Especificación de Código

Función de Código	Plantillas de Código
run : **Programa** → Instruccion*	run [[**programa** → *elemento**]] = #SOURCE\{sourceFile}\ CALL main halt define[[elemento]]
define : **Elemento** → Instruccion*	define[[**funcion** → *String parametros:definicion* tipo atributo* sentencia**]] = #LINE {start.line} {String}: ENTER { Σ atributo$_i$.tipo.tamaño} ejecuta[[sentencias$_i$]] Si tipo == tipovoid RET 0, Σ atributo$_i$.tipo.tamaño, Σ parametros$_i$.tipo.tamaño define[[**struct** → String definicion*]] = define[[**atributo** → definicion]] =
ejecuta : **Sentencia** → Instruccion*	ejecuta [[**expresionBinaria** → *left*:expresion String *right*:expresion]] = #LINE {end.line} direccion[[left]] valor[[right]] STORE<left.tipo> ejecuta [[**print** → expresion]] = #LINE {end.line} valor[[expresion]] OUT<expresion.tipo> ejecuta [[**read** → expresion]] = #LINE {end.line} direccion[[expresion]] IN<expresion.tipo> STORE<expresion.tipo> ejecuta[[**if** → condic:expresion verdadero:Sentencia* falso:Sentencia*]]= #LINE {end.line} valor[[condic]] Jz else{n} ejecuta[[verdadero]] Jmp fin_if{n} else{n}: ejecuta[[falso]] fin_if{n}: ejecuta[[**while**:sentencia → expresion sentencia*]] = #LINE {start.line} inicio_while{n}: valor[[expresion]] jz fin_while{n} ejecuta[[sentencia]] jmp inicio_while{n} fin_while{n}: ejecuta[[**invocarSentencia**→ string expresion*]] = #LINE {end.line}

valor[[expresion]]
CALL {string}
Si expresion.definicion.tipo ≠ Tipovoid
POP<expresion.definicion.tipo>

ejecuta[[**return** → expresion]] =
valor[[expresion]]
RET return.miFuncion.tipo.size, Σ $atributo_i$.tipo.tamaño, Σ $parametros_i$.tipo.tamaño

valor: **Expresion** → Instruccion*

valor[[**expresionBinaria** → left:expresion string right:expresion]]=
Si string ≠ "="
valor[[left]]
valor[[right]]
Si string == "+"
ADD<left.tipo>
Si string == "-"
SUB<left.tipo>
Si string == "*"
MUL<left.tipo>
Si string == "/"
DIV<left.tipo>
Si string == "=="
EQ<left.tipo>
Si string == "!="
NE<left.tipo>
Si string == ">"
GT<left.tipo>
Si string == ">="
GE<left.tipo>
Si string == "<"
LT<left.tipo>
Si string == "<="
LE<left.tipo>
Sino
direccion[[left]]
valor[[right]]
STORE<tipo>

valor[[**cast** → tipo expresion]]=
valor[[expresion]]
<expresion.tipo>2<cast.tipo>

valor[[**litent** → valor:int]]=
PUSH {valor}

valor[[**litchar** → valor:string]]=
Si string ≠ " '\\n' "
PUSHB {valor.codePointAt(1)}
Sino
PUSHB 10

valor[[**litreal** → valor:string]]=
PUSHF {valor}

valor[[**var** → nombre:string]]=
direccion[[var.definicion.direccion]]
LOAD<var.tipo>

valor[[**expresionUnaria** → expresion]]=
valor[[expresion]]
NOT

valor[[**expresionLogica** → left:expresion string right:expresion]]=
valor[[left]]
valor[[right]]
Si string == ”&&”
AND
Si string == “||”
OR

valor[[**accesoArray** → contenedor:expresion posicion:expresion]]=
direccion[[accesoArray]]
LOAD<contenedor.tipo.tipo>

valor[[**accesoStruct** → contenedor:expresion atributo:string]]=
direccion[[accesoStruct]]
LOAD
<contenedor.tipo.definicion.definiciones[definicion.nombre==atributo].tipo>

valor[[**entreParentesis** → contenido:expresion]]=
valor[[expresion]]

valor[[**invocarFuncion** → string expresion*]]=
valor[[expresion]]
CALL {string}

direccion:E**xpresion** → Instruccion*

direccion[[**var** → nombre:string]]=
Si (var.definicion.direccion < 0) OR (var.definicion.esParametro)
PUSHA BP
PUSH {var.definicion.direccion}
ADD
Sino
PUSHA {var.definicion.direccion}

direccion[[**accesoStruct** → contenedor:expresion atributo:string]] =
direccion[[contenedor]]
PUSH
contenedor.tipo.definicion.definiciones[definicion.nombre==atributo].direccion
ADD

direccion[[**accesoArray** → contenedor:expresion posicion:expresion]] =
direccion[[contenedor]]
valor[[posicion]]
PUSH contenedor.definicion.tipo.tipo.size
MUL
ADD