



**Universidad
Europea de Madrid**

LAUREATE INTERNATIONAL UNIVERSITIES

GENERACIÓN DE CÓDIGO INTERMEDIO

EJEMPLOS PARA DISTINTAS ESTRUCTURAS DE DATOS

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

Índice

Presentación	4
Expresiones booleanas	5
DDS para expresiones booleanas	7
Proposiciones de control de flujo	10
DDS para if-then	12
DDS para if-then-else	13
DDS para while	14
Relleno de etiquetas por retroceso	15
Llamadas a funciones	17
Resumen	18

The logo for 'Cartagena99' features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue, abstract background that resembles a stylized 'C' or a wave. Below the text, there is a horizontal orange and yellow gradient bar.

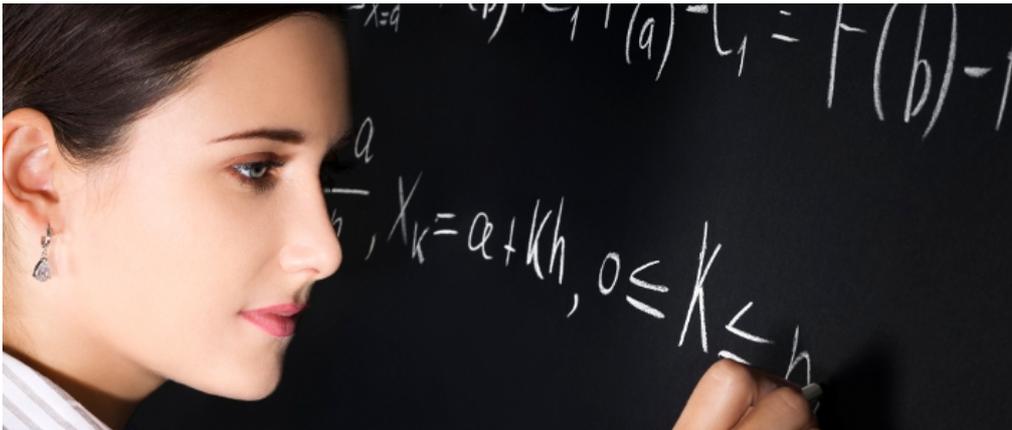
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

Presentación

El objetivo de este tema es comprender cómo aplicar la generación de código intermedio, mediante el código de tres direcciones a determinadas proposiciones. Para todo ello se alcanzarán los siguientes objetivos:

- Comprender cuáles son las funciones de las expresiones booleanas, así como sus formas de codificación.
- Obtener una *Definición Dirigida por la Sintaxis (DDS)* razonada para las expresiones booleanas.
- Conocer las proposiciones de control de flujo.
- Obtener una DDS para las proposiciones de control de flujo más importantes: **if-then**, **if-then-else** y **while**.
- Entender la problemática del relleno de los valores de las etiquetas y su solución.
- Comprender el funcionamiento de las **llamadas a funciones y procedimientos**.



Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Expresiones booleanas

Las funciones principales de las expresiones booleanas son:

- Cálculo de valores lógicos.
- Expresiones de condición: se utilizan en las proposiciones de control de flujo (if-then-else, while, for, etc.).

Los operadores que se aplican en este tipo de expresiones se denominan **operadores booleanos** y estos son: and, or y not, que en C y C# se escriben como `&`, `||` y `!`, respectivamente.

Estos operadores booleanos se aplican a variables booleanas.

 [Expresiones relacionales y aritméticas](#)
En detalle

Hay que tener en cuenta cómo vamos a representar los valores de una expresión booleana. Por parte de Aho *et al* (1986) se proponen dos métodos:

Codificación numérica de true y false	<p>Codificar numéricamente los valores true y false y evaluar una expresión booleana igual que una aritmética. Podemos asignar a <i>true</i> el valor 1 y a <i>false</i> el valor 2 y ver el siguiente ejemplo: if (x > y) then a = 3</p> <p>50: if x > y goto 53</p> <p>51: t:= 0</p> <p>52: goto 55</p> <p>53: t:=1</p> <p>54: a:= 3</p> <p>55:</p>
---------------------------------------	--



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

EJEMPLOS PARA DISTINTAS ESTRUCTURAS DE DATOS

Control de flujo	<p>Consiste en representar el valor de una expresión booleana mediante una posición alcanzada en un programa. Este método es muy útil para expresiones booleanas en proposiciones de control como <i>if-then-else</i> o <i>while</i>.</p> <p>Supongamos el caso del <i>if-then</i>, que se representaría así:</p>

En detalle

Expresiones relacionales y aritméticas

También se pueden construir expresiones relacionales (E_1 oprel E_2), cuyo resultado será un valor lógico (true o false), que en este contexto además serán los atributos que utilizaremos para evaluar estas expresiones. Estas expresiones relacionales usan los siguientes operadores relacionales: $>$, $>=$, $<$, $<=$, $==$ y $!=$. Estos a su vez se aplican sobre expresiones aritméticas (E_1 y E_2), que finalmente pueden ser identificadores o números (ej: $a > 3$).



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

DDS para expresiones booleanas

La gramática para las expresiones booleanas es la siguiente:

$E \rightarrow E \text{ and } E \mid E \text{ or } E \mid \text{not } E \mid E \text{ oprel } E \mid \text{true} \mid \text{false}$

Utilizamos los siguientes atributos sintetizados.

Producción		
$E_1 \rightarrow E_2 \text{ and } E_3$	Regla semántica	x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones será: $t_1 := x \text{ and } y$, donde t_1 será una variable temporal. Por tanto, lo primero es obtener esa variable temporal con $E_1.\text{nombre} = \text{tempnuevo}()$, a la que dependiendo de la evaluación de la expresión booleana (and, or o not) le asignaremos un true o un false a su valor, representado por E.lugar.
$E_1 \rightarrow E_2 \text{ or } E_3$	Regla semántica	
$E_1 \rightarrow \text{not } E_2$	Regla semántica	
$E_1 \rightarrow (E_2)$	Regla semántica	
$E_1 \rightarrow E_2 \text{ oprel } E_3$	Regla semántica	Ahora estamos en el caso de una operación relacional: 50: if x > y goto 53 51: t=0 52: goto 55 53: t=1 54:
$E \rightarrow \text{true}$	Regla semántica	
$E \rightarrow \text{false}$	Regla semántica	

Producción		
$E_1 \rightarrow E_2 \text{ and } E_3$	Regla semántica	x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones será: $t_1 := x \text{ and } y$, donde t_1 será una
<pre> E1.nombre = tempnuevo() if E2.lugar = true and E3.lugar = true then E1.lugar = true else E1.lugar = false escribe (E1.lugar := E2.lugar "&&" E3.lugar) </pre>		

Producción		
$E_1 \rightarrow E_2 \text{ or } E_3$		x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones
<pre> E1.nombre = tempnuevo() if E2.lugar = true or E3.lugar = true then </pre>		



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

EJEMPLOS PARA DISTINTAS ESTRUCTURAS DE DATOS

Producción

$E_1 \rightarrow \text{not } E_2$

```

E1.nombre = tempnuevo( )
if E2.lugar = true then
    E1.lugar = false
else
    E1.lugar = true
escribe (E1.lugar := "!" E2.lugar )
    
```

Regla semántica

53: t=1
54:

x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones

Producción

$E_1 \rightarrow (E_2)$

```

E1.lugar = E2.lugar
    
```

Regla semántica

valor, representado por E.lugar.

$E_1 \rightarrow (E_2)$

$E_1 \rightarrow E_2 \text{ oprel } E_3$

$E \rightarrow \text{true}$

$E \rightarrow \text{false}$

Regla semántica

Regla semántica

Regla semántica

Regla semántica

Ahora estamos en el caso de una operación relacional:

```

50: if x > y goto 53
51: t = 0
52: goto 55
53: t = 1
54: ....
    
```

Producción

$E_1 \rightarrow E_2 \text{ oprel } E_3$

```

E1.nombre = tempnuevo( )
escribe (if E2.lugar "oprel.op" E3.lugar "goto" nextprop + 3)
escribe (E1.lugar := "0")
escribe (goto nextprop + 2)
escribe (E1.lugar := "1")
    
```

Regla semántica

Regla semántica

una operación relacional:

```

50: if x > y goto 53
51: t = 0
52: goto 55
53: t = 1
54: ....
    
```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

EJEMPLOS PARA DISTINTAS ESTRUCTURAS DE DATOS

Producción
 $E_1 \rightarrow E_2 \text{ and } E_3$
Regla semántica
 x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones será: $t_1 := x \text{ and } y$, donde t_1 será una variable temporal. Por tanto, lo primero es

Producción
 $E \rightarrow \text{true}$
 $E_1.\text{nombre} = \text{tempnuevo}()$
 escribe ($E_1.\text{lugar} := "1"$)

Regla semántica
 and operación relaciona:
 50: if x > y goto 53
 51: t= 0
 52: goto 55
 53: t=1
 54:

Producción
 $E \rightarrow \text{false}$
 $E_1.\text{nombre} = \text{tempnuevo}()$
 escribe ($E_1.\text{lugar} := "0"$)

Regla semántica
 x and y, (lo mismo x or y junto con !x) expresado en código de tres direcciones será: $t_1 := x \text{ and } y$, donde t_1 será una variable temporal. Por tanto, lo primero es obtener esa variable temporal con $E_1.\text{nombre} = \text{tempnuevo}()$, a la que dependiendo de la

Regla semántica
 51: t= 0
 52: goto 55
 53: t=1
 54:

[DDS para expresiones booleanas](#)
 Documentos

Atributos sintetizados

Utilizamos los siguientes atributos sintetizados:

- **E.nombre:** es el nombre que tendrá la variable temporal.
- **E.lugar:** es el nombre que tendrá el valor lógico de E (true o false).
- **Función tempnuevo():** devolverá el nombre del siguiente temporal.
- **Función escribe():** va a escribir en un fichero el código intermedio que se va generando.



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70**

Proposiciones de control de flujo

Las proposiciones de control de flujo se distinguen por la necesidad de generar etiquetas a las que saltará la secuencia del programa según se cumpla o no la condición o expresión booleana. Esta generación de etiquetas provoca un problema debido a que en el momento en el que se generan no se sabe dónde van a estar ubicadas.

Además de las proposiciones de control como el **if-then**, **if-then-else** o **while**, tenemos el bucle **for**, la proposición **case**, o **repeat**, las cuales son muy parecidas a estas tres que definimos.

Las principales proposiciones de control de flujo son:

- if E then S.
- if E then S else S.
- while E do S.

E es la expresión booleana que hay que evaluar y convertir a código de tres direcciones. Utilizaremos una función *nuevaeti()* que nos devolverá una nueva etiqueta cada vez que se llama. Estas etiquetas serán *E.true* y *E.false*, y es donde irá el control de flujo una vez se evalúe la mencionada expresión booleana.

 [Atributo código](#)
En detalle

Como nos indica Aho *et al* (2008) este *código* puede generarse directamente, sin necesidad de generar el árbol sintáctico y retrasar esta generación de código cuando se recorra el árbol.

También necesitamos un atributo heredado denominado *siguiente*, que nos proporciona el valor donde empieza la primera instrucción después del código correspondiente a S.



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Cartagena99

En detalle

Atributo *código*

También vamos a necesitar otro atributo denominado código (E.código), que será el que tiene el código de tres direcciones de las instrucciones, siendo por tanto del tipo cadena.

The logo for 'Cartagena99' features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the rest of the text. The logo is set against a light blue background with a white arrow pointing to the right, and a yellow shadow is cast beneath the text.

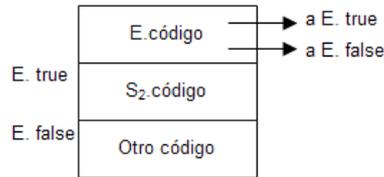
**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

DDS para if-then

La generación de código para las sentencias *if-then* e *if-then-else* pasa por entender a través de un esquema la forma en la que este se genera.

Empecemos por el de la sentencia $S_1 \rightarrow \text{if } E \text{ then } S_2$:



El patrón de código para *if (x > y) then a = 3* será:

```
t1 = true o false (dependiendo de la evaluación de x > y).
If t1 = false goto Etiq1
  Código para S2 (a:= 3)
Etiq1
Otro código
```

 [Implementación en forma de regla semántica](#)
En detalle

Hay que tener en cuenta que las expresiones booleanas y sus reglas semánticas complementan a las proposiciones de control de flujo en el proceso de evaluación de la condición.

En detalle

Implementación en forma de regla semántica

Su implementación en forma de regla semántica será:

Producción	$S_1 \rightarrow \text{if } E \text{ then } S_2$
Regla semántica	<pre>E.true := nuevaeti() E.false := S1.siguiente S2.siguiente := S1.siguiente S1.código = E.código escribe(E.true:) S2.código</pre>



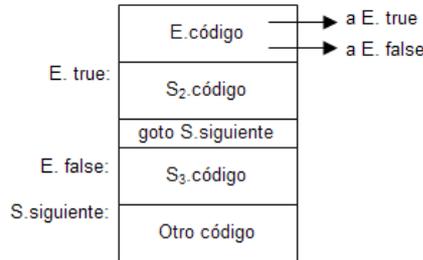
CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

- - -

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

DDS para if-then-else

Como en el caso anterior, se tiene que partir del esquema para $S_1 \rightarrow \text{if } E \text{ then } S_2 \text{ else } S_3$:



El patrón de código para *if (x > y) then a = 3 else a = 2* será:

```

t1 = true o false (dependiendo de la evaluación de x > y).
If t1 = false goto Etiq1
  Código para S2 (a:= 3)
  Goto Etiq2
Etiq1
  Código para S3 (a:= 2)
Etiq2
  Otro código
  
```

 [Implementación en forma de regla semántica](#)
En detalle

En detalle

Implementación en forma de regla semántica

Su implementación en forma de regla semántica será:

Producción	$S_1 \rightarrow \text{if } E \text{ then } S_2 \text{ else } S_3$
Regla semántica	<pre> E.true := nuevaetiq() E.false := nuevaetiq() S2.siguiente := S1.siguiente S3.siguiente := S1.siguiente S1.código = E.código escribe(E.true:) S2.código escribe("goto" S1.siguiente) escribe(E.false:) S3.código </pre>

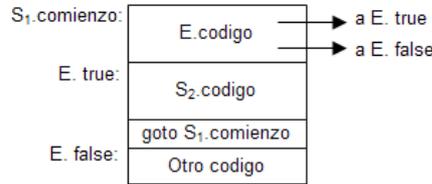


**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70**

DDS para while

Realizamos el esquema para $S_1 \rightarrow \text{while } E \text{ do } S_2$:



El patrón de código para *while* ($x > y$) *do* $a = a + 1$ será:

```

Etiq1
t1 = true o false (dependiendo de la evaluación de x > y).
If t1 = false goto Etiq2
  Código para S2 (a:= a + 1)
  goto Etiq1
Etiq2
Otro código
    
```

Su implementación en forma de regla semántica será:

[Implementación](#)
En detalle



En detalle

Implementación

Implementación en forma de regla semántica:

Producción	$S_1 \rightarrow \text{while } E \text{ do } S_2$
Regla semántica	<pre> S1.comienzo := nuevaetiq() E.true := nuevaetiq() E.false := S1.siguiente S2.siguiente := S1.comienzo S1.codigo = escribe (S1.comienzo:) E.codigo escribe(E.true:) S2.codigo escribe("goto" S1.comienzo) </pre>



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Relleno de etiquetas por retroceso

Como se indicó anteriormente, el principal problema de las proposiciones de control de flujo es rellenar estas etiquetas con un valor que todavía no conocemos, puesto que hasta que no está todo el código no se sabe dónde irán los saltos. Este problema se evita de una forma sencilla, aunque poco eficiente, dando dos pasadas.



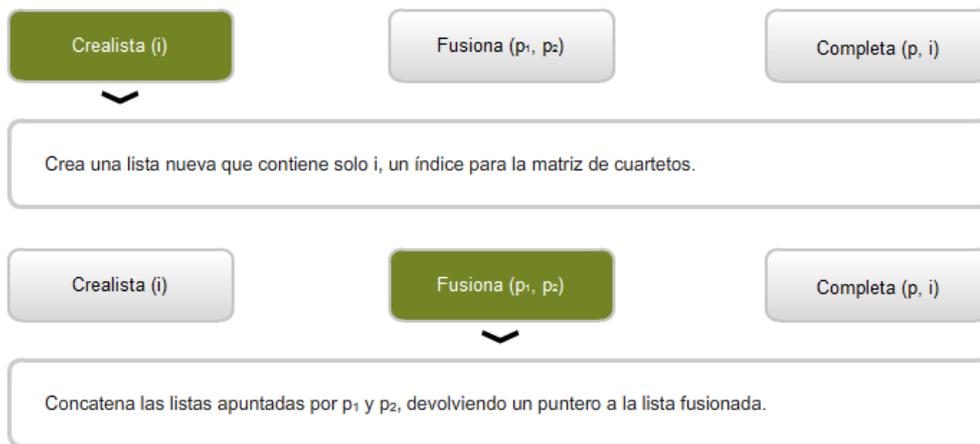
Una solución más eficiente a este problema (Aho *et al*, 1986) consiste en generar las proposiciones de bifurcación, pero sin especificar los destinos de los saltos, e ir completando una **lista de proposiciones**

goto, cuyas etiquetas se rellenarán cuando se pueda determinar la etiqueta adecuada. Esto se denomina **relleno por retroceso**.

Esto requiere que el código generado se mantenga en un buffer, de manera que se pueda rellenar mediante una única pasada.

Louden (2004) indica que el proceso de relleno puede generar otros problemas debido a las arquitecturas de las máquinas, puesto que muchas permiten dos variedades de saltos: un salto corto (dentro del alcance de 128 bytes), y un salto largo que requiere más espacio de código.

Para manejar las listas de etiquetas, Aho *et al* (1986) proponen tres funciones:

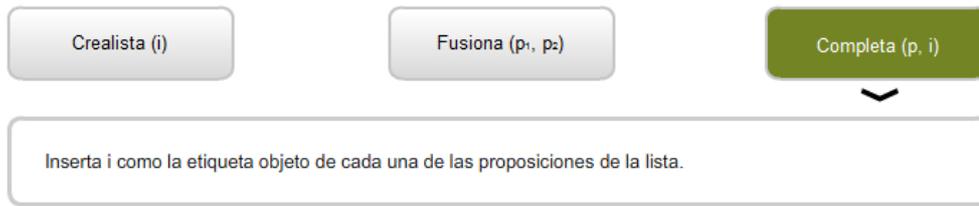


CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
 CALL OR WHATSAPP:689 45 44 70

GENERACIÓN DE CÓDIGO INTERMEDIO

EJEMPLOS PARA DISTINTAS ESTRUCTURAS DE DATOS



En Aho *et al* (1986 y 2008) hay ejemplos de implementación incluyendo las reglas semánticas necesarias.

Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

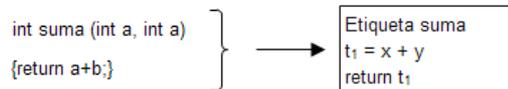
Llamadas a funciones

Las tareas más importantes en las funciones o procedimientos son la gestión de las llamadas y de los retornos. Para poder hacer la llamada a una función antes se tiene que haber realizado la declaración o definición de la misma.

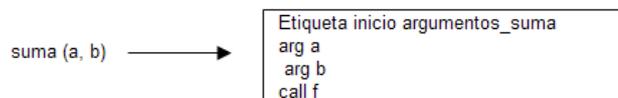
Hay que recordar que la arquitectura de memoria de la máquina es importante en este proceso, puesto que interviene en la forma en la que se reserva el espacio para la función, así como para tener control del valor devuelto por la función o procedimiento. Por todo ello, las tareas a realizar, aparte de la generación del código de la función en cuestión, son:

Declaración de la función	Esto implica introducir en la tabla de símbolos el nombre de la función, el tipo que devuelve y los parámetros o argumentos de la función. Ejemplos de declaración de funciones son los siguientes: <i>int nuevatemporal()</i> , o bien <i>boolean expresión(int a, char oprel, int b)</i> .
Llamada a la función	<ol style="list-style-type: none"> 1. Secuencia de llamada: incluir espacio para el registro de activación, argumentos y variables locales (incluyendo las variables temporales). 2. Secuencia de retorno: ubicación del valor de retorno, liberación de registros y liberación del espacio ocupado por el registro de activación.

El código de tres direcciones para la declaración de una función sencilla como:



Para la llamada a la función hay que indicar el inicio del cálculo del argumento, los argumentos para los que utilizamos la instrucción arg y, por último, la instrucción de llamada a funciones que llamamos call. Un ejemplo sería:



En Louden (2004) se define una gramática y algunos ejemplos de implementación.

Memoria



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70

Resumen

En este tema hemos aprendido cómo se construye el **código de tres direcciones** para las expresiones booleanas y las de control de flujo.

En primer lugar, se han definido los esquemas necesarios para entender el proceso, así como el patrón de código correspondiente al mismo, con el objeto de que el código que se genera mediante las reglas semánticas sea totalmente trazable y entendible.

También se ha visto **cómo obtener variables temporales para los cálculos y las etiquetas para los saltos de flujo**. Estos saltos se realizan una vez se ha evaluado la condición o expresión booleana correspondiente y hay que obtener el valor de la dirección de memoria a la que se va a saltar. Para resolver este problema, se ha visto una solución denominada **relleno por retroceso**.

Por último se han identificado las tareas necesarias para generar código de tres direcciones para las llamadas a funciones y las declaraciones de las mismas.

The logo for 'Cartagena99' features the text 'Cartagena99' in a stylized, blue, serif font. The '99' is significantly larger and more prominent than the 'Cartagena' part. The text is set against a light blue background with a subtle gradient and a soft shadow effect.

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS
CALL OR WHATSAPP:689 45 44 70