Plantillas de código

Función de código	Plantilla de código
run: Programa -> Instrucción*	run[[Programa -> definiciones:Definicion*]]: #SOURCE ficheroEntrada CALL main HALT define[[definiciones$_i$]]

define: Definicion -> Instrucción*	define[[DefinicionVariable -> nombre:String tipo:Tipo]]:
	#VAR <nombre>:<tipo.nombreMAPL>
	
	define[[DefinicionFuncion -> nombre:String retorno:Tipo parametros:DefinicionVariable* definicionesVariable:DefinicionVariable* sentencias:Sentencia*]]:
	#FUNC <nombre>
	#PARAM <parametros$_i$.nombre>:<parametros$_i$.tipo.nombreMAPL>
	#LOCAL <definicionesVariable$_i$.nombre>:
	definicionesVariable$_i$.tipo.nombreMAPL>
	SI retorno != NULL
	#RET <retorno.nombreMAPL>
	<nombre>:
	enter <Σ definicionesVariable$_i$.tipo.size>
	ejecuta[[sentencias$_i$]]
	SI retorno == NULL
	RET 0,<Σ definicionesVariable$_i$.tipo.size>,<Σ locales$_i$.tipo.size>
	
	define[[DefinicionStruct `-> nombre:String listaCampos:Campo* primitivo:boolean`]]:
	#TYPE <nombre>:{
	define[[listaCampos$_i$]]
	}
	
	define[[Campo -> nombre:String tipo:Tipo]]:
	<nombre>:<tipo.nombreMAPL>

ejecuta: Sentencia -> Instrucción*	ejecuta[[Asignacion -> izquierda:Expresion derecha:Expresion]]: #Line {linea} direccion[[izquierda]] valor[[derecha]] STORE<izquierda.tipo.sufijo> ejecuta[[IF -> expresion:Expresion sentenciasIF:Sentencia* sentenciasElse:Sentencia*]]: #Line {linea} valor[[expresion]] JZ else<n> ejecuta[[sentenciasIF$_i$]] JMP fin_if<n> else<n>:

	ejecuta[[sentenciasElse$_i$]] fin_if<n>: ejecuta[[InvocacionProcedimiento -> nombre:String expresiones:Expresion*]]: #Line {linea} valor[[expresiones$_i$]] CALL <nombre> ejecuta[[Print -> expresion:Expresion*]]: #Line {linea} valor[[expresion]] OUT<expresion.tipo.sufijo> ejecuta[[Read -> expresion:Expresion*]]: #Line {linea} direccion[[expresion]] IN<expresion.tipo.sufijo> STORE<expresion.tipo.sufijo> ejecuta[[Return -> expresion:Expresion*]]: #Line {linea} SI expresion != NULL valor[[expresion]] RET expresion.tipo.size, <Σdefinicion.locales$_i$.tipo.size>, <Σdefinicion.parametros$_i$.tipo.size> ELSE RET 0, <Σdefinicion.locales$_i$.tipo.size>, <Σdefinicion.parametros$_i$.tipo.size> ejecuta[[While -> expresion:Expresion* sentencias:Sentencia*]]: #Line {linea} inicioWhile<n>: valor[[expresion]] JZ finWhile<n>: ejecuta[[sentencias$_i$]] JMP inicioWhile<n> finWhile<n>:
valor: Expresion-> Instrucción*	valor[[AccesoArray -> izquierda:Expresion derecha:Expresion]]: dirección[[AccesoArray]] load<izquierda.tipo.tipo.sufijo> valor[[AccesoCampo -> izquierda:Expresion derecha:Expresion]]: dirección[[AccesoCampo]] load<izquierda.tipo.campo(derecha.nombre).tipo.sufijo > valor[[Aritmetica -> op1:Expresion op2:Expresion operador:String]]: valor[[op1]] valor[[op2]] SI operador == '+'

```
            ADD<tipo.sufijo>
        SI operador == '-'
            SUB<tipo.sufijo>
        SI operador == '*'
            MUL<tipo.sufijo>
        SI operador == '/'
            DIV<tipo.sufijo>

valor[[Cast -> tipoCast:Tipo expresion:Expresion]]:
    valor[[expresion]]
    cast(expresión.tipo, tipoCast)

valor[[Comparacion -> operando1:Expresion
operando2:Expresion operador:String]]:
    valor[[op1]]
    valor[[op2]]
    SI operador == '>'
        GT<op1.tipo.sufijo>
    SI operador == '<'
        LT< op1.tipo.sufijo>
    SI operador == '>='
        GE< op1.tipo.sufijo>
    SI operador == '<='
        LE< op1.tipo.sufijo>
    SI operador == '=='
        EQ< op1.tipo.sufijo>
    SI operador == '!='
        NE< op1.tipo.sufijo>

valor[[InvocacionFuncion-> identificador:String
listaExpresiones:Expresion*]]:
    valor[[expresiones_i]]
    CALL <nombre>

valor[[LiteralCaracter-> caracter:Character]]:
    PUSH<tipo.sufijo> <caracter>

valor[[LiteralEntero-> valor:Int ]]:
    PUSH<tipo.sufijo> <valor>

valor[[LiteralReal-> valor:Double]]:
    PUSH<tipo.sufijo> <valor>

valor[[Logica -> operando1:Expresion operando2:Expresion
operador:String]]:
    valor[[operando1]]
    valor[[operando2]]
    SI operador == 'and'
        AND
    SI operador == 'or'
        OR

valor[[Negacion -> expresion:Expresion]]:
    valor[[expresion]]
```

	NOT valor[[Variable -> nombre:String definicion:Definicion]]: direccion[[Variable]] LOAD<tipo.sufijo>
direccion: Expresion-> Instrucción*	dirección[[AccesoArray -> izquierda:Expresion derecha:Expresion]]: direccion[[izquierda]] valor[[derecha]] PUSH <izquierda.tipo.tipo.size> MUL ADD direccion[[AccesoCampo -> izquierda:Expresion derecha:Expresion]]: dirección[[izquierda]] PUSH <izquierda.tipo.offsetCampo(derecha)> ADD direccion[[Variable -> nombre:String definicion:Definicion]]: SI definicion.ambito == 'GLOBAL' PUSHA <definicion.direccion> SI NO PUSHA BP PUSH <definicion.direccion> ADD

Método	Función
cast(Tipo tipoOrigen, Tipo tipoDestino)	Método que permite imprimir las instrucciones que realizan el cast entre dos tipos, en función del tipo origen y el tipo destino.