Code Specification

Función	Plantillas de Código
run[[**program**]]	run[[**program** → *definitions*:definition*]] = #SOURCE {file} call main HALT define[[definitions]]
define[[**structField**]]	define[[**structField** → *name*:String *type*:type]] = {name} : {type}
define[[**definition**]]	define[[**definitionVariable** → *name*:String *type*:type]] = #GLOBAL {name} : {type}

	define[[**definitionStruct** → *name*:String *structFields*:structField*]] =
	#TYPE {name} : {
	define[[structFields]]
	}

	define[[**definitionFunction** → *name*:String *definitionFunctionParams*:definitionVariable* *type*:type *localVariables*:definitionVariable* *sentences*:sentence*]] = #FUNC {name} defineParam[[definitionFunctionParams]] #RET {type} defineLocalVariable[[localVariables]] {name}: enter {localVariables.size} execute[[sentences]] *if (!definitionFunction.hasGoodReturn) {* ret (0, localVariables.size, definitionFunctionParams.size} *}*
defineLocalVariable[[**definitionVariable**]]	defineLocalVariable[[**definitionVariable** → *name*:String *type*:type]] = #LOCAL {name} : {type}
defineParam[[**definitionVariable**]]	defineParam[[**definitionVariable** → *name*:String *type*:type]] = #PARAM {name} : {type}
execute[[**sentence**]]	execute[[**sentencePrint** → *expression*:expression]] = #LINE {end.line} *if (expression!= Ø) {* value[[expression]] out{expression.type.suffix} *}*
	execute[[**sentencePrintsp** → *expression*:expression]] = #LINE {end.line} *if (expression!= Ø) {* value[[expression]]

	out{expression.type.suffix} *}* pushb 32 outb
	execute[[**sentencePrintln** → *expression*:expression]] = #LINE {end.line} *if (expression!= Ø) {* value[[expression]] out{expression.type.suffix} *}* pushb 10 outb
	execute[[**sentenceReturn** → *expression*:expression]] = #LINE {end.line} *if (fatherFunction.type== TypeVoid) {* ret 0, {fatherFunction.localVariables.size}, {fatherFunction.definitionFunctionParams.size} *} else {* value[[expression]] ret {expression.type.size}, {fatherFunction.localVariables.size}, {fatherFunction.definitionFunctionParams.size} *}*
	execute[[**sentenceRead** → *expression*:expression]] = #LINE {end.line} address[[expression]] in{expression.type.suffix} store{expression.type.suffix}
	execute[[**sentenceAssignment** → *left*:expression *right*:expression]] = #LINE {end.line} address[[left]] value[[right]] store{left.type.suffix}
	execute[[**sentenceCallFunction** → *name*:String *callFunctionParams*:expression*]] = #LINE {end.line} value[[callFuntionParams]] call {definition.name} *if (definition.type!=TypeVoid) {* pop{definition.type.suffix} *}*
	execute[[**sentenceIf** → *condition*:expression *ifSentences*:sentence* *elseSentences*:sentence*]] = #LINE {start.line} value[[condition]] *if (hasElse) {* jz {elseLabel} *} else {* jz {endLabel} *}* execute[[ifSentences]] *if (hasElse) {* jmp {endLabel} {elseLabel}: execute[[elseSentences]] *}* {endLabel}:

	execute[[**sentenceWhile** → *condition*:expression *sentences*:sentence*]] = #LINE {start.line} {initLabel}: value[[condition]] jz {endLabel} execute[[sentences]] jmp {initLabel} {endLabel}
value[[**expression**]]	value[[**expressionConstantInt** → *value*:String]] = pushi {value}
	value[[**expressionConstantFloat** → *value*:String]] = pushf {value}
	value[[**expressionConstantChar** → *value*:String]] = pushfb{value}
	value[[**expressionCallFunction** → *name*:String *callFunctionParams*:expression*]] = #LINE {start.line} value[[callFunctionParams]] call {name}
	value[[**expressionUnary** → *operator*:String *expression*:expression]] = value[[expression]] {getOperation(operator, type.suffix)}
	value[[**expressionCast** → *newType*:type *expression*:expression]] = value[[expression]] {getCast(expression.type.suffix,newType.suffix)}
	value[[**expressionArithmetic** → *left*:expression *operator*:String *right*:expression]] = value[[left]] value[[right]] {getOperation(operator, left.type.suffix)}
	value[[**expressionRelational** → *left*:expression *operator*:String *right*:expression]] = value[[left]] value[[right]] {getOperation(operator, left.type.suffix)}
	value[[**expressionLogical** → *left*:expression *operator*:String *right*:expression]] = value[[left]] value[[right]] {getOperation(operator, left.type.suffix)}
	value[[**expressionVariable** → *name*:String]] = address[[this]] load{type.suffix}
	value[[**expressionStructField** → *struct*:expression *name*:String]] = address[[this]] load{type.suffix}

	value[[**expressionArray** → *array*:expression *index*:expression]] = address[[this]] load{type.suffix}
address[[**expression**]]	address[[**expressionVariable** → *name*:String]] = *if(isLocal) {* pusha BP push definition.address add *} else {* pusha definition.address *}*
	address[[**expressionStructField** → *struct*:expression *name*:String]] = address[[struct]] push {getRelativeAddress} add
	address[[**expressionArray** → *array*:expression *index*:expression]] = address[[array]] value[[index]] push {type.size} mul add