Code Specification

Función	Plantillas de Código
run[[**program**]]	run[[**program** → *definitions*:definition*]] = #SOURCE {file} CALL main HALT define[[definitions$_i$]]

define[[**definition**]]	define[[**varDefinition** → *name*:String *type*:type]] =
	 #{varDefinition.scope} {name}:{MAPLType(type)}
	
	 define[[**structDefinition** → *name*:varType *definitions*:structField*]] =
	 #TYPE {name.type}:
	 {define[[definitions$_i$]]}
	
	 define[[**funDefinition** → *name*:String *params*:definition*
	   return_t:type *definitions*:varDefinition* *sentences*:sentence*]] =
	 {name}:
	 #FUNC {name}
	 #RET {MAPLType(return_t)}
	 address[[params$_i$]]
	 value[[return_t]]
	 address[[definitions$_i$]]
	 ENTER {∑definitions$_i$.type.size}
	 execute[[sentences$_i$]]
	 si return_t == voidType
	      RET {return_t.size},{∑definitions$_i$.type.size},{∑params$_i$.type.size}
	
	 define[[**structField** → *name*:String *type*:type]] =
	 {name}:{MAPLType(type)}

execute[[**sentence**]]	execute[[**print** → *expression*:expression]] = #LINE {end.line} value[[expression]] OUT<expression.type> execute[[**printsp** → *expression*:expression]] = #LINE {end.line} value[[expression]] OUT<expression.type> PUSHB 32 OUTB execute[[**println** → *expression*:expression]] = #LINE {end.line} value[[expression]] OUT<expression.type> PUSHB 10 OUTB execute[[**read** → *expression*:expression]] =

#LINE {end.line}
address[[expression]]
IN<expression.type>
STORE<expression.type>

execute[[**assignment** → *left*:expression *right*:expression]] =
#LINE {end.line}
address[[left]]
value[[right]]
STORE<left.type>

execute[[**return** → *expression*:expression]] =
#LINE {end.line}
value[[expression]]
RET {expression.type.size},{∑return.definition.definitions$_i$.type.size},{∑
return.definition .params$_i$.type.size}

execute[[**ifElse** → *expression*:expression *if_s*:sentence* *else_s*:sentence*]] =
#LINE {end.line}
value[[expression]]
Si else_s ¡= null
JZ else_{n}
Sino
JZ end_if_else_{n}
Si if_s ¡= null
execute[[if_s$_i$]]
JMP end_if_else_{n}
Si else_s ¡= null
else_{n}:
execute[[else_s$_i$]]
end_if_else_{n}:

execute[[**while** → *expression*:expression *sentence*:sentence*]] =
#LINE {end.line}
while_{n}:
value[[expression]]
JZ end_while_{n}
si sentence ¡= null
execute[[sentence$_i$]]
JMP while_{n}
while_{n}:

execute[[**funcInvocation** → *name*:String *args*:expression*]] =
#LINE {end.line}
value[[args$_i$]]
CALL {name}
Si funInvocation.definition.return_t ¡= voidType
POP< funInvocation.definition.return_t>

value[[**expression**]]   value[[**variable** → *name*:String]] =
address[[variable]]

LOAD<variable.type>

value[[**intConstant** → *value*:String]] =
        PUSH {value}

value[[**realConstant** → *value*:String]] =
        PUSHF {value}

value[[**charConstant** → *value*:String]] =
        PUSHB {value}

value[[**funcInvocationExpression** → *name*:String *params*:expression*]] =
        value[[params]]
        CALL {name}

value[[**arithmeticExpression** → *left*:expression *operator*:String *right*:expression]] =
        value[[left]]
        value[[right]]
        si operator == "+"
                ADD<arithmeticExpression.type>
        si operator == "-"
                SUB<arithmeticExpression.type>
        si operator == "*"
                 MUL<arithmeticExpression.type>
        si operator == "/"
                DIV<arithmeticExpression.type>

value[[**logicalExpression** → *left*:expression *operator*:String *right*:expression]] =
        value[[left]]
        value[[right]]
        si operator == "&&"
                AND
        si operator == "||"
                OR

value[[**unaryExpression** → *operator*:String *expr*:expression]] =
        value[[expr]]
        si operator == "!"
                NOT

value[[**comparableExpression** → *left*:expression *operator*:String *right*:expression]] =
        value[[left]]
        value[[right]]
        si operator == ">"
                GT<comparableExpression.type>
        si operator == "<"
                LT< comparableExpression.type>
        si operator == ">="
                 GE< comparableExpression.type>
        si operator == "<="
                LE< comparableExpression.type>
        si operator == "=="
                EQ< comparableExpression.type>

si operator == "!="
      NE< comparableExpression.type>

value[[**castExpression** → *type*:type *expr*:expression]] =
    value[[expr]]
    <expr.type>2<type>

value[[**fieldAccessExpression** → *expr*:expression *name*:String]] =
    address[[fieldAccessExpression]]
    LOAD< fieldAccessExpression.type>

value[[**indexExpression** → *expr*:expression *index*:expression]] =
    address[[indexExpression]]
    LOAD<indexExpression.type>

address[[**expression**]]   address[[**variable** → *name*:String]] =
    Si variable.definition.scope == GLOBAL
      PUSHA {variable.definition.address}
    Sino
      PUSHA BP
      PUSH {variable.definition.address}
      ADD

address[[**fieldAccessExpression** → *expr*:expression *name*:String]] =
    address[[expr]]
    PUSH {expr.type.field(name).address}
    ADD

address[[**indexExpression** → *expr*:expression *index*:expression]] =
    address[[expr]]
    value[[index]]
    PUSH {indexExpression.type.size}
    MUL
    ADD