Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1

MANUAL TÉCNICO

Henry Ronely Mendoza Aguilar
Carné: 202004810
PROYECTO 2

Objetivos y Alcances del Sistema

Objetivo Principal

- Brindar la solución óptima para la compilación de un lenguaje de programación empleando conocimientos de análisis léxico y sintactico.

Objetivos Específicos

- Diseñar una plataforma agradable e intuitiva por medio del lenguaje de programación JavaScript y Librería React JS.
- Diseñar un servidor empleando Node JS
- Programar un sistema funcional para cualquier usuario y hardware en el que se utilice.

Requisitos del Hardware

- 512 MB de RAM
- 20MB de disco duro para almacenamiento del software (ya que puede aumentar con la generación de imágenes).
- Procesador Pentium 2 a 266Mhz

Requisitos del Software

- Node JS
- Graphviz
- Windows 10 (64-bit), Linux o Mac OS
- Google Chrome o Mozilla Firefox

Librerías Empleadas

Graphviz:

Graphviz (abreviatura de Graph Visualization Software) es un paquete de herramientas de código abierto iniciado por AT&T Labs Research para dibujar gráficos especificados en scripts de lenguaje DOT que tienen la extensión de nombre de archivo "gv". También proporciona bibliotecas para que las aplicaciones de software utilicen las herramientas. Graphviz es un software gratuito con licencia de Eclipse Public License.

Jison:

Jison toma una gramática libre de contexto como entrada y produce código JavaScript capaz de parsear el lenguaje descrito por dicha gramática. Una vez se tenga el script generado podemos usarlo para parsear la entrada y aceptarla, rechazarla o ejecutar acciones con base en la entrada. Si se está familiarizado con Bison, Yacc o algún otro similar ya se está listo para iniciar. Jison genera tanto el analizador léxico como el analizador sintáctico.

Node JS

Node.js es un entorno de tiempo de ejecución JavaScript back-end , multiplataforma y de código abierto que se ejecuta en el motor V8 y ejecuta código JavaScript fuera de un navegador web . Node.js permite a los desarrolladores usar JavaScript para escribir herramientas de línea de comandos y para secuencias de comandos del lado del servidor: ejecutar secuencias de comandos del lado del servidor para producir contenido de página web dinámico antes de que la página se envíe al navegador web del usuario. En consecuencia, Node.js representa un paradigma de "JavaScript en todas partes", [6] que unifica las aplicaciones web desarrollo en torno a un único lenguaje de programación, en lugar de diferentes lenguajes para scripts del lado del servidor y del lado del cliente.

Node.js tiene una arquitectura basada en eventos capaz de E/S asíncrona. Estas elecciones de diseño apuntan a optimizar el rendimiento y la escalabilidad en aplicaciones web con muchas operaciones de entrada/salida, así como para aplicaciones web en tiempo real (por ejemplo, programas de comunicación en tiempo real y juegos de navegador).

React JS

React (también conocido como React.js o ReactJS) es una biblioteca de JavaScript front-end gratuita y de código abierto para crear interfaces de usuario basadas en componentes de interfaz de usuario. Lo mantiene Meta (anteriormente Facebook) y una comunidad de desarrolladores y empresas individuales. [4] [5] [6] React se puede utilizar como base en el desarrollo de aplicaciones de una sola página, móviles o renderizadas por servidor con marcos como Next.js. Sin embargo, React solo se preocupa por la gestión del estado y la representación de ese estado en el

DOM. por lo que la creación de aplicaciones React generalmente requiere el uso de bibliotecas adicionales para el enrutamiento, así como ciertas funciones del lado del cliente.

Arquitectura Cliente-Servidor

El modelo cliente-servidor es una estructura de aplicación distribuida que divide tareas o cargas de trabajo entre los proveedores de un recurso o servicio, denominados servidores, y los solicitantes del servicio, denominados clientes . [1] A menudo, los clientes y los servidores se comunican a través de una red informática en hardware independiente, pero tanto el cliente como el servidor pueden residir en el mismo sistema. Un anfitrión del servidor ejecuta uno o más programas de servidor, que comparten sus recursos con los clientes. Un cliente normalmente no comparte ninguno de sus recursos, pero solicita contenido o servicio de un servidor. Los clientes, por lo tanto, inician sesiones de comunicación con los servidores, que esperan las solicitudes entrantes. Ejemplos de aplicaciones informáticas que utilizan el modelo cliente-servidor son el correo electrónico, la impresión en red y la World Wide Web.

SIngleton

En ingeniería de software, el patrón singleton es un patrón de diseño de software que restringe la creación de instancias de una clase a una instancia "única". Esto es útil cuando se necesita exactamente un objeto para coordinar acciones en todo el sistema. El término proviene del concepto matemático de singleton.

El patrón de diseño singleton es uno de los veintitrés patrones de diseño conocidos de la "banda de los cuatro" que describen cómo resolver problemas de diseño recurrentes para diseñar software orientado a objetos flexible y reutilizable con el objetivo de facilitar la implementación, el cambio y la implementación. probar y reutilizar objetos.

El patrón de diseño singleton resuelve problemas al permitirle:

- Asegúrese de que una clase solo tenga una instancia
- Acceda fácilmente a la única instancia de una clase
- Controlar su instanciación
- Restringir el número de instancias
- Acceder a una variable global

BACKEND

Para el desarrollo del backend se utilizaron las librerías de Express para la realización de las peticiones http

```
const express = require('express')
const morgan = require('morgan')
const cors = require('cors')
const routes = require('./routes')

const app = express()

app.set('port', process.env.PORT || 9000)

app.use(express.json())
app.use(cors())
app.use(morgan())

app.listen(app.get('port'), ()=>{
    console.log('server running on port', app.get('port'))
})


app.use('/',routes)
```

Asu vez se implementó la exportación de rutas para tener un mejor orden al momento de realizar las peticiones.

```
const express = require('express')
const routes = express.Router()
const ENV = require('../src/Symbol/Env')
const Parser = require('../src/Grammar/grammar')
const Singleton = require('../src/Pattern/Singleton')
const fs = require('fs')
const exec = require('child_process')

var entrada = "";

var s = Singleton.Singleton.getInstance()
var env = new ENV.ENV(null)
var eject = false
var index = 0

routes.get('/', (req, res)=>{
    res.json({mensaje:"Hola"})
})

routes.get('/entrada', (req, res)=>{
    res.json({mensaje:s.getConsola()})
})

routes.post('/entrada', (req, res)=>{
    entrada = ""
    entrada = req.body.mensaje;
    s.reset()
    let ast = Parser.parse(entrada)
```

Uso de Jison

El analizador para el compilador se implementó dentro del backend por medio de la sintaxis propuesta por jison se desarrolló la gramática.

```
INSTRUCCION
    : DECLARACION "puntocoma"   {$$ = $1}
    | ASIGNACION "puntocoma"    {$$ = $1}
    | INDECRE "puntocoma"       {$$ = $1}
    | MODIFICACIONV "puntocoma" {$$ = $1}
    | IFSENTENCIA               {$$ = $1}
    | SWITCHSENTENCIA           {$$ = $1}
    | CICLOWHILE                {$$ = $1}
    | CICLOFOR                  {$$ = $1}
    | CICLODOWHILE              {$$ = $1}
    | FUNCTION
    | METODO                    {$$ = $1}
    | LLAMADA "puntocoma"       {$$ = $1}
    | FUNPRINT "puntocoma"      {$$ = $1}
    | FUNPRINTLN "puntocoma"    {$$ = $1}
    | EJECUTAR "puntocoma"      {$$ = $1}
    | error                     {console.log("Error sintactico "+$1+" linea "+@1.first_line+" columna "+@1.first_
                                var s = Singleton.getInstance()
                                s.addError(new Errores("Sintactico",$1,@1.first_line,@1.first_column))}
    |                           {$$ = null}
;

EJECUTAR
    : "run" "identificador" "parena" LISTAEXP "parenc"  {$$ = new RUN($2,$4,@1.first_line, @1.first_column)}
    | "run" "identificador" "parena" "parenc"           {$$ = new RUN($2,null,@1.first_line, @1.first_column)}
```

Ejecución de Instrucciones

Para la ejecución de instrucciones se utilizó un modelo de clases abstractas para brindar diferentes métodos dependiendo de la situación del programa.

```
export abstract class Expression {

    constructor(public line: number, public column: number) {
        this.line = line
        this.column = column + 1
    }

    public abstract run(env:ENV):Return

    public abstract save(env:ENV):any

    public abstract ast(line:number,column:number):string
}
```

Implementación de clases

Para cada instrucción se decidió crear una clase que ejecutaría sus funciones dependiendo el tipo ya sea Instrucción, Expresión o Función.

```
export class Asignacion extends Instruction {
    constructor(
      public nombre: string,
      public expresion:Expression,
      line: number,
      column: number
    ) {
      super(line, column);
    }

    public run(env: ENV) {

      let exp= this.expresion.run(env)
      var s = Singleton.getInstance()
      if(env.search(this.nombre)){
        if (env.getType(this.nombre) == exp.type){
          env.setVar(this.nombre, exp.value)
        }else{
          s.addError(new Errores("Semantico","Tipo de dato incompatible, n
        }
      }else{
        s.addError(new Errores("Semantico","La variable no ha sido creada"
      }

    }

    public save(env: ENV) {}
```

Creación de Ambiente

Para manejar los datos se utilizó la clase ENV que realiza diferentes métodos para el control de la tabla de símbolos del compilador

```
export class ENV{
    private tablaSimbolos: Map<string, Symbol>;

    constructor(public anterior: ENV | null) {
        this.tablaSimbolos = new Map();
    }

    public getEnv(){
        return this.tablaSimbolos
    }

    public saveVar(nombre: string, valor: any, type: Type,ins:Array
        if(!this.search(nombre)){
          this.tablaSimbolos.set(nombre, new Symbol(valor, nombre,
          return true
        }
        return false
    }
```

Uso de Singleton

Se implementó el patrón singleton para el almacenamiento de errores, las salidas de la consola y la ejecución del árbol AST

```
export class Singleton{

    private static instance: Singleton

    private consola: string = ""
    private symbols:Map<string, Symbol> = new Map()
    private errores:Array<Errores> = new Array()
    private ast:string = "nodoPrincipal[label = \"Lista Instrucciones\"];\n"

    private constructor() { }

    public static getInstance(): Singleton {
        if (!Singleton.instance) {
            Singleton.instance = new Singleton();
        }
        return Singleton.instance;
    }

    public addConsola(data:string){
        this.consola+= data
    }
    public getConsola():string{
        return this.consola
    }
```

FRONTEND

El frontend se basó principal mente en el uso de react y las diferentes herramientas que posee.

```
import React, {useState, useContext} from "react";
import { Container, Info, InputDiv, Name,Send ,Entry} from "./EntradaElements";
import { FaPlay } from "react-icons/fa";
import { IconContext } from "react-icons";
import { FiRefreshCw } from "react-icons/fi";
import { InputContext } from './tools/Context'

function Entrada(){

    const {EntradaState,setEntradaState} = useContext(InputContext)

    const [entrada, setentrada] = useState({
      mensaje: ""
    })

    console.log(EntradaState)
    console.log(entrada)
    const handleChange = e => {
      setentrada({
          ...entrada,
          [e.target.name]: e.target.value
      })
      setEntradaState(e.target.value)
      console.log(entrada.mensaje)
    }
    let {mensaje} = entrada
```

Se utilizaron las llamadas fetch para realizar las peticiones al BackEnd

```
//consulta
const requestInit = {
  method: 'POST',
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify(json)
}
fetch('http://localhost:9000/entrada', requestInit)
.then(res => res.text())
.then(res => console.log(res))

//reiniciando state
setentrada({mensaje : EntradaState.toString()})
alert('Se ha creado el curso correctamente')
```