**UNIVERSIDAD DEL VALLE DE GUATEMALA**

**CAMPUS CENTRAL**

**FACULTAD DE INGENIERIA**

**CATEDRATICO: M.A. BIDKAR POJOY**

**CONSTRUCCION DE COMPILADORES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombres de los estudiantes** | | **Carné** | **Sección** |
| **Jonnathan Alejandro Juárez Velásquez** | | **15377** | **10** |
| **Nombre de la tarea** | **Documentación Proyecto #2** | | |

Diagrama de Flujo:

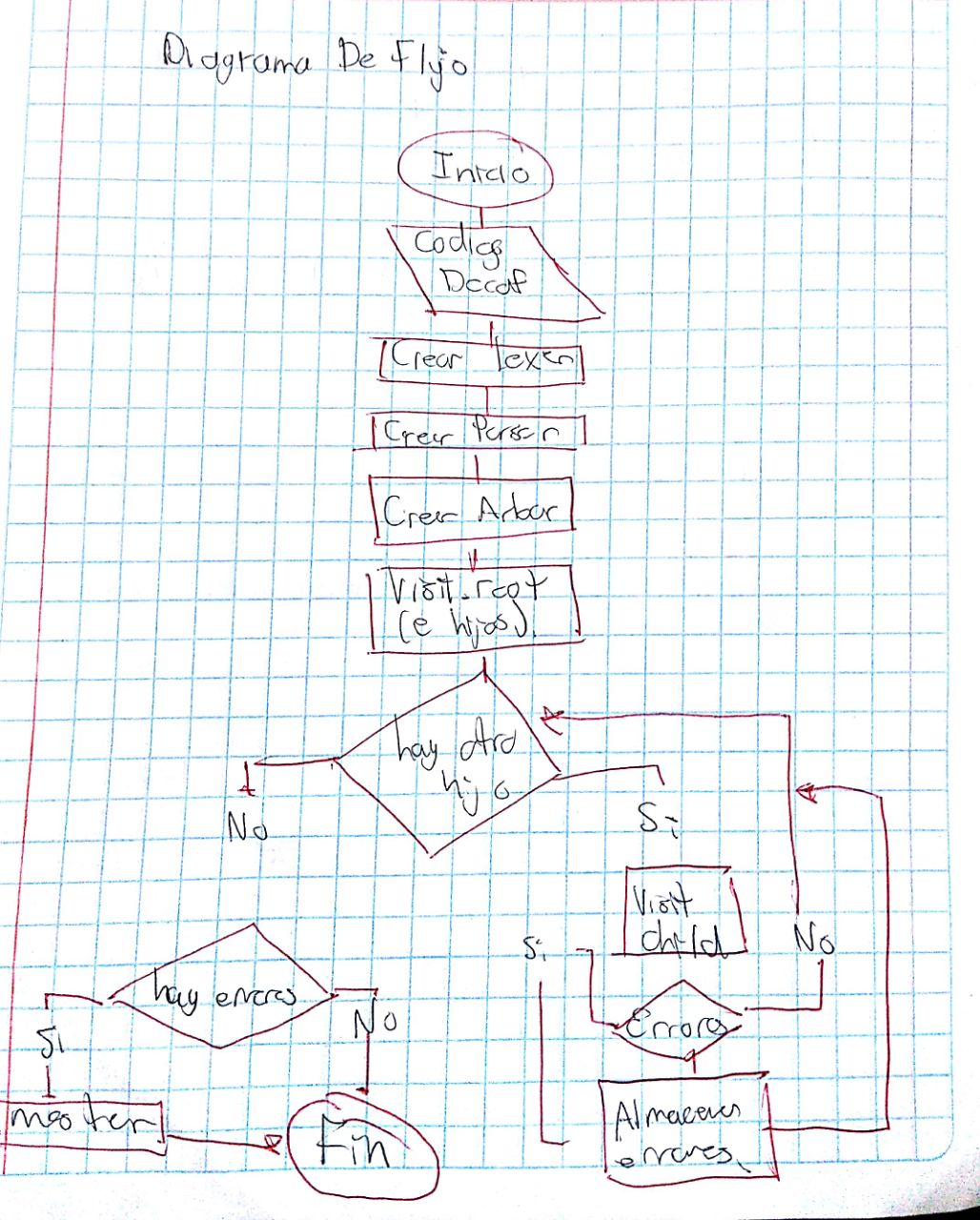
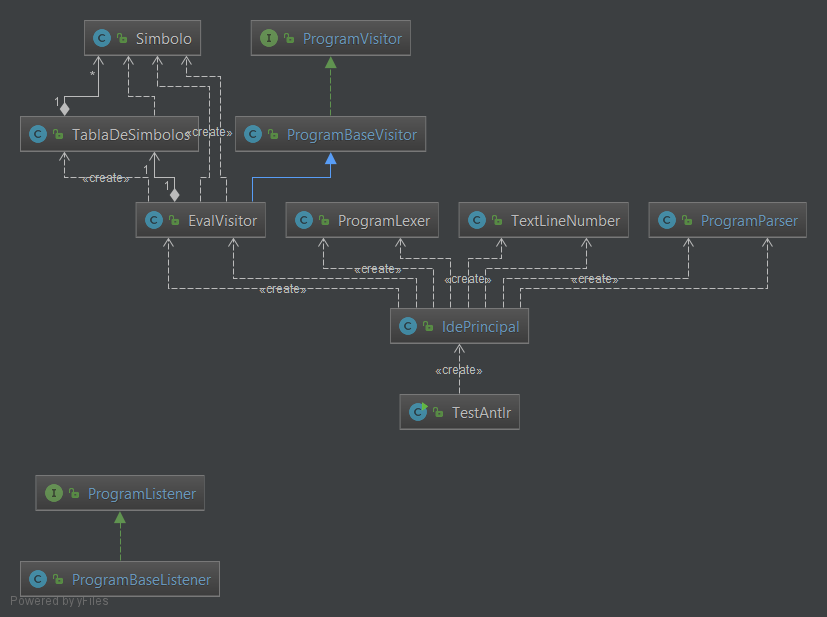


Diagrama de Clases simplificado



Esta versión únicamente incluye las relaciones entre las clases del programa, esto debido a que agregar parámetros y métodos aumenta de manera a considerable el tamaño y sería complicado visualizarlo.

Sistema de tipos:

proram

: 'class' ID '{' (declaration)\* '}' #progDeclarattion

;

declaration

: structDeclaration #aStructDec

| varDeclaration #aVarDec

| methodDeclaration #aMethodDec

;

varDeclaration

: varType ID ';' #simpleVarTyoe

| varType ID '[' NUM ']' ';' #arrayVarType

;

structDeclaration

: STRUCT ID '{' (varDeclaration)\* '}' #declarationStruct

;

varType

: INT #vartypeInt

| CHAR #vartypeChar

| BOOLEAN #vartypeBoolean

| STRUCT ID #vartypestrucID

| structDeclaration #vartypestruc

| VOID #vartypeVoid

;

methodDeclaration

: methodType ID '(' (parameter |( parameter (',' parameter)\*))? ')' block #methodDecl

;

methodType

: INT #methodTypeInt

| CHAR #methodTypeChar

| BOOLEAN #methodTypeBoolean

| VOID #methodTypeIntVoid

;

parameter

: parameterType ID #parameterID

| parameterType ID '[' NUM ']' #parameterArray

;

parameterType

: INT #parameterTypeInt

| CHAR #parameterTypeChar

| BOOLEAN #parameterTypeBoolean

;

block

: '{' (varDeclaration | statement)\* '}' #blockDeclaration

;

statement

: 'if' '(' expression ')' block (statementElse)? #statementIF

| WHILE '(' expression ')' block #statementWhile

| 'return' (expression | ) ';' #statementReturn

| methodCall ';' #statementMethodCall

| block #statementBlock

| location '=' expression ';' #statementLocation

| (expression)?';' #statementExpression

;

statementElse

:

ELSE block #statemElse

;

location

: ID | ID '[' expression ']'

| '.' location

;

expression

: andExpr

| expression cond\_op\_or andExpr

;

andExpr

: eqExpr #andExprEqExpr

| andExpr cond\_op\_and eqExpr #andExprCondOpAnd

;

eqExpr

: relationExpr #eqExprRelationExpr

| eqExpr eq\_op relationExpr #eqExprEqOp

;

relationExpr

: addExpr #relExprAddExpre

| relationExpr rel\_op addExpr #relExprRelOp

;

addExpr

: multExpr #addExprMultExpr

| addExpr minusplus\_op multExpr #addExprMinusPlusOp

;

multExpr

: unaryExpr #multExprUnary

| multExpr multdiv\_op unaryExpr #multExprMultDivOp

;

unaryExpr

: '('(INT|CHAR)')' value

| '-' value

| '!' value

| value

;

value

: location #valueLocation

| methodCall #valueMethodCall

| literal #valueLiteral

| '(' expression ')' #valueExprWithParent

;

methodCall

: ID '(' (parameter |( parameter (',' parameter)\*))? ')'#methodCalldecl

;

arg

: expression

;

minusplus\_op

: '+'

| '-'

;

multdiv\_op

: '\*'

| '/'

| '%'

;

rel\_op

: '<'

| '>'

| '<='

| '>='

;

eq\_op

: '=='

| '!='

;

cond\_op\_or : '||';

cond\_op\_and: '&&';

literal

: int\_literal

| char\_literal

| boolean\_literal

;

int\_literal

: NUM

;

char\_literal

: Char

;

boolean\_literal

: 'true'

| 'false'

;

Reglas Semánticas validadas

* Ningún identificador es declarado dos veces en el mismo ámbito
* Ningún identificador es utilizado antes de ser declarado.
* El programa contiene una definición de un método main sin parámetros, en donde se empezará la ejecución del programa.
* num en la declaración de un arreglo debe de ser mayor a 0.
* El número y tipos de argumentos en la llamada a un método deben de ser los mismos que los argumentos formales, es decir las firmas deben de ser idénticas.
* Si un método es utilizado en una expresión, este debe de devolver un resultado.
* La instrucción return no debe de tener ningún valor de retorno, si el método se declara del tipo void.
* El valor de retorno de un método debe de ser del mismo tipo con que fue declarado el método.
* Si se tiene la expresión id[<expr>], id debe de ser un arreglo y el tipo de <expr> debe de ser int.
* El tipo de <expr> en la estructura if y while, debe de ser del tipo boolean.
* Los tipos de operandos para los operadores <arith\_op> y <rel\_op> deben de ser int.
* Los tipos de operandos para los operadores <eq\_ops> deben de ser int, char o boolean, y además ambos operandos deben de ser del mismo tipo.
* Los tipos de operandos para los operadores <cond\_ops> y el operador ! deben de ser del tipo boolean
* El valor del lado derecho de una asignación, debe de ser del mismo tipo que la locación del lado izquierdo.

Oportunidades de mejora

* Validar estructuras mas complejas.
* En los tipos, validar el resto de estructuras de expresiones. En los la evaluaciond e retorno
* Comprobar tipos en los argumentos

Comentarios personales

Considero que el análisis semítico es una parte del compilador muy importante para prevenir que se ingreses acciones que causaran problemas en la generación de código intermedio. Adema vale la pena mencionar que es una parte bastante tediosa en la parte del diseño de un compilador.

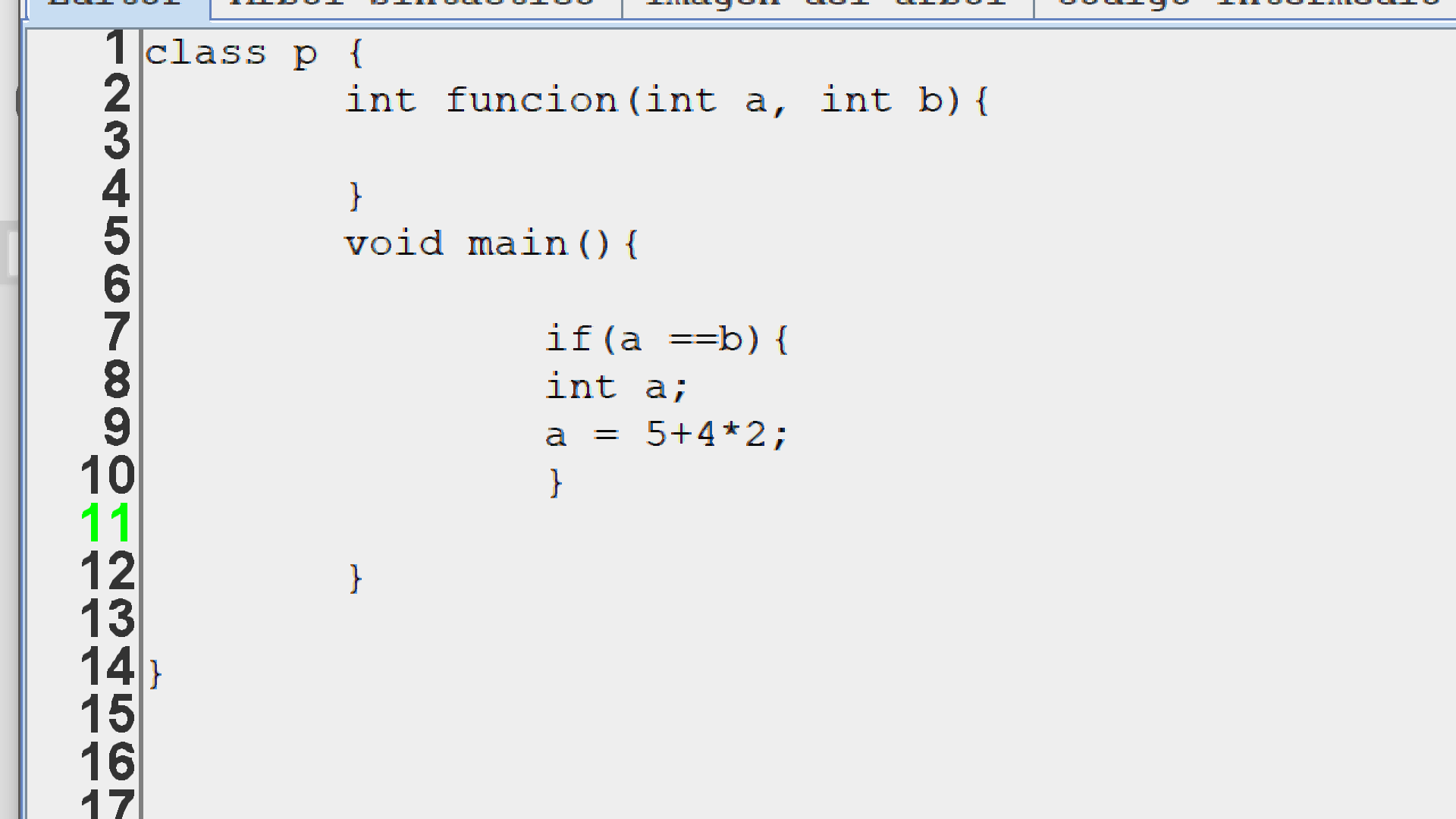
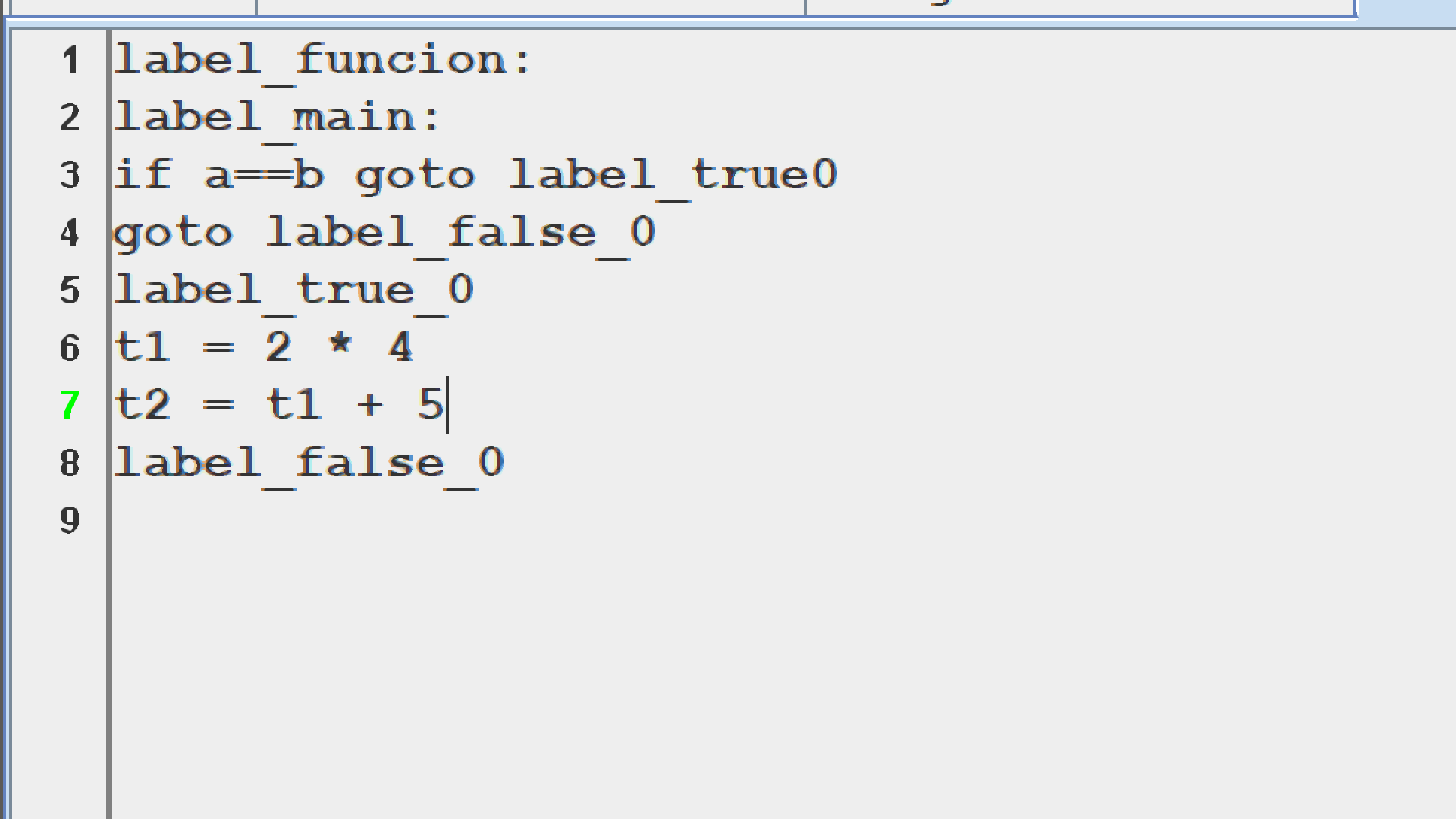
**Fase #2: código intermedio:**

Lenguaje utilizado: código de tres direcciones. Esto debido a su similitud con el lenguaje de Assembler en ARM. Este consiste en una dirección de destino, y 2 que son las temporales. En el caso de sentencias if, se genera etiquetas de salto condicional.

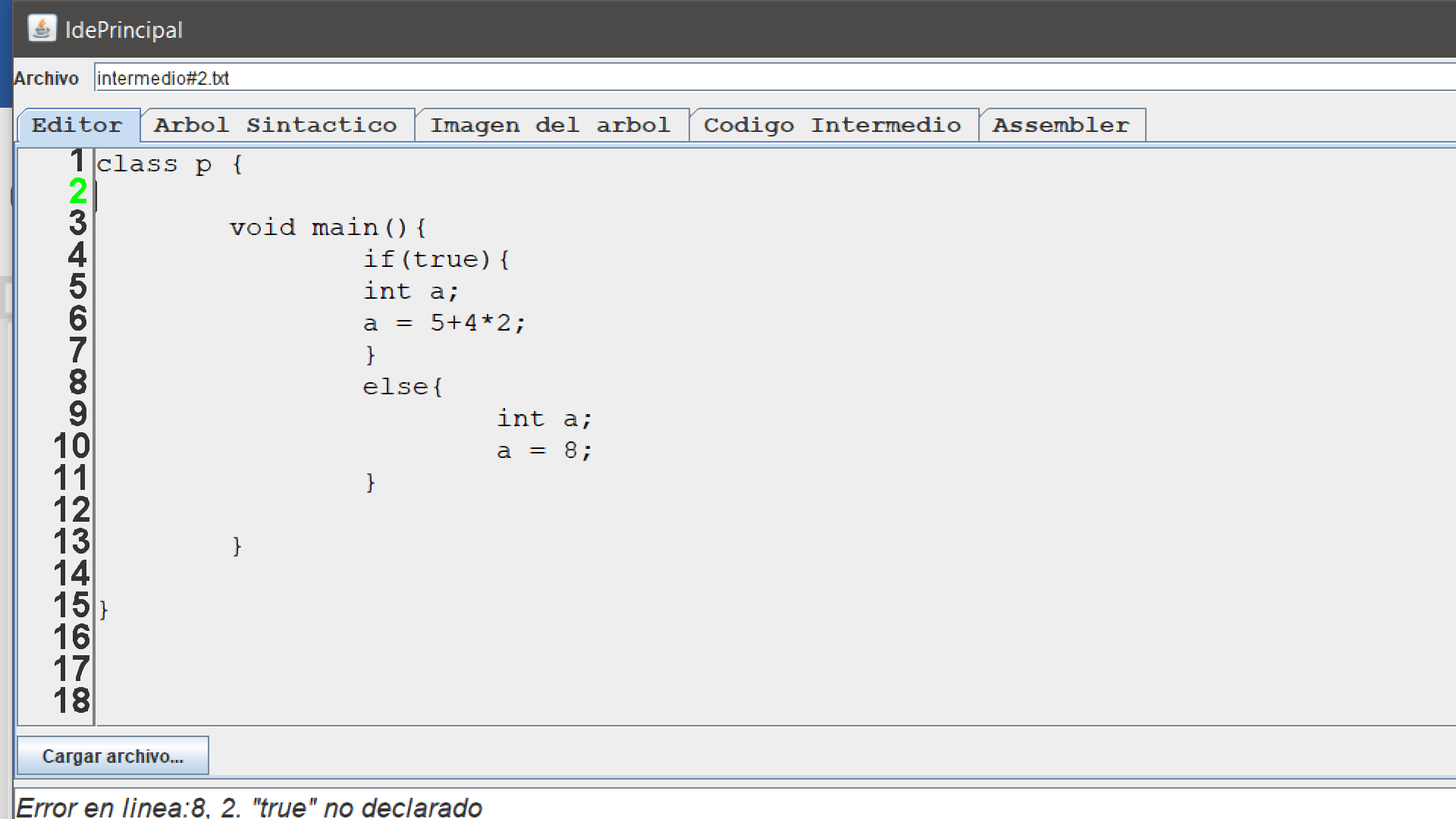
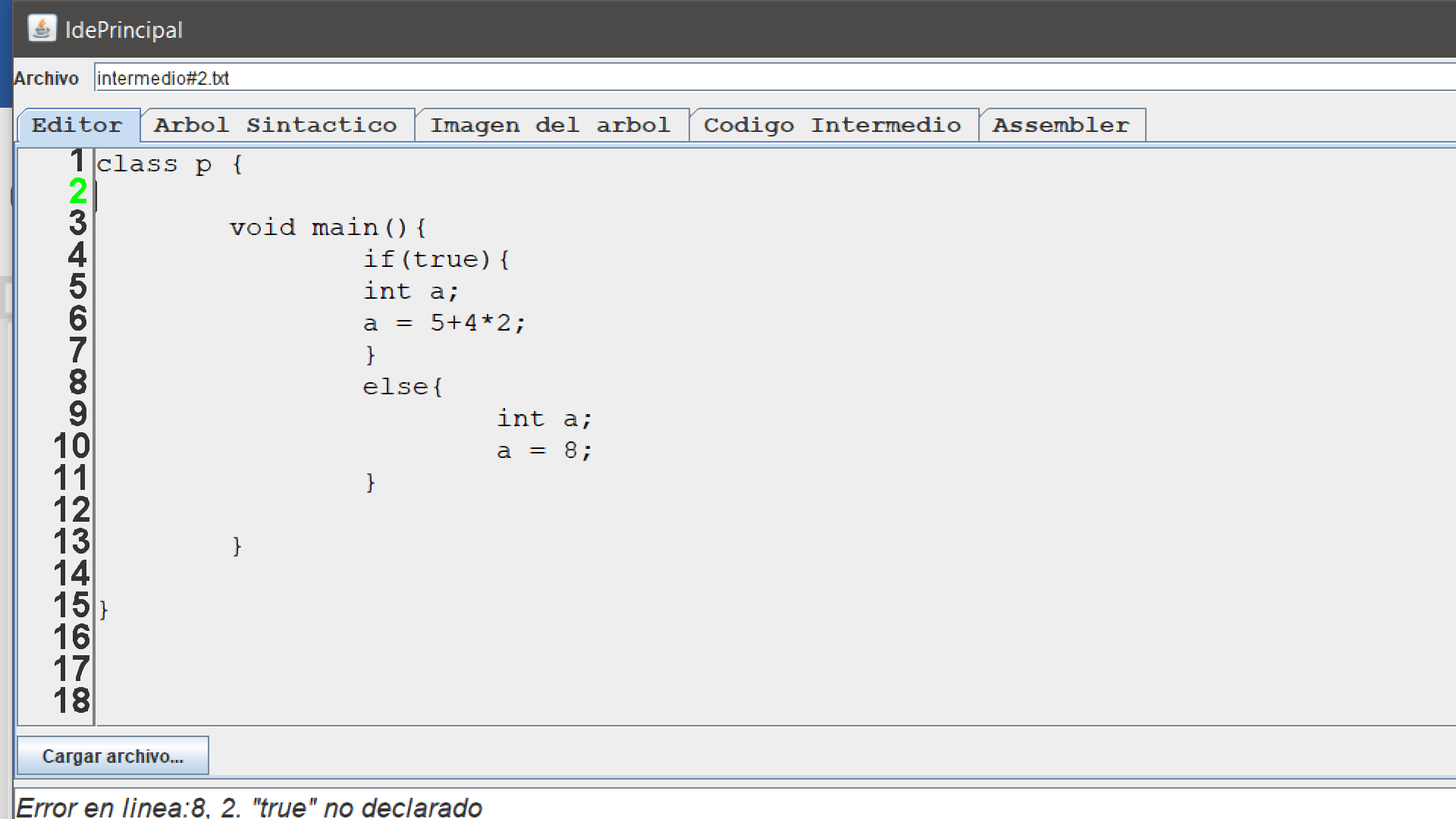
**Ejemplos de funcionamiento:**

**A continuación, se presenta como se generan algunas estructuras de código intermedio**

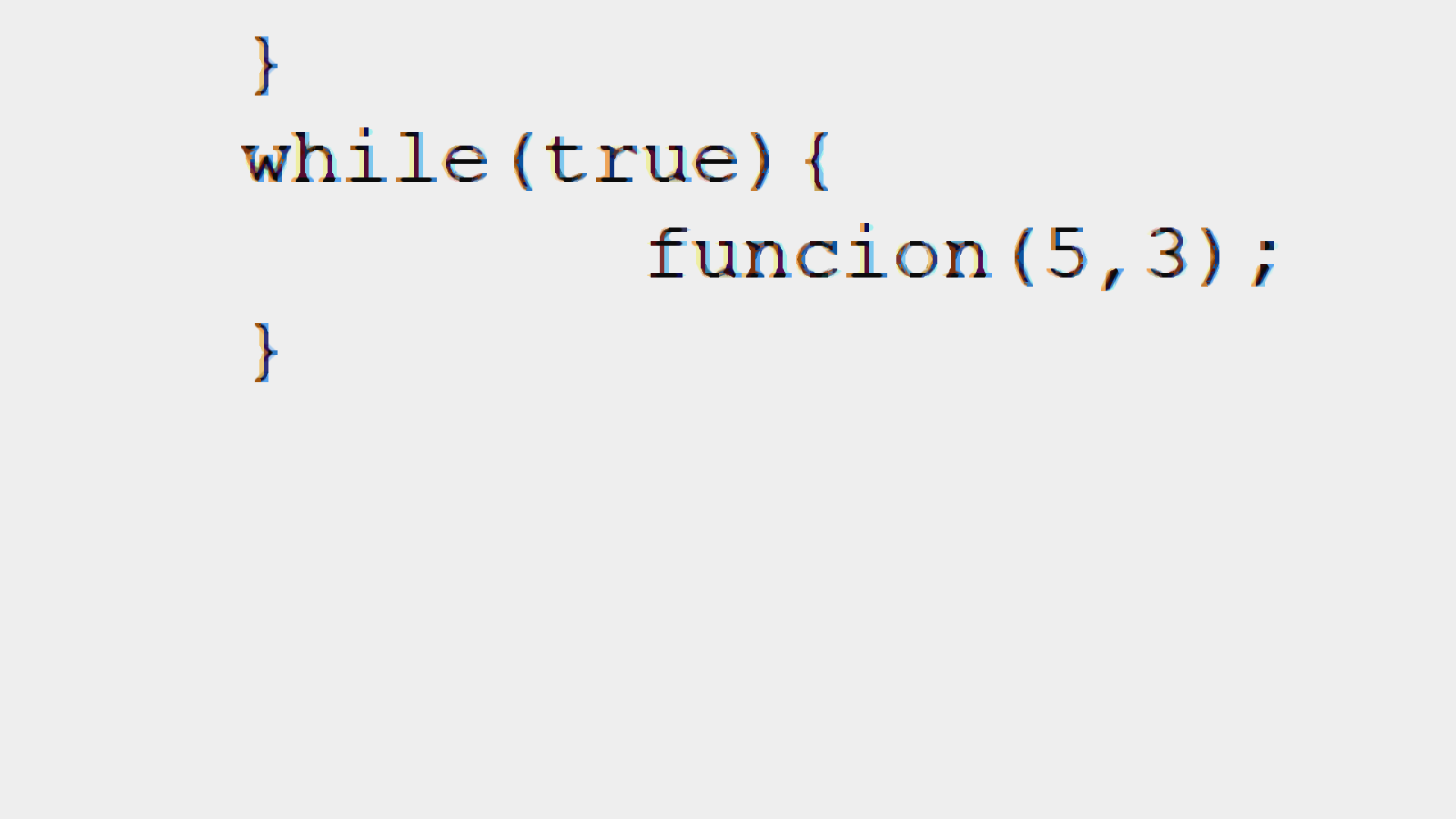
**Ejemplos de if:**

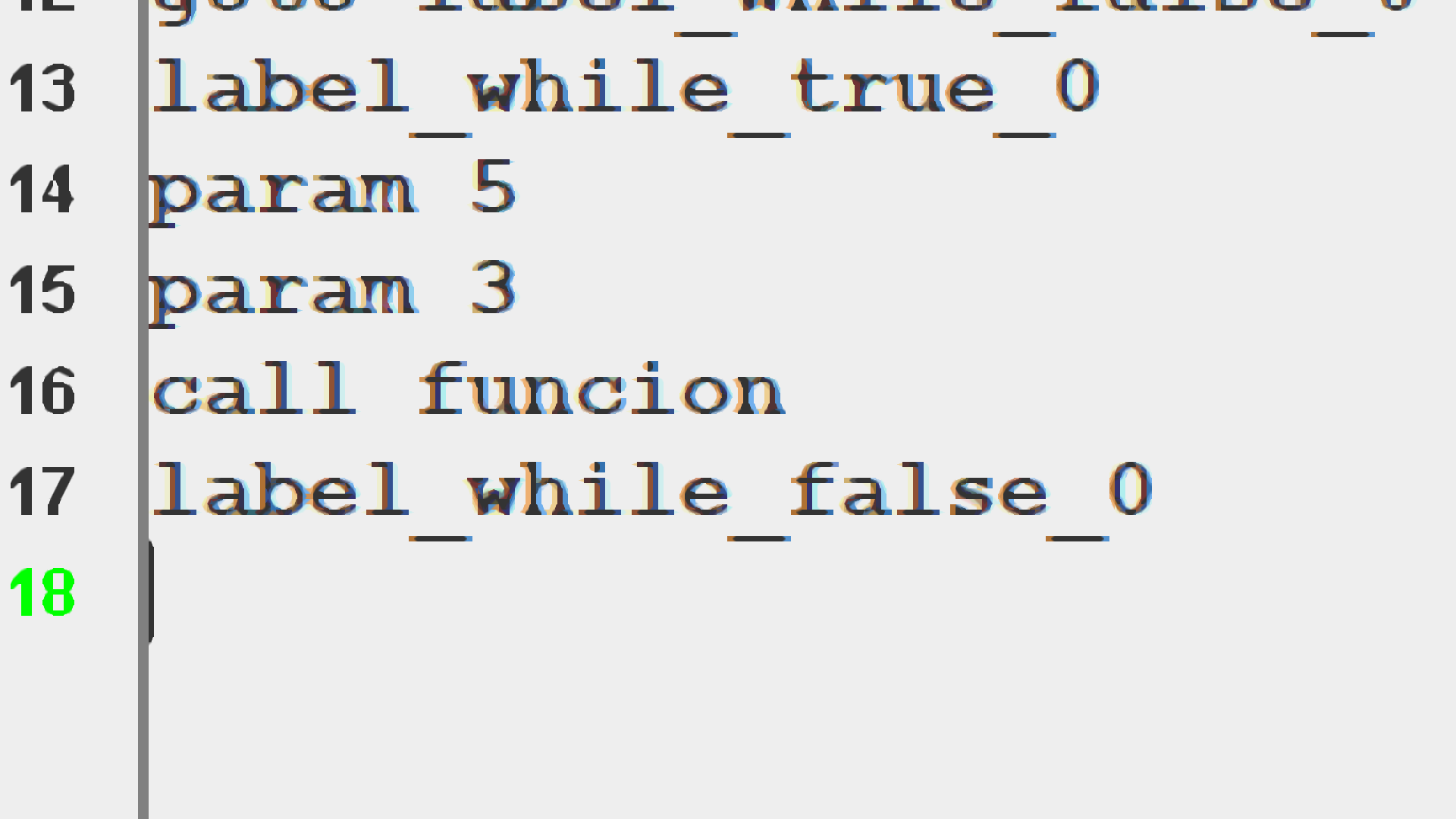
 

**Ejemplos de if y else:**

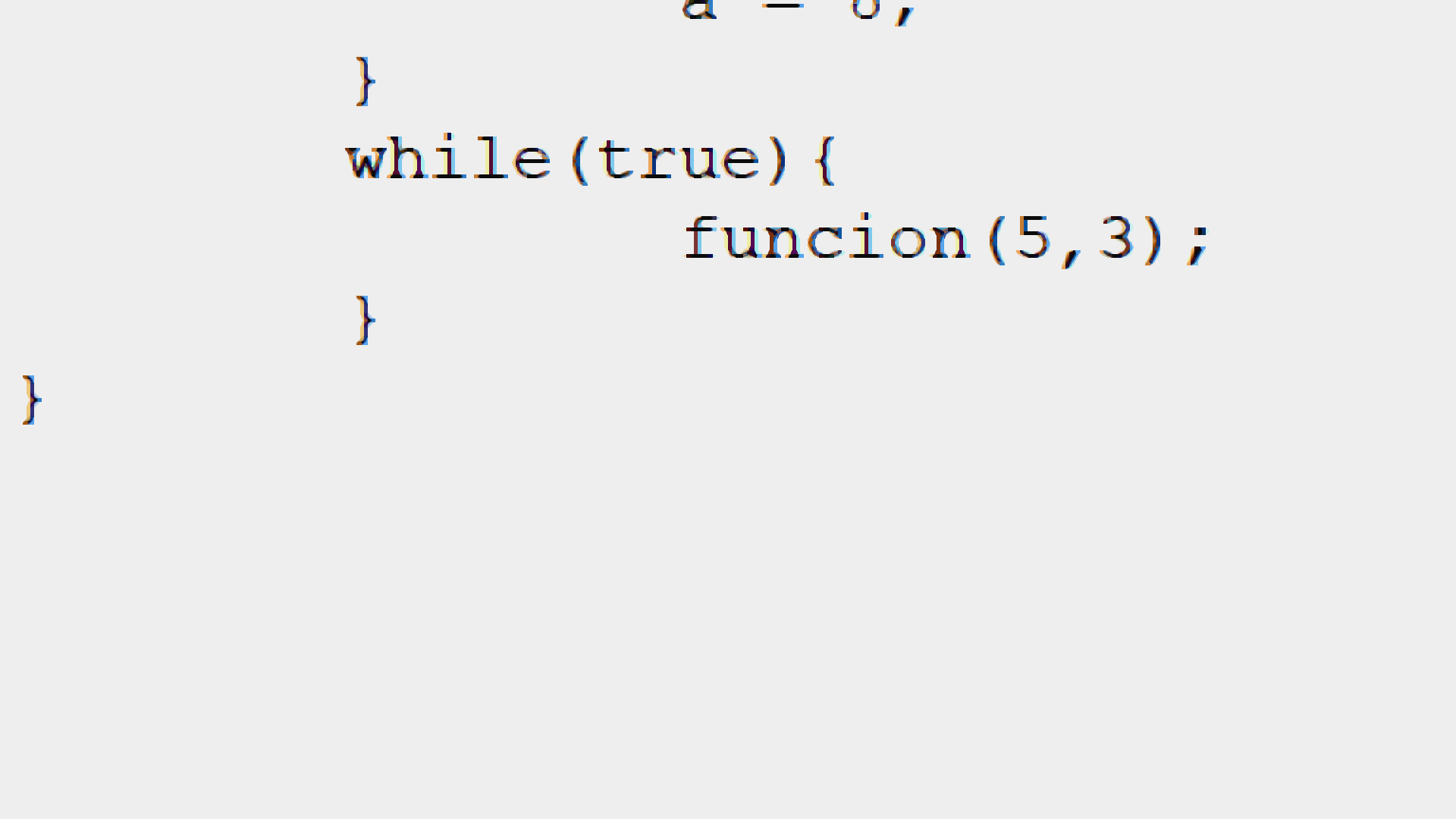
 

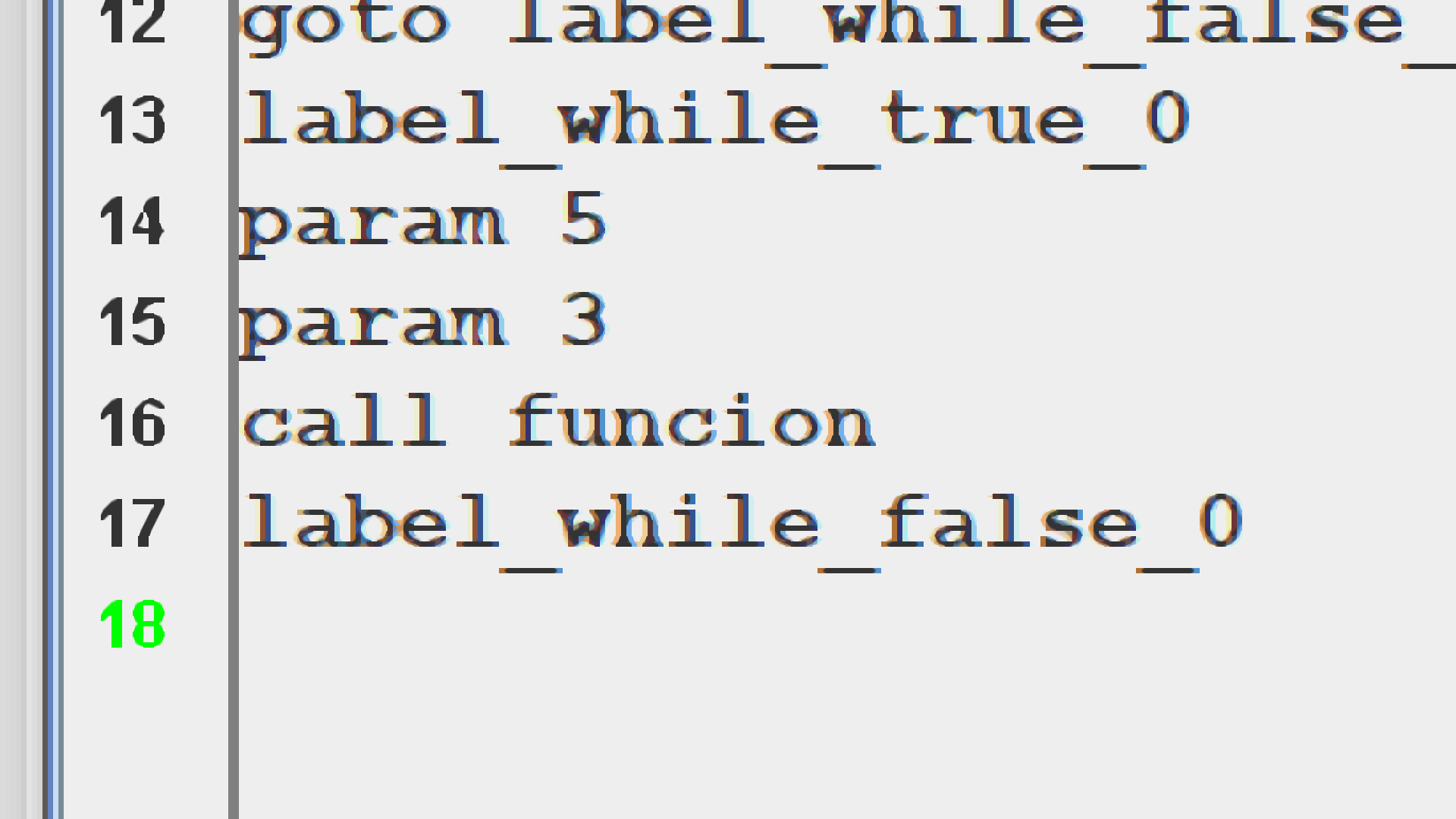
**Llamda a métodos**



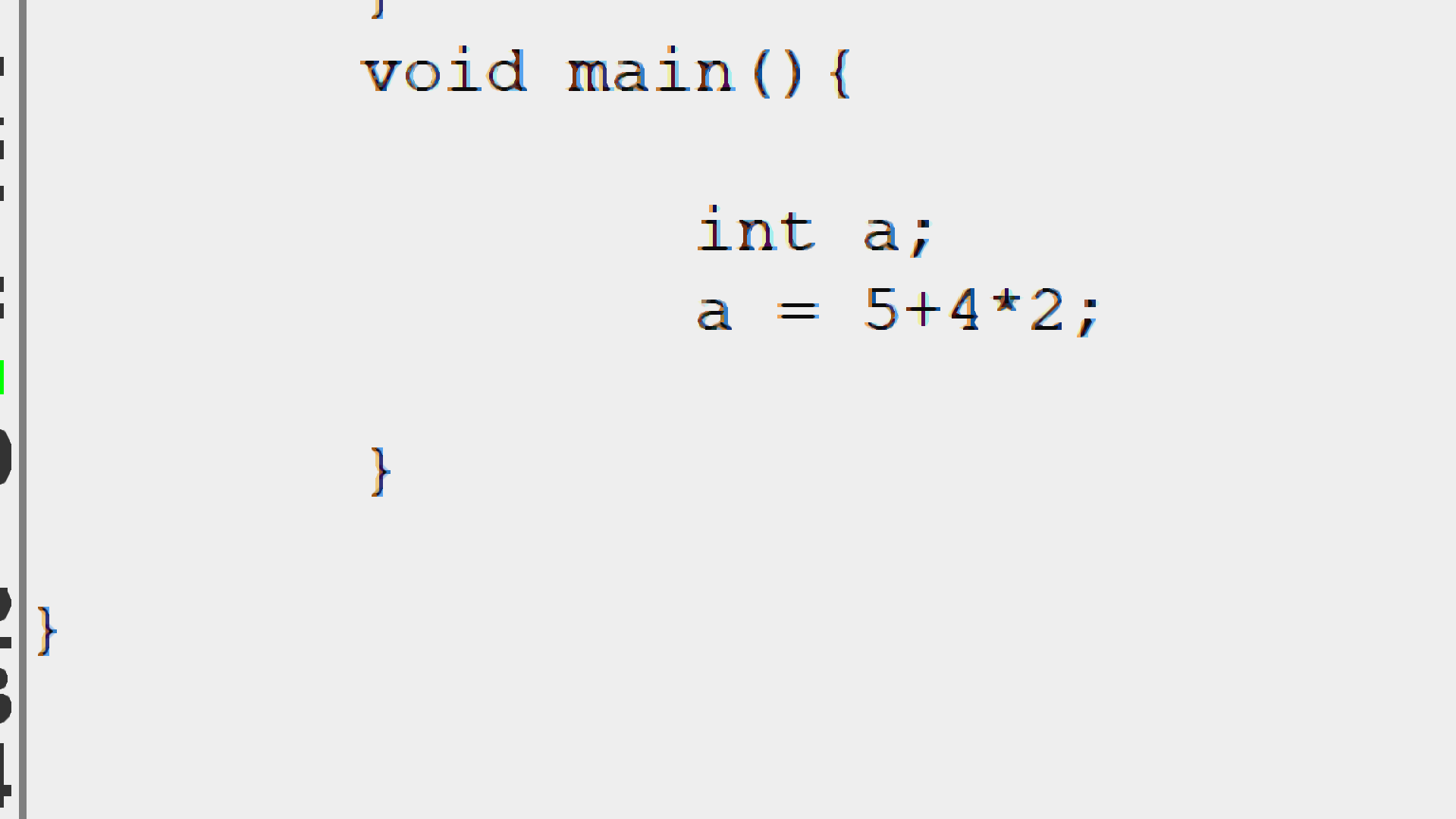


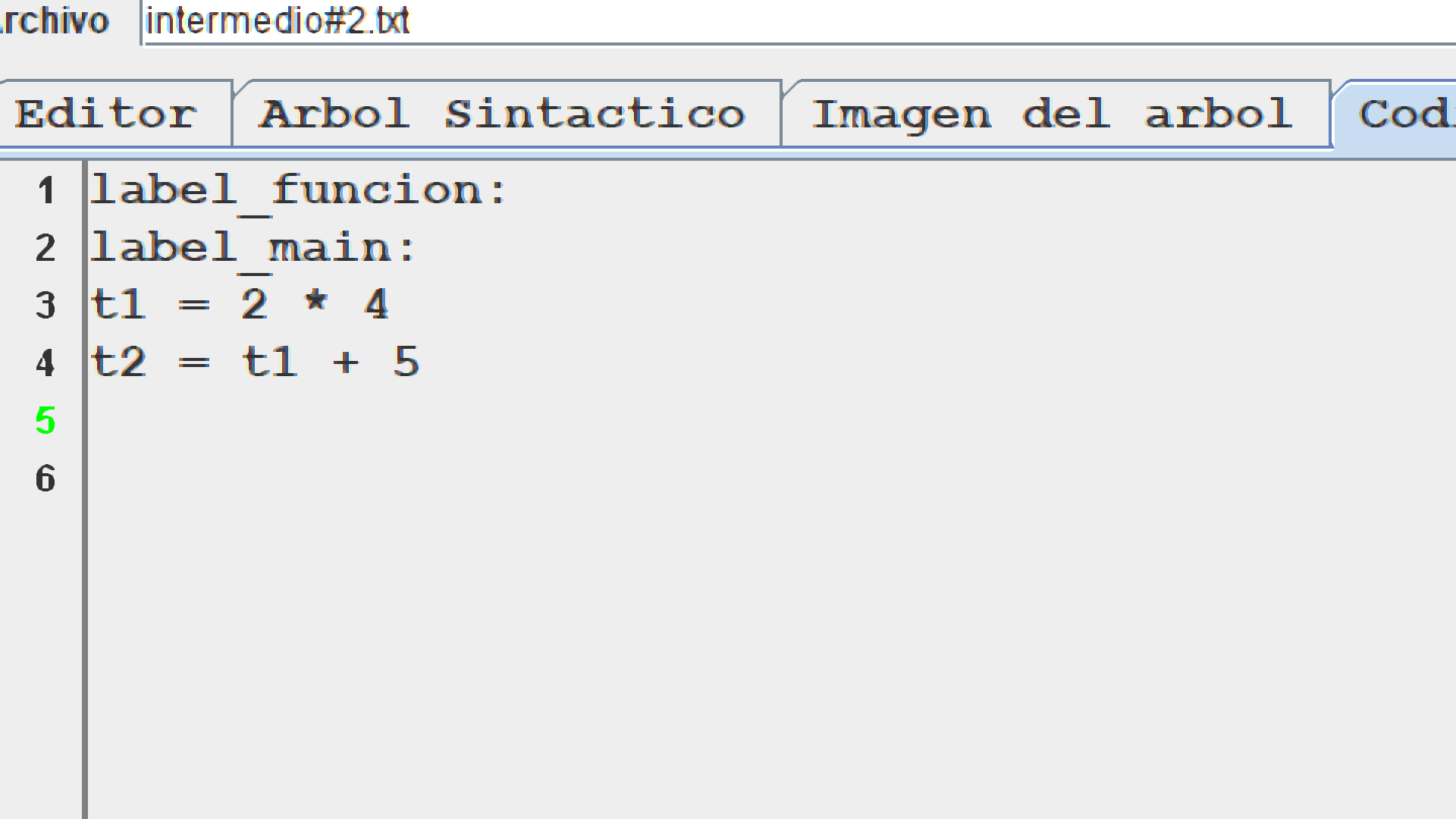
**While**



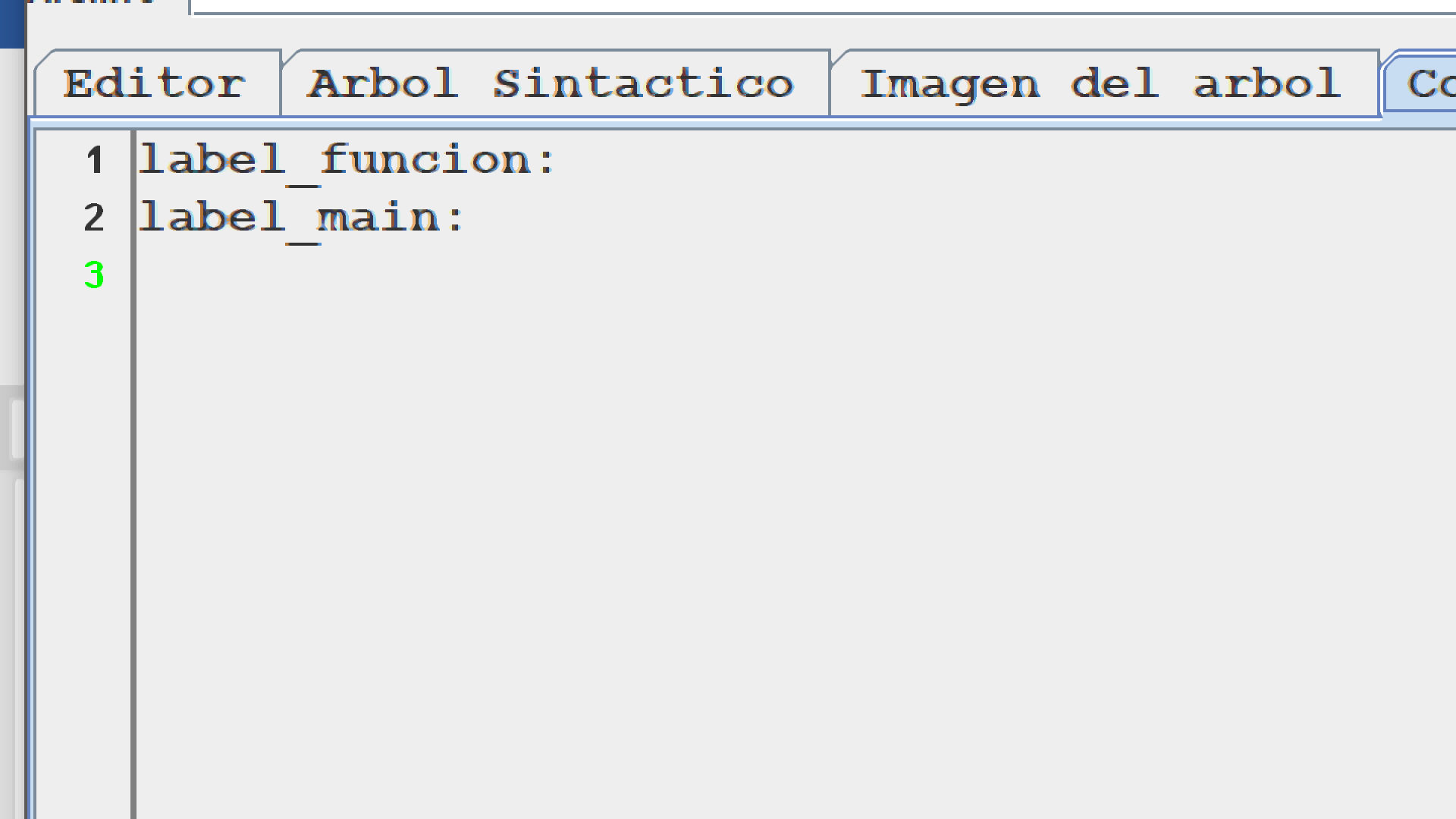
–

**Instrucciones aritméticas**





**Etiquetas de métodos**



**Generación de código objeto (Assembler ARM)**

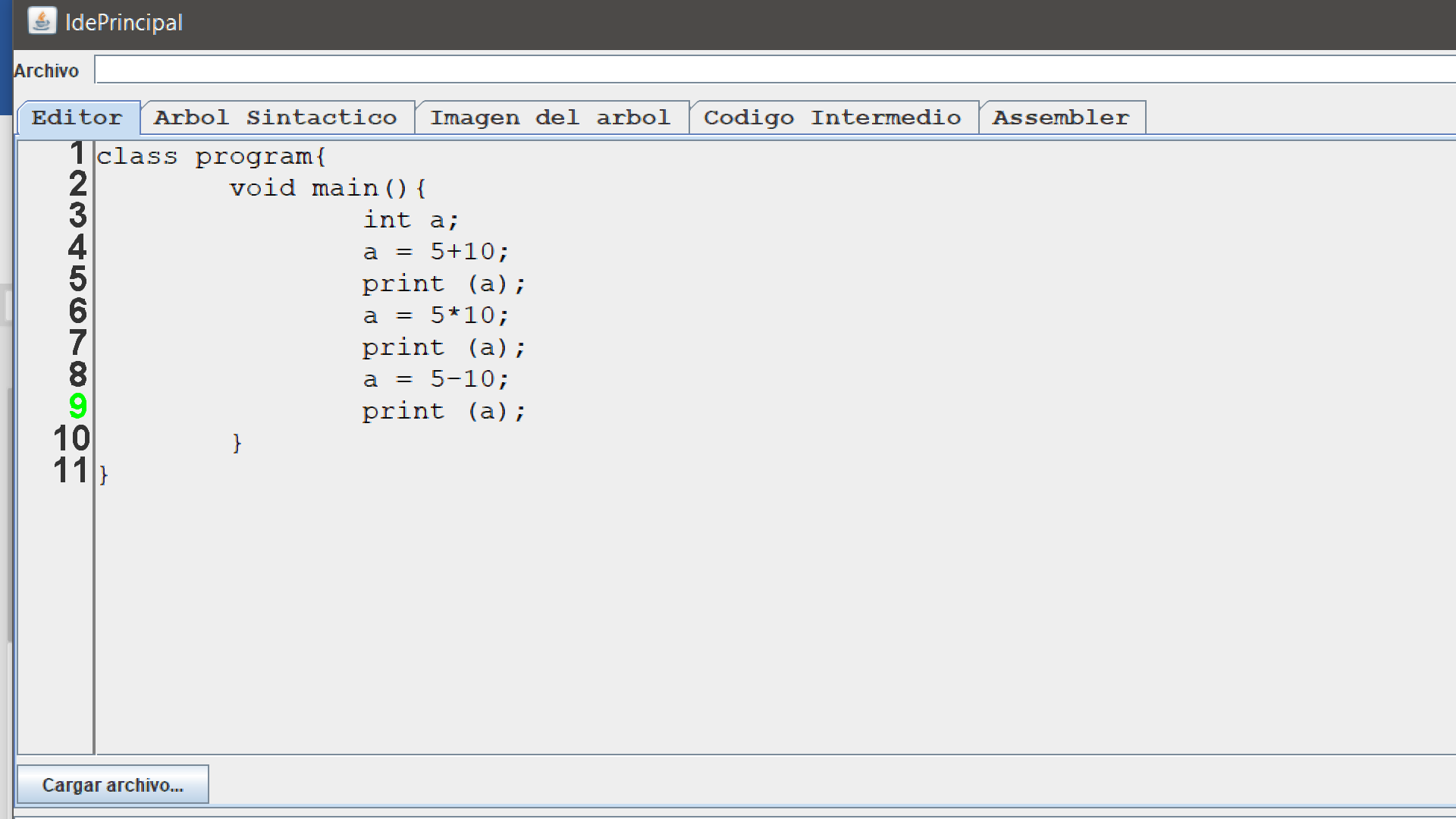
Es un lenguaje de bajo nivel desarrollado para procesadores móviles, cuenta con 15 registros de los cuales son utilizables 12 (R0-R12) con un stack pointer (R14), un link register (R13) y un program counter (R15). Cuenta con un espacio de memoria cuyo tamaño de localidad máxima es un word de 32bits.

**Complemento del diseño de la aplicación.**

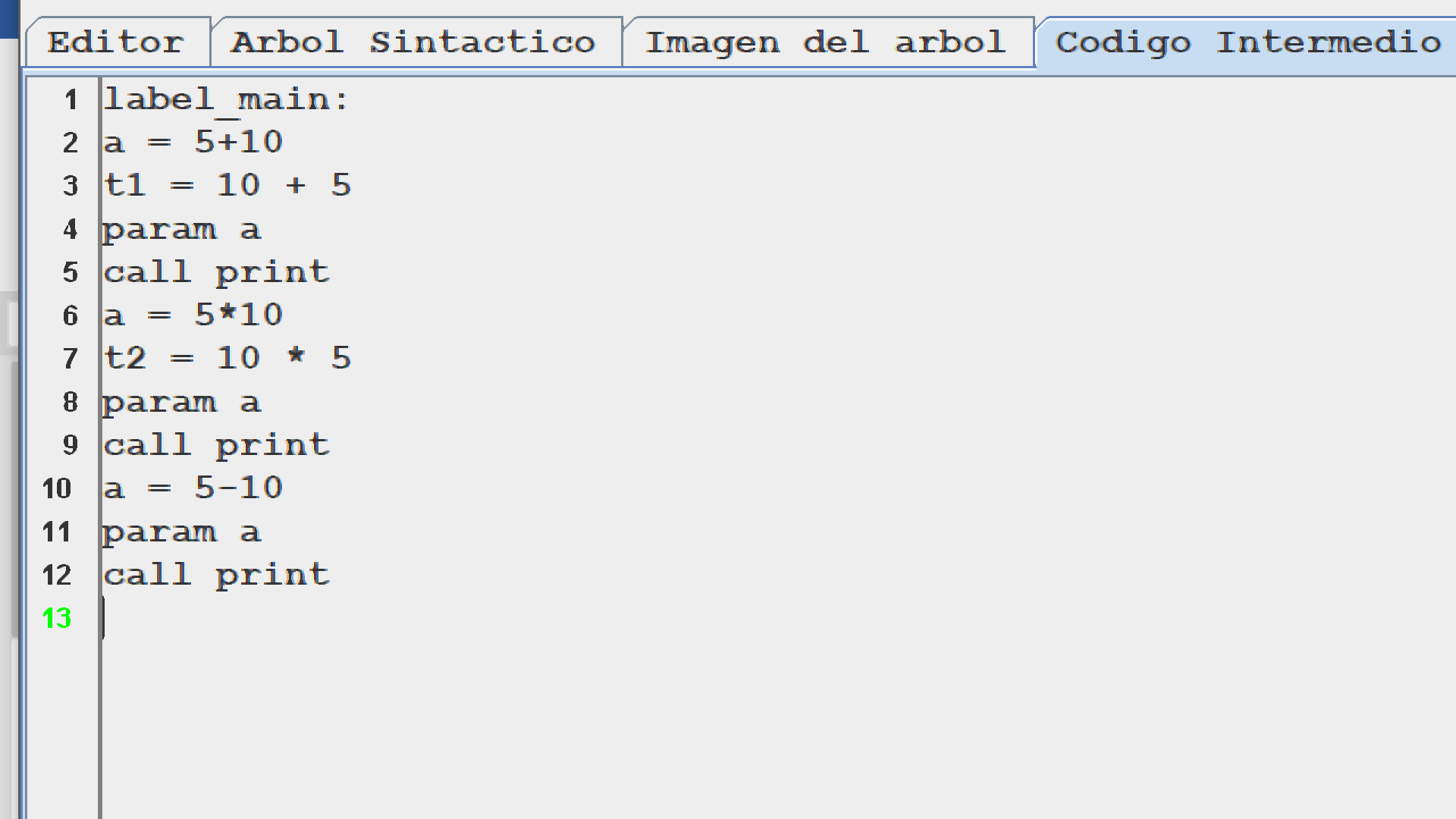
En esta última fase únicamente se habilito el panel de generación de código Assembler, en donde se muestra el resultado de la compilación.

**Pruebas de ejecución de código.**

**Código original**



**Codigo Intermedio**



**Código Assembler ARM**

