Attribute Grammar

Símbolo	Predicados	Reglas Semánticas
Programa → *definiciones*:Definicion*	tabla[main]!=null	
Campo → *identificador*:String *tipo*:Tipo		
DefVariable:Definicion →*identificador*:String *tipo*:Tipo	tabla[identificador]==null	tabla[identificador]=DefVariable
TipoVoid:Tipo →		
TipoInt:Tipo →		
TipoReal:Tipo →		
TipoChar:Tipo →		
TipoArray:Tipo → *tam*:String *tipo*:Tipo		
TipoFuncion:Tipo →*parametros*:Definicion* *tipoRetorno*:Tipo*defLocales*:Definicion**sentencias*:Sentencia*		
TipoStruct:Tipo → *campos*:Campo*	campoi.identificador!=campoj.identificador	
TipoComplejo:Tipo → *nombreTipo*:String	tabla[nombreTipo]!=null	TipoComplejo.struct=tabla[nombreTipo]
SentenciaAsignacion:Sentencia →*left*:Expresion *right*:Expresion	left.direccion mismaClase(left.tipado, right.tipado)	
SentenciaWrite:Sentencia →*expresion*:Expresion*	sonTiposSimples(expresion)	
SentenciaRead:Sentencia →*expresion*:Expresion	expresion.direccion	
SentenciaReturn:Sentencia →*expresion*:Expresion*	SentenciaReturn.funcion.tipoRetorno==expresion0.tipado	
SentenciaWhile:Sentencia →*condicion*:Expresion *sentencias*:Sentencia*	expresion.tipado==TipoInt	
SentenciaIf:Sentencia →*condicion*:Expresion*sentenciasIf*:Sentencia**sentenciasElse*:Sentencia*	expresion.tipado==TipoInt	
SentenciaProcedimiento:Sentencia → *identificador*:String *entrada*:Expresion*	tabla[identificador]!=null |entrada|==|SentenciaProcedimiento.definicion.tipo.parametros| entradai.tipado== SentenciaProcedimiento.definicion.tipo.parametrosi.tipo	SentenciaProcedimiento.definicion=tabla[identificador]

ExpresionCast:Expresion → *tipo*:Tipo*expresion*:Expresion	~mismaClase(tipo, expresion.tipado) sonTiposSimples(tipo, expresion.tipado)	ExpresionCast.tipado=tipo
ExpresionChar:Expresion → *tipo*:Tipo *value*:String		ExpresionChar.tipado=TipoChar
ExpresionInt:Expresion → *tipo*:Tipo*value*:String		Expresionint.tipado=TipoInt
ExpresionReal:Expresion → *tipo*:Tipo*value*:String		ExpresionReal.tipado=TipoReal
ExpresionAritmetica:Expresion →*left*:Expresion *symbol*:String*right*:Expresion	mismaClase(left.tipado,right.tipado) left.tipado!=TipoChar	ExpresionAritmetica.tipado=left.tipado
ExpresionVariable:Expresion →*variable*:String	tabla[identificador]!=null	ExpresionVariable.definicion=tabla[identificador] ExpresionVariable.tipado=ExpresionVariable.definicion.tipo ExpresionVariable.direccion=TRUE
ExpresionAccesoArray:Expresion →*expresion1*:Expresion*expresion2*:Expresion	expresion1.tipado==TipoArray expresion2.tipado==TipoInt	ExpresionAccesoArray.tipado=expresion1.tipado.tipo ExpresionAccesoArray.direccion=TRUE
ExpresionAccesoCampo:Expresion → *expresion*:Expresion *nombre*:String	expresion.tipado==TipoComplejo expresion.tipado.struct.campo[nombre]!=null	ExpresionAccesoCampo.tipado=expresion.tipado.struct.campo[nombre].tipo ExpresionAccesoCampo.direccion=TRUE
ExpresionLlamadaFuncion:Expresion →*identificador*:String *entrada*:Expresion*	tabla[identificador]!=null |entrada|==|ExpresionLlamadaFuncion.definicion.tipo.parametros| entradai.tipado==ExpresionLlamadaFuncion.definicion.tipo.parametrosi.tipo	ExpresionLlamadaFuncion-definicion=tabla[identificador] ExpresionLlamadaFuncion.tipado=ExpresionLlamadaFuncion.definicion.tipo.tipoRetorno
ExpresionComparacion:Expresion →*left*:Expresion *symbol*:String*right*:Expresion	mismaClase(left.tipado, right.tipado) left.tipado!=TipoChar	ExpresionComparacion.tipado=TipoInt
ExpresionLogica:Expresion →*left*:Expresion *symbol*:String*right*:Expresion	mismaClase(left.tipado, right.tipado) left.tipado==TipoInt	ExpresionLogica.tipado=TipoInt
ExpresionNegacion:Expresion →*expresion*:Expresion	expresion.tipado==TipoInt	ExpresionNegacion.tipado=TipoInt
ExpresionMenosUnario:Expresion →*expresion*:Expresion	expresion.tipado==TipoInt	EpresionMenosUnario.tipado=expresion.tipado

Atributos

Nodo/Categoría	Atributo	Dominio (tipo)	Heredado/Sintetizado
ExpresionVariable	definicion	Definicion	h
ExpresionLlamadaFuncion	definicion	Definicion	h
SentenciaLlamadaprocedimiento	definicion	Definicion	h
TipoComplejo	struct	TipoStruct	h
Expresion	tipado	Tipo	h
Expresion	direccion	bool	s
SentenciaReturn	funcion	TipoFuncion	h

Conjuntos auxiliares

tabla TablaSimbolos

Funciones auxiliares

mismaClase(Tipo1, Tipo2): TRUE cuando son tipos simples y además el mismo

sonTiposSimples(Tipo[]): TRUE cuando todos son tipos simples

Code Specification

Función de Código	Plantillas de Código
Ejecutar: **Programa** → Instruccion*	∫ [[**Programa** → *definiciones*:Definicion*]] = #source "nombreFichero" call main halt for(def in definiciones) ejecutar[[def]]

ejecutar : **Definicion** → Instruccion*	ejecutar [[**DefVariable** → *identificador*:String *tipo*:Tipo]] =
	if(tipo instaceof tipoFuncion)
	#func identificador ejecutar[[tipo]]
	else if(tipo instaeof tipoStruct)
	#type identificador { ejecutar[[tipo]]
	else
	#var identificador : ejecutar[[tipo]]

ejecutar: **Tipo** → Instruccion*	ejecutar [[**TipoVoid** →]] = void ejecutar [[**TipoInt** →]] = int ejecutar [[**TipoReal** →]] = float ejecutar [[**TipoChar** →]] = byte ejecutar [[**TipoArray** → *tam*:String *tipo*:Tipo]] = tam*ejecutar[[tipo]] ejecutar [[**TipoFuncion** → *parametros*:Definicion* *tipoRetorno*:Tipo *defLocales*:Definicion* *sentencias*:Sentencia*]] = for(def in parametros) #param def.identificador : ejecutar[[def.tipo]]

```
    #ret ejecutar[[tipoRetorno]]
    for(def in defLocales)
      #local def.identificador : ejecutar[[def.tipo]]
    enter bytesLoc
    for(sent in sentencias)
      ejecutar[[sent]]
    if(tipoRetorno is TipoVoid)
      ret bytesReturn, bytesLoc, bytesParam
  ejecutar [[ TipoStruct → campos:Campo* ]] =
    for(campo in campos)
      campo.identificador : ejecutar[[campo.tipo]]
    }
  ejecutar [[ TipoComplejo → nombreTipo:String ]] =
    nombreTipo
```

ejecutar : **Sentencia** → Instruccion*

```
  ejecutar [[ SentenciaAsignacion → left:Expresion right:Expresion ]] =
    #line línea
    dirección[[left]]
    valor[[right]]
    storeleft.sufijo
  ejecutar [[ SentenciaWrite → expresion:Expresion* ]] =
    #line línea
    for(exp in expresion)
      writeexp.sufijo valor[[exp]]
  ejecutar [[ SentenciaRead → expresion:Expresion ]] =
    #line línea
    direccion[[expresion]]
    inexpresion.sufijo
    storeexpresion.sufijo
  ejecutar [[ SentenciaReturn → expresion:Expresion* ]] =
    #line línea
    ret bytesReturn, bytesLoc, bytesParam
  ejecutar [[ SentenciaWhile → condicion:Expresion sentencias:Sentenci
  a* ]] =
    etetiqueta:
    valor[[condición]]
    jz etetiqueta +1
    for(sent in sentencias)
      ejecutar[[sent]]
    jmp etetiqueta
    etetiqueta+1:
  ejecutar [[ SentenciaIf → condicion:Expresion sentenciasIf:Sentencia*
  sentenciasElse:Sentencia* ]] =
     valor[[condicion]]
     jnz etetiqueta
    for(sent int sentenciasIf)
      ejecutar[[sent]]
    jmp etetiqueta+1
    etetiqueta:
    for(sent in sentenciasElse)
      ejecutar[[sent]]
    etetiqueta+1:
  ejecutar [[ SentenciaProcedimiento → identificador:String entrada:E
  xpresion* ]] =
    #line línea
    for(exp in entrada)
      valor[[exp]]
```

```
  call identificador
  if( ~ (tipoFuncion.tipoRetorno is void) )
    poptipoFuncion.tipoRetorno.sufijo
```

valor : **Expresion** → Instruccion*

```
valor [[ ExpresionCast → tipo:Tipo expresion:Expresion ]] =
  valor[[expresion]]
  conversion(expresión.tipo, tipo)
valor [[ ExpresionChar → tipo:Tipo value:String ]] =
  pushb value
valor [[ ExpresionInt → tipo:Tipo value:String ]] =
  pushi value
valor [[ ExpresionReal → tipo:Tipo value:String ]] =
  pushf value
valor [[ ExpresionAritmetica → left:Expresion symbol:String right:Ex
presion ]] =
  valor[[left]]
  valor[[right]]
  simbolo(symbol)ExpresionAritmetica.tipo.sufijo
valor [[ ExpresionVariable → variable:String ]] =
  direccion[[ExpresionVariable]]
  loadExpresionVarable.tipo.sufijo
valor [[ ExpresionAccesoArray → expresion1:Expresion expresion2:E
xpresion ]] =
  direccion[[ExpresionAccesoArray]]
  loadExpresionAccesoArray.tipo.sufijo
valor [[ ExpresionAccesoCampo → expresion:Expresion nombre:Strin
g ]] =
  direccion[[ExpresionAccesoCampo]]
  loadExpresionAccesoCampo.tipo.sufijo
valor [[ ExpresionLlamadaFuncion → identificador:String entrada:Ex
presion* ]] =
  for(exp in entrada)
    valor[[exp]]
  call identificador
valor [[ ExpresionComparacion → left:Expresion symbol:String right:
Expresion ]] =
  valor[[left]]
  valor[[right]]
  simbolo(symbol)left.tipo.sufijo
valor [[ ExpresionLogica → left:Expresion symbol:String right:Expresi
on ]] =
  valor[[left]]
  valor[[right]]
  simbolo(symbol)
valor [[ ExpresionNegacion → expresion:Expresion ]] =
  valor[[expresion]]
  not
valor [[ ExpresionMenosUnario → expresion:Expresion ]] =
  valor[[expresion]]
  dupexpresion.tipo.sufijo
  dupexpresion.tipo.sufijo
  addexpresion.tipo.sufijo
  sub expresion.tipo.sufijo
```

direccion : **Expresion** → Instruccion*

```
direccion [[ ExpresionAccesoArray → expresion1:Expresion expresion2:Expresion ]] =
  direccion[[epresion1]]
  valor[[expresion2]]
  push ExpresionAccesoArray.definicion.tipo.tipo.size
  mul
  add

direccion [[ ExpresionAccesoCampo → expresion:Expresion nombre:String ]] =
  direccion[[expresión]]
  push ExpresionAccesoCampo.definicion.tipo.campos[nombre].offset
  add

direccion [[ ExpresionVariable → variable:String ]] =
  if(ExpresionVariable)
    pusha BP
    push ExpresionVariable.definicion.direccion
    add
  else
    push ExpresionVariable.definicion.direccion
```

simbolo(String): el operador adecuado (ejemplo: “+” → “add”)
conversion(Tipo1, Tipo1): la conversión entre los tipos (ejemplo: float a byte → “f2i\ni2b”)