

Code Specification

Functions	Code Templates
run[[program]]	<pre>run[[program → declaraciones*]] = #SOURCE\t" + "\" + getSpecification().getSourceFile() <call main> <Halt> Define[[declaraciones*]]</pre>
define[[declaraciones]]	<pre>define[[declaracionstructs:declaraciones → nombre:string definicion*]] = #type + declaracionstructs.getNombre() + ": {" Definicion*.foreach(definicion -> definicion.getIdENT() + ":" + definicion.getTipo().tipoMapl()); "}" define [[declaracionglobales:declaraciones → definicion]] = #global + declaracionglobales.getDefinicion().getIdENT() + ":" + declaracionglobales.getDefinicion().getTipo().tipoMapl() define [[declaracionfuncion:declaraciones → nombre:string argumento:definicion* tipo? variablesLocales:definicion* sentencia*]] = #func declaracionfuncion.getNombre() declaracionfuncion.getNombre() + ":" argumento*.foreach(arg -> #param + arg.getIdENT() + ":" + arg.getTipo().tipoMapl()); int acumuladorParam = argumento*.stream().mapToInt(p -> p.getTipo().getSize()).sum(); Int acumuladorVar = variablesLocales*.stream().mapToInt(p -> p.getTipo().getSize()).sum(); #ret + declaracionfuncion.getTipofunc().tipoMapl(); <enter> acumuladorVar ejecuta[[sentencia*]]</pre>

	<pre> If(declaracionfuncion.getTipo().isEmpty()) <ret 0,> acumuladorVar <,> acumuladorParam </pre>
ejecuta[[sentencia]]	<pre> ejecuta[[printSentencia:sentencia → expression*]] = Comment(printSentencia) expression*.forEach(expr -> { valor[[expr]] <out> expr.getTipoexpresion().sufijo(); }); ejecuta [[readSentencia:sentencia → expression*]] = Expression valor = expression*.getFirst(); Direccion[[valor]] <in> valor.getTipoexpresion.sufijo(); <store> valor.getTipoexpresion.sufijo(); ejecuta [[printspSentencia:sentencia → expression*]] = Comment(printspSentencia); Expression*.foreach(expr -> { Valor[[expr]] <out> expr.getTipoexpresion.sufijo(); <pushb 32> <outb> }); If(printspSentencia.getExpressions().size() == 0){ <pushb 32> </pre>

```

<outb>

}

ejecuta [[printlnSentencia:sentencia → expression*]] =

Comment(printlnSentencia)

Expression*.forEach(expr -> {

    Valor[[expr]]

    <out> expr.getTipoexpresion.sufijo()

    <pushb 10>

    <outb>

    });

If(printlnSentencia.getExpressions.size() == 0){

    <pushb 10>

    <outb>

    }

ejecuta [[funcionSentencia:sentencia → nombre:string argumento:expression*]] =

Valor[[argumento*]]

<call> nombre

If(funcionSentencia.getDeclaracionfuncion().getTipo().isPresent() &&
funcionSentencia.getDeclaracionfuncion().getTipo().get().getClass() != VoidTipo.class){

    < pop > funcionSentencia.getDeclaracionfuncion().getTipo().get().sufijo(); }

ejecuta [[asignacionSentencia:sentencia → left:expression expression]] =

Comment(asignacionSentencia) ;

Direccion[[left]]

Valor[[expression]]

```

```

<store>asignacionSentencia.getLeft().getTipoExpresion().sufijo();

f4[returnSentencia:sentencia → expression?] =

If(returnSentencia.getExpression().isPresent()){

Valor[[returnSentencia.getExpression().get()]]

Int returnSize = expression.getTipoexpresion().getSize()

int acumuladorParam = returnSentencia.getDeclaFuncion().argumento*.stream().mapToInt(p -
> p.getTipo().getSize()).sum();

Int acumuladorVar = r eturnSentencia.getDeclaFuncion().
variablesLocales*.stream().mapToInt(p->p.getTipo().getSize()).sum();

<ret >returnSize <, > acumuladorVar <, > acumuladorParam

}

ejecuta [ifSentencia:sentencia → condicion:expression entonces:sentencia* otro:sentencia*] =

Coment(ifSentencia);

Valor[[condicion]]

Int valorAleatorio = r.nextInt(0,50000);

String etiquetaElse = “else_”+valorAleatorio;

String etiquetaFinal = “end_if_else_”+valorAleatorio;

If(ifSentencia.otro().count() != 0){

<jz > etiquetaElse

Valor[[entonces*]]

<jmp > etiquetaFinal

Valor[[otro*]]

}else{

<jz > etiquetaFinal

Valor[[entonces*]]

}

```

	<p>EtiquetaFinal <: ></p> <p>ejecuta [[whileSentencia:sentencia → condicion:expression entonces:sentencia*]] =</p> <p>Int numero = r.nextInt(0,50000)</p> <p>String inicio_condicion = "inicio_condicion" + numero;</p> <p>String final_condicion = "final_condicion"+numero</p> <p>inicio_condicion <: ></p> <p>Valor[[condicion]]</p> <p><jz> final_condicion</p> <p>Ejecuta[[entonces*]]</p> <p><jmp > inicio_condicion</p> <p>Final_condicion <: ></p>
valor[[expression]]	<p>valor [[intExpresion:expression → intValue:int]] =</p> <p><pushi > intValue.getIntValue();</p> <p>valor [[realExpresion:expression → doubleValue:double]] =</p> <p><pushf > realExpresion.getDoubleValue();</p> <p>valor [[identificadorExpresion:expression → name:string]] =</p> <p>Direccion[[identificadorExpresion]]</p> <p><load> espression.getTipoExpresion().sufijo();</p> <p>valor [[charExpresion:expression → name:string]] =</p> <p>Int charValor = char.expresion.getName().charAt(0);</p> <p><pushb > charValor</p> <p>valor [[accesoArrayExpresion:expression → acceso:expression indice:expression]] =</p>

```
Direccion[[accesoArrayExpresion]]
```

```
<load>accesoArrayExpresion.getTipoexpresion().sufijo();
```

```
valor [[parentesisExpresion:expression → expression]] =
```

```
Valor[[expression]]
```

```
valor [[castExpresion:expression → tipo expression]] =
```

```
Valor[[expression]]
```

```
String desde = castExpresion.getExpresion().getTipoexpresion().sufijo();
```

```
String hacia = castExpresion.getTipoexpresion().sufijo();
```

```
if(desde == 'f' && hacia == 'b') {
```

```
    desde<2i>
```

```
    <i2>hacia
```

```
}else if(desde == 'b' && hacia == 'f') {
```

```
    desde<2i>
```

```
    <i2>hacia
```

```
}else {
```

```
    desde<2>hacia
```

```
}
```

```
valor [[negacionExpresion:expression → expression]] =
```

```
Valor[[expression]]
```

```
<not>
```

```
valor [[arithmeticExpresion:expression → left:expression operator:string right:expression]] =
```

```
Valor[[left]]
```

```
Valor[[Right]]
```

```

String ins = "";

switch (operator) {

case "+": {

ins = "add"; break;

}

case "-": {

ins = "sub"; break;

}

case "*": {

ins = "mul"; break;

}

case "/": {

ins = "div"; break;

}

case "%": {

ins = "mod"; break;

}

}

ins+arithmeticExpresion.getTipoexpresion().sufijo());

valor [[logicExpression:expression → left:expression operator:string right:expression]] =

Valor[[left]]

Valor[[Right]]

String op = operator

String inst = "";

```

```
switch (op) {  
  
  case "<":{  
  
    inst = "lt";break;  
  
  }  
  
  case ">":{  
  
    inst = "gt"; break;  
  
  }  
  
  case "<=": {  
  
    inst = "le"; break;  
  
  }  
  
  case ">=": {  
  
    inst = "ge"; break;  
  
  }  
  
  case "==":{  
  
    inst = "eq"; break;  
  
  }  
  
  case "!=":{  
  
    inst = "ne"; break;  
  
  }  
  
}  
  
inst+logicExpression.getLeft().getTipoexpresion().sufijo());  
  
valor[[boolExpression:expression → left:expression operator:string right:expression]] =
```


	<pre> Valor[[left]] Valor[[Right]] switch(operator) { case "&&":{ <and>; break; } case " ":{ <or>; break; } } valor[[acederCap:expression → left:expression right:string]] = Direccion[[acederCap]] <load>acederCap.getTipoExpresion().sufijo() valor[[funcionExpresion:expression → nombre:string argumentos:expression*]] = Valor[[argumentos*]] <call > nombre </pre>
direccion[[expressi on]]	<pre> direccion [[intExpresion:expression → intValue:int]] = error direccion [[realExpresion:expression → doubleValue:double]] = error direccion [[identificadorExpresion:expression → name:string]] = Definicion aux = identificadorExpresion.getDefinicion(); if(aux.scope == 0) { <pusha > aux.getAdress() </pre>

```
}else if(aux.scope == 1){
```

```
<pusha BP>
```

```
<pusha > aux.getAddress()
```

```
<add>
```

```
}else if(aux.scope == 2){
```

```
<pusha BP>
```

```
<pusha > -aux.getAddress()
```

```
<subi>
```

```
}
```

```
direccion [[charExpresion:expression → charValue:char]] =  
error
```

```
direccion [[accesoArrayExpresion:expression → acceso:expression indice:expression]] =  
direccion[[acceso]]
```

```
valor[[indice]]
```

```
<pushi >accesoArrayExpresion.getTipoexpresion().sufijo();
```

```
<mul>
```

```
<add>
```

```
direccion [[parentesisExpresion:expression → expression]] =  
If( parenthesisExpresion.getExpresion().isLvalue())
```

```
    Direccion[[expression]]
```

```
direccion [[castExpresion:expression → tipo expression]] =
```

```
If( castExpresion.getExpresion.isLvalue)  
    Direccion[[expression]]
```

```
direccion [[negacionExpresion:expression → expression]] =  
error
```

```
direccion [[arithmeticExpresion:expression → left:expression operator:string right:expression]] =  
error
```

```
direccion [[logicExpresion:expression → left:expression operator:string right:expression]] =
```

	<p>error</p> <p>direccion [[boolExpression:expression → left:expression operator:string right:expression]] = error</p> <p>direccion [[acederCap:expression → left:expression right:string]] = direccion[[left]] StringTipo struct = (StringTipo) acederCap.getLeft().getTipoExpresion(); Struct.getDefinicions().forEach(p -> { If(p.getIdENT().equals(acederCap.getRight())){ <pushi > p.getAddress() } }); <add></p> <p>direccion [[funcionExpresion:expression → nombre:string argumentos:expression*]] = Error</p>
--	---

Auxiliary Functions

Name	Description
TipoMapl()	Devuelve el tipo para que MAPL entienda.
Coment(AbstractSentencia abstractSentencia)	Devuelve una linea con la informacion del AST
Sufijo()	Devuelve el sufijo del tipo para que MAPL lo entienda.