Universidad Nacional Autónoma de México

Facultad de Ingeniería

Compiladores

Semestre 2020-1

Práctica 6: Definición Dirigida por Sintaxis y Esquema de traducción

Profesor:

Adrián Ulises Mercado Martínez

Alumnos:

Cárdenas Cárdenas Jorge

Murrieta Villegas Alfonso

Reza Chavarria Sergio Gabriel

Valdespino Mendieta Joaquin

Objetivo

Para la gramática de la sección inferior obtener la definición por sintaxis y su respectivo esquema de traducción.

Gramática

1. programa → declaraciones **\n** funciones
2. declaraciones → tipo lista_var **\n** declaraciones
   | **registro \n inicio** declaraciones **\n fin \n** declaraciones
   | ε
3. tipo → base tipo_arreglo
4. base → **ent** | **real** | **dreal** | **car** | **sin**
5. tipo_arreglo → **[num]** tipo_arreglo | ε
6. lista_var → lista_var **,** **id** | **id**
7. funciones → **func** tipo **id(** argumentos **)** **inicio \n** declaraciones sentencias **\n fin \n** funciones | ε
8. argumentos → listar_arg | **sin**
9. lista_arg → lista_arg arg | arg
10. arg → tipo **id**
11. sentencias → sentencias**\n** sentencia | sentencia
12. sentencia → **si** expresion_booleana **entonces \n** sentencias **\n fin**
    | **si** expresion_booleana **\n** sentencias **\n sino \n** sentencias **\n fin**
    | **mientras \n** expresion_booleana **hacer \n** sentencias **\n fin**
    | **hacer \n** sentencia **\n mientras que** expresion_booleana
    | **id :=** expresion | **escribir** expresion | **leer** variable | **devolver**
    | **devolver** expresion | **segun (** expresion **) \n** casos predeterminado **\n fin**| **terminar**
13. casos → **caso num: \n** sentencias **\n** | casos **\n caso num: \n** sentencias **\n**
14. predeterminado → **predet:\n** sentencias | ε
15. expresion_booleana → expresion_booleana **oo** expresion_booleana
    | expresion_booleana **yy** expresion_booleana
    | **no** expresion_booleana
    | relacional | **verdadero** | **falso**
16. relacional → relacional **<** relacional | relacional **>** relacional | relacional **<=** relacional
    | relacional **>=** relacional | relacional **==** relacional | relacional **<>** relacional | expresion
17. expresion → expresion **+** expresion | expresion **-** expresion
    | expresion ***** expresion | expresion **/** expresion
    | expresion **%** expresion | **(**expresion **)**
    | variable | **num** | **cadena** | **caracter** | **id(** parametros **)**
18. param_arr → **id**[ ] | param_arr [ ]
19. variable → **id** parte_arreglo | **id.id**
20. parte_arreglo → [expresion] parte_arreglo | ε
21. parametros → lista_param | ε
22. lista_param → lista_param **,** expresion | expresion

Definición Dirigida por Sintaxis

program → declaraciones ***\n*** funciones	CODIGO=crearListaCuadruplas(); TablaTiposGlobal=crearTablaTipos(); TablaSimbolosGlobal=crearTablaSimbolos(); PilaTablaTipos=CrearPilaTablaTipos(); PilaTablaSimbolos=CrearPilaTablaSimbolos(); PilaTablaSimbolos.add(TablaSimbolosGlobal); PilaTablaTipos.add(TablaTiposGlobal); Program.code=crearListaCuadruplas(); GenCode(listaCuadruplas); TipoFuncion=sin;
declaraciones → tipo lista var ***\n*** declaraciones(1)	BASES=B.tipo Declaraciones.tipo=lista var.tipo TIPO=lista var .tipo
declaraciones → **registro** ***\n*** **inicio** declaraciones(1) ***\n*** **fin** ***\n*** declaraciones(2)	PTSpush(newTSimbolos()); PTTpush(newTTipos()); PilaDir.push(dir); DIR=0 TS=PTS.pop() TT=PTT.pop() Ts.addTT(tt); Declaraciones.tipo=cimaPTT.add(ts); Dir=PilaDir().pop()
declaraciones →ε	Base=declaraciones.tipo;

tipo → base tipo arreglo	If(base.tipo!=sin) {
	BASES=base.tipo
	tipo.tipo=tipo arreglo.tipo}
	else{
	error(“Base no valida”)

base → **ent**	Base.tipo= ent
base → **real**	Base.tipo=real
base → **dreal**	Base.tipo=dreal
base → **car**	Base.tipo=car
base → **sin**	

tipo arreglo → **[num]** tipo arreglo(1)	If (getTipoNum(num)==ent){
	If (num.val>=0){
	Tipo arreglo.tipo=arreglo(num.val, tipo arreglo(1))
	}Else error(“Numero menor que 0”);
	}Else Error(“No es número entero”);

tipo arreglo → ε	tipo arreglo.tipo=base
lista var → lista var(1)**, id**	If(cimaTS.add(id.id, lista var.tipo, dir, “var”)!=-1

	Dir=dir+TT.getTamanio(id.id) Else Error("No se a agregado a la tabla de símbolos")
lista var → **id**	If(cimaTS.add(id.id, lista var.tipo, dir, "var")!=-1){ Dir=dir+TT.getTamanio(id.id) }Else Error("No se a agregado a la tabla de símbolos");

Funciones → func tipo id(argumentos) inicio \n declaraciones sentencias \n fin \n funciones	TipoFunciones=tipo;
	NewTablaSimbolos=crearTablaSimbolos();
	NewTablaTipos=crearTablaTipos();
	PTS.add(NewTablaSimbolos);
	PTT.add(NewTablaTipos);
	If(cimaTS.add(id.id, tipo, dir, "funcion",argumentos)!=-1){
	If(sentencias.listaDevolver.tipos==funciones.tipo){
	If(declaraciones.contains("registros"){
	Dir=dir+TT.getTamanio(id,id);
	}Else Error("No se puede declarar registros");
	}Else Error("Valores de retorno invalidos");
	}Else Error("Funcion ya declarada);
	Funciones.code= id ||inicio || declaraciones || sentencias||fin
	TabSim=PTS.pop();
	PTT..addCimaTablatTipos(TabSim);

Funciones → ε	
sentencias → sentencias(1)\n sentencia(2)	Sentencia.code= sentencia(1) || sentencias.next() || sentencias(2)
sentencias → sentencias(1)	Sentencia.code= sentencia(1)

sentencia → **si** expresión booleana **entonces** \n sentencias(1) \n **fin**	backpatch(newLabel(), indice);
	sentencia.siglista = combinar(B.hstafalse, sentencias.siglista);
	Sentencia.code= expresión_booleana || etiqueta(Expresión booleana.listatrue) || sentencias(1)

sentencia → **si** expresion booleana \n sentencias(1) \n **sino** \n sentencias(2)	backpatch (newLabel(), indice1);
	backpatch (newLabel(), indice2);
	temp = combinar (sentencias(1).siglista, N.siglista);
	sentancia.siglista = combinar (temp, sentencias(2).siglista);
	Sentencia.code= expresión_booleana || etiqueta(Expresión booleana.listatrue) || sentencias(1)|| etiqueta(Expresión booleana.listafalse) || sentencias(2)

sentencia → **mientras** \n expresion booleana **hacer** \n sentencias(1) \n **ftn**	backpatch(newLabel(), indice1); backpatch(newLabel(), indice2); sentencia.siglista = expresion booleana.listafalse;

	Sentencia.code=etiqueta(Sentencia0.siguiente) || expresión booleana|| etiqueta(Expresión booleana.verdadero) || sentencias(1) GenCode('goto' I0);
sentencia → **hacer** \n sentencia(1) \n **mientras que** expresion booleana	backpatch(newLabel(), indice1); backpatch(newLabel(), indice2); sentencia.siglista = expresion booleana.listafalse; Sentencia.code=etiqueta(Sentencia0.siguiente) || sentencias(1) || etiqueta(Expresión booleana.verdadero) || expresión booleana GenCode('goto' I0);
sentencia → **id :=** expresion	If cimaPTS.get(id)== -1 If cimaPTS.getTipo==expresión.tipo Sentencia.code=add_quat("=",expresión.dir,"","id") Error("Incompatibilidad") Else Error("Variable no declarada")Error("Variable no declarada")
sentencia → **escribir** expresión	Sentencia.code=expresión
sentencia → **leer** variable	Sentencia.code=obtenerInfo(variable)
sentencia → **devolver**	If(tipoFunciones!=sin) Error("Funcion sin");
sentencia → **devolver** expresión	If(tipoFunciones!=expresión.tipo) Error("Tipo de dato diferente al de la función")
Sentencia → segun(expresion)\n **n** casos predeterminado fin	If(expresión.tipo!=ent){ Error(); }
sentencia → terminar	genCode(goto "sentencia.next")
casos → caso num: \n sentencias \n	Caso=newEtiqueta(); Caso.next()=sentencias
casos → casos \n caso num: \n sentencias \n	Caso=newEtiqueta(); Caso.next()=sentencias
predeterminado → predet: \n sentencias \n	Predeterminado.next=newEtiqueta();
predeterminado →ε	
expresion booleana → expresion booleana(1) **oo** expresion booleana(2)	backpatch(newLabel(), indice) expression booleana.listatrue = combinar(expression booleana(1).listatrue, expression booleana(2).listatrue); expression booleana.listafalse = expression booleana(2).listafalse; expresion booleana.code= expresion booleana(1) || etiqueta (expresion booleana(1). falso)|| expresion booleana(2)

expresion booleana → expresion booleana **yy** expresion booleana	backpatch(newLabel(), indice);
	expresion booleana.listatrue = expresion booleana(2).listatrue;
	expresion booleana.listafalse = combinar(expresion booleana(1).listafalse, expresion booleana(2).listafalse);
	expresion booleana.code= expresion booleana(1) || etiqueta(expresion booleana.vardadero) || expresion booleana(2)

expresion booleana → **no** expresion booleana(1)	Expresion booleana.listatrue = expresion booleana(1).listafalse; Expresion booleana.listafalse = expresion booleana(1).listatrue;
expresion booleana → relacional	Expresion booleana.listatrue = crearlista(indice); Expresion booleana.listafalse = crearlista(indice); expresion booleana.etiqueta=relacional.code genCode('goto' _'I0);

expresion booleana → **verdadero**	expresion booleana.listatrue = crearlista(indice);
	expresion booleana.code=goto(expresion booleana.listatrue
	genCode('goto' _'I0);

expresion booleana → **falso**	expresion booleana.listafalse = crearlista(indice);
	expresion booleana.code=goto(expresion booleana.listafalse
	genCode('goto' _' I0);

relacional → relacional(1) < relacional(2)	Relacional.code=relacional(1)|| relacional(2) Add_quat (‘<’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → relacional(1) > relacional(2)	Relacional.code=relacional(1)|| relacional(2) Add_quat (‘>’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → relacional(1) <= relacional(2)	Relacional.code=relacional(1)|| relacional(2) Add_quat (‘<=’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → relacional(1) >= relacional(2)	Relacional.code=relacional(1)|| relacional(2) Add_quat (‘>=’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → relacional(1) == relacional(2)	Relacional.code=relacional(1)|| relacional(2) Add_quat (‘==’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → relacional(1) <> relacional(2)	Relacional.code=relacional(1)|| relacional(2)

	Add_quat (‘<>’ relacional(1).dir , relacional(2).dir, ‘I0’); ||genCode(‘goto’ relacional.verdadero)||genCode(‘goto’ relacional.falso
relacional → expresión	Relacional.code=expresión
expresion → expresión(1) **+** expresion(2)	Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo) Expresión.dir=newTemporal() A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir) A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir) Expresion.code= expresión(1).code|| expresión(2)|| addCuad(“+“,A1,A2,E.dir)
expresion → expresion(1) **–** expresión(2)	Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo) Expresión.dir=newTemporal() A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir) A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir) Expresion.code= expresión(1).code|| expresión(2)|| addCuad(“-“,A1,A2,E.dir)
expresion → expresion(1) ***** expresion(2)	Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo) Expresión.dir=newTemporal() A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir) A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir) Expresion.code= expresión(1).code|| expresión(2)|| addCuad(“*“,A1,A2,E.dir)
expresion → expresion(1) **/** expresion(2)	Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo) Expresión.dir=newTemporal() A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir) A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir) Expresion.code= expresión(1).code|| expresión(2)|| addCuad(“+“,A1,A2,E.dir)
expresion → expresion(1) **%** expresion(2)	If expresión(1).tipo==ent &&expresión(2).tipo==ent Expresión.dir=newTemporal() Expresion.code= expresión(1).code|| expresión(2)|| addCuad(“%“,A1,A2,E.dir) Else Error(“Deben ser enteros”)

expresion → **(**expresion(1) **)**	Expresión.dir= expresión(1).dir expresión.code=expresión(1)
expresion → variable	Expresión.dir=variable.dir
expresion → **num**	Expresión.dir=newTemporal() Expresión.code=genCode('Expresión.dir=num')
expresion → **cadena**	Expresión.dir=newTemporal() Expresión.code=genCode('Expresión.dir=cadena')
expresion → **caracter**	Expresión.dir=newTemporal() Expresión.code=genCode('Expresión.dir=caracter')
expresion → **id(** parametros **)**	Expresión.dir=newTemporal() Expresión.code=genCode('Expresión.dir=num')
param arr → **id[ ]**	If(cimaTS.add(id.id, tipo, dir, "funcion",parametros)!=-1){ Dir= Dir=dir+TT.getTamanio(id.id); }Else Error("Parametro ya declarado")
param arr → param arr **[ ]**	
variable → **id** parte arreglo	If(getTipoParam(id)!=-1) CimaTS.add(id) Else Error();
variable → **id.id**	If(getTipoParam(id)!=-1) CimaTS.add(id) Else Error();
parte arreglo → **[** expresion **]** parte arreglo	If (getTipoNum(expresión.tipo)==ent){ Tipo arreglo.tipo=arreglo(num.val, tipo arreglo(1)) }Else Error("No es número entero");
parte arreglo → ε	
parametros → lista param	Parámetros.lista=crearListaParametros();
parametros → ε	
lista param → lista param **,** expresion	Lista param.listaParametros=newListaParametros() Lista_param.listaParametros.add(Expresion.param)
lista param → expresion	Lista param.listaParametros=newListaParametros() Lista_param.listaParametros.add(Expresion.param)

Esquema de Traducción

1. programa → **{** CODIGO=crearListaCuadruplas(); TablaTiposGlobal=crearTablaTipos(); TablaSimbolosGlobal=crearTablaSimbolos(); PilaTablaTipos=CrearPilaTablaTipos(); PilaTablaSimbolos=CrearPilaTablaSimbolos();PilaTablaSimbolos.add(TablaSimbolosGlobal); PilaTablaTipos.add(TablaTiposGlobal); Program.code=crearListaCuadruplas(); } declaraciones \n funciones { GenCode(listaCuadruplas); TipoFuncion=sin;}

2. declaraciones → tipo { BASES=B.tipo} lista var \n declaraciones **{** Declaraciones.tipo=lista var.tipo; TIPO=lista var .tipo **}**| **registro \n inicio** declaraciones **\n ftn \n** declaraciones | ε

3. tipo → base tipo arreglo { If(base.tipo!=sin) {BASES=base.tipo; tipo.tipo=tipo arreglo.tipo} else{ error("Base no valida")}}

4. base → ent{ Base.tipo= ent } | **real** { Base.tipo=real } | dreal { Base.tipo=dreal } | **car**{ Base.tipo=car} | **sin**

5. tipo arreglo → [num] tipo arreglo { If (getTipoNum(num)==ent){If (num.val>=0){Tipo arreglo.tipo=arreglo(num.val, tipo arreglo(1))}Else error("Numero menor que 0");}Else Error("No es número entero");} | ε { tipo arreglo.tipo=BASES }

6. lista var → lista var , id { If(cimaTS.add(id.id, lista var.tipo, dir, "var")!=-1 Dir=dir+TT.getTamanio(id.id)}Else{Error("No se a agregado a la tabla de símbolos");} | id{ If(cimaTS.add(id.id, lista var.tipo, dir, "var")!=-1 Dir=dir+TT.getTamanio(id.id)}Else{Error("No se a agregado a la tabla de símbolos");}funciones → func tipo { TipoFunciones=tipo; }id(argumentos) { NewTablaSimbolos=crearTablaSimbolos(); NewTablaTipos=crearTablaTipos(); PTS.add(NewTablaSimbolos); PTT.add(NewTablaTipos); If(cimaTS.add(id.id, tipo, dir, "funcion",argumentos)!=-1) {If(sentencias.listaDevolver.tipos==funciones.tipo){If(declaraciones.contains("registros"){ Dir=dir+TT.getTamanio(id,id);}Else Error("No se puede declarar registros");}Else Error("Valores de retorno invalidos"); }Else Error("Funcion ya declarada);} inicio \n declaraciones sentencias { etiqueta(funciones.next) } \n ftn \n funciones | ε

7. argumentos → listar arg | sin

8. lista arg → lista arg arg | arg
   arg → tipo id

9. sentencias → sentencias { sentencias.next(); }\n sentencia | sentencia

10. sentencia → { backpatch(newLabel(), indice);}si expresion booleana { etiqueta(Expresión booleana.listatrue) } entonces \n sentencias { sentencia.siglista = combinar(B.hstafalse, sentencias.siglista); etiqueta(Expresión booleana.listatrue) }\n ftn | si { backpatch (newLabel(), indice1); backpatch (newLabel(), indice2); } expresion booleana \n sentencias { temp = combinar (sentencias(1).siglista, N.siglista);} \n sino { etiqueta(Expresión booleana.listafalse) } \n sentencias \n ftn | mientras{ backpatch(newLabel(), indice1); backpatch(newLabel(), indice2); } \n expresion booleana { sentencia.siglista = expresion booleana.listafalse; }hacer \n sentencias \n ftn | hacer { backpatch(newLabel(), indice1); backpatch(newLabel(), indice2); }\n sentencia \n mientras que expresion booleana | id := expresión{ If (cimaPTS.get(id)== -1)If(cimaPTS.getTipo==expresión.tipo) Sentencia.code=add_quat("=",expresión.dir,"","id"); Error("Incompatibilidad") Else Error("Variable no declarada")} | escribir expresión { Sentencia.code=expresión }| leer variable {Sentencia.code=obtenerInfo(variable)} | devolver { If(tipoFunciones!=sin); Error("Funcion sin"); } | devolver expresión {If(tipoFunciones!=expresión.tipo); Error("Tipo de dato diferente al de la función"); } | segun (expresion) { If(expresión.tipo!=ent){Error();}} \n casos predeterminado \n ftn| terminar {genCode(goto "sentencia.next"); }

11. casos → caso { Caso=newEtiqueta(); Caso.next()=sentencias;} num: \n sentencias \n | casos \n caso { Caso=newEtiqueta(); Caso.next()=sentencias;} num: \n sentencias \n

12. predeterminado → predet:\n{ Predeterminado.next=newEtiqueta();} sentencias | ε

13. expresion booleana → { backpatch(newLabel(), indice); expression booleana.listatrue = combinar(expression booleana(1).listatrue, expression booleana(2).listatrue); expression booleana.listafalse = expression booleana(2).listafalse;} expresion booleana **oo** { etiqueta (expresion booleana(1). falso) }expresion booleana | { backpatch(newLabel(), indice); expresion booleana; expresion booleana.listatrue = expresion booleana(2).listatrue; expresion booleana.listafalse = combinar(expresion booleana(1).listafalse, expresion booleana(2).listafalse); } yy { etiqueta(expresion booleana.vardadero) } expresion booleana | no expresion booleana { Expresion booleana.listatrue = expresion booleana(1).listafalse; Expresion booleana.listafalse = expresion booleana(1).listatrue;} | relacional { Expresion booleana.listatrue = crearlista(indice); Expresion booleana.listafalse = crearlista(indice);} | verdadero { expresion booleana.listatrue = crearlista(indice);} | falso { expresion booleana.listafalse = crearlista(indice); }

14. relacional → relacional < relacional { Add_quat ('<' relacional(1).dir , relacional(2).dir, 'I0'); } | relacional > relacional { Add_quat ('>' relacional(1).dir , relacional(2).dir, 'I0'); } | relacional <= relacional { Add_quat ('<=' relacional(1).dir , relacional(2).dir, 'I0'); } | relacional >= relacional { Add_quat ('>=' relacional(1).dir , relacional(2).dir, 'I0'); } | relacional == relacional { Add_quat ('==' relacional(1).dir , relacional(2).dir, 'I0'); } | relacional <> relacional { Add_quat ('<>' relacional(1).dir , relacional(2).dir, 'I0'} | expresión

15. expresion → expresion + expresión {Expresión.tipo=máximo(Expresión.tipo, Expresión.tipo); Expresión.dir=newTemporal(); A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir); A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir); addCuad("+",A1,A2,E.dir) } | expresion – expresión { Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo); Expresión.dir=newTemporal(); A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir); A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir); addCuad("-",A1,A2,E.dir)} | expresion * expresión { Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo); Expresión.dir=newTemporal(); A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir); A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir); addCuad("*",A1,A2,E.dir)} | expresion / expresión { Expresión.tipo=máximo(Expresión(1).tipo, Expresión(2).tipo); Expresión.dir=newTemporal(); A1=ampliar(Expresión(1).tipo, Expresión.tipo, Expresión(1).dir); A2=ampliar(Expresión(2).tipo, Expresión.tipo, Expresión(2).dir); addCuad("+",A1,A2,E.dir) } | expresion % expresión { If expresión(1).tipo==ent &&expresión(2).tipo==ent; Expresión.dir=newTemporal();addCuad("%",A1,A2,E.dir); Else Error("Deben ser enteros"); } | (expresion) { Expresión.dir= expresión(1).dir } | variable { Expresión.dir=variable.dir} | num { Expresión.dir=newTemporal(); Expresión.code=genCode('Expresión.dir=num')} | cadena{ Expresión.dir=newTemporal(); Expresión.code=genCode('Expresión.dir=cadena'); } | carácter {Expresión.dir=newTemporal(); Expresión.code=genCode('Expresión.dir=caracter') } | id(parametros)

16. param arr → id[ ] { If(cimaTS.add(id.id, tipo, dir, "funcion",parametros)!=-1){Dir= Dir=dir+TT.getTamanio(id.id);}Else Error("Parametro ya declarado")} | param arr [ ]

17. variable → id parte arreglo | id.id{ If(cimaTS.add(id.id, variable.tipo, dir, "funcion",parametros)!=-1){Dir= Dir=dir+TT.getTamanio(id.id);} Else Error("Parametro ya declarado")}

18. parte arreglo → [expresion] parte arreglo { If (getTipoNum(expresión.tipo)==ent){Tipo arreglo.tipo=arreglo(num.val, tipo arreglo(1))}Else Error("No es número entero");} | ε

19. parametros → lista param { Parámetros.lista=crearListaParametros(); } | ε

20. lista param → lista param , expresión { Listaparam.listaParametros=newListaParametros()Lista_param.listaParametros.add(Expresion.param)} | expresión { Listaparam.listaParametros=newListaParametros()Lista_param.listaParametros.add(Expresion.param)}

Anexo: Cuádruplas

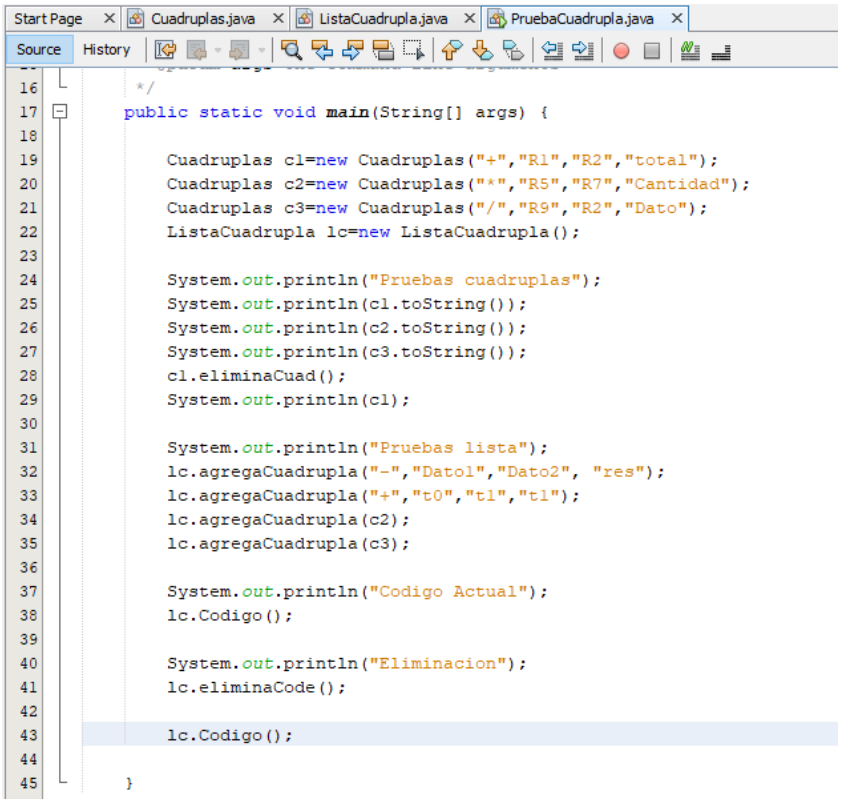
En el presente apartado se muestran todas las clases que fueron necesarias, además también se muestran capturas de pantallas de pruebas de escritorio.

```java
public class Cuadruplas {

    String operador;
    String argumento1;
    String argumento2;
    String resultado;

    /*Creación de cuadrupla*/
    public Cuadruplas(String op, String arg1, String arg2, String res){
        this.operador=op;
        this.argumento1=arg1;
        this.argumento2=arg2;
        this.resultado=res;
    }

    /*Eliminación de info*/
    public void eliminaCuad(){
        this.operador=null;
        this.argumento1=null;
        this.argumento2=null;
        this.resultado=null;
    }
    @Override
    public String toString(){
        if(this.argumento1!=null){
            return this.resultado+"="+this.argumento1+this.operador+this.argumento2;
        }else
            return "Sin datos en la cuadrupla";
    }
}
```

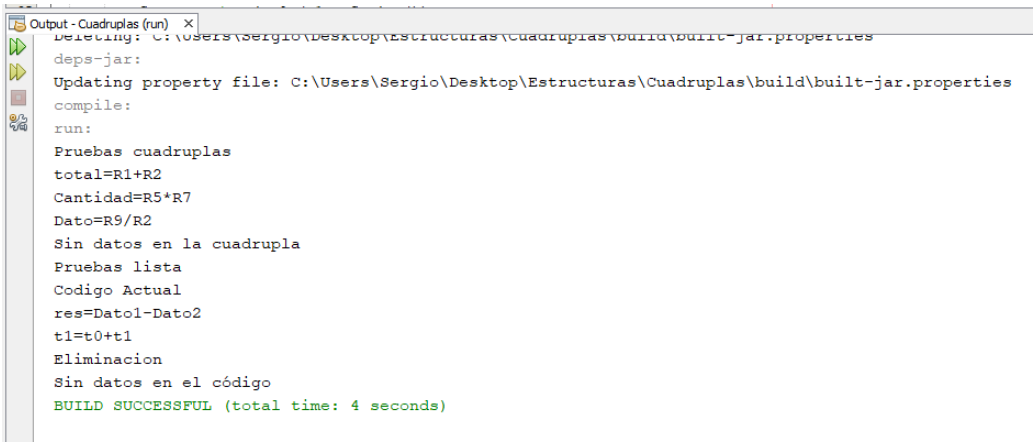
Código 1: Creación de cuádruplas, generación y eliminación

```java
*/
public class ListaCuadrupla {
    LinkedList <Cuadruplas> listaCuad;
    int numInstrucciones;

    /*Creacion de la lista*/
    public ListaCuadrupla(){
        listaCuad=new LinkedList<Cuadruplas>();
        this.numInstrucciones=0;
    }
    /*Agrega a la instancia una nueva instancia de cuadrupla*/
    public void agregaCuadrupla(String op, String arg1, String arg2, String res){
        listaCuad.add(new Cuadruplas(op,arg1,arg2,res));
        this.numInstrucciones=listaCuad.size();
    }
    public void agregaCuadrupla(Cuadruplas c1){
        listaCuad.add(c1);
    }

    public int lineasCode(){
        return this.numInstrucciones;
    }
    public void eliminaCode(){
        listaCuad.clear();
        this.numInstrucciones=0;
    }

    public void Codigo(){
        if(this.numInstrucciones!=0){
            for (int i = 0; i < this.numInstrucciones; i++) {
            System.out.println(listaCuad.get(i).toString());
            }
        }else{
            System.out.println("Sin datos en el código");
        }
    }

}
```

Código 2: Código de la lista de cuadruplas, creación, manejo y eliminación de las listas



Código 3: Código de pruebas



Resultados 1: Ejecución en la creación de cuádruplas, creación de las listas, agregado y eliminación de la lista.

NOTAS:

Es necesario destacar que para poder correr el presente código es necesario el IDE Apache NeatBeans o alguna versión del compilador de Java superior a la 8.

En el caso de no contar con el IDE, es necesario reescribir el apartado de los paquetes dentro las 3 clases programas.

Conclusiones

Cárdenas Cárdenas Jorge

Esta práctica nos permitió generar la definición dirigida por sintaxis y su respectivo esquema de traducción para la gramática definida en este mismo documento.

A través de la obtención de dicho esquema de traducción y reglas semánticas de manera general, hemos podido aplicar los conocimientos adquiridos en las clases teóricas, así como complementarlos con la investigación de algunos de ellos, tal como con el concepto de backpatch.

Después de esta práctica podemos concluir que lo más importante y complejo en un compilador radica en el análisis semántico de la gramática definida, pues no solo se encarga de revisar la coherencia del código, sino que además funge como base para generar tanto el código intermedio como el código ensamblador; por lo que definir correctamente todas las reglas en el esquema de traducción será medular para la construcción de nuestro compilador.

Murrieta Villegas Alfonso

En la presente práctica a través de dos conceptos principales como son la *definición dirigida por sintaxis* y el *esquema de traducción* es como se pudo generalizar y abarcar la mayor parte del análisis semántico de nuestro futuro compilador.

Para poder obtener ambos elementos fue necesario recurrir a los conocimientos adquiridos en nuestras clases teóricas donde aprendimos a poder obtener estos elementos de una gramática, cabe destacar que algo importante de esta práctica fue el emplear el concepto de ***backpatch*** debido a que en un primer momento nuestra definición dirigida por sintaxis estaba conformada por atributos tanto sintetizados como heredados lo cual debido al diseño que se tomó para nuestro compilador no resultaba compatible, fue por ello que se tuvo que omitir mediante diferentes recursos todos los atributos heredados.

Por último, una vez que se ha pasado por el análisis semántico la representación usada es un código intermedio conocido como *código de tres direcciones* o *TAC*, el cual al ser el recurso intermedio entre el compilador y ensamblador requiere una forma de representarse, es por ello por lo que incluso en esta práctica (En el apartado del anexo) se empleó un código para poder representar el código de tres direcciones mediante *cuádruplas*, las cuales representan código mediante un operador, 2 operandos y un resultado .

Reza Chavarría Sergio Gabriel

En la práctica realizada se pudo realizar y desarrollo de la parte de la definición dirigida por sintaxis y del esquema de traducción.

Para la definición dirigida por sintaxis que desempeño en obtener y reconocer las diferentes acciones que el programa llegará a realizar durante el proceso del frontend. Esto se dio con respecto a las acciones como la creaciones y manejo de las tablas de símbolos y de tipos, entre otras acciones.

Y con esto se pudo dar a la creación del esquema de traducción, el cual nos ayudará a entender en que tiempo se darán los procesos de las acciones en el reconocimiento de las etapas y poder revisar las

diferentes condiciones por las que el programa a traducir puede llegar a generar, como los diferentes tipos de errores, generación de datos creados por el usuario, entre otros casos.

Valdespino Mendieta Joaquín

En la presente practica se realizó la definición dirigida por sintaxis y su esquema de traducción, como parte fundamental del compilador, ya que como el nombre lo indica y se ha logrado observar todo parte de la parte sintáctica. Además con esto pudimos definir gran parte de las acciones semánticas que operara nuestro compilador, partiendo de una gramática y conceptos vistos en clase, tales como atributos heredados, sintetizados, donde se tuvo que analizar la parte del backpatch debido a incompatibilidad de los atributos heredados a la hora de programarlo, Por ultimo cabe destacar que el uso de estructura de datos es indispensable para desarrollar un compilador, añadiendo la interpretación y conocimientos sobre lenguaje ensamblador.