Especificación de Código

Función de Código	Plantillas de Código
run⟦**Programa**⟧	run⟦**program** → definition*⟧ = #SOURCE {file} #LINE {end.line} <call main> <halt> definition.forEach(definition -> execute⟦definiciones$_i$⟧)

execute⟦**Sentencia**⟧	execute[[**FunctionDefiniton**:definition -> name: string params:varDefinition* type: type? definitions:varDefinition* statements:statement*]] =
	
	<name>:
	int cte1 = type.size;
	int cte2 = definitions.mapToInt(vardef -> vardef.type.size).sum();
	int cte3 = params.mapToInt(param -> param.type.size).sum();
	
	statements.forEach(stmt -> execute[[stmt]](cte1, cte2, cte3));
	
	if(cte1 == 0) ret <cte1>,<cte2>,<cte3>
	
	execute ⟦**read:statement** → *expression*⟧ =
	#LINE {end.line}
	address[[*expression*]]
	in<expression.type>
	store<expression.type>
	
	execute ⟦**print**:statement → expression*⟧ =
	#LINE {end.line}
	expression*.forEach(exp → {
	value[[exp]];
	out <exp.type>
	});
	
	execute ⟦**println**:statement → expression*⟧ =
	#LINE {end.line}
	expression*.forEach(exp → {
	value[[exp]]
	out <exp.type>
	
	pushb 10 //Código ASCII del salto de línea
	outb
	});
	
	if(!expression){
	pushb 10
	outb
	}
	execute ⟦printsp:statement → expression*⟧ =
	#LINE {end.line}
	expression*.forEach(exp → {
	value[[exp]]
	out <exp.type>
	pushb 32 //Código ASCII del espacio
	outb
	});
	
	if(!expression){
	pushb 32
	outb
	}
	execute[[**return:statement** -> expression: expression?]] (cte1, cte2, cte3) =
	
	#LINE{end.line}
	if(cte1 != 0){
	ret <cte1>,<cte2>,<cte3>
	}
	
	execute ⟦**assignment:statement** → *left*:Expresion *right*:Expresion⟧ =
	
	#LINE {end.line}
	Address[[left]]
	value[[right]]
	store<left.type>
	
	execute[[while:statement → expression statement*]] =
	#LINE {end.line}
	<loop:>

```
                value[[expression]]
                <jz exit>
                statement*.forEach(stmt→ execute[[stmt]])
                <jmp loop>
                <exit:>

execute[[Ifelse:statement → expression tr: statement* fs:statement*]] =
                String jzLabel = "label" + ifelseActualLabel++;
               String jmpLabel = "label" + ifelseActualLabel++;
                #LINE {end.line}
                value[[expression]]
                <jz> jzLabel
                tr.forEach(stmt → execute[[stmt]])
                <jmp> jmpLabel
                fs.forEach(stmt → execute[[stmt]])
                 jmpLabel<:>

execute[[FunctionCallStatement:statement → ID expression*]] =
                #LINE {end.line}
                value[[functionCallStatement]]

                if(functionCallStatement.defininition.type){
                        pop <functionCallStatement.type>
                }
```

value[[**Expresion**]]

```
value [[intLiteral:expression → intValue:int]] =
                pushi{intValue}

value [[floatLiteral:expression → floatValue:float]] =
                pushf{floatValue}

value [[charLiteral:expression → name:string]] =
                pushb{name}

value [[arrayAccess:expression → expr1:expression expr2:expression]] =
                address[[arrayAccess]]
                <load>arrayAccess.type

value [[fieldAccess:expression → expr:expression name:string]] =
                address[[fieldAccess]]
                <load>fieldAccess.attrDefinition.type

value [[not:expression → expression]] =
                value[[expression]]
                <not>

value [[logic:expression → left:expression operator:string right:expression]] =
                value[[left]]
                value[[right]
                if operator == “&&”
                        and
                if operator == “||”
                        or
                if operator == “>=”
                        ge<arithmetic.type>
                if operator == “<=”
                        le<arithmetic.type>
                if operator == “>”
                        gt<arithmetic.type>
                if operator == “<”
                        lt<arithmetic.type>
                if operator == “==”
                        eq<arithmetic.type>
                if operator == “!=”
                        ne<arithmetic.type>

value [[arithmetic:expression → left:expression operator:string right:expression]] =
                value[[left]]
                value[[right]]
                if operator == "+"
                        ADD<arithmetic.type>
                if operator == "-"
                        SUB<arithmetic.type>
                if operator == "*"
                        MUL<arithmetic.type>
                if operator == "/"
                        DIV<arithmetic.type>
                If operator == “%”
                        MOD<arithmetic.type>

value [[variable:expression → name:string]] =
                if(variable.varDefinition.scope == 1){
                        address[[variable]]
```

```
            } else {
                <push bp>
                address[[variable]]
                <addi>
            }
            <load>variable.type

value ⟦cast:expression → type expression⟧ =
            value[[expression]]
            if(type != expression.type) <arithmetic.type>b< type>

value ⟦functionCallExpression:expression → name:string expression*⟧ =
            expression*.forEach(expr → value[[expr]])
            <call> name

value ⟦Return:expression → exp:expression?⟧ =
            if(exp){
                value[[expr]]
            }
            <call> name

value ⟦FunctionCallStatement:expression → name:string expression*⟧ =
            expression*.forEach(expr → value[[expr]])
            <call> name
```

address⟦expression⟧

```
address ⟦intLiteral:expression → intValue:int⟧ =
            error
address ⟦floatLiteral:expression → floatValue:float⟧ =
            error
address ⟦charLiteral:expression → name:string⟧ =
            error
address ⟦arrayAccess:expression → expr1:expression expr2:expression⟧=
            address[[expr1]]
            value[[expr2]]

            <pushi>arrayAccess.type.size
            <muli>
            <addi>

address ⟦fieldAccess:expression → expr:expression  string⟧ =
            address[[expr]]
            <pushi>fieldAccess.attrDefinition.address
            <addi>

address ⟦not:expression → expression⟧ =
            error
address ⟦logic:expression → left:expression operator:string right:expression⟧ =
            error
address ⟦arithmetic:expression → left:expression operator:string right:expression⟧ =
            error
address ⟦variable:expression → name:string⟧ =
            if(variable.varDefinition.scope = 1){
                <push> variable.varDefinition.address
            } else {
                <push bp>
                <push>variable.varDefinition.address
                <addi>
            }

address ⟦cast:expression → type expression⟧ =
            error
address ⟦functionCallExpression:expression → name:string expression*⟧ =
```