Especificación de Código

Función de Código	Plantillas de Código

run[[**Programa**]]	run[[**Programa** → *definiciones*:Definicion*]] =
	
	  #SOURCE {file}
	  CALL main
	  HALT
	  define⟦definiciones$_i$⟧

define[[**Definicion**]]	define[[**FunctionDeclaration** → *nombre*:String *parametros*:DefinicionVariable* *retorno*:Tipo *locales*:DefinicionVariable* *sentencias*:Sentencia*]] =
	  #LINE {end.line}
	  {nombre}
	  ENTER {defVariable.type.size}
	  Sentencias.forEach(s -> execute[[s]])
	  IF sentencias.size > 0
	    IF tipo == Void && !(lastStatement instanceOf Return)
	      RET {0, sizeLocales, sizeParameters}
	  ELSE
	    RET {0, sizeLocales, sizeParameters}
	
	define[[**VariableDeclaration** -> ID:String tipo:Type]] =
	  #GLOBAL {end.line}
	
	define[[**StructDeclaration** -> ID:String def:VaraibleDeclarations]] =
	  #TYPE {end.line}

execute[[**Sentencia**]]	

	execute[[**Print** -> exps:List<Expression>]] =
	  exps.forEach(e -> {value[[e]]; OUT + e.tipo.suffix})
	
	execute[[**PrintSp** -> exps:List<Expression>]] =
	  IF exps.isEmpty()
	    PUSH 32
	    OUTB}
	  ELSE
	    exps.forEach(e -> {
	        value[[e]]
	        OUT + e.tipo.suffix
	        PUSH 32
	        OUTB
	    })
	
	execute[[**PrintLn** -> exps:List<Expression>]] =
	  IF exps.isEmpty()
	    PUSH 10
	    OUTB}
	  ELSE
	    exps.forEach(e -> {
	        value[[e]]
	        OUT + e.tipo.suffix
	        PUSH 10
	        OUTB
	    })
	
	execute[[**Read** -> exp:Expression]] =
	  address[[exp]];
	  IN + exp.tipo.suffix;
	  STORE + exp.tipo.suffix

```
execute[[If -> exp:Expression, s1:List<Statement>, s2:List<Statement>]] =
        value[[exp]];

        if s2.isEmpty() then
                {JZ "finIf" + ifID}
        else
                {JZ "else" + ifID};

        s1.forEach(st -> execute[[st]]);

        if !(s1.lastElement instanceof Return)
                {JMP "finIf" + ifID};

        if !s2.isEmpty() {
                {LABEL "else" + ifID};
                s2.forEach(st -> execute[[st]]);
        }

        {"finIf" + ifID}

execute[[While -> exp:Expression, statements:List<Statement>]] =
        {LABEL "while" + whileID};

        value[[exp]];

        {JZ "finWhile" + whileID};

        statements.forEach(st -> execute[[st]]);

        {JMP "while" + whileID};

        {"finWhile" + whileID}

execute[[Return -> expression:Optional<Expression>,
funcion:FunctionDeclaration]] =

        if !expression.isPresent()
                {RET 0, sizeLocales, sizeParameters}
        else {
                value[[expression.get()]];
                {RET funcion.tipo.size, sizeLocales, sizeParameters}
         }

execute[[Asignacion -> e1:Expression, e2:Expression]] =
        address[[e1]];
        value[[e2]];
        STORE + e1.tipo.suffix

execute[[FuncionLlamada -> ID:String, exps:List<Expression>,
functionDeclaration:FunctionDeclaration]] =
        if exps != null then
                value[[exps]];

        {CALL ID};

         if !(functionDeclaration.tipo instanceof VoidType)
                {POP + functionDeclaration.tipo.suffix}
```

value[[**Expresion**]]	value[[**Cast** -> targetType:Type, expression:Expression]] =
	    value[[expression]];
	
	    if suffExp == "f" && suffTarget == "b" then
	        {F2I; I2B}
	    else if suffExp == "b" && suffTarget == "f" then
	        {B2I; I2F}
	    else
	    {suffExp + "2" + suffTarget}
	
	value[[**StructAccess** -> expression:Expression, ID:String]] =
	    address[[StructAccess]];
	    LOAD + tipo.suffix
	
	value[[**ArrayAccess** -> e1:Expression, e2:Expression]] =
	    address[[this]];
	    LOAD + tipo.suffix
	
	value[[**ExpresionLlamada** -> ID:String, expressions:List<Expression>]] =
	    value[[expressions]];
	    CALL ID
	
	value[[**Not** -> expression:Expression]] =
	    value[[expression]];
	    NOT
	
	value[[**ExpresionAritmetica** -> e1:Expression, op:String, e2:Expression]] =
	    value[[e1]];
	    value[[e2]];
	
	    switch op
	        "+" -> {ADD + e1.tipo.suffix}
	        "-" -> {SUB + e1.tipo.suffix}
	        "*" -> {MUL + e1.tipo.suffix}
	        "/" -> {DIV + e1.tipo.suffix}
	        "%" -> {MOD + e1.tipo.suffix}
	
	value[[**ExpresionLogica** -> e1:Expression, op:String, e2:Expression]] =
	    value[[e1]];
	    value[[e2]];
	
	    switch op of
	        "&&" -> {AND}
	        "||" -> {OR}
	        "<" -> {LT + e1.tipo.suffix}
	        "<=" -> {LE + e1.tipo.suffix}
	        ">" -> {GT + e1.tipo.suffix}
	        ">=" -> {GE + e1.tipo.suffix}
	        "==" -> {EQ + e1.tipo.suffix}
	        "!=" -> {NE + e1.tipo.suffix}
	
	value[[**Variable** -> ID:String, variableDeclaration:VariableDeclaration]] =
	    address[[Variable]];
	    LOAD + variableDeclaration.tipo.suffix
	
	value[[**LitEnt** -> LITENT:String]] =
	    PUSH LITENT
	
	value[[**LitReal** -> LITREAL:String]] =
	    PUSHF LITREAL
	
	value[[**LitChar** -> CHAR_LITERAL:String]] =
	    PUSHB CHAR_LITERAL

address[[**Expresion**]]	address[[**StructAccess** -> expr:Expresion ID:String]] = addres[[ID]] PUSH {ID.tipo.definicion.defVariable[ID].address} ADD
	address[[**ArrayAccess** -> exp1:Expresion exp2:Expresion]]= address[[exp1]] PUSH {exp1.tipo.size} value[[exp2]] MUL ADD

	address[[**Variable** -> ID:String]] =
	IF Variable.definicion.ambito == “parametro”
	PUSHA BP
	PUSH {variable.definition.address}
	ADD
	ELSE
	IF Variable.definicion.ambito == “local”
	PUSHA BP
	PUSH {variable.definition.address}
	ADD
	ELSE
	PUSH {variable.definition.address}