**Code Specification**

|  |  |
| --- | --- |
| **Función** | **Plantillas de Código** |
| run[[**program**]] | run[[**program** → *definitions*:definition\* ]] = |
|  | #SOURCE {file} |
|  | CALL main |
|  | HALT |
|  | define[[definitionsi]] |
| define[[**definition**]] | define[[**varDefinition** → *name*:String *type*:type ]] = |
|  | #{varDefinition.scope} {name}:{MAPLType(type)} |
|  | define[[**structDefinition** → *name*:varType *definitions*:structField\* ]] = |
|  | #TYPE {name.type}: |
|  | {define[[definitionsi]]} |
|  | define[[**funDefinition** → *name*:String *params*:definition\* |
|  | *return\_t*:type *definitions*:varDefinition\* *sentences*:sentence\*]] = |
|  | {name}: |
|  | #FUNC {name} |
|  | #RET {MAPLType(return\_t)} |
|  | address[[paramsi]] |
|  | value[[return\_t]] |
|  | address[[definitionsi]] |
|  | ENTER {∑definitionsi.type.size} |
|  | execute[[sentencesi]] |
|  | si return\_t == voidType |
|  | RET {return\_t.size},{∑definitionsi.type.size},{∑paramsi.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.definitionsi.type.size},{∑ return.definition .paramsi.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\_si]]

JMP end\_if\_else\_{n}

Si else\_s ¡= null

else\_{n}:

execute[[else\_si]]

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[[sentencei]]

JMP while\_{n}

while\_{n}:

execute[[**funcInvocation** → *name*:String *args*:expression\* ]] = #LINE {end.line}

value[[argsi]]

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