Code Specification

Functions	Code Templates
run⟦**program**⟧	run⟦**program** → declaraciones*⟧ = #SOURCE\t" + "\"" + getSpecification().getSourceFile() <call main> <Halt> Define[[declaraciones*]]

define⟦**declaraciones**⟧	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*]]

	If(declaracionfuncion.getTipo().isEmpty())
	
	<ret 0,> acumulador var <, > acumuladorParam
	
	else
	
	<ret >tipo.get().getSize()<, >acumuladorParam

ejecuta⟦**sentencia**⟧	ejecuta⟦**printSentencia**:sentencia → expression*⟧ =
	
	Comment(printSentencia)
	
	expression*.forEach(expr -> {
	
	valor[[expr]]
	
	<out> expr.getTipoexpresion().sufijo();
	
	});
	
	ejecuta ⟦**readSentencia**:sentencia → expression*⟧ =
	
	Expresion 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>
	
	<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[[argumeto*]]
	
	<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>asignacionSentecia.getLeft().getTipoExpresion().sufijo();
	
	f_4⟦**returnSentencia**:sentencia → expression?⟧ =
	
	If(returnSentencia.getExpression().isPresent()){
	
	Valor[[returnSentencia.getExpression().get()]]
	
	}
	
	ejecuta ⟦**ifSentencia**:sentencia → condicion:expression entonces:sentencia* otro:sentencia*⟧ =
	
	Coment(ifSentencia);
	
	Valor[[condicion]]
	
	Int valorAleatorio = r.nextInt(0,50000);
	
	String etiquetaElse = “else_”+valorAletorio;
	
	String etiquetaFinal = “end_if_else_”+valorAletorio;
	
	If(ifSentencia.otro().count() != 0){
	
	<jz > etiquetaElse
	
	Valor[[entonces*]]
	
	<jmp > etiquetaFinal
	
	Valor[[otro*]]
	
	}else{
	
	<jz > etiquetaFinal
	
	Valor[[entonces*]]
	
	}
	
	EtiquetaFinal <: >

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

	Valor[[expression]]
	
	valor ⟦**castExpresion**:expression → tipo expression⟧ =
	
	Valor[[expression]]
	
	String desde = castExpresion.getExpression().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]] =
	
	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

direccion⟦**expression**⟧	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.getAdrress()
	}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 ⟦**accessoArrayExpresion**:expression → acceso:expression indice:expression⟧ =
direccion[[acceso]]

valor[[indice]]

<pushi >accessoArrayExpresion.getTipoexpresion().sufijo();

<mul>

<add>

direccion ⟦**parentesisExpresion**:expression → expression⟧ =
If(parentesisExpresion.getExpression().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 ⟦**logicExpression**:expression → left:expression operator:string right:expression⟧ =
error

direccion ⟦**boolExpression**:expression → left:expression operator:string right:expression⟧ =
error

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.getAddres()
	}
	});
	<add>
	
	direccion [[**funcionExpresion**:expression → nombre:string argumentos:expression*]] =
	Error

Auxiliary Functions

Name	Description
TipoMapl()	Devulve 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.