Especificación de código

Borja Rodriguez Lorenzo
UO258643@UNIOVI.ES | DISEÑO DE LENGUAJES DE PROGRAMACIÓN

Función	Plantillas de Código

run[[program**]]**	run[[**program** → definitions:definition*]] =
	  # SOURCE {file}
	  CALL main
	  HALT
	  define[[definitions]]

define[[definition**]]**	define[[**defVariable** → name:String *type*:type scope]] = # defVariable.scope defVariable.name: defVariable.maplName

	define [[**defStruct** → name:varType *param*:structField*]] =
	  #TYPE {defStruct.name.type}:{
	  define[[param$_i$]]
	  }

	define [[**structField** → name:String *type*:type]] = \t {structField.name}:{structField.type.maplName}

	define [[**defFunc** → name:String *args*:defVariable* returnType:type *definitions*:defVariable* *sentences*:sentence*]] =
	  {name}:
	  #FUNC {name}
	  #RET{returnType.maplName}
	  dirección[[args$_i$]]
	  valor[[returnType]]
	  dirección[[definitions$_i$]]
	  ENTER{∑definitions$_i$.type.size}
	  ejecuta[[sentences]]
	
	  si returnType == voidType // caso del main por ejemplo
	  RET {returnType.size}, {∑definitions$_i$.type.size}, {∑params$_i$.type.size}

ejecuta [[sentence**]]**	ejecuta [[**assignment** → left:expression *right*:expression]] = #LINE end.line direction[left] valor[right] STORE<left.type>
	ejecuta [[**ifElse** → expression:expression *if_sent*:sentence* else_*sent*:sentence*]] = valor[[expression]] JZ else {n} ejecuta[[if_sent$_i$]] JMP endElse {n} else {n}: ejecuta[[else_sent$_i$]] endElse {n}:
	ejecuta [[**while** → param:expression *sentence*:sentence*]] = #LINE {end.line} startWhile {n}: valor[[param]] JZ endWhile {n} ejecuta[[sentence$_i$]] JMP startWhile {n} endWhile {n}:
	ejecuta [[**return** → expression:expression]] = #LINE {end.line} valor[[expression]] RET {returnType.size}, {$\sum$definitions$_i$.type.size}, {$\sum$params$_i$.type.size}
	ejecuta [[**read** → expression:expression]] = PUSHA {expression.def.dir} in<expression.def.type>
	ejecuta [[**print** → expression:expression]] = #LINE (end.line)

	valor[[expression]] OUT_{<expresssion.type>}
	ejecta [[**println** → expression:expression]] = #LINE (end.line) Value[[expression]] OUT_{<expresssion.type>}
	ejecta [[**printsp** → expression:expression]] = #LINE (end.line) Value[[expression]] OUT_{<expresssion.type>}
	ejecta [[**funcSentence** → name:String *args*:expression*]] =
valor [[expression**]]**	valor [[**intConstant** → value:String]] = PUSHI{value}
	valor [[**realConstant** → value:String]] = PUSHF{value}
	valor [[**charConstant** → value:String]] = PUSHB{value}
	valor [[**variable** → valué: String]] = dirección[variable] LOAD_{<variable.type>}
	valor [[**arrayCall** → index:expression *expr*:expression]] =
	valor [[**fieldAccess** → expression:expression *name*:String]] = direction[[arrayCall]] LOAD_{<fieldAccess.type>}

valor [[**arithmeticExpr** → left:expression *operator*:String *right*:expression]] =
value[[left]]
value[[right]]
si operator == ”+”
ADD<arithmethicExpression.type>
Si operator == “-“
SUB<arithmethicExpression.type>
Si operator == “*”
MUL<arithmethicExpression.type>
Si operator == “/”
DIV<arithmethicExpression.type>

valor [[**comparationExpr** → left:expression *operator*:String *right*:expression]] =
value[[left]]
value[[right]]
si operator == ”>”
GT<left.type>
Si operator == “<“
LT<left.type>
Si operator == “>=”
GE<left.type>
Si operator == “<=”
LE<left.type>

valor [[**logicExpr** → left:expression *operator*:String *right*:expression]] =
valor[[left]]
valor[[right]]
si operator == “&&”
AND
si operator == “||”
OR

valor [[**negationExpr** → operator:String *expression*:expression]] =
valor[[expression]]
si operator == “!”

NOT_{<left.type>}
valor [[**castExpr** → type:type *expression*:expression]] = valor[[expression]] si type != intType AND
valor [[**funcExpr** → name:String *args*:expression*]] = valor[[args]] CALL {name}