# Code Specification

| **Función de Código** | **Plantillas de Código** |
| --- | --- |
| run[[**program**]] | run[[**programa** → ast*:*ast*\**]] =  #SOURCE {sourceFile}  call main  halt  [[ast]] |
| metadata[[def]] | metadata[[**defVar** → *type:*type *name:*String ]] =  #GLOBAL {name} : {type}  metadata[[**defStruct** → name*:*stringparameter\*]] =  #type {name} :{  {parameteri.name}:{parameter.type}  }  metadata [[ **func** → *name*:String  parameter\* *retorno*:type *defVar\**  sentence\* ]] =  {name}:  #FUNC {name}  #RET {retorno.name}  #PARAM parameteri  #LOCAL defVari  ENTER { Σ defVari.type.size }  ejecuta[[sentencei]]  if retorno == null  RET 0, {Σ defVari.type.size}, {Σ parameteri.definition.type.size} |
| execute[[**sentence**]] | execute[[**print** → string *expr* ]] =  value[[expr]]  out<expr.type>  if string == println  pushb 10  outb  else if string == printsp  pushb 32  outb  execute[[**read** → *expr* ]] =  address[[expr]]  in<expr.type>  store<expr.type>  execute[[**asignment** → *left*:*expr* *right*:*expr*]] =  address[[left]]  value[[right]]  store<left.type>  execute[[**ifSentence** → *condition*:*expr* *iftrue*:*sentence\** ]] =  int label = getLabel  value[[condition]]  jz “label” label:  execute[[ifTruei]]  “label” label :  execute[[**ifElseSentence** → *condition*:*expr* *iftrue*:*sentence\* else1*:*sentence\** ]] =  int label = getLabel  value[[condition]]  jz “label” label:  execute[[ifTruei]]  jmp “label” label+1  “label” label :  execute[[else1i]]  “label” label+1 :  Execute[[**whileSentence** → *condition*:*expr* *sentence\**]] =  int label = getLabel  “label” label :  Value[[condition]]  jz “label” label + 1  execute[[sentencei]]  jmp “label” label  “label” label +1:  Execute[[**returnNode** → *expr*]] =  if expr != null  valor[[expr]]  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 ]] =  value[[left]]  value[[right]]  codeSelection(op)<left.type>  value[[**exprLogica** → *left*:expression **op**:String *right*:expression ]] =  value[[left]]  value[[right]]  codeSelection(op)<left.type>  value[[**exprLogicaNe** → *left*:expression]] =  value[[left]]  not  value[[**variable** → *name*:String ]] =  address[[variable]]  LOAD<variable.type>  value[[**litEnt** → *value*:String ]] =  pushi {value}  value[[**litReal** → *value*:String ]] =  pushf {value}  value[[**litChar** → *value*:String ]] =  pushb {value}  value[[**parameter** → *name*:String type ]]=  address[[parameter]]  LOAD<parameter.type>  value[[**acces** → *left:*expr *right:*expr ]]=  address[[acces]]  LOAD<acces.type>  value[[**arrayAcces** → *left:*expr *right:*expr ]]=  address[[arrayAcces]]  LOAD<arrayAcces.type>  Value[[**cast** → TypeToConvert*:*type expr]] =  Value[[expr]]  {expr.type}2{typeTojConvert}  Value[[**methodCallExpr** → name:string args:expr\*]] =  Valor[[args]]  CALL{ name} |
| address[[**expr**]] | address[[**variable** → *name*:String ]] =  if defVar.ambito == GLOBAL  PUSHA {defVar.address}  Else  PUSHA bp  PUSH {defVar.address}  add  address[[**parameter** → *name*:String type ]]=  pusha {defVar.address}  address[[**acces** → *left:*expr *right:*expr ]]=  address[[left]]  push {left.type.params(right.text).address}  add  address[[**arrayAcces** → *left:*expr *right:*expr ]]=  address[[left]]  push {left.type.size}  value[[right]]  mul  add |

## 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 |

## Notas

**Nota**:

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

Ejemplos:

LOAD<int> → LOADI

LOAD<real> → LOADF

**Metadata**:

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