Memoria práctica 4

GRUPO 06:

Jaime Delgado Linares

Juan Luis García Flores

Sintaxis abstracta

* **Representación de programas**

programa: Decs x Instrucciones -> Programa

* **Representación de declaraciones**

listDecs: LDecs -> Decs

listDecsVacia: -> Decs

seqSimpleDecs: Declaracion -> LDecs

seqCompuestaDecs: Decs x Declaracion -> LDecs

decV: DecVar -> Declaracion

decT: DecTipo -> Declaracion

decF: DecFun -> Declaracion

decVariable: DTipo0 x **String** -> DecVar

decTipo: DTipo0 x **String** -> DecTipo

decFun: Cabecera x Programa x **String** -> DecFun

* **Representación de procedimientos**

cabe: String x Parametros x TipoRet -> Cabecera

tReturn: DTipo0 -> TipoRet

sinReturn: -> TipoRet

params: LParametros -> Parametros

paramsVacios: -> Parametros

seqSimpleParams: Parametro -> LParametros

seqCompuestaParams: LParametros x Parametro -> Params

param: DTipo0 x **String** -> Parametro

paramRef: DTipo0 x **String** -> Parametro

* **Representación de expresiones de tipo**

dTipoPuntero: DTipoPuntero -> DTipo0

dTipo1: DTipo1 -> DTipo0

tInt: -> DTipo1

tReal: -> DTipo1

tipoid: **String** -> DTipo1

dTipoRegistro: DTipoRegistro -> DTipo1

dTipoObjeto: DTipoObjeto -> DTipo1

array: DTipo1 x numeroEntero -> DTipo1

pointer: DTipo0 -> DTipoPuntero

registro: LCampos -> DTipoRegistro

seqSimpleVars: DecVar -> LCampos

seqCompuestaVars: LCampos x DecVar -> DecVars

objeto: Superclase x Campos -> DTipoObjeto

superclase: **String** -> Superclase

superclaseVacia: -> Superclase

seqSimpleCampos: Campo -> Campos

seqCompuestaCampos: Campos x Campo -> Campos

campo: DecVar -> Campo

campoMet: DecMet -> Campo

decMet: Cabecera x Programa x **String** -> DecMet

* **Representación de instrucciones**

seqSimpleIs: Instruccion -> Instrucciones

seqComuestaIs: Instrucciones x Instruccion -> Instrucciones

iAsig: IAsig -> Instruccion

iIn: IIn -> Instruccion

iOut: IOut -> Instruccion

iAlloc: IAlloc -> Instruccion

iFree: IFree -> Instruccion

iIf: IIf -> Instruccion

iWhile: IWhile -> Instruccion

iReturn: IReturn -> Instruccion

invocFunMet: InvocacionFunMet -> Instruccion

iasig: Designador x Exp0 -> IAsig

iin: Designador -> IIn

iout: Designador -> IOut

ialloc: Designador -> IAlloc

ifree: Designador -> IFree

iIf: Casos x ParteElse -> IIf

seqSimpleCasos: Caso -> Casos

seqCompuestaCasos: Casos x Caso -> Casos

caso: Exp0 x Is -> Caso

instrucciones: Instrucciones -> Is

sinInstrucciones: -> Is

pElse: Is -> ParteElse

sinpElse: -> ParteElse

iwhile: Exp0 x Is -> IWhile

iReturn: Exp0 -> IReturn

llamada: Designador x ParamsInvoc -> InvocacionFunMet

paramsInvoc: LParamsInvoc -> ParamsInvoc

paramsInvocVacios: -> ParamsInvoc

seqSimpleExps: Exp0 -> LParamsInvoc

seqCompuestaExps: LParamsInvoc x Exp0 -> LParamsInvoc

* **Representación de designadores**

var: **String** -> Designador

otroDesignador: OtroDesignador -> Designador

desigArray: DesigArray -> Designador

desigAtributo: DesigAtributo -> Designador

desigPuntero: DesigPuntero -> Designador

dThis: -> OtroDesignador

dSuper: -> OtroDesignador

selCampo: Designador x **String** -> DesigAtributo

indxElem: Designador x Exp -> DesigArray

deref: Designador -> DesigPuntero

* **Representación de expresiones**

exp01: Exp1 x Op0NA x Exp1 -> Exp0

exp02: Exp1 -> Exp0

exp11: Exp1 x Op1 x Exp2 -> Ex1

exp12: Exp2 -> Exp1

exp21: Exp2 x Op2 x Exp3 -> Exp2

exp22: Exp3 -> Exp2

exp31: Op3 x Exp3 -> Exp3

exp32: Op3NA x Exp4 -> Exp3

exp33: Exp4 -> Exp3

numEntero: numeroEntero -> Exp4

numReal: numeroReal -> Exp4

mem: Designador -> Exp4

eNull: -> Exp4

invocFunMet: InvocacionFunMet -> Exp4

exp41: Exp0 -> Exp4

op0naMenor: -> Op0NA

op0naMayor: -> Op0NA

op0naMenorIgual -> Op0NA

op0naMayorIgual -> Op0NA

op0naIgual -> Op0NA

op0naDistinto: -> Op0NA

op1Or: -> Op1

op1Mas -> Op1

op1Menos: -> Op1

op2Por: -> Op2

op2Div: -> Op2

op2Mod: -> Op2

op2And: -> Op2

op3Menos: -> Op3

op3Not: -> Op3

op3naCastInt : -> Op3NA

op3naCastReal: -> Op3NA

Gramática de atributos

Programa -> Decs # Instrucciones

Programa.v = programa(Decs.v, Instrucciones.v)

Decs -> LDecs

Decs.v = listDecs(LDecs.v)

Decs -> ε

Decs .v = listDecsVacia()

LDecs -> LDecs ; Declaracion

LDecs.v = seqCompuestaDecs(LDecs.v, Declaracion.v)

LDecs -> Declaracion

LDecs.v = seqSimpleDecs(Declaracion.v)

Declaracion -> DecVariable

Declaracion.v = decV(DecVariable.v)

Declaracion -> DecTipo

Declaracion.v = decT(DecTipo.v)

Declaracion -> DecFun

Declaracion.v = decF(DecFun.v)

DecVariable -> DTipo0 Identificador

DecVariable.v = decVariable(DTipo0.v, Identificador.lex)

DecTipo -> Tipo DTipo0 Identificador

DecTipo.v = decTipo(DTipo0.v, Identificador.lex)

DTipo0 -> DTipoPuntero

DTipo0.v = dTipoPuntero(DTipoPuntero.v)

DTipo0 -> DTipo1

DTipo0.v = dTipo1(DTipo1.v)

DTipo1 -> Int

DTipo1.v = tInt()

DTipo1 -> Real

DTipo1.v = tReal()

DTipo1 -> Identificador

DTipo1.v = tipoid(Identificador.lex)

DTipo1 -> DTipoRegistro

DTipo1.v = dTipoRegistro(DTipoRegistro.v)

DTipo1 -> DTipoObjeto

DTipo1.v = dTipoObjeto(DTipoObjeto.v)

DTipo1 -> DTipo1 [ NumEntero ]

DTipo1.v = array(DTipo1.v, NumEntero.lex)

DTipoRegistro -> Rec LCampos EndRec

DTipoRegistro.v = registro(LCampos.v)

LCampos -> LCampos ; DecVariable

LCampos.v = seqCompuestaVars(LCampos.v, DecVariable.v)

LCampos -> DecVariable

LCampos.v = seqSimpleVars(DecVariable.v)

DTipoPuntero -> Pointer DTipo0

DTipoPuntero = pointer(DTipo0.v)

DTipoObjeto -> Object Superclase LCamposOMet EndObject

DTipoObjeto.v = objeto(SuperClase.v, LCamposOMet.v)

Superclase -> Extends Identificador

Superclase.v = superclase(Identificador.lex)

Superclase -> ε

Superclase.v = superclaseVacia()

LCamposOMet -> LCamposOMet ; Campo

LCamposOMet.v = seqCompuestaCampos(LCamposOMet.v, Campo.v)

LCamposOMet -> Campo

LCamposOMet.v = seqSimpleCampos(Campo.v)

Campo -> DecVariable

Campo.v = campo(DecVariable.v)

Campo -> DecMet

Campo.v = campo(DecMet.v)

DecFun -> Fun Cabecera Programa End Identificador

DecFun = decFun(Cabecera.v, Programa.v, Identificador.lex)

DecMet -> Method Cabecera Programa End Identificador

DecMet = decMet(Cabecera.v, Programa.v, Identificador.lex)

Cabecera -> Identificador Pap Parametros Pcierre TipoRet

Cabecera.v = cabe(Identificador.lex, Parametros.v, TipoRet.v)

Parametros -> LParametros

Parametros.v = params(LParametros.v)

Parametros -> ε

Parametros.v = paramsVacios()

LParametros -> LParametros, Parametro

LParametros.v = seqCompuestaParams(LParametros.v, Parametro.v)

LParametros -> Parametro

LParametros.v = seqSimpleParams(Parametro.v)

Parametro ->DTipo0 Identificador

Parametro.v = param(DTipo0.v, Identificador.lex)

Parametro -> DTipo0 & Identificador

Parametro.v = paramRef(DTipo0.v, Identificador.lex)

TipoRet -> Returns DTipo0

TipoRet.v = tReturn(DTipo0.v);

TipoRet -> ε

TipoRet.v = sinReturn()

Designador -> Identificador

Designador.v = var(Identificador.lex)

Designador -> OtroDesignador

Designador.v = otroDesignador(OtroDesignador.v)

Designador -> DesigArray

Designador.v = desigArray(DesigArray.v)

Designador -> DesigAtributo

Designador.v = desigAtributo(DesigAtributo.v)

Designador -> DesigPuntero

Designador.v = desigPuntero(DesigPuntero.v)

OtroDesignador -> This

OtroDesignador.v = dThis()

OtroDesignador -> Super

OtroDesignador.v = dSuper()

DesigArray -> Designador Cap Exp0 Ccierre

DesigArray.v = indxElem(Designador.v, Exp0.v)

DesigAtributo -> Designador. Identificador

DesigAtributo.v = selCampo(Designador.v, Identificador.lex)

DesigPuntero -> Designador^

DesigPuntero.v = deref(Designador.v)

Instrucciones -> Instrucciones ; Instruccion

Instrucciones.v = seqCompuestaIs(Instrucciones.v, Instruccion.v)

Instrucciones -> Instruccion

Instrucciones.v = seqSimpleIs(Instruccion.v)

Instruccion -> IAsig

Instruccion.v = iAsig(IAsig.v)

Instruccion -> IIn

Instruccion.v = iIn(IIn.v)

Instruccion -> IOut

Instruccion.v = iOut(IOut.v)

Instruccion -> IAlloc

Instruccion.v = iAlloc(IAlloc.v)

Instruccion -> IFree

Instruccion.v = iFree(IFree.v)

Instruccion -> IIf

Instruccion.v = iIf(IIf.v)

Instruccion -> IWhile

Instruccion.v = iWhile(IWhile.v)

Instruccion -> IReturn

Instruccion.v = iReturn(IReturn.v)

Instruccion -> InvocacionFunMet

Instruccion.v = invocFunMet(InovacionFunMet.v)

IAsig -> Designador = Exp0

IAsig.v = asig(Designador.v, Exp0.v)

IIn -> In Designador

IIn.v = iin(Designador.v)

IOut -> Out Designador

IOut.v = iout(Designdor.v)

IAlloc -> Alloc Designador

IAlloc.v = ialloc(Designador.v)

IFree -> Free Designador

IFree.v = ifree(Designador.v)

IIf -> If Casos ParteElse Endif

IIf.v = iIf(Casos.v, ParteElse.v)

ParteElse -> Else Is

ParteElse.v = pElse(Is.v)

ParteElse -> ε

ParteElse.v = sinpElse()

Casos -> Casos Elsif Caso

Casos0.v = seqCompuestaCasos(Casos1.v, Caso.v)

Casos -> Caso

Casos.v = seqSimpleCasos(Caso.v)

Caso -> Exp0 Then Is

Caso.v = caso(Exp0.v, Is.v)

Is -> Instrucciones

Is.v = instrucciones(Instrucciones.v)

Is -> ε

Is = sinInstrucciones()

IWhile -> While Exp0 Do Is Endwhile

IWhile.v = iwhile(Exp0.v, Is.v)

IReturn -> Return Exp0

IReturn.v = iReturn(Exp0.v);

Exp0 -> Exp1 Op0NA Exp1

Exp0.v = exp01(Exp10.v, Op0NA.v, Exp11.v)

Exp0 -> Exp1

Exp0.v = exp02(Exp1.v)

Exp1 -> Exp1 Op1 Exp2

Exp1.v = exp11(Exp1.v, Op1.v, Exp2.v)

Exp1 -> Exp2

Exp1.v = exp12(Exp2.v)

Exp2 -> Exp2 Op2 Exp3

Exp2.v = exp21(Exp2.v, Op2.v, Exp3.v)

Exp2 -> Exp3

Exp2.v = exp22(Exp3.v)

Exp3 -> Op3 Exp3

Exp3.v = exp31(Op3.v, Exp3.v)

Exp3 -> Op3NA Exp4

Exp3.v = exp32(Op3NA.v, Exp4.v)

Exp3 -> Exp4

Exp3.v = exp33(Exp4.v)

Exp4 -> NumEntero

Exp4.v = numEntero(NumEntero.lex)

Exp4 -> NumReal

Exp4.v = numReal(NumReal.lex)

Exp4 -> Designador

Exp4.v = mem(Designador.v)

Exp4 -> Null

Exp4.v = null()

Exp4 -> InvocacionFunMet

Exp4.v = invocFunMet(InvocacionFunMet.v)

Exp4 -> (Exp0)

Exp4.v = exp41(Exp0.v)

InvocacionFunMet -> Designador Pap ParamsInvoc Pcierre

InvocacionFunMet.v = llamada(Designador.v, ParamsInvoc.v)

ParamsInvoc -> LParamsInvoc

ParamsInvoc.v = paramsInvoc(LParamsInvoc.v)

ParamsInvoc -> ε

ParamsInvoc.v = paramsInvocVacios()

LParamsInvoc -> LParamsInvoc , Exp0

LParamsInvoc.v = seqCompuestaExps(LParamsInvoc.v, Exp0.v)

LParamsInvoc -> Exp0

LparamsInvoc.v = seqSimpleExps(Exp0.v)

Op0NA -> <

Op0NA.v = op0naMenor()

Op0NA -> >

Op0NA.v = op0naMayor()

Op0NA -> <=

Op0NA.v = op0naMenorIgual()

Op0NA -> >=

Op0NA.v = op0naMayorIgual()

Op0NA -> ==

Op0NA.v = op0naIgual()

Op0NA -> !=

Op0NA.v = op0naDistinto()

Op1 -> Or

Op1.v = op1Or()

Op1 -> +

Op1.v = op1Mas()

Op1 -> -

Op1.v = op1Menos()

Op2 -> \*

Op2.v = op2Por()

Op2 -> /

Op2.v = op2Div()

Op2 -> %

Op2.v = op2Mod()

Op2 -> And

Op2.v = op2And()

Op3 -> -

Op3.v = op3Menos()

Op3 -> !

Op3.v = op3Not()

Op3NA -> CastInt

Op3NA.v = op3naCastInt()

Op3NA -> CastReal

Op3NA.v = op3naCastReal()

Restricciones semánticas

* Registros
  + No se permiten nombres de campos duplicados en registros.
* Objetos
  + Para hacer referencia a una superclase debe estar declarada previamente.
  + No se permiten nombres de campos o métodos duplicados.
* Tipos
  + Para describir un tipo se utiliza un identificador de tipo que debió ser declarado previamente.
* Funciones y métodos
  + No puede haber parámetros duplicados.
  + El nombre de la función o método debe coincidir con el nombre indicado en el end de la misma.
  + Los nombres de los parámetros debe ser distinto de los nombres de variables, tipos y funciones declarados en la sección de declaraciones de la función o método.
  + El nombre la función o método no puede coincidir con el nombre de ninguno de los parámetros. Tampoco con los nombres de variables, tipos o funciones declarados en su sección de declaraciones.
* Operadores de comparación (<, >, <=, >=, ==, !=)
  + El tipo del primer y segundo argumento han de ser enteros o reales.
* Operadores lógicos binarios (||, &&)
  + El tipo del primer y segundo argumento han de ser enteros.
* Módulo (%):
  + El tipo de ambos operandos ha de ser entero.
* Negación lógica (!):
  + El tipo del operando ha de ser entero.
* Invocación a función
  + La función invocada deberá existir, el número de los parámetros reales deberá coincidir con el número de los correspondientes parámetros formales, y cada parámetro real deberá poder ser asignado a cada parámetro formal.
* Asignación
  + Para asignar un objeto a otro, estos deben tener la misma estructura de tipos.
    - Un objeto de tipo int puede asignarse a otro objeto de tipo int o a uno de tipo real.
    - Un objeto de tipo real únicamente puede asignarse a otro objeto de tipo real.
    - Un objeto de tipo array puede asignarse únicamente a otro objeto de tipo array. Además, ambos objetos deben tener el mismo número de elementos y tener tipos base estructuralmente compatibles.
    - Un objeto de tipo registro puede asignarse únicamente a otro objeto de tipo registro. Ambos registros deben tener exactamente el mismo número de campos.
    - Un objeto de tipo puntero puede asignarse únicamente a otro objeto de tipo puntero siempre y cuando los tipos base sean estructuralmente compatibles.
    - Un objeto de tipo objeto puede asignarse a un objeto del mismo tipo o de alguna de sus superclases.