Code Specification

Función	Plantillas de Código
run[[**program**]]	run[[**program** → *definitions*:definition*]] = #SOURCE {sourceFile} Call main halt

define[[**definition**]]	define[[**varDef** → *type*:type *name*:String]] =
	
	si ambito == 0
	#GLOBAL {name}: {getMAPLName(type)}
	si ambito == 1
	#LOCAL {name}: {getMAPLName(type)}
	si ambito == 2
	#PARAM {name}: { getMAPLName (type)}
	
	define[[**funcDef** → *ident*:String *type*:type
	localDefs:varDef* *sentence*:sentence*]] =
	
	#FUNC {ident}
	
	Ident :
	
	Si getLocalBytes() > 0
	Enter getLocalBytes()
	
	Si type.tipoRetorno == null
	Ret 0, getLocalBytes(), type.getSize()
	
	define[[**structDef** → *name*:String *records*:recordDef*]] =
	
	#TYPE {name} : {
	records$_i$.name : { getMAPLName (records$_i$.type)}
	}

ejecuta[[**sentence**]]	ejecuta[[**print** → *expression*:expression *tipoPrint*:int]] = value [[expression]] out<expression.tipo> si tipoPrint == 2

pushb {32} //print blank space
out

si tipoPrint == 2 //print new line
pushb {10}
out

ejecuta[[**assignment** → *left*:expression *right*:expression]] =

address[[left]]
value[[right]]
store<left.tipo>

ejecuta[[**callProcedure** → *ident*:String *arguments*:expression*]] =

value [[arguments$_i$]]
call {ident}
si callProcedure.definicion.tipo.tipoRetorno ¡= null
pop<callProcedure.definicion.tipo.tipoRetorno>

ejecuta[[**ifElse** → *condition*:expression *ifSentences*:sentence* *elseSentences*:

Value [[expression]]
Jz else$_i$
Ejecuta[[ifSentences$_i$]]
Jmp fin_else$_i$
Else$_i$:
Ejecuta [[elseSentences$_i$]]
Fin_else$_i$:

ejecuta[[**read** → *expression*:expression]] =

address[[expression]]
in<expression.tipo>
store<expression.tipo>

ejecuta[[**return** → *expression*:expression]] =

value [[expression]]
ret {return.tipoRetorno.size},{return.definicion.localBytes},
{return.definicion.tipo.size}

ejecuta[[**while** → *condition*:expression *whileSentences*:sentence*]] =

while$_i$:
value [[condition]]
jz fin_while$_i$
ejecuta [[whileSentences$_i$]]
jmp while$_i$

fin_while$_i$:

value[[**expression**]] value[[**arithmetic** → *left*:expression *operator*:String *right*:expression]] =

value [[left]]
value [[right]]
si operador == +
add<left.tipo>
si operador == -
sub<left.tipo>
si operador == *
mul<left.tipo>
si operador == /
div<left.tipo>

value[[**callFunction** → *ident*:String *arguments*:expression*]] =

value [[arguments$_i$]]
call {ident}

value[[**cast** → *castType*:type *expression*:expression]] =

{expression.tipo.suffix}2{castType.suffix}

value[[**comparison** → *left*:expression *operator*:String *right*:expression]] =

value [[left]]
value [[right]]
si operador == >
gt<left.tipo>
si operador == >=
ge <left.tipo>
si operador == <
lt<left.tipo>
si operador == <=
le<left.tipo>
si operador == ==
eq<left.tipo>
si operador == ¡=
ne<left.tipo>

value[[**fieldAccess** → *ident*:expression *fieldName*:String]] =

address [[fieldAccess]]
load<fieldAccess.tipo>

value[[**indexing** → *ident*:expression *index*:expression]] =

address[[indexing]]
load<indexing.tipo>

value[[**logic** → *left*:expression *operator*:String *right*:expression]] =

value [[left]]
value [[right]]
si operador == &&
and
si operador == ||
or

value[[**not** → *expression*:expression]] =

value [[expression]]
not

value[[**unaryMinus** → *expression*:expression]] =

value [[expression]]
push<expression.tipo> -1
mul<expression.tipo>

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

address[[variable]]
load<variable.tipo>

value[[**charConstant** → *value*:String]] =

pushb {value}

value[[**intConstant** → *value*:String]] =

push {value}

value[[**realConstant** → *value*:String]] =

pushf {value}

Address[[expression]] address[[**indexing** → *ident*:expression *index*:expression]] =

address[[ident]]
value[[index]]
push {indexing.tipo.size}
mul

add

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

si ámbito > 0 //param o local
pusha bp
push definición.address
add
si ámbito == 0 //global
pusha definición.address

address[[**fieldAccess** → *ident*:expression *fieldName*:String]] =

address[[ident]]

push {getRecordByName().address}

add

Funciones auxiliares:

Para cada tipo que lo necesite:

```
@Override
public String getMAPLName() {
    return "int";
}
```

En FunctionDefinition

```
public int getLocalBytes() {
    int total = 0;
    for (Definition definition : localDefs) {
        VarDefinition vdef = (VarDefinition) definition;
        total += vdef.getType().getSize();
    }
    return total;
}
```

En FunctionType

```java
@Override
public int getSize() {
    int bytes = 0;
    for (Definition definition : paramDefs) {
        if(definition instanceof VarDefinition)
        bytes += ((VarDefinition) definition).getType().getSize();
    }
    return bytes;
}
```

En StructType

```java
@Override
public int getSize() {
    int valor = 0;
    StructDef sdef = (StructDef) definicion;
    for(Definition d : sdef.getRecords()) {
        RecordDef rdef = (RecordDef) d;
        valor += rdef.getTipo().getSize();
    }
    return valor;
}

public RecordDef getRecordByName( String nombre ) {
    StructDef sdef = (StructDef) definicion;
    for(Definition d : sdef.getRecords()) {
        RecordDef rdef = (RecordDef) d;
        if(nombre.equals(rdef.getName())) {
            return rdef;
        }
    }
    return null;
}
```