# 1 Code Specification

| Función de Código | Plantillas de Código |
|---|---|
| run[[**program**]] | run[[**programa** → ast:ast*]] =<br>#SOURCE {sourceFile}<br>call main<br>halt<br>[[ast]] |
| metadata[[def]] | metadata[[**defVar** → *type:*type *name:*String ]] =<br>#GLOBAL {name} : {type}<br>metadata[[**defStruct** → name:string parameter*]] =<br>#type {name} :{<br>{parameter$_i$.name}:{parameter.type}<br>}<br>metadata [[ **func** → *name*:String parameter* *retorno*:type *defVar** sentence* ]] =<br>{name}:<br>#FUNC {name}<br>#RET {retorno.name}<br>#PARAM parameter$_i$<br>#LOCAL defVar$_i$<br>ENTER { Σ defVar$_i$.type.size }<br>ejecuta[[sentence$_i$]]<br>if retorno == null<br>RET 0, {Σ defVar$_i$.type.size}, {Σ<br>parameter$_i$.definition.type.size} |
| execute[[**sentence**]] | execute[[**print** → string *expr* ]] =<br>value[[expr]]<br>out<expr.type><br>if string == println<br>pushb 10<br>outb<br>else if string == printsp<br>pushb 32<br>outb<br>execute[[**read** → *expr* ]] =<br>address[[expr]]<br>in<expr.type><br>store<expr.type><br>execute[[**asignment** → *left*:*expr* *right*:*expr*]] =<br>address[[left]]<br>value[[right]]<br>store<left.type><br>execute[[**ifSentence** → *condition*:*expr* *iftrue*:*sentence** ]] =<br>int label = getLabel<br>value[[condition]]<br>jz “label” label:<br>execute[[ifTrue$_i$]]<br>“label” label :<br>execute[[**ifElseSentence** → *condition*:*expr* *iftrue*:*sentence** *else1*:*sentence** ]] =<br>int label = getLabel<br>value[[condition]]<br>jz “label” label: |

| Función de Código | Plantillas de Código |
|---|---|
| | execute[[ifTrue$_i$]]<br>jmp “label” label+1<br>“label” label :<br>execute[[else1$_i$]]<br>“label” label+1 :<br>Execute[[**whileSentence** → *condition*:*expr sentence**]] =<br>int label = getLabel<br>“label” label :<br>Value[[condition]]<br>jz “label” label + 1<br>execute[[sentence$_i$]]<br>jmp “label” label<br>“label” label +1:<br>Execute[[**returnNode** → *expr*]] =<br>if expr != null<br>valor[[expr]]<br>RET {expr.type.size},{ Σ expr.func.varDef.type.size},{ Σ expr.func.parameter.definition.type.size} |
| value[[**expr**]] | value[[**exprAritmetica** → *left*:expression **op**:String *right*:expression ]] =<br>value[[left]]<br>value[[right]]<br>codeSelection(op)<left.type><br>value[[**exprLogica** → *left*:expression **op**:String *right*:expression ]] =<br>value[[left]]<br>value[[right]]<br>codeSelection(op)<left.type><br><br>value[[**exprLogicaNe** → *left*:expression]] =<br>value[[left]]<br>not<br><br>value[[**variable** → *name*:String ]] =<br>address[[variable]]<br>LOAD<variable.type><br><br>value[[**litEnt** → *value*:String ]] =<br>pushi {value}<br><br>value[[**litReal** → *value*:String ]] =<br>pushf {value}<br><br>value[[**litChar** → *value*:String ]] =<br>pushb {value}<br><br>value[[**parameter** → *name*:String type ]]=<br>address[[parameter]]<br>LOAD<parameter.type><br>value[[**acces** → *left:*expr *right:*expr ]]=<br>address[[acces]]<br>LOAD<acces.type><br><br>value[[**arrayAcces** → *left:*expr *right:*expr ]]=<br>address[[arrayAcces]]<br>LOAD<arrayAcces.type><br>Value[[**cast** → TypeToConvert*:*type expr]] =<br>Value[[expr]]<br>{expr.type}2{typeTojConvert}<br>Value[[**methodCallExpr** → name:string args:expr*]] =<br>Valor[[args]]<br>CALL{ name} |

| Función de Código | Plantillas de Código |
|---|---|
| address[[**expr**]] | address[[**variable** → *name*:String ]] =<br>if defVar.ambito == GLOBAL<br>PUSHA {defVar.address}<br>Else<br>PUSHA bp<br>PUSH {defVar.address}<br>add<br>address[[**parameter** → *name*:String type ]]=<br>pusha {defVar.address}<br>address[[**acces** → *left*:expr *right*:expr ]]=<br>address[[left]]<br>push {left.type.params(right.text).address}<br>add<br>address[[**arrayAcces** → *left*:expr *right*:expr ]]=<br>address[[left]]<br>push {left.type.size}<br>value[[right]]<br>mul<br>add |

## 1.1 Funciones auxiliares

| Función Auxiliar | Definición |
|---|---|
| codeSelection | Convierte los operadores de nuestro lenguaje en los de mapl |
| getLabel | Retorna un numero que va aumentando cada vez que se llama a la función |

## 1.2 Notas

**Nota**:

La notación Instruccion<sub><expresión de tipo></sub> representa a la versión adecuada de la instrucción para el tipo indicado.

Ejemplos:

LOAD<sub><int></sub> → LOADI
LOAD<sub><real></sub> → LOADF

**Metadata**:

Todas las líneas con metadatos (prefijadas con el símbolo #) son opcionales (ver el tutorial de MAPL para más información).