Practica 4:

Construcción de árboles de sintaxis abstracta

Grupo 11:

Youssef El Faqir El Rhazoui

Enrique Ávila Rodríguez

1. Conjunto de funciones constructoras

Prog: Sec_Dec **x** Sec_Ins → Prog

Sec_Dec: LDs → Prog

Sec_Ins: LIs → Prog

LD_simp: String **x** String → D

LD_comp: String **x** String **x** LDs → LDs

LI_simp: String **x** Exp → I

LI_comp: String **x** Exp **x** LIs → LIs

Mas: Exp **x** Exp → Exp

Menos: Exp **x** Exp → Exp

And: Exp **x** Exp → Exp

Or: Exp **x** Exp → Exp

Distinto: Exp **x** Exp → Exp

Igual: Exp **x** Exp → Exp

Menor_que: Exp **x** Exp → Exp

Menor_igual_que: Exp **x** Exp → Exp

Mayor_que: Exp **x** Exp → Exp

Mayor_igual_que: Exp **x** Exp → Exp

Mul: Exp **x** Exp → Exp

Div: Exp **x** Exp → Exp

Not: Exp → Exp

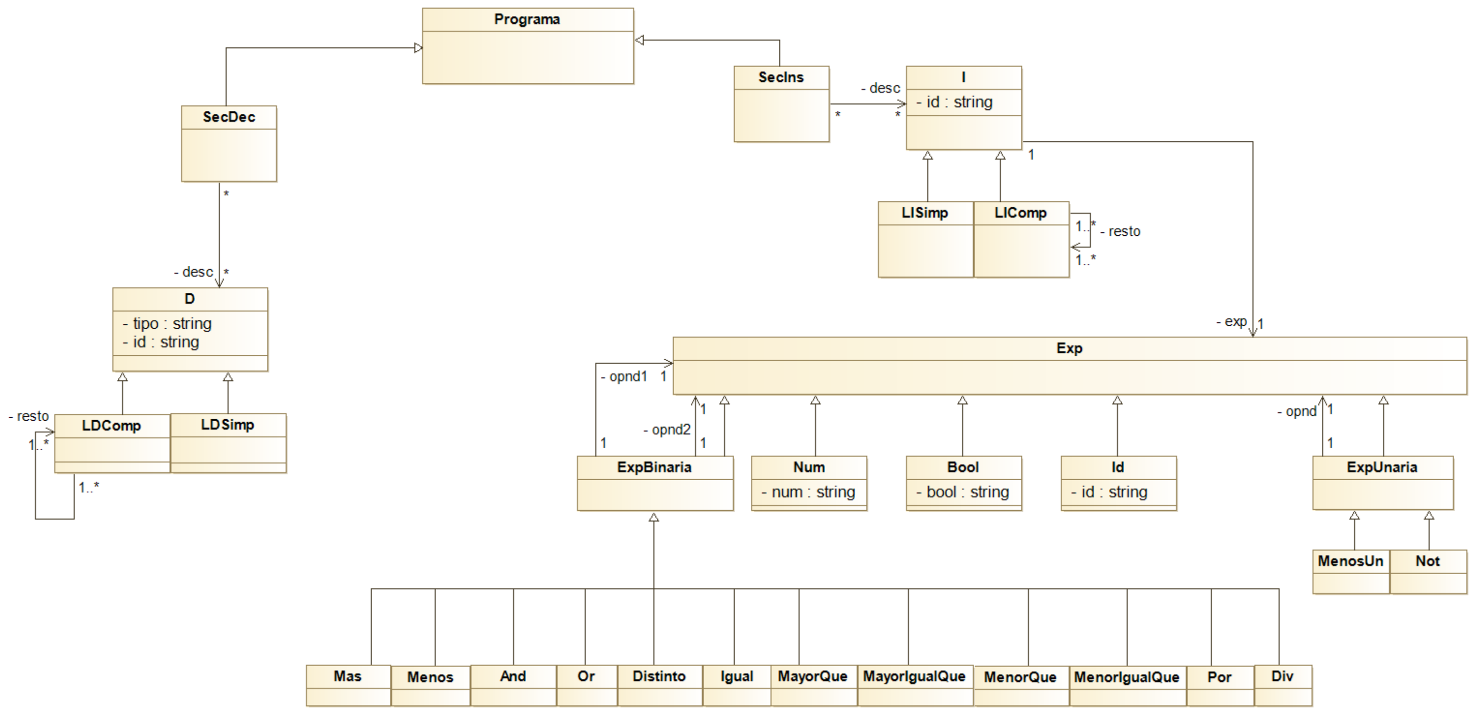
Menos_unario: Exp → Exp

Num: String → Exp

Bool: String → Exp

Id: String → Exp

2. Diagrama de clases



3. Gramática de atributos

Prog → Sec_Dec && Sec_Ins

Prog.a = prog(Sec_Dec.a, Sec_Ins.a)

Sec_Dec → Sec_Dec; D

Sec_Dec_0.a = ldCompuesta(D.tipo, D.id, Sec_Dec_1.a)

Sec_Dec → D

Sec_Dec.a = ldSimple(D.tipo, D.id)

Sec_Ins → Sec_Ins; I

Sec_Ins_0.a = liCompuesta(I.id, I.exp, Sec_Ins_1.a)

Sec_Ins → I

Sec_Ins.a = liSimple(I.id, I.exp)

D → tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I → identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.v

Exp0 → Exp0 Op0 Exp1

$Exp0_0$.v = mkexp(Op0.op, $Exp0_1$.v, Exp1.v)

Exp0 → Exp1

Exp0.v = Exp1.v

Exp1 → Exp2 and Exp1

$Exp1_0$.v = and(Exp2.v, $Exp1_1$.v)

Exp1 → Exp2 or Exp2

Exp1.v = or($Exp2_0$.v, $Exp2_1$.v)

Exp1 → Exp2

Exp1.v = Exp2.v

Exp2 → Exp3 Op2 Exp3

Exp2.v = mkexp(Op2.op, $Exp3_0$.v, $Exp3_1$.v)

Exp2 → Exp3

Exp2.v = Exp3.v

Exp3 → Exp3 Op3 Exp4

$Exp3_0$.v = mkexp(Op3.op, $Exp3_1$.v, Exp4.v)

Exp3 → Exp4

Exp3.v = Exp4.v

Exp4 → - Exp4

$Exp4_0$.v = menos_unario($Exp4_1$.v)

Exp4 → not Exp5

Exp4.v = not(Exp5.v)

Exp4 → Exp5

Exp4.v = Exp5.v

Exp5 → numero

Exp5.v = num(numero.lex)

Exp5 → booleano

Exp5.v = bool(booleano.lex)

Exp5 → identificador

Exp5.v = id(identificador.lex)

Exp5 → (Exp0)

Exp5.v = Exp0.v

Op0 → +

Op0.op = “+”

Op0 → -

Op0.op = “-”

Op2 → !=

Op2.op = “!=”

Op2 → ==

Op2.op = “==”

Op2 → <

Op2.op = “<”

Op2 → <=

Op2.op = “<=”

Op2 → >

Op2.op = “>”

Op2 → >=

Op2.op = “>=”

Op3 → *

Op3.op = “*”

Op3 → /

Op3.op = “/”

Definimos la función mkexp como sigue:

```
fun mkexp(op, opnd1,opnd2) {
	switch(op) {
		"+" => return suma(opnd1,opnd2)
		"-" => return resta(opnd1,opnd2)
		"!=" => return distinto(opnd1,opnd2)
		"==" => return igual(opnd1,opnd2)
		"<" => return menorQue(opnd1,opnd2)
		"<=" => return menorIgualQue(opnd1,opnd2)
		">" => return mayorQue(opnd1,opnd2)
		">=" => return mayorIgualQue(opnd1,opnd2)
		"*" => return mul(opnd1,opnd2)
		"/" => return div(opnd1,opnd2)
	}
}
```

4. Acondicionamiento para imp descendente

Prog → Sec_Dec && Sec_Ins

Prog.a = prog(Sec_Dec.a, Sec_Ins.a)

Sec_Dec → D PDec

PDec.ah = ldSimple(D.tipo, D.id)

Sec_Dec.a = PDec.a

PDec → ; D PDec

$PDec_1$.a = ldCompuesta($PDec_0$.ah, D.a)

$PDec_0$.a = $PDec_1$.a

PDec → ε

PDec.a = PDec.ah

Sec_Ins → I PIns

PIns.ah = liSimple(I.id, I.exp)

Sec_Ins.a = PIns.a

PIns → ; I PIns

$PIns_1$.a = liCompuesta($PIns_0$.ah, I.a)

$PIns_0$.a = $PIns_1$.a

PIns → ε

PIns.a = PIns.ah

D → tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I → identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.v

Exp0 → Exp1 RExp0

RExp0.vh = Exp1.v

Exp0.v = RExp0.v

RExp0 → Op0 Exp1 RExp0

$RExp0_1$.vh = mkexp(Op0.op, $RExp0_0$.vh, Exp1.v)

$RExp0_0$.v = $RExp0_1$.v

RExp0 → ε

RExp0.v = RExp0.vh

Exp1 → Exp2 RExp1

RExp1.vh = Exp2.v

Exp1.v = RExp1.v

RExp1 → and Exp1

RExp1.v = and(RExp.vh, Exp1.v)

RExp1 → or Exp2

RExp1.v = or(RExp.vh, Exp2.v)

RExp1 → ε

RExp1.v = RExp1.vh

Exp2 → Exp3 RExp2

RExp2.vh = Exp3.v

Exp2.v = RExp2.v

RExp2 → Op2 Exp3

RExp2.v = mkexp(Op2.op, RExp2.vh, Exp3.v)

RExp2 → ε

RExp2.v = RExp2.vh

Exp3 → Exp4 RExp3

RExp3.vh = Exp4.v

Exp3.v = RExp3.v

RExp3 → Op3 Exp4 RExp3

$RExp3_1$.vh = mkexp(Op3.op, $RExp3_0$.vh, Exp4.v)

$RExp3_0$.v = $RExp3_1$.v

RExp3 → ε

RExp3.v = RExp3.vh

Exp4 → - Exp4

$Exp4_0$.v = menos_unario($Exp4_1$.v)

Exp4 → not Exp5

Exp4.v = not(Exp5.v)

Exp4 → Exp5

Exp4.v = Exp5.v

Exp5 → numero

Exp5.v = num(numero.lex)

Exp5 → booleano

Exp5.v = bool(booleano.lex)

Exp5 → identificador

Exp5.v = id(identificador.lex)

Exp5 → (Exp0)

Exp5.v = Exp0.v

Op0 → +

Op0.op = "+"

Op0 → -

Op0.op = "-"

Op2 → !=

Op2.op = “!=”

Op2 → ==

Op2.op = “==”

Op2 → <

Op2.op = “<”

Op2 → <=

Op2.op = “<=”

Op2 → >

Op2.op = “>”

Op2 → >=

Op2.op = “>=”

Op3 → *

Op3.op = “*”

Op3 → /

Op3.op = “/”