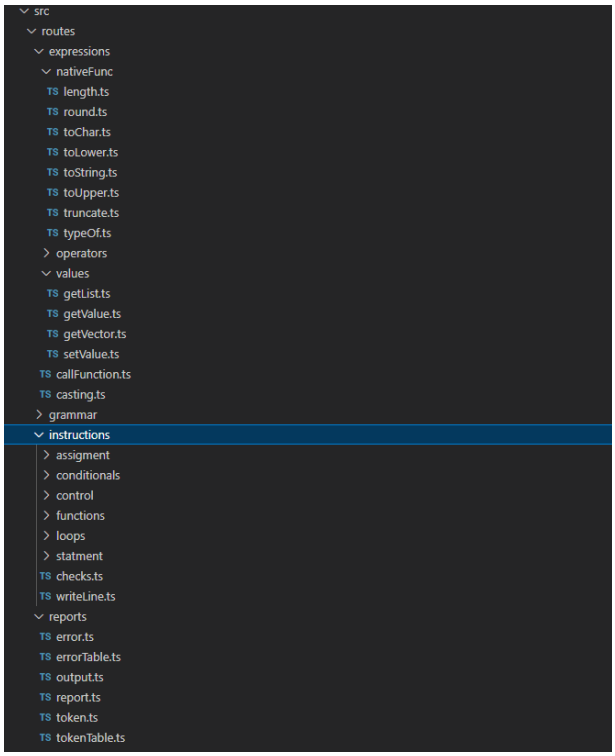
Gramática

Desarrollada con la herramienta JISON, se compone únicamente de un archivo, el cuál cumple la función de análisis léxico y sintáctico.



```
<<EOF>>                     return 'EOF';

.   { output.setOutput(`-->Léxico, caracter: ${yytext} no pertenece al lenguaje (${yylloc.first_line}:${yylloc.first_column}).`);
      errors.add(new Error("Léxico", `Caracter: ${yytext} no pertenece al lenguaje.`, yylloc.first_line, yylloc.first_column)); }

/lex

//Precedencias
%left '?' ':'
%left '||'
%left '&&'
%right '!'
%left '==' '!=' '>=' '<=' '>' '<'
%left '+' '-'
%left '*' '/' '%'
%left '^'
%left '++' '--'
%right UCAST
%left UMINUS

// Análisis Sintactico
%start init

%%

init:           globalBody EOF                                          { return $1; }
;

globalBody:     globalBody global                                       { $1.push($2); $$ = $1; }
              | global                                                  { $$ = [$1]; }
;

global:         statment                                                { $$ = $1; }
              | method                                                  { $$ = $1; }
              | assigment                                               { $$ = $1; }
              | main                                                    { $$ = $1; }
              | error ';'                                               { output.setOutput(`-->Sintáctico, se esperaba: ${yytext} (${this.
                                                                        errors.add(new Error("Sintáctico", `Se esperaba: ${yytext}`, this.
              | error '}'                                               { output.setOutput(`-->Sintáctico, se esperaba, ${yytext} (${this.
                                                                        errors.add(new Error("Sintáctico", `Se esperaba: ${yytext}`, this.
;
```

El proyecto consta de varias clases abstractas, pero para efectos prácticos nos enfocaremos en las principales para su explicación



Run.ts

Se encarga de tomar los valores obtenidos a través del método *main*, generando así el AST, la tabla de símbolos, errores y llamando a las demás clases.

```
import { Main } from '../instructions/functions/main'
import { MethodTable } from '../symbols/methodTable'
import { SymbolTable } from '../symbols/symbolTable'
import { Instruction } from '../types/instruction'
import { errors, output } from '../reports/report'
import { Error } from '../reports/error'

export function getAST(ast:any, globalST:SymbolTable, methods:MethodTable){
    const main:any[] = []
    for(const instruction of ast){
        try {
            if(instruction instanceof Instruction)
                instruction.run(globalST, globalST, methods, "-")

            else if(instruction instanceof Main)
                instruction.run(main)

        } catch (error) {
            if(error instanceof Error) errors.add(error)
        }
    }
    return main
}

export function runMain(main:any, globalST:SymbolTable, localST:SymbolTable, methods:MethodTable){
    if(main.length === 1){
        main[0].run(globalST, localST, methods, main[0].id)
    }
    else if(main.length > 1){
        output.setOutput(`-->Semántico, solo puede existir un metodo main (${main[1].line}:${main[1].column}).`)
        throw new Error(`Semántico`, `Solo puede existir un metodo main`, main[1].line, main[1].column)
    }
    else if(main.length < 1){
        output.setOutput(`-->Semántico, debe de existir un metodo main.`)
        throw new Error(`Semántico`, `Debe de existir un metodo main`, 0, 0)
    }
}
```

```
import { MethodTable } from "../symbols/methodTable"
import { Instruction } from "../types/instruction"

export function getDOT(ast:any, methods:MethodTable, id:string){
    let dot:string = ``
    for(const instruction of ast){
        if(instruction instanceof Instruction){
            const aux = instruction.getAST(methods)
            dot += `${aux.ast}
            ${id} -> ${aux.id}; `
        }
    }
    return dot
}
```

while.ts

Hace uso de la clase run, envía el listado de tokens al ast, verifica las sentencias y distribuye la información que viene dentro de la clase.

```
export class While extends Instruction {

    constructor(private condition:Expression, private body:Instruction[], line:number, column:number){
        super(line, column)
    }

    public run(globalST:SymbolTable, localST:SymbolTable, methods:MethodTable, environment:string){
        var condition = this.condition.run(globalST, localST, methods, environment)
        this.setError(condition)

        while(condition.value){
            var localST2 = new SymbolTable(localST.symbols)
            const control = this.runInstrucctions(globalST, localST2, methods, environment+'_while')
            //Sentencias de control
            if(control !== null){
                if(control.type === ControlType.BREAK){
                    break
                }
                else if(control.type === ControlType.CONTINUE){
                    condition = this.condition.run(globalST, localST, methods, environment)
                    continue
                }
                else if(control.type === ControlType.RETURN){
                    return control
                }
            }
            condition = this.condition.run(globalST, localST, methods, environment)
            this.setError(condition)
        }
    }

    private setError(condition:Value){
        if(condition.type != ValueType.BOOLEAN){
            output.setOutput(`-->Semántico, condicion de if no es de tipo BOOLEAN (${this.line}:${this.column}).`)
            throw new Error("Semántico", `Condicion de if no es de tipo BOOLEAN`, this.line, this.column)
        }
    }
```

Al detectar errores los apila en la tabla de reportes, y envía la información a la generación del AST

```
private setError(condition:Value){
    if(condition.type != ValueType.BOOLEAN){
        output.setOutput(`-->Semántico, condicion de if no es de tipo BOOLEAN (${this.line}:${this.column}).`)
        throw new Error("Semántico", `Condicion de if no es de tipo BOOLEAN`, this.line, this.column)
    }
}

private runInstrucctions(globalST:SymbolTable, localST:SymbolTable, methods:MethodTable, environment:string):any{
    for(var i in this.body){
        const control = this.body[i].run(globalST, localST, methods, environment)
        if(control !== null && control !== undefined){
            return control
        }
    }
    return null
```

Grammar.controller.ts
Centraliza las exportaciones que serán requeridas por el frontend

```ts
import { RequestHandler } from 'express'
import { run, getDot } from './compiler'
import { errors, tokens } from './reports/report'

export const testGrammar:RequestHandler = (req, res) => {
    res.json(run(req.body.code))
}

export const getErrors:RequestHandler = (req, res) => {
    res.json(errors.get())
}

export const getTokens:RequestHandler = (req, res) => {
    res.json(tokens.get())
}

export const getAST:RequestHandler = (req, res) => {
    res.json(getDot())
}
```

Grammar.routes.ts
Alista las peticiones get del backend para ser consumidas

```ts
import { Router } from 'express'
import * as grammarCtrl from './grammar.controller'

const router = Router()

router.post('/', grammarCtrl.testGrammar)

router.get('/Tokens', grammarCtrl.getTokens)

router.get('/Errores', grammarCtrl.getErrors)

router.get('/AST', grammarCtrl.getAST)

export default router
```

Frontend

El frontend tiene sus archivos de configuración para levantar el servidor, y las peticiones que le realizará al backend.

Index.tsx

Prepara la comunicación entre componentes del frontend

```tsx
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter, Route, Switch } from 'react-router-dom'
import reportWebVitals from './reportWebVitals';

import Navbar from './Components/Navbar/Navbar'
import TabBar from './Components/Tabs/TabBar'
import Tokens from './Components/Table/Tokens/Tokens'
import Errors from './Components/Table/Errors/Errors'
import AST from './Components/AST/ast'

ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <Navbar/>
      <Switch>
        <Route exact path="/" component={TabBar} />
        <Route exact path="/Errores" component={Errors} />
        <Route exact path="/Tokens" component={Tokens} />
        <Route exact path="/AST" component={AST} />
      </Switch>
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

ast.tsx y services.ts

Estos dos archivos van de la mano, el ast.tsx es el componente que se renderiza en el servidor, y el services es el encargado de extraer la información del backend por medio de axios.

TS Services.ts

Frontend > typewise > src > Components > AST > TS Services.ts > path

```ts
import axios from "axios";

const path = 'http://localhost:3000/'

export const getDot = async() => {
    return await axios.get(path + 'AST')
}
```

```tsx
import React, { useState, useEffect } from 'react'
import { Graphviz } from 'graphviz-react'
import { getDot } from './Services'

const AST = () => {
    const [ast, setDot] = useState<string>('diagraph g { }')

    const loadDot = async () => {
        const res = await getDot()
        setDot(res.data)
    }

    useEffect(() => {
        loadDot()
    }, [])

    return (
        <div>
            <Graphviz
            options={{
                fit: true,
                height: 1000,
                width: 2130,
                zoom: true
            }}
            dot={ast} />
        </div>
    )
}

export default AST
```

CodeEditor.tsx

Este componente renderiza la consola dentro del frontend, envía la información al backend y muestra las respuestas del mismo.

```tsx
const CodeEditor = () => {
    const [code, setCode] = useState<string>("")
    const [output, setOutput] = useState<string>("")
    const [fileName, setFielName] = useState<string>("Typewise.tw")

    const sendCode = async () => {
        const res = await getOutput(code)
        setOutput(res.data.toString())
    }

    const handleClick = () => {
        setOutput("")
        sendCode()
    }

    const handleFile = (e:any) => {
        let files = e.target.files
        setFielName(files[0].name)
        let reader = new FileReader()
        reader.onload = (i:any) => {
            const text = i.target.result
            setCode(text)
        }
        reader.readAsText(files[0])
    }

    const handleSave = () => {
        download(fileName, code)
    }
```

```tsx
import axios from 'axios'

const path = 'http://localhost:3000/'

export const getOutput = async(code:string) => {
    return await axios.post(path, {code: code})
}
```

Tokens.tsx

Renderiza el apartado de la tabla de símbolos dentro del frontend.

```
const loadTokens = async () => {
    const res = await getTokens()
    setTokens(res.data)
}

const columns = useMemo(() => getColumns(), [])
const data = tokens

useEffect(() => {
    loadTokens()
}, [])

const {
    getTableProps,
    getTableBodyProps,
    headerGroups,
    rows,
    prepareRow
  } = useTable<Token>({ columns, data }, useSortBy);

return (
<div className="app-containe">
    <table {...getTableProps()}>
        <thead>
            {headerGroups.map(headerGroup => {
                return(
                <tr {...headerGroup.getHeaderGroupProps()}>
                    {headerGroup.headers.map(column =>{
                        return(
                            <th {...column.getHeaderProps(column.getSortByToggleProps())}
                                className={
                                    column.isSorted
                                    ? column.isSortedDesc
                                    ? "desc"
                                    : "asc"
                                    : ""
                                }>
                                {column.render("Header")}
                            </th>
                        )
                    })}
                </tr>
            )})}
        </thead>
```

```
import axios from 'axios'

const path = 'http://localhost:3000/'

export const getTokens = async() => {
    return await axios.get(path  + 'Tokens')
}

export const getErrors = async() => {
    return await axios.get(path + 'Errores')
}
```

Diseño del frontend