Código intermedio

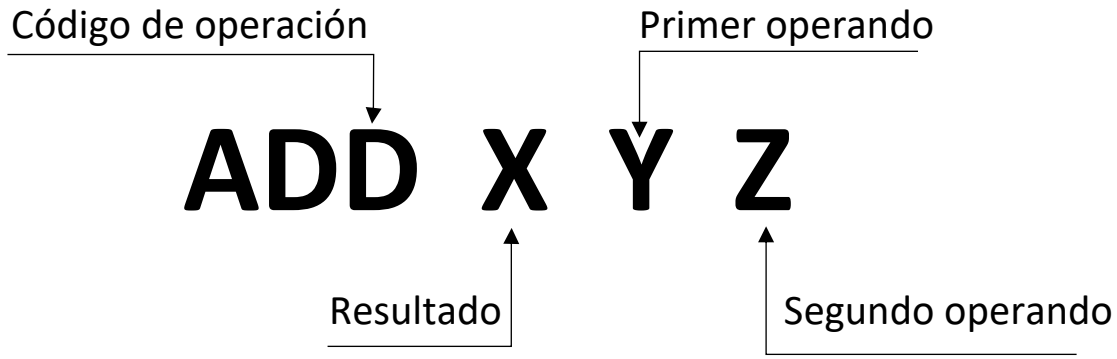
Fases de un compilador

- Etapa de Análisis
  - Análisis léxico
  - Análisis sintáctico
  - Análisis semántico
- Etapa de Síntesis
  - **Generación de código intermedio**
  - Generación de código objeto
  - Optimización del código objeto

El código intermedio son una secuencia de instrucciones cercanas o parecidas al código final o código destino, pero independiente de la arquitectura física o máquina.

El código intermedio, aunque forme parte de la etapa de síntesis se puede considerar una parte del **front-end**, mientras que el código máquina se considera parte del **back-end**.

En la práctica se va a usar un lenguaje de cuartetos o cuadruplas o quadruplas:



El resultado, primer operando y segundo operando son opcionales.

La secuencia de instrucciones que se van a ir generando debe de propagarse hasta el **AXIOMA**.

Es muy recomendable que el alumno lea detenidamente el punto **3.3 Código intermedio** de las directrices de implementación de la práctica, así como las transparencias del material de estudio de Javier Vélez Reyes y sus ejemplos:

- Tema 8 Generación de código intermedio. Sentencias y expresiones.
- Tema 9 Generación de código intermedio II Activación de subprogramas.

Diseño de un lenguaje de Código Intermedio de Javier Vélez Reyes para un lenguaje tipo Pascal o PascalUned. Lo usaremos para los ejemplos siguientes.

Cuarteto	Descripción	Cuarteto	Descripción
NOP	Nada	NOT x y	x := !y
ADD x y z	x := y + z	BR L	Salto a L
SUB x y z	x := y - z	BRT x L	Si x, salto a L
MUL x y z	x := y * z	BRF x L	Si !x, salto a L
DIV x y z	x := y / z	INL L	Insertar L:
MOD x y z	x := y % z	MV x y	x := y
INC x y	x := y + 1	MVP x y	x := *y
DEC x y	x := y - 1	MVA x y	x := &y
NEG x y	x := - y	STP x y	*x := y
GR x y z	x := (y > z) ? 1 : 0	STA x y	&x := y
EQ x y z	x := (y == z) ? 1 : 0	PUSH x	*SP := x ; SP++
LS x y z	x := (y < z) ? 1 : 0	POP x	x := *SP ; SP --
AND x y z	x := y && z	CALL f	Llamada a función f
OR x y z	x := y || z	RET x	Retorno de f con valor x
XOR x y z	x := y ^ z	HALT	Stop

Referencia: Tema 8 Generación de código intermedio. Sentencias y expresiones. Javier Vélez Reyes. Departamento de Lenguajes y Sistemas Informáticos

Añadir en las clases que contengan temporales

Dentro de las clases que hemos definido en el paquete "*compiler*"

C:\..\ArquitecturaPLII-cursoXX\src\compiler\syntax\nonTerminal vamos añadir los siguientes métodos de acceso para el código intermedio generado y la lista de temporales.

```
import es.uned.lsi.compiler.intermediate.TemporalIF;
import es.uned.lsi.compiler.intermediate.QuadrupleIF;
private TemporalIF temporal;
private List<QuadrupleIF> intermediateCode;

public List<QuadrupleIF> getIntermediateCode ()  {
      return intermediateCode;
}

public void setIntermediateCode (List<QuadrupleIF> intermediateCode) {
      this.intermediateCode = intermediateCode;
}

public TemporalIF getTemporal () {
         return temporal;
}

public void setTemporal (TemporalIF temporal) {
         this.temporal = temporal;
}
```

O bien podemos ampliar la clase *nonTerminal* donde ya nos viene dada una lista de cuádruplas con sus métodos de acceso, añadiendo un *TemporalIF* con sus métodos *get* y *set*.

```
package compiler.syntax.nonTerminal;

import es.uned.lsi.compiler.intermediate.TemporalIF;

public abstract class NonTerminal implements NonTerminalIF
{
   . . .
   . . .
   private TemporalIF temporal;

   public void setTemporal(TemporalIF temporal)
         {
               this.temporal = temporal;
         }

         public TemporalIF getTemporal() {
               return temporal;
         }
   . . .
   . . .
}
```

Ejemplo:

programa test:

  variables x : **entero**;

comienzo

  x = 5;

fin.

Después de las acciones semánticas:

- Creación de los ámbitos del programa principal “test”.
- Registrar la variable x de tipo entero en la tabla de símbolos del ámbito test.
- Comprobar la compatibilidad de tipos en la sentencia de asignación.

```
finalTest:
[java] [SYNTAX INFO] - Input file > C:\Users\josep\Documents\ArquitecturaPLIIGrupoA-2019-
2020_Tipos-v2\doc\test\test.muned
[java] [SYNTAX INFO] - Output file > C:\Users\josep\Documents\ArquitecturaPLIIGrupoA-2019-
2020_Tipos-v2\doc\test\test.ens
[java] [SYNTAX INFO] - Starting parsing...
[java] Nuevo ambito abierto de nombre: TEST de nivel: 0 y su id es: 0
[java] Variable X aun No declarada
[java] Variable Tipo  = Type - TypeSimple [scope = TEST, name = ENTERO] - {}
[java] sentAsign - IDENTIFICADOR X en linea 4 esta declarado
[java] sentAsign coinciden los tipo
[java] cierre del ambito: TEST de nivel: 0 y su id es: 0
[java] [SYNTAX INFO] - Parsing process ended.
[java] [SEMANTIC DEBUG] - ** TYPE TABLES **
[java] [SEMANTIC DEBUG] - Type - TypeSimple [scope = TEST, name = ENTERO] - {}
[java] [SEMANTIC DEBUG] - Type - TypeSimple [scope = TEST, name = LOGICO] - {}
[java] [SEMANTIC DEBUG] - ** SYMBOL TABLES **
[java] [SEMANTIC DEBUG] - SYMBOL TABLE [TEST]
[java] [SEMANTIC DEBUG] - Symbol - SymbolVariable [scope = TEST, name = X, type = ENTERO] - {}
BUILD SUCCESSFUL
Total time: 4 seconds
```

Procedemos a la creación del código intermedio.

Para ello vamos a utilizar los siguientes cuartetos del lenguaje diseñado por Javier Vélez Reyes e iremos creando temporales para ir almacenado los valores y las direcciones de las variables.

MV x y	->	x := y	Quadruple - [MV T_0, 5, null]
MVA x y	->	x := &y	Quadruple - [MVA T_1, x, null]
STP x y	->	*x := y	Quadruple - [STP T_1, T_0, null]

El código intermedio generado en la sentencia de asignación **x := 5** sería:

[Quadruple - [MV T_0, 5, null], Quadruple - [MVA T_1, x, null], Quadruple - [STP T_1, T_0, null]]

Temporales, etiquetas y código intermedio

Temporal:

```
TemporalFactory tf = new TemporalFactory(scope);
TemporalIF temporal = tf.create();
```

Etiqueta:

```
LabelFactory lF = new LabelFactory();
LabelIF lb = lF.create();
```

Código intermedio:

```
IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
cb.addQuadruple("MV", temporal, Integer.parseInt(num.getLexema()));
```

C:\..\ArquitecturaPLII-cursoXX\src\compiler\intermediate\

- Label.java
- Procedure.java
- Temporal.java
- Value.java
- Variable.java

Expresiones

- Propagar el temporal generado en la expresión.
    - En las expresiones que son identificadores habrá que comprobar si el símbolo al que hace mención el identificador es un constante, una variable o un parámetro.
        - En el caso de ser una constante habrá que obtener su valor el cual se conoce en tiempo de compilación y no cambia durante la ejecución del programa (“C:\..\ArquitecturaPLII-cursoXX\src\compiler\intermediate\Value.java”).
        - En el caso de que se trate de una variable o parámetro de una función crearemos el objeto de la variable (“C:\..\ArquitecturaPLII-cursoXX \src\compiler\intermediate\Variable.java”).
- Propagar las cuádruplas de código intermedio generado en la expresión.

expresion ::= ID:id

```
{:      Expresion e = new Expresion();
        ScopeIF scope = scopeManager.getCurrentScope();
        //Comprobaciones semánticas
        //Código intermedio
        TemporalFactory tf = new TemporalFactory(scope);
        IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
        TemporalIF temp = tf.create();
        SymbolIF simbolo = scopeManager.searchSymbol(id.getLexema());
        if (simbolo instanceof SymbolConstant) {
                Value valor = new Value(Integer.parseInt(((SymbolConstant)simbolo).getValor()));
                cb.addQuadruple("MV", temp, valor);
        } else if (simbolo instanceof SymbolVariable) {
                Variable var = new Variable(id.getLexema(), simbolo.getScope());
                cb.addQuadruple("MVP", temp, var);
        } else {
        //enviar mensaje de error no se trata ni de una variable ni de una constante.
        }
        //añadir el temporal y el código intermedio generado a la expression
        e.setTemporal(temp);
        e.setIntermediateCode(cb.create());

        //printamos el código intermedio generado (opcional)
        System.out.println("Codigo intermedio: " + e.getIntermediateCode());
        Result = e;
:}
```

OP -> puede ser cualquier operación aritmética o lógica como ADD, SUB, MUL, DIV, GR, EQ, LS, etc.

expresion ::= expresion:e1 OP expresion:e2

```
{:
        Expresion e = new Expresion();
        ScopeIF scope = scopeManager.getCurrentScope();
        //Comprobaciones semánticas
        //Código intermedio
        TemporalFactory tf = new TemporalFactory(scope);
        IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
        TemporalIF temp1 = e1.getTemporal();
        TemporalIF temp2 = e2.getTemporal();
        TemporalIF temp = tf.create();
        cb.addQuadruples(e1.getIntermediateCode());
        cb.addQuadruples(e2.getIntermediateCode());
        cb.addQuadruple("OP", temp, temp1, temp2);
        e.setTemporal(temp);
        e.setIntermediateCode(cb.create());

        //printamos el código intermedio generado
        System.out.println("Codigo intermedio: " + e.getIntermediateCode());

        Result = e;
:}
```

Vectores

programa test:

tipos

tvector = **vector** [1..3] **de entero**;

variables

x, z: **entero**;

candidato: tvector;

v, w: **booleano**;

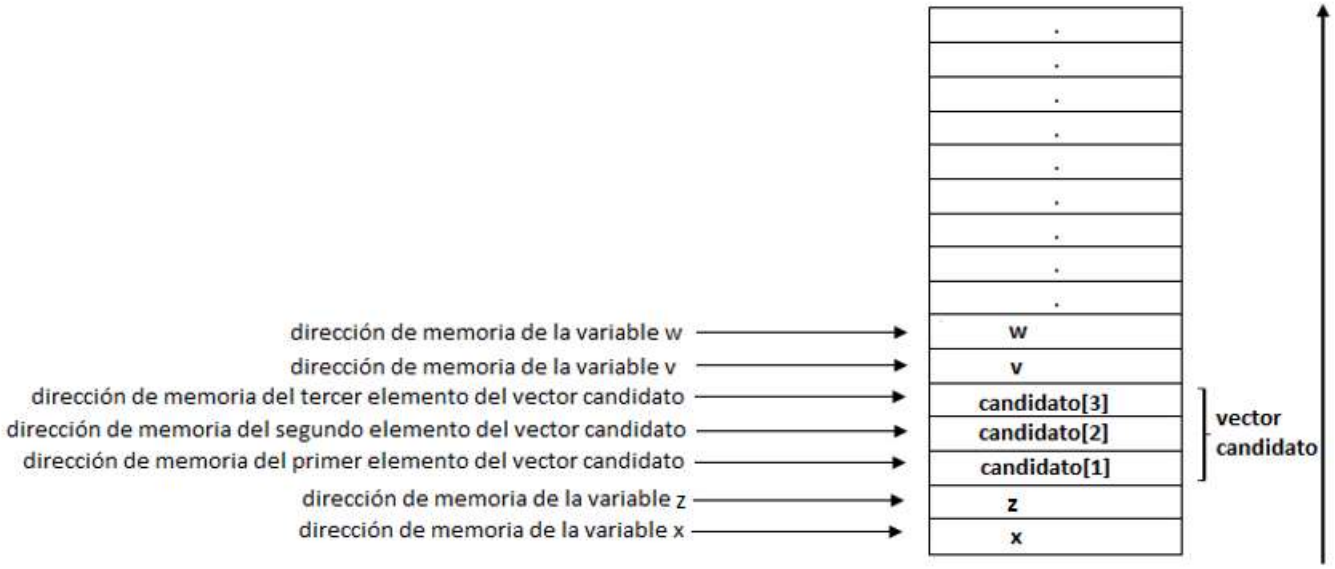
comienzo

candidato[1] = 1;

candidato[2] = 2;

candidato [3] = candidato [2] + candidato [1];

fin.



expresion ::= IDENTIFICADOR:id CORCHIZQ expresion:ex CORCHDER

```
{:      Expresion e = new Expresion();
        ScopeIF scope = scopeManager.getCurrentScope();
        //Comprobaciones semánticas
        ……
        //Expresión propaga la posición del elemento del vector
        //Código intermedio
        TemporalFactory tF = new TemporalFactory(scope);
        IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
        //añadir al codigo intermedio las cuadruplas de la expresiones
        cb.addQuadruples (ex.getIntermediateCode());
        TemporalIF temp = tF.create();
        TemporalIF temp1 = tF.create();
        TemporalIF temp2 = tF.create();
        //Temporal para el desplazamiento (índice vector), para candidato[1] su desplazamiento será 0
        TemporalIF tempPosicion = ex.getTemporal();
        SymbolIF sV = scopeManager.searchSymbol(id.getLexema());
        Variable var = new Variable(id.getLexema(), sV.getScope());
        cb.addQuadruple("MVA", temp1, var);                 //dirección base de la variable
        //Sumamos el desplazamiento si la memoria va de posiciones inferiores a superiores
        //si la memoria va de posiciones superiores a inferiores usamos  SUB en lugar de ADD
        cb.addQuadruple("ADD/SUB", temp2, temp1, tempPosicion);
        cb.addQuadruple("MVP", temp, temp2);
        e.setTemporal(temp);
        e.setIntermediateCode (cb.create());
        //printamos el código intermedio generado (opcional)
        System.out.println("Codigo intermedio: " + e.getIntermediateCode());
        RESULT e;
:}
```

Sentencias

- Las sentencias no propagan temporales.
- Las sentencias solo propagan el código intermedio generado.
- Deberemos añadir el código intermedio generado en una sentencia en la lista de cuádruplas del resto de sentencias respetando el orden en que se van generando.

sentencias ::= sentencia sentencias | ;

sentencia ::= sentenciaAsignacion

| sentenciaIf

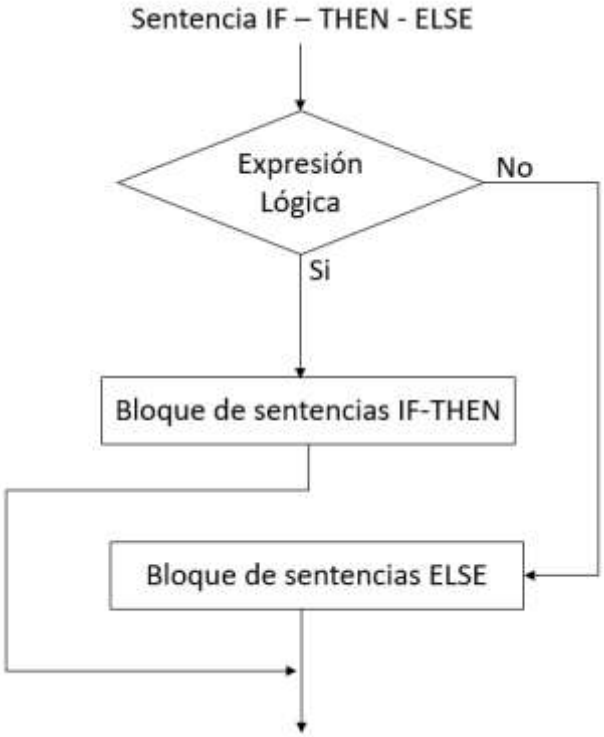
| sentenciaImprimeCadena

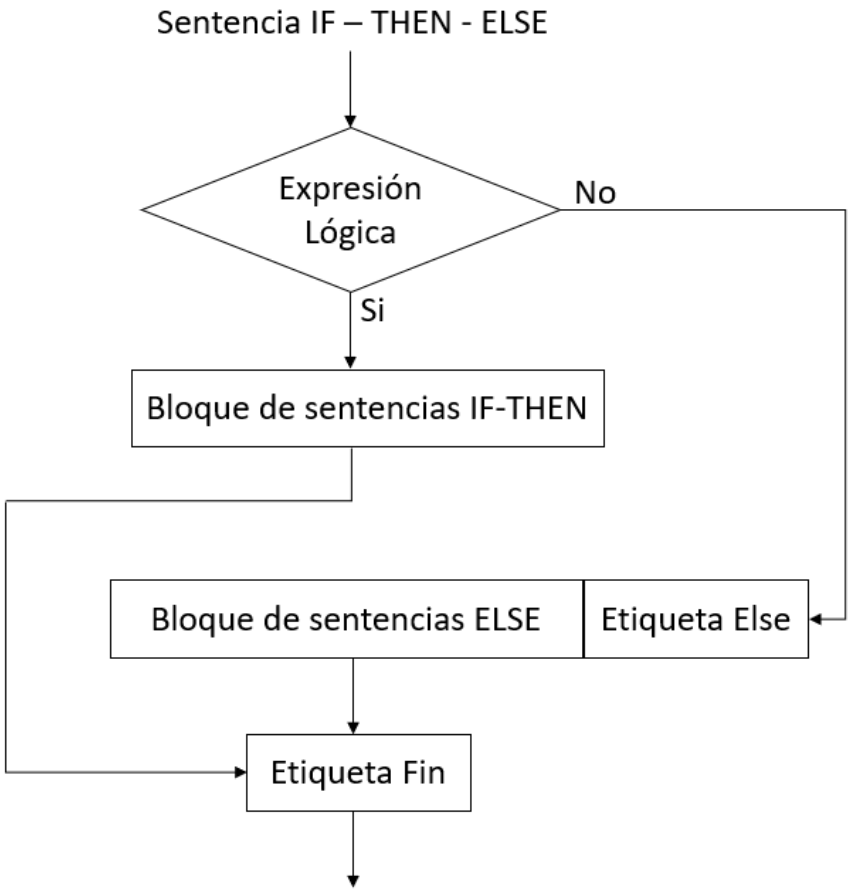
| . . . ;

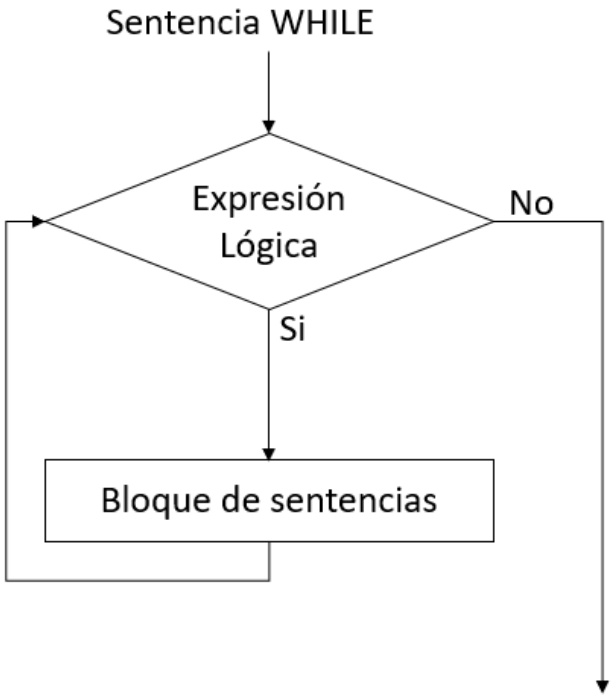
Sentencia asignación

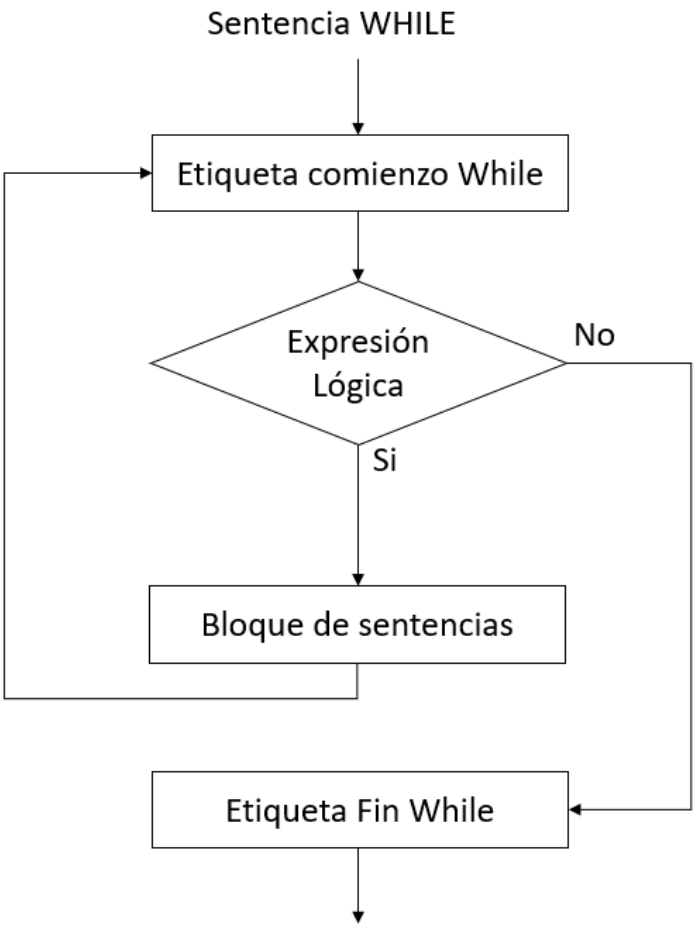
```
sentenciaAsignacion ::= ID:id IGUAL expresion:e
{:
        SentenciaAsignacion sA= new SentenciaAsignacion();
        ScopeIF scope = scopeManager.getCurrentScope();
        //Comprobaciones semanticas
        ……
        //Código intermedio
        TemporalFactoryIF tF = new TemporalFactory(scope);
        IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
        TemporalIF temp = tF.create();
        TemporalIF eTemp = e.getTemporal();
        cb.addQuadruples(e.getIntermediateCode());
        SymbolIF sV = scopeManager.searchSymbol(id.getLexema());
        Variable var = new Variable(id.getLexema(), sV.getScope());
        cb.addQuadruple("MVA", temp, var);
        cb.addQuadruple("STP", temp, eTemp);
        sA.setIntermediateCode(cb.create());
        //printamos el código intermedio generado
        System.out.println("Codigo intermedio: " + sA.getIntermediateCode());

        RESULT = sA;
:}
```

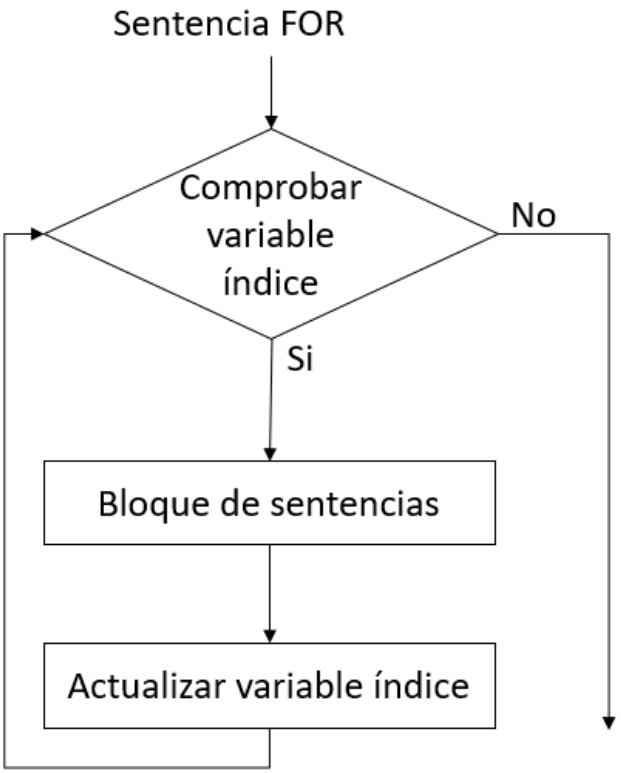
Sentencia condicional IF-ELSE

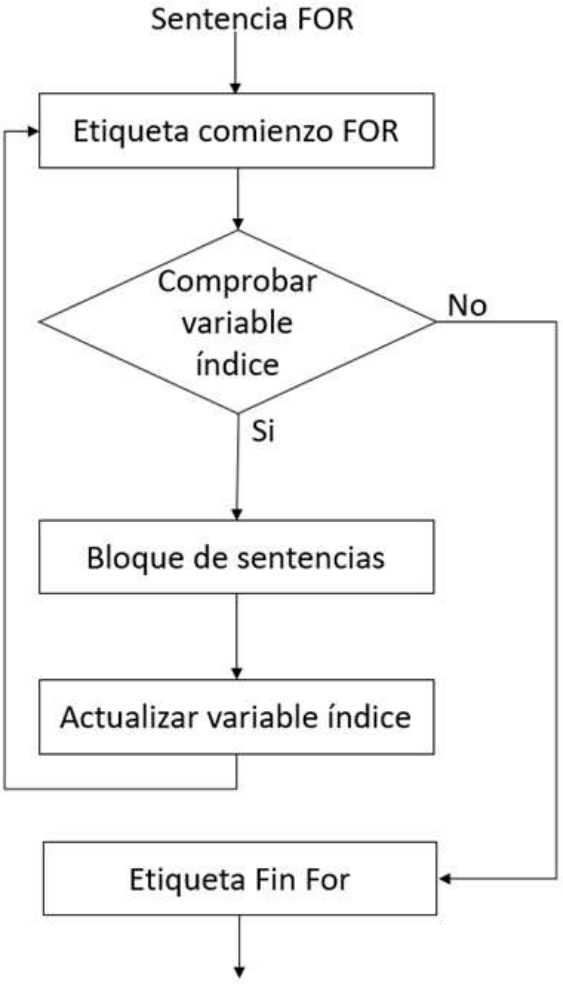





Sentencia WHILE
Expresión
Lógica
No
Si
Bloque de sentencias


Sentencia WHILE
Etiqueta comienzo While
Expresión
Lógica
No
Si
Bloque de sentencias
Etiqueta Fin While


Sentencia FOR
Comprobar variable índice
No
Si
Bloque de sentencias
Actualizar variable índice


Sentencia FOR
Etiqueta comienzo FOR
Comprobar variable índice
No
Si
Bloque de sentencias
Actualizar variable índice
Etiqueta Fin For

sentencia_IF ::= IF OBRACKET expresion:e CBRACKET THEN sentencias:s1 ELSE sentencias:s2

```
{:
        Sentencia_If senIf= new Sentencia_If();
        ScopeIF scope = scopeManager.getCurrentScope();
        //Comprobaciones semánticas
        //Código intermedio
        LabelFactory lF = new LabelFactory();
        LabelIF etiquetaFinIf = lF.create();                    //Etiqueta para fin de las sentenicas IF
        LabelIF etiquetaElse = lF.create();                     //Etiqueta de comienzo parte Else
        TemporalIF expTemp =  e.getTemporal();
        IntermediateCodeBuilder cb= new IntermediateCodeBuilder(scope);
        cb.addQuadruples(e.getIntermediateCode());          //añadir el código de la expresión
        cb.addQuadruple("BRF", expTemp, etiquetaElse);  //si no se cumple la condicion salto etiquetaElse
        cb.addQuadruples (s1.getIntermediateCode());        // añadir código sentencias parte IF
        cb.addQuadruple("BR", etiquetaFinIf);               //salto a la etiquetaFinIf
        cb.addQuadruple("INL", etiquetaElse);               // insertar etiquetaElse
        cb.addQuadruples(s2.getIntermediateCode());         //añadir código sentencias parte Else
        cb.addQuadruple("INL", etiquetaFinIf);              // insertar etiquetaFinIf
        senIf.setIntermediateCode(cb.create());
        //printamos el código intermedio generado
        System.out.println("Codigo intermedio: " + senIf.getIntermediateCode());

        RESULT=senIf;
:};
```

Ejemplo:

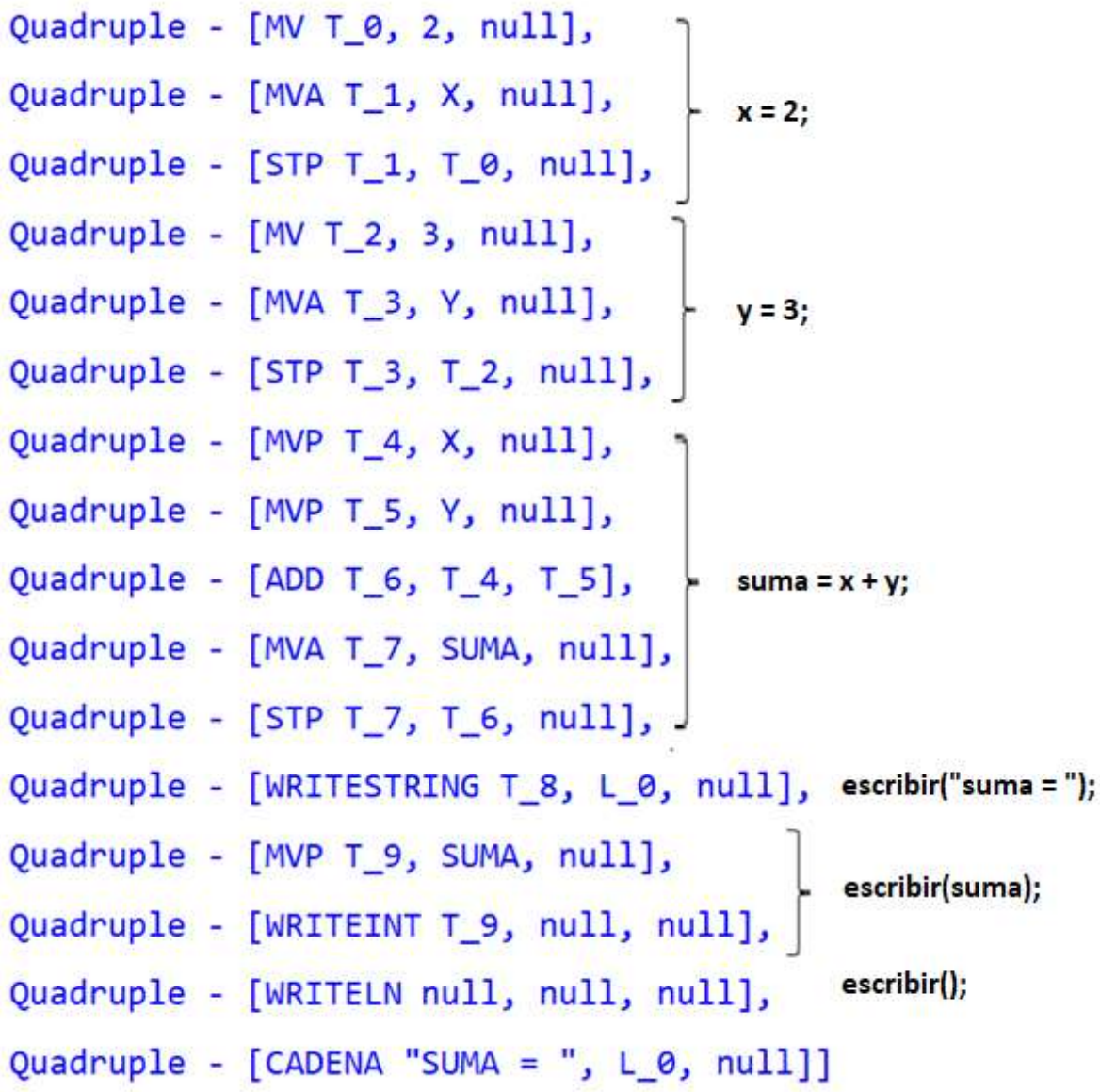
```
programa test:
  variables
        suma, x, y : entero;
comienzo
   x =2;
   y =3;
   suma  = x + y;
   escribir ("suma = ");
   escribir(suma);
   escribir();
fin.
```

[java] [SYNTAX INFO] - Starting parsing...
[java] Codigo intermedio: [Quadruple - [MV T_0, 2, null]]
[java] Codigo intermedio: [Quadruple - [MV T_0, 2, null], Quadruple - [MVA T_1, X, null], Quadruple - [STP T_1, T_0, null]]
[java] Codigo intermedio: [Quadruple - [MV T_2, 3, null]]
[java] Codigo intermedio: [Quadruple - [MV T_2, 3, null], Quadruple - [MVA T_3, Y, null], Quadruple - [STP T_3, T_2, null]]
[java] Codigo intermedio: [Quadruple - [MVP T_4, X, null]]
[java] Codigo intermedio: [Quadruple - [MVP T_5, Y, null]]
[java] Codigo intermedio: [Quadruple - [MVP T_4, X, null], Quadruple - [MVP T_5, Y, null], Quadruple - [ADD T_6, T_4, T_5], Quadruple - [MVA T_7, SUMA, null], Quadruple - [STP T_7, T_6, null]]
[java] Codigo intermedio: [Quadruple - [WRITESTRING T_8, L_0, null]]
[java] Codigo intermedio: [Quadruple - [MVP T_9, SUMA, null]]
[java] Codigo intermedio: [Quadruple - [MVP T_9, SUMA, null], Quadruple - [WRITEINT T_9, null, null]]
[java] Codigo intermedio: [Quadruple - [WRITELN null, null, null]]
[java] **Codigo intermedio Axioma: [Quadruple - [MV T_0, 2, null], Quadruple - [MVA T_1, X, null], Quadruple - [STP T_1, T_0, null], Quadruple - [MV T_2, 3, null], Quadruple - [MVA T_3, Y, null], Quadruple - [STP T_3, T_2, null], Quadruple - [MVP T_4, X, null], Quadruple - [MVP T_5, Y, null], Quadruple - [ADD T_6, T_4, T_5], Quadruple - [MVA T_7, SUMA, null], Quadruple - [STP T_7, T_6, null], Quadruple - [WRITESTRING T_8, L_0, null], Quadruple - [MVP T_9, SUMA, null], Quadruple - [WRITEINT T_9, null, null], Quadruple - [WRITELN null, null, null], Quadruple - [CADENA "SUMA = ", L_0, null]]**
[java] [SYNTAX INFO] - Parsing process ended.

[java] [SEMANTIC DEBUG] - ** TYPE TABLES **
[java] [SEMANTIC DEBUG] - Type - TypeSimple [scope = TEST, name = ENTERO] - {}
[java] [SEMANTIC DEBUG] - Type - TypeSimple [scope = TEST, name = LOGICO] - {}
[java] [SEMANTIC DEBUG] - ** SYMBOL TABLES **
[java] [SEMANTIC DEBUG] - SYMBOL TABLE [TEST]
[java] [SEMANTIC DEBUG] - Symbol - SymbolVariable [scope = TEST, name = SUMA, type = ENTERO] - {}
[java] [SEMANTIC DEBUG] - Symbol - SymbolVariable [scope = TEST, name = X, type = ENTERO] - {}
[java] [SEMANTIC DEBUG] - Symbol - SymbolVariable [scope = TEST, name = Y, type = ENTERO] - {}
BUILD SUCCESSFUL
Total time: 9 seconds

parser.cup

```
start with program;

program ::=
 {: syntaxErrorManager.syntaxInfo ("Starting parsing...");
 :}
 axiom:ax
 {:
                // No modificar esta estructura, aunque se pueden añadir más acciones semánticas
                //printamos el código intermedio generado
                System.out.println("Codigo intermedio Axioma: " + ax.getIntermediateCode());
                //List intermediateCode = ax.getIntermediateCode ();
                //finalCodeFactory.create (intermediateCode);
                // En caso de no comentarse las dos sentencias anteriores puede generar una excepcion
                // en las llamadas a cupTest si el compilador no está completo. Esto es debido a que
                // aún no se tendrá implementada la generación de código intermedio ni final.
                // Para la entrega final deberán descomentarse y usarse.
                syntaxErrorManager.syntaxInfo ("Parsing process ended.");
 :};
```



Para las cadenas de caracteres como en el ejemplo `"SUMA = "` podemos crear una lista para almacenar estas cadenas e introducirlas al final.

```
// Declaración del código de usuario
action code   {:
        SyntaxErrorManager   syntaxErrorManager   = CompilerContext.getSyntaxErrorManager();
        SemanticErrorManager semanticErrorManager = CompilerContext.getSemanticErrorManager ();
        ScopeManagerIF       scopeManager         = CompilerContext.getScopeManager ();
        FinalCodeFactoryIF   finalCodeFactory     = CompilerContext.getFinalCodeFactory ();

        //Lista para almacenar las cadenas de caracteres
        List<QuadrupleIF> listaCadenas = new ArrayList<QuadrupleIF>();
:}
```

```
sentenciaImprimeCadena  ::= ESCRIBIR PARENIZQ CADENA:cadena PARENDER
{:
SentenciaImprimeCadena sic = new SentenciaImprimeCadena ();
ScopeIF scope = scopeManager.getCurrentScope();

//Generacion de Codigo Intermedio
TemporalFactory tF = new TemporalFactory(scope);
IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
String texto = cadena.getLexema();
LabelFactory lF = new LabelFactory();
LabelIF lb = lF.create();
TemporalIF temp = tF.create();
cb.addQuadruple("WRITESTRING", temp, lb);
//Guardamos la cadena de caracteres en "listaCadena" y la recuperaremos al final del programa principal
listaCadenas.add(new Quadruple("CADENA", new Label(texto), lb));
sic.setIntermediateCode(cb.create());
//printamos el código intermedio generado
System.out.println("Codigo intermedio: " + sic.getIntermediateCode());
RESULT = sic;
:}



bloqueSentencias ::= BEGIN sentencias:ss END
{:
        . . .
        . . .
        cb.addQuadruples(ss.getIntermediateCode());
        //Comprobamos que estamos en el ámbito del programa principal ámbito de nivel 0  y
        //añadimos las listas de  cadenas al final del código intermedio generado
        if (scope.getLevel() == 0) {
                for (int i=0; i< listaCadenas.size(); i++) {
                        cb.addQuadruple(listaCadenas.get(i));
                }
        }
        . . .
        . . .
:}
```