

**Práctica 3**

**Procesadores de Lenguajes**

Borja Aday Guadalupe Luis

Diego Enrique de Miguel López

Grupo 13

# Índice

[Índice 2](#_Toc72355488)

[1. Sintaxis abstracta. 3](#_Toc72355489)

[2. Constructor de árboles de sintaxis abstracta (ASTs). 5](#_Toc72355490)

[2.1. Funciones semánticas. 9](#_Toc72355491)

[3. Acondicionamiento para implementación descendente. 10](#_Toc72355492)

# Sintaxis abstracta.

|  |  |
| --- | --- |
| **Gramática Tiny 1** | **Sintaxis abstracta** |
| Programa -> LDecs **&&** LIns  Programa -> LIns  LDecs -> LDecs **;** Dec  LDecs -> Dec  LDecs -> ε  Dec -> **var** Tipo **identificador**  Dec -> **type** Tipo **identificador**  Dec -> **proc identificador** ParForm Bloque  ParForm -> **(** LParams **)**  LParams -> LParams **,** Param  LParams -> Param  LParams -> ε  Param -> Tipo Referencia **identificador**  Referencia -> **&**  Referencia -> ε  Bloque -> **{** ProgramaBloque **}**  ProgramaBloque -> Programa  ProgramaBloque -> ε  Tipo -> **int**  Tipo -> **real**  Tipo -> **bool**  Tipo -> **string**  Tipo -> **identificador**  Tipo -> **array [ numEnt ] of** Tipo  Tipo -> **record {** LCampos **}**  LCampos -> LCampos **;** Campo  LCampos -> Campo  Campo -> Tipo **identificador**  Tipo -> **pointer** Tipo  LIns -> LIns **;** Ins  LIns -> Ins  Ins -> E0 **=** E0  Ins -> **if** E0 **then** LInsV **endif**  Ins -> **if** E0 **then** LinsV **else** LinsV **endif**  Ins -> **while** E0 **do** LinsV **endwhile**  LinsV -> LinsV **;** Ins  LinsV -> Ins  LinsV -> ε  Ins -> **read** E0  Ins -> **write** E0  Ins -> **nl**  Ins -> **new** E0  Ins -> **delete** E0  Ins -> **call identificador (** LParamsReales **)**    LParamsReales -> LParamsReales **,** E0  LParamsReales -> E0  LParamsReales -> ε  Ins -> Bloque  E0 -> E1 **+** E0  E0 -> E1 ­**- ­**E1  E0 -> E1  E1 -> E1 OpN1 E2  E1 -> E2  E2 -> E2 OpN2 E3  E2 -> E3  E3 -> E4 OpN3 E4  E3 -> E4  E4 -> **-** E5  E4 -> **not** E4  E4 -> E5  E5 -> E5 **[** E0 **]**  E5 -> E5 **. identificador**  E5 -> E5 **-> identificador**  E5 -> E6  E6 -> **\*** E6  E6 -> E7  E7 -> **identificador**  E7 -> **numEnt**  E7 -> **numReal**  E7 -> **true**  E7 -> **false**  E7 -> **cadena**  E7 -> **null**  E7 -> **(** E0 **)**  OpN1 -> **and**  OpN1 -> **or**  OpN2 -> **<**  OpN2 -> **>**  OpN2 -> **<=**  OpN2 -> **>=**  OpN2 -> **==**  OpN2 -> **!=**  OpN3 -> **\***  OpN3 -> **/**  OpN3 -> **%** | programa\_sin\_decs: LIns -> Programa  programa\_con\_decs: LDecs x LIns -> Programa  decs\_1: Dec -> LDecs  decs\_muchas: LDecs x Dec -> LDecs  dec\_var: Tipo x string -> Dec  dec\_type: Tipo x string -> Dec  dec\_proc\_sin\_pf: string x Bloque -> Dec  dec\_proc\_con\_pf: string x LParams x Bloque -> Dec  bloque\_sin\_programa: -> Bloque  bloque\_con\_programa: Programa -> Bloque  param\_con\_referencia: Tipo x Referencia x string -> Param  param\_sin\_referencia: Tipo x string -> Param  referencia: -> Referencia  lparams\_1: Param -> Lparams  lparams\_muchos: Lparams x Param -> LParams  tipo\_int: -> Tipo  tipo\_real: -> Tipo  tipo\_bool: -> Tipo  tipo\_string: -> Tipo  tipo\_id: string -> Tipo  tipo\_array: -> string x Tipo -> Tipo  tipo\_pointer: Tipo -> Tipo  tipo\_record: LCampos -> Tipo  campo: Tipo x string -> Campo  lcampos\_1: Campo -> Lcampos  lcampos\_muchos: LCampos x Campo -> LCampos  lins\_1: Ins -> LIns  lins\_muchas: LIns x Ins -> LIns  ins\_asignacion: Exp x Exp -> Ins  ins\_read: Exp -> Ins  ins\_write: Exp -> Ins  ins\_nl: -> Ins  ins\_new: Exp -> Ins  ins\_delete: Exp -> Ins  ins\_bloque: Bloque -> Ins  ins\_call\_con\_params: string x LParamsReales -> Ins  ins\_call\_sin\_params: string -> Ins  preales\_1: Exp -> LParamsReales  preales\_muc: -> LparamsReales x Exp -> LParamsReales  ins\_if\_con\_ins: Exp x LInsV -> Ins  ins\_if\_sin\_ins: Exp -> Ins  ins\_if\_else\_con\_ins: Exp x LInsV x LinsV -> Exp  ins\_if\_else\_insarg0: Exp x LInsV -> Exp  ins\_if\_else\_insarg1: Exp x LInsV -> Exp  ins\_if\_else\_sin\_ins: Exp -> Ins  ins\_while\_con\_ins: Exp x LinsV -> Ins  ins\_while\_sin\_ins: Exp -> Ins  linsv\_1: Ins -> LInsV  linsv\_muchas: LInsV x Ins -> LInsV  suma: Exp x Exp -> Exp  resta: Exp x Exp -> Exp  and: Exp x Exp -> Exp  or: Exp x Exp -> Exp  menor: Exp x Exp -> Exp  mayor: Exp x Exp -> Exp  menor\_igual: Exp x Exp -> Exp  mayor\_igual: Exp x Exp -> Exp  igualdad: Exp x Exp -> Exp  distinto: Exp x Exp -> Exp  mul: Exp x Exp -> Exp  div: Exp x Exp -> Exp  mod: Exp x Exp -> Exp  menos: Exp -> Exp  not: Exp -> Exp  index: Exp x Exp -> Exp  access\_reg\_flecha: Exp x string -> Exp  access\_reg\_punto: Exp x string -> Exp  indireccion: Exp -> Exp  identificador: string -> Exp  numEnt: string -> Exp  numReal: string -> Exp  cadena: string -> Exp  true: -> Exp  false: -> Exp  null: -> Exp |

# 2.Constructor de árboles de sintaxis abstracta (ASTs).

|  |
| --- |
| **Constructor de árboles de sintaxis abstracta.**  **Gramática S-Atribuida** |
| Programa -> LDecs **&&** LIns  Programa.a = programa\_con\_decs(LDecs.a, LIns.a)  Programa -> LIns  Programa.a = programa\_sin\_decs(LIns.a)  LDecs -> LDecs **;** Dec  LDecs0.a = decs\_muchas(LDecs1.a, Dec.a)  LDecs -> Dec  LDecs.a = decs\_1(Dec.a)  LDecs -> ε  LDecs.a = **null**  Dec -> **var** Tipo **identificador**  Dec.a = dec\_var(Tipo.a, **identificador**.lexema)  Dec -> **type** Tipo **identificador**  Dec.a = dec\_type(Tipo.a, **identificador**.lexema)  Dec -> **proc identificador** ParForm Bloque  Dec.a = dec\_proc(**identificador**.lexema, ParForm.a, Bloque.a)  ParForm -> **(** LParams **)**  ParForm.a = LParams.a  LParams -> LParams **,** Param  LParams0.a = lparams\_muchos(LParams1.a, Param.a)  LParams -> Param  LParams.a = lparams\_1(Param.a)  LParams -> ε  LParams.a = **null**  Param -> Tipo Referencia **identificador**  Param.a = param(Tipo.a, Referencia.a, **identificador**.lex)  Referencia -> **&**  Referencia.a = referencia()  Referencia -> ε  Referencia.a = **null**  Bloque -> **{** ProgramaBloque **}**  Bloque.a = bloque(ProgramaBloque.a)  ProgramaBloque -> Programa  ProgramaBloque.a = Programa.a  ProgramaBloque -> ε  ProgramaBloque.a = **null**  Tipo -> **int**  Tipo.a = tipo\_int()  Tipo -> **real**  Tipo.a = tipo\_reall()  Tipo -> **bool**  Tipo.a = tipo\_bool()  Tipo -> **string**  Tipo.a = tipo\_string()  Tipo -> **identificador**  Tipo.a = tipo\_id(**identificador**.lexema)  Tipo -> **array [ numEnt ] of** Tipo  Tipo.a = tipo\_array(**numEnt**.lexema, Tipo.a)  Tipo -> **record {** LCampos **}**  Tipo.a = tipo\_record(LCampos.a)  LCampos -> LCampos **;** Campo  LCampos0.a = lcampos\_muchos(LCampos1.a, Campo.a)  LCampos -> Campo  LCampos.a = lcampos\_1(Campo.a)  Campo -> Tipo **identificador**  Campo.a = campo(Tipo.a, **identificador**.lexema)  Tipo -> **pointer** Tipo  Tipo.a = tipo\_pointer(Tipo.a)  LIns -> LIns **;** Ins  LIns0.a = ins\_muchas(Lins1.a, Ins.a)  LIns -> Ins  Lins.a = ins\_1(Ins.a)  Ins -> E0 **=** E0  Ins.a = ins\_asignacion(E00.a, E01.a)  Ins -> **if** E0 **then** LInsV **endif**  Ins.a = ins\_if(E0.a, LInsV.a)  Ins -> **if** E0 **then** LinsV **else** LInsV **endif**  Ins.a = ins\_if\_else(E0.a, LInsV0.a, LInsV.a)  Ins -> **while** E0 **do** LInsV **endwhile**  Ins.a = ins\_while(E0.a, LInsV.a)  LinsV -> LInsV **;** Ins  LinsV0.a = linsv\_muchas(LinsV1.a, Ins.a)  LinsV -> Ins  LinsV.a = linsv\_1(Ins.a)  LinsV -> ε  LinsV.a = **null**  Ins -> **read** E0  Ins.a = ins\_read(E0.a)  Ins -> **write** E0  Ins.a = ins\_write(E0.a)  Ins -> **nl**  Ins.a = ins\_nl()  Ins -> **new** E0  Ins.a = ins\_new(E0.a)  Ins -> **delete** E0  Ins.a = ins\_delete(E0.a)  Ins -> **call identificador (** LParamsReales **)**  Ins.a = ins\_call(**identificador**.lexema, LParamsReales.a)    LParamsReales -> LParamsReales **,** E0  LParamsReales0.a = preales\_muc(LParamsReales1.a, E0.a)  LParamsReales -> E0  LParamsReales.a = preales\_1 (E0.a)  LParamsReales -> ε  LParamsReales.a = **null**  Ins -> Bloque  Ins.a = ins\_bloque(Bloque.a)  E0 -> E1 **+** E0  E00.a = suma(E1.a, E01.a)  E0 -> E1 ­**- ­**E1  E0.a = resta(E10.a, E11.a)  E0 -> E1  E0.a = E1.a  E1 -> E1 OpN1 E2  E10.a = exp(OpN1.op, E11.a, E2.a)  E1 -> E2  E1.a = E2.a  E2 -> E2 OpN2 E3  E20.a = exp(OpN2.op, E21.a, E3.a)  E2 -> E3  E2.a = E3.a  E3 -> E4 OpN3 E4  E3.a = exp(OpN3.op, E40.a, E41.a)  E3 -> E4  E3.a = E4.a  E4 -> **-** E5  E4.a = menos(E5.a)  E4 -> **not** E4  E40.a = not(E41.a)  E4 -> E5  E4.a = E5.a  E5 -> E5 **[** E0 **]**  E50.a = index(E51.a, E0.a)  E5 -> E5 **. identificador**  E50.a = access\_reg\_punto(E51.a, **identificador**.lexema)  E5 -> E5 **-> identificador**  E50.a = access\_reg\_flecha(E51.a, **identificador**.lexema)  E5 -> E6  E5.a = E6.a  E6 -> **\*** E6  E60.a = indireccion(E61.a)  E6 -> E7  E6.a = E7.a  E7 -> **identificador**  E7.a = identificador(**identificador**.lexema)  E7 -> **numEnt**  E7.a = identificador(**numEnt**.lexema)  E7 -> **numReal**  E7.a = identificador(**numReal**.lexema)  E7 -> **true**  E7.a = true()  E7 -> **false**  E7.a = false()  E7 -> **cadena**  E7.a = cadena(**cadena**.lexema)  E7 -> **null**  E7.a = null()  E7 -> **(** E0 **)**  E7.a = E0.a  OpN1 -> **and**  opN1.a = “and”  OpN1 -> **or**  opN1.a = “or”  OpN2 -> **<**  opN2.a = “<”  OpN2 -> **>**  opN2.a = “>”  OpN2 -> **<=**  opN2.a = “<=”  OpN2 -> **>=**  opN2.a = “>=”  OpN2 -> **==**  opN2.a = “==”  OpN2 -> **!=**  opN2.a = “!=”  OpN3 -> **\***  opN3.a = “\*”  OpN3 -> **/**  opN3.a = “/”  OpN3 -> **%**  opN3.a = “%” |

## Funciones semánticas.

fun exp(Op,Arg0,Arg1) {

**switch** Op

**case “**and“: **return** and(Arg0, Arg1)

**case “**or“: **return** or(Arg0, Arg1)

**case “**<“: **return** menor(Arg0, Arg1)

**case “**>“: **return** mayor(Arg0, Arg1)

**case “**<=“: **return** menor\_igual(Arg0, Arg1)

**case “**>=“: **return** mayor\_igual(Arg0, Arg1)

**case “**==“: **return** igualdad(Arg0, Arg1)

**case “**=!“: **return** distinto(Arg0, Arg1)

**case “\***“: **return** mul(Arg0, Arg1)

**case “**/“: **return** div(Arg0, Arg1)

**case** “%”: **return** mod(Arg0, Arg1)

}

fun programa(LDecs, LIns) {

**if** LDecs == **null then return** programa\_sin\_decs(LIns)

**else return** programa\_con\_decs(LDecs, LIns)

}

fun dec\_proc(Id, LParams, Bloque) {

**if** LParams == **null then return** dec\_proc\_sin\_pf(Id, Bloque)

**else return** dec\_proc\_con\_pf(Id, LParams, Bloque)

}

fun param(Tipo, Ref, Id) {

**if** Ref == **null then return** param\_sin\_referencia(Tipo, Id)

**else return** param\_con\_referencia(Tipo, Referencia, Id)

}

fun ins\_if(Exp, LInsV) {

**if** LInsV == **null then return** ins\_if\_sin\_ins(Exp)

**else return** ins\_if\_con\_ins(Exp, LInsV)

}

fun ins\_if\_else(Exp, LInsV1, LInsV2) {

**if** LInsV1 == **null** && LInsV2 == **null then return** ins\_if\_else\_sin\_ins(Exp)

**else if** LInsV1 != **null** && LInsV2 != **null** **then return** ins\_if\_else\_con\_ins(Exp, LInsV1, LInsV2)

**else if** LInsV1 != **null** **then return** ins\_if\_else\_insarg0(Exp, LInsV1)

**else return** ins\_if\_else\_insarg1(Exp, LInsV2)

}

fun ins\_while(Exp, LInsV) {

**if** LInsV == **null then return** ins\_while\_sin\_ins(Exp)

**else return** ins\_while\_con\_ins(Exp, LInsV)

}

fun ins\_call(Id, LParamsReales) {

**if** LParamReales == **null then return** ins\_call\_sin\_params(Id)

**else return** ins\_call\_con\_params(Id, LParamsReales)

}

fun bloque(Programa) {

**if** Programa == **null then return** bloque\_sin\_programa()

**else return** bloque\_con\_programa(Programa)

}

# Acondicionamiento para implementación descendente.

|  |
| --- |
| **Constructor de árboles de sintaxis abstracta.**  **Gramática S-Atribuida** |
| Programa -> LDecs **&&** LIns  Programa.a = programa\_con\_decs(LDecs.a, LIns.a)  Programa -> LIns  Programa.a = programa\_sin\_decs(LIns.a)  LDecs -> Dec RLDecs  RLDecs.ah = decs\_1(Dec.a)  LDecs.a = RLDecs.a  LDecs -> ε  LDecs.a = **null**  RLDecs -> **;** Dec RLDecs  RLDecs1.ah = decs\_muchas(RLDecs0.ah, Dec.a)  RLDecs0.a = RLDecs1.a  RLDecs -> ε  RLDecs.a = RLDecs.ah  Dec -> **var** Tipo **identificador**  Dec.a = dec\_var(Tipo.a, **identificador**.lexema)  Dec -> **type** Tipo **identificador**  Dec.a = dec\_type(Tipo.a, **identificador**.lexema)  Dec -> **proc identificador** ParForm Bloque  Dec.a = dec\_proc(**identificador**.lexema, ParForm.a, Bloque.a)  ParForm -> **(** LParams **)**  ParForm.a = LParams.a  LParams -> Param RLParams  RLParams.ah = lparams\_1(Param.a)  LParams.a = RLParams.a  LParams -> ε  LParams.a = **null**  RLParams -> **,** Param RLParams  RLParams1.ah = lparams\_muchos(RLParams0.ah, Param.a)  RLParams0.a = RLParams1.a  RLParams -> ε  RLParams.a = RLParams.ah  Param -> Tipo Referencia **identificador**  Param.a = param(Tipo.a, Referencia.a, **identificador**.lexema)  Referencia -> **&**  Referencia.a = referencia()  Referencia -> ε  Referencia.a = **null**  Bloque -> **{** ProgramaBloque **}**  Bloque.a = bloque(ProgramaBloque.a)  ProgramaBloque -> Programa  ProgramaBloque.a = Programa.a  ProgramaBloque -> ε  ProgramaBloque.a = **null**  Tipo -> **int**  Tipo.a = tipo\_int()  Tipo -> **real**  Tipo.a = tipo\_real()  Tipo -> **bool**  Tipo.a = tipo\_bool()  Tipo -> **string**  Tipo.a = tipo\_string()  Tipo -> **identificador**  Tipo.a = tipo\_id(**identificador**.lexema)  Tipo -> **array [ numEnt ] of** Tipo  Tipo.a = tipo\_array(**numEnt**.lexema, Tipo.a)  Tipo -> **record {** LCampos **}**  Tipo.a = tipo\_record(LCampos.a)  LCampos -> Campo RLCampos  RLCampos.ah = lcampos\_1 (Campo.a)  LCampos.a = RLCampos.a  RLCampos -> **;** Campo RLCampos  RLCampos1.ah = lcampos\_muchos(RLCampos0.ah, Campo.a)  RLCampos0.a = RLCampos1.a  RLCampos -> ε  RLCampos.a = RLCampos.ah  Campo -> Tipo **identificador**  Campo.a = campo(Tipo.a, **identificador**.lexema)  Tipo -> **pointer** Tipo  Tipo.a = tipo\_pointer(Tipo.a)  LIns -> Ins RLIns  RLIns.ah = ins\_1 (Ins.a)  LIns.a = RLIns.a  RLIns -> **;** Ins RLIns  RLIns1.ah = ins\_muchas(RLIns0.ah, Ins.a)  RLIns0.a = RLIns1.a  RLIns -> ε  RLIns.a = RLIns.ah  Ins -> E0 **=** E0  Ins.a = ins\_asignacion(E00.a, E01.a)  Ins -> **if** E0 **then** LInsV RInsIfThen  RInsIfThen.ah0 = E0.a  RInsIfThen.ah1 = LInsV.a  Ins.a = RInsIfThen.a  RInsIfThen -> **endif**  RInsIfThen.a = ins\_if(RInsIfThen.ah0, RInsIfThen.ah1)  RInsIfThen -> **else** LInsV **endif**  RInsIfThen.a = ins\_if\_else(RInsIfThen.ah0, RInsIfThen.ah1, LInsV.a)  Ins -> **while** E0 **do** LInsV **endwhile**  Ins.a = ins\_while(E0.a, LInsV.a)  LInsV -> Param RLInsV  RLInsV.ah = linsv\_1(Ins.a)  LInsV.a = RLInsV.a  LInsV -> ε  LInsV.a = **null**  RLInsV -> **;** Param RLInsV  RLInsV1.ah = linsv\_muchas(RLInsV0.ah, Ins.a)  RLInsV0.a = RLInsV1.a  RLInsV -> ε  RLInsV.a = RLInsV.ah  Ins -> **read** E0  Ins.a = ins\_read(E0.a)  Ins -> **write** E0  Ins.a = ins\_write(E0.a)  Ins -> **nl**  Ins.a = ins\_nl()  Ins -> **new** E0  Ins.a = ins\_new(E0.a)  Ins -> **delete** E0  Ins.a = ins\_delete(E0.a)  Ins -> **call identificador (** LParamsReales **)**  Ins.a = ins\_call(**identificador**.lexema, LParamsReales.a)  LParamsReales -> E0 RLParamsReales  RLParamsReales.ah = preales\_1(E0.a)  LParamsReales.a = RLParamsReales.a  LParamsReales -> ε  LParamsReales.a = **null**  RLParamsReales -> **,** E0 RLParamsReales  RLParamsReales1.ah = preales\_muc(RLParamsReales0.ah, E0.a)  RLParamsReales0.a = RLParamsReales1.a  RLParamsReales -> ε  RLParamsReales.a = RLParamsReales.ah  Ins -> Bloque  Ins.a = ins\_bloque(Bloque.a)  E0 -> E1 RE0  RE0.ah = E1.a  E0.a = RE0.a  RE0 -> **+** E0  RE0.a = suma(RE0.ah, E0.a)  RE0 -> **-** E1  RE0.a = resta(RE0.ah, E1.a)  RE0 -> ε  RE0.a = RE0.ah  E1 -> E2 RE1  RE1.ah = E2.a  E1.a = RE1.a  RE1 -> OpN1 E2 RE1  RE11.ah = exp(opN1.op, RE10.ah, E2.a)  RE1 -> ε  RE1.a = RE1.ah  E2 -> E3 RE2  RE2.ah = E3.a  E2.a = RE2.a  RE2 -> OpN2 E3 RE2  RE21.ah = exp(opN2.op, RE20.ah, E3.a)  RE2 -> ε  RE2.a = RE2.ah  E3 -> E4 RE3  RE3.ah = E4.a  E3.a = RE3.a  RE3 -> OpN3 E4  RE3.a = exp(opN3.op, RE3.ah, E4.a)  RE3 -> ε  RE3.a = RE3.ah  E4 -> **-** E5  E4.a = menos(E5.a)  E4 -> **not** E4  E40.a = not(E41.a)  E4 -> E5  E4.a = E5.a  E5 -> E5 RE5  RE5.ah = E5.a  E5.a = RE5.a  RE5 -> **[** E0 **]**  RE5.a = index(RE5.ah, E0.a)  RE5 -> **. identificador**  RE5.a = access\_reg\_punto(RE5.ah, **identificador**.lexema)  RE5 -> **-> identificador**  RE5.a = access\_reg\_flecha(RE5.ah, **identificador**.lexema)  E5 -> E6  E5.a = E6.a  E5 -> E6 RRE5    RRE5.ah = E6.a    E5.a = RRE5.a  RRE5 -> RE5 RRE5    RRE51.ah = RE5.a    RRE50.a = RRE51.a    RE5.ah = RRE50.ah  RRE5 -> ε    RRE5.a = RRE5.ah  RE5 -> [ E0 ]    RE5.a = index(RE5.ah, E0.a)  RE5 -> **. identificador**    RE5.a = access\_reg\_punto(RE5.ah, i**dentificador**.lexema)  RE5 -> **-> identificador**    RE5.a = access\_reg\_flecha(RE5.ah, i**dentificador**.lexema)  E6 -> **\*** E6  E60.a = indireccion(E61.a)  E6 -> E7  E6.a = E7.a  E7 -> **identificador**  E7.a = identificador(**identificador**.lexema)  E7 -> **numEnt**  E7.a = numEnt(**numEnt**.lexema)  E7 -> **numReal**  E7.a = numReal(**numReal**.lexema)  E7 -> **true**  E7.a = true()  E7 -> **false**  E7.a = false()  E7 -> **cadena**  E7.a = cadena(**cadena**.lexema)  E7 -> **null**  E7.a = null()  E7 -> **(** E0 **)**  E7.a = E0.a  OpN1 -> **and**  opN1.a = “and”  OpN1 -> **or**  opN1.a = “or”  OpN2 -> **<**  opN2.a = “<”  OpN2 -> **>**  opN2.a = “>”  OpN2 -> **<=**  opN2.a = “<=”  OpN2 -> **>=**  opN2.a = “>=”  OpN2 -> **==**  opN2.a = “==”  OpN2 -> **!=**  opN2.a = “!=”  OpN3 -> **\***  opN3.a = “\*”  OpN3 -> **/**  opN3.a = “/”  OpN3 -> **%**  opN3.a = “%” |