Universidad de San Carlos de Guatemala

Facultad de ingeniería

Escuela de Ciencias y Sistemas

Organización de lenguajes y compiladores 1

Manual técnico

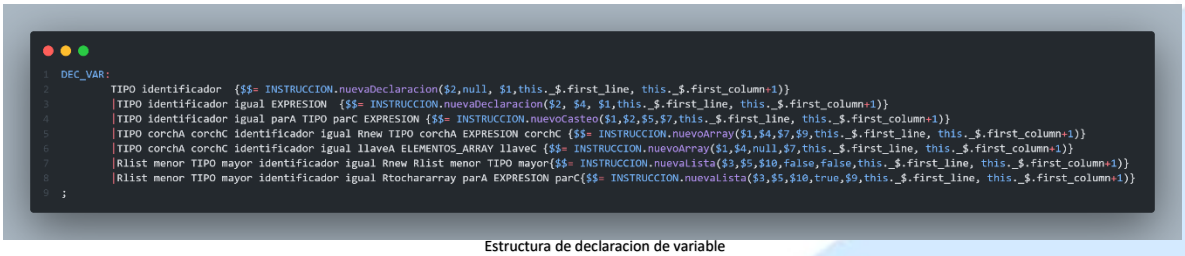
Julio Alfredo Fernández Rodríguez

201902416

TypeWise

El objetivo de este proyecto es aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la realización de un intérprete sencillo, con las funcionalidades principales para que sea funcional.

El proyecto fue desarrollado en JavaScript y se utilizo nodejs para poder modularizar el código ,express de nodejs para el desarrollo de la api y jison para el desarrollo de la gramatica.La parte fundamental del proyecto es la gramatica ya que es aca donde se definen todos los tokens y las estructuras de reconocimiento.



```
DEC_VAR:
        TIPO identificador  {$$= INSTRUCCION.nuevaDeclaracion($2,null, $1,this._$.first_line, this._$.first_column+1)}
        |TIPO identificador igual EXPRESION  {$$= INSTRUCCION.nuevaDeclaracion($2, $4, $1,this._$.first_line, this._$.first_column+1)}
        |TIPO identificador igual parA TIPO parC EXPRESION {$$= INSTRUCCION.nuevoCasteo($1,$2,$5,$7,this._$.first_line, this._$.first_column+1)}
        |TIPO corchA corchC identificador igual Rnew TIPO corchA EXPRESION corchC {$$= INSTRUCCION.nuevoArray($1,$4,$7,$9,this._$.first_line, this._$.first_column+1)}
        |TIPO corchA corchC identificador igual llaveA ELEMENTOS_ARRAY llaveC {$$= INSTRUCCION.nuevoArray($1,$4,null,$7,this._$.first_line, this._$.first_column+1)}
        |Rlist menor TIPO mayor identificador igual Rnew Rlist menor TIPO mayor{$$= INSTRUCCION.nuevaLista($3,$5,$10,false,false,this._$.first_line, this._$.first_column+1)}
        |Rlist menor TIPO mayor identificador igual Rtochararray parA EXPRESION parC{$$= INSTRUCCION.nuevaLista($3,$5,$10,true,$9,this._$.first_line, this._$.first_column+1)}
;
```

Estructura de declaracion de variable

Para procesar las estructuras se definió un archivo js llamado Instrucción.js donde recibe las partes importantes de la estructura como por ejemplo si se va a analizar int edad=23; la gramática enviara al archivo el tipo de dato (int) el identificador (edad) y el valor (23) y este retorna un diccionario con una llave respecto a los que se le este asignando,por ejemplo para retornar un identificador(id: identificador) para que al analizar todas estructuras se pueda claseificar según su tipo,y asi hacer las operaciones necesarias para su funcionamiento.

```
nuevaDeclaracion: function (_id, _valor,_tipo, _linea, _columna) {
        return{

            tipo: TIPO_INSTRUCCION.DECLARACION,
            id: _id,
            valor: _valor,
            tipo_dato: _tipo,
            linea: _linea,
            columna: _columna
        }
    }
```

Estructura declaracion de variable

Luego de procesar todas las instrucciones se prodece a analizarlas en el archivo llamado bloque,donde este analiza todas y cada una de ellas y las clasificara.Por ejemplo al declarar una variable el bloque procesara la instrucción tipo declaración de variable y enviara dicha instrucción al archivo declaración.js para que este agregar la misma al ámbito.

```
else if (instruccion.tipo === TIPO_INSTRUCCION.ASIGNACION) {
            //console.log("++++++++++++",instruccion)
            var mensaje = Asignacion(instruccion, _ambito)
            if (mensaje != null) {
                cadena += mensaje
            }
        }
```

Procesando declaracion de variable

Procesando declaracion de variable

Con la misma lógica se procesan todas las demás operaciones. Ahora bien para la parte de los ámbitos se creo una clase donde estará la tabla de símbolos y tabla de métodos,con ámbito se entiende como un espacio donde se pueden crear variables y métodos sin afectar si se declaran mas de estos con el mismo nombre pero en otro ámbito,este principio se utilizo para crear un ámbito global donde se pueden declarar variables,métodos y funciones,donde cada método tiene su ámbito,con sus propias variables e instrucciones. A cada sentencia de control también se le asigno un nuevo ámbito.

```
function Declaracion(_instruccion, _ambito){
    console.log("tipo",_instruccion.tipo_dato)
    if (_instruccion.tipo_dato === TIPO_DATO.DECIMAL) {
        var valor = 0.0
        if (_instruccion.valor != null) {
            var op = Operacion(_instruccion.valor, _ambito)
            tipo = op.tipo;
            if (tipo === TIPO_DATO.DECIMAL) {
                valor = op.valor;
            }
        }

        const nuevoSimbolo = new Simbolo(_instruccion.id, valor, TIPO_DATO.DECIMAL, _instruccion.linea, _instruccion.columna)

        if (_ambito.existeSimboloAmbitoActual(nuevoSimbolo.id) != false) {
            return "Error 1: La variable " + nuevoSimbolo.id + " ya existe linea: " + nuevoSimbolo.linea + " columna: " + nuevoSimbolo.columna;
        }
        _ambito.addSimbolo(nuevoSimbolo.id, nuevoSimbolo)
        // console.log(_ambito)
        return null
    }else if (_instruccion.tipo_dato === TIPO_DATO.ENTERO) {

        var valor = 0
        if (_instruccion.valor != null) {

            console.log("Op: ",_instruccion.valor)
            try {
                var op = Operacion(_instruccion.valor, _ambito)
                tipo = op.tipo;
                if (tipo === TIPO_DATO.ENTERO) {
                    valor = op.valor;
                }
            } catch (error) {
                var op=OperadorTernario(_instruccion.valor,_ambito)
                console.log("Ternario: ",op)
                valor = op.valor;
            }



        }
        const nuevoSimbolo = new Simbolo(_instruccion.id, valor, TIPO_DATO.ENTERO, _instruccion.linea, _instruccion.columna)

        if (_ambito.existeSimboloAmbitoActual(nuevoSimbolo.id) != false) {

            return "Error 2: La variable " + nuevoSimbolo.id + " ya existe linea: " + nuevoSimbolo.linea + " columna: " + nuevoSimbolo.columna;
        }

        _ambito.addSimbolo(nuevoSimbolo.id, nuevoSimbolo)
        //console.log(_ambito)
        return null
    } else if (_instruccion.tipo_dato === TIPO_DATO.CHAR) {
        var valor = ''
        if (_instruccion.valor != null) {
            try {
                var op = Operacion(_instruccion.valor, _ambito)
                tipo = op.tipo;
                if (tipo === TIPO_DATO.CHAR) {
                    valor = op.valor;
                }
            } catch (error) {
                var op=OperadorTernario(_instruccion.valor,_ambito)
                console.log("Ternario: ",op)
                valor = op.valor;
            }
        }
        const nuevoSimbolo = new Simbolo(_instruccion.id, valor, TIPO_DATO.CHAR, _instruccion.linea, _instruccion.columna)

        if (_ambito.existeSimboloAmbitoActual(nuevoSimbolo.id) != false) {

            return "Error 3: La variable " + nuevoSimbolo.id + " ya existe linea: " + nuevoSimbolo.linea + " columna: " + nuevoSimbolo.columna;
        }
        _ambito.addSimbolo(nuevoSimbolo.id, nuevoSimbolo)
        //console.log(_ambito)
        return null
    } else if (_instruccion.tipo_dato === TIPO_DATO.BOOL) {
        var valor = true
        if (_instruccion.valor != null) {

            try {
                var op = Operacion(_instruccion.valor, _ambito)
                tipo = op.tipo;
                if (tipo === TIPO_DATO.BOOL) {
                    valor = op.valor;
                }
            } catch (error) {
                var op=OperadorTernario(_instruccion.valor,_ambito)
                console.log("Ternario: ",op)
                valor = op.valor;
            }

        }
        const nuevoSimbolo = new Simbolo(_instruccion.id, valor, TIPO_DATO.BOOL, _instruccion.linea, _instruccion.columna)

        if (_ambito.existeSimboloAmbitoActual(nuevoSimbolo.id) != false) {
            return "Error 4: La variable " + nuevoSimbolo.id + " ya existe linea: " + nuevoSimbolo.linea + " columna: " + nuevoSimbolo.columna;
        }
        _ambito.addSimbolo(nuevoSimbolo.id, nuevoSimbolo)
        // console.log(_ambito)
        return null
    } else if (_instruccion.tipo_dato === TIPO_DATO.CADENA) {

        var valor = ""
        // console.log(_instruccion)
        if (_instruccion.valor != null) {
            try {
                var op = Operacion(_instruccion.valor, _ambito)
                tipo = op.tipo;
                if (tipo === TIPO_DATO.CADENA) {
                    valor = op.valor;
                }
            } catch (error) {
                var op=OperadorTernario(_instruccion.valor,_ambito)
                console.log("Ternario: ",op)
                valor = op.valor;
            }

        }
        console.log("Valor: ",valor)
        const nuevoSimbolo = new Simbolo(_instruccion.id, valor, TIPO_DATO.CADENA, _instruccion.linea, _instruccion.columna)

        if (_ambito.existeSimboloAmbitoActual(nuevoSimbolo.id) != false) {
            return "Error 5: La variable " + nuevoSimbolo.id + " ya existe linea: " + nuevoSimbolo.linea + " columna: " + nuevoSimbolo.columna;
        }
        _ambito.addSimbolo(nuevoSimbolo.id, nuevoSimbolo)
        //console.log(_ambito)
        return null
    }
}
```

Ambito

Para el proyecto de utilizo en analizador de 3 pasadas donde primero ejecuta la verificación que solo venga un main en el ámbito global.

```
var contadorMain = 0;
    for (let i = 0; i < _instrucciones.length; i++) {
        graficarSimbolos(_ambito,false)
       if(_instrucciones[i].tipo == TIPO_INSTRUCCION.MAIN){
          contadorMain++;
       }

    }
     if(contadorMain>1){
        return "Solo puede haber 1 main"
    }else if(contadorMain==0){
        return "No hay main"
    }
```

Primera pasada

La segunda pasa se encarga de declarar métodos, funciones y variables.

```
for (let i = 0; i < _instrucciones.length; i++) {


       if(_instrucciones[i].tipo===TIPO_INSTRUCCION.DECLARACION){
           var mensaje = Declaracion(_instrucciones[i],_ambito)
           if(mensaje!=null){
              cadena+= mensaje+ "\n"
           }
       }else if(_instrucciones[i].tipo===TIPO_INSTRUCCION.PRINT){
           var mensaje = Print(_instrucciones[i],_ambito)
           if(mensaje!=null){
              cadena+= mensaje+ "\n"
           }
       }else if(_instrucciones[i].tipo===TIPO_INSTRUCCION.ASIGNACION){
           var mensaje = Asignacion(_instrucciones[i],_ambito)
           if(mensaje!=null){
              cadena+= mensaje+ "\n"
           }
       }else if (_instrucciones[i].tipo === TIPO_INSTRUCCION.DEC_METODO) {
           var mensaje = DecMetodo(_instrucciones[i], _ambito)
           if (mensaje != null) {
               cadena += mensaje + "\n"
           }
       }else if (_instrucciones[i].tipo === TIPO_INSTRUCCION.DEC_FUNCION) {
           var mensaje = DecFuncion(_instrucciones[i], _ambito)
           if (mensaje != null) {
               cadena += mensaje + "\n"
           }
       }

   }
```

Segunda pasada

Y la tercera pasada se encarga de ejecutar el main.

```
for (let i = 0; i < _instrucciones.length; i++) {
        if(_instrucciones[i].tipo===TIPO_INSTRUCCION.MAIN){
            var mensaje = Main(_instrucciones[i],_ambito)
            if(mensaje!=null){
                cadena+= mensaje+ "\n"
        }
            break
        }

    }
```

Tercera pasada

Backus-Naur

Backus-Naur gramática utilizada en el proyecto en notación Backus-Naur.

<INICIO>: <OPCIONESCUERPO> <EOF>;

<OPCIONESCUERPO>: <OPCIONESCUERPO> <CUERPO>

|<CUERPO>;

<CUERPO>: <DEC_VAR> <ptcoma>

|<ASIG_VAR> <ptcoma>

|<METODOS>

|<MAIN>

|<FUNCIONES>;

<METODOS>:

<Rvoid> <identificador> <parA> <parC> <llaveA> <INSTRUCCIONES> <llaveC>

|<Rvoid> <identificador> <parA> <LIST_PARAMETROS> <parC> <llaveA> <INSTRUCCIONES> <llaveC>;

<LIST_PARAMETROS>: <LIST_PARAMETROS> <coma> <PARAMETROS>

|<PARAMETROS>;

<PARAMETROS>: <TIPO> <identificador>;

<MAIN>:

<Rmain> <identificador> <parA> <parC> <ptcoma>

|<Rmain> <identificador> <parA> <PARAMETROS_LLAMADA> <parC> <ptcoma>;

<PARAMETROS_LLAMADA>: <PARAMETROS_LLAMADA> <coma> <EXPRESION>

|<EXPRESION>;

<DEC_VAR>:

<TIPO> <identificador>

|<TIPO> <identificador> <igual> <EXPRESION>

|<TIPO> <identificador> <igual> <parA> <TIPO> <parC> <EXPRESION>

|<TIPO> <corchA> <corchC> <identificador> <igual> <Rnew> <TIPO> <corchA> <EXPRESION> <corchC>

|<TIPO> <corchA> <corchC> <identificador> <igual> <llaveA> <ELEMENTOS_ARRAY> <llaveC>

|<Rlist> <menor> <TIPO> <mayor> <identificador> <igual> <Rnew> <Rlist> <menor> <TIPO> <mayor>

|<Rlist> <menor> <TIPO> <mayor> <identificador> <igual> <Rtochararray> <parA> <EXPRESION> <parC>;

<ELEMENTOS_ARRAY>: <ELEMENTOS_ARRAY> <coma> <EXPRESION>

|<EXPRESION>

;

<ASIG_VAR>:

<identificador> <igual> <EXPRESION>

|<identificador> <masmas>

|<identificador> <menosmenos>

|<identificador> <suma> <EXPRESION>

|<identificador> <menos> <EXPRESION>;

<TIPO>: <Rint>

|<Rdouble>

|<Rchar>

|<Rboolean>

|<Rstring>;

<INSTRUCCIONES>: <INSTRUCCIONES> <INSTRUCCION>

|<INSTRUCCION>;

<INSTRUCCION>: <DEC_VAR> <ptcoma>

|<ASIG_VAR> <ptcoma>

|<PRINT>

|<IF>

|<FOR>

|<while>

|<DO>

|<SWITCH>

|<LLAMADAS> ptcoma>

|<RETURN> ptcoma>

|<MODIFICAR_POSARRAY> ptcoma>

|<ADDLISTA> ptcoma>

|<MODIFICARLISTA> ptcoma>

;

<FUNCIONES>:

<TIPO> <identificador> <parA> <LIST_PARAMETROS> <parC> <llaveA> <INSTRUCCIONES> <llaveC>

|<TIPO> <identificador> <parA> <parC> <llaveA> <INSTRUCCIONES> <llaveC> ;

<LLAMARFUNCION>:

<identificador> <parA> <parC> <ptcoma>

|<identificador> <parA> <PARAMETROS_LLAMADA> <parC> <ptcoma>;

<LLAMADAS>:

<identificador> <parA> <parC>

|<identificador> <parA> <PARAMETROS_LLAMADA> <parC>;

<MODIFICARLISTA>:

<identificador> <corchA> <corchA> <EXPRESION> <corchC> <corchC> <igual> <EXPRESION>;

<ADDLISTA>:

<identificador> <punto> <Radd> <parA> <EXPRESION> <parC>;

<MODIFICAR_POSARRAY>:

<identificador> <corchA> <EXPRESION> <corchC> <igual> <EXPRESION>;

<RETURN>:<Rreturn> <EXPRESION>;

<IF>: <Rif> <parA> <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC>

|<Rif> <parA <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC> <Relse> <llaveA> <INSTRUCCIONES> <llaveC>

| <Rif> <parA> <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC> <ELSEIF>

| <Rif> <parA> <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC> <ELSEIF> <Relse> <llaveA> <INSTRUCCIONES> <llaveC>;

<ELSEIF>:<ELSEIF> <CONEIF>

| <CONEIF>;

<CONEIF>: <Relse> <Rif> <parA> <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC>;

<FOR>:

<Rfor> <parA> <DEC_VAR> <ptcoma> <EXPRESION> <ptcoma> <ASIG_VAR> <parC> <llaveA> <INSTRUCCIONES> <llaveC>

|<Rfor> <parA> <ASIG_VAR> <ptcoma> <EXPRESION> <ptcoma> <ASIG_VAR> <parC> <llaveA> <INSTRUCCIONES> <llaveC> ;

<while>:

<Rwhile> <parA> <EXPRESION> <parC> <llaveA> <INSTRUCCIONES> <llaveC>;

<DO>: <Rdo> <llaveA> <INSTRUCCIONES> <llaveC> <Rwhile> <parA> <EXPRESION> <parC> <ptcoma>;

<SWITCH>:

<Rswitch> <parA> <EXPRESION> <parC> <llaveA> <CASES> <llaveC>;

<CASES>:<CASES> <CASE>

|<CASE>;

<CASE>:

<Rcase> <EXPRESION> <dospuntos> <INSTRUCCIONES> <Rbreak> <ptcoma>

|<Rdefault> <dospuntos> <INSTRUCCIONES> <Rbreak> <ptcoma>;

<PRINT>: <Rprint> <parA> <EXPRESION> <parC> <ptcoma>;

<EXPRESION: <EXPRESION> <suma> <EXPRESION>

| <EXPRESION> <menos> <EXPRESION>

| <EXPRESION> <multi> <EXPRESION>

| <EXPRESION> <EXPRESION>

| <EXPRESION> <exponente> <EXPRESION>

| <EXPRESION> <modulo> <EXPRESION>

| <EXPRESION> <menor> <EXPRESION>

| <EXPRESION> <mayor> <EXPRESION>

| <EXPRESION> <menorIgual> <EXPRESION>

| <EXPRESION> <mayorIgual> <EXPRESION>

| <EXPRESION> <diferente> <EXPRESION>

| <EXPRESION> <and> <EXPRESION>

| <EXPRESION> <or> <EXPRESION>

| <menos> <EXPRESION> <umenos>

| <not> <EXPRESION>

| <parA> <EXPRESION <parC>

| <EXPRESION> <igualigual> <EXPRESION>

| <decimal>

| <entero>

| <Rtrue>

| <Rfalse>

| <string>

| <identificador>

| <char>

| <EXPRESION> <interrogacion> <EXPRESION> <dospuntos> <EXPRESION>

| <identificador> <corchA> <EXPRESION> <corchC>

| <Rlower> <parA> <EXPRESION> <parC>

| <Rup> <parA> <EXPRESION> <parC>

| <Rlen> <parA> <EXPRESION> <parC>

| <Rtrunc> <parA> <EXPRESION> <parC>

| <Rround> <parA> <EXPRESION> <parC>

| <Rtype> <parA> <EXPRESION> <parC>

| <RtoString> <parA> <EXPRESION> <parC>

;