

## Tema 2.7. Subprogramas. Traducción

Pedro Javier Rodríguez Rodrigo, Víctor Cuadrado Juan

9 de mayo de 2014

# Índice

Índ.

## Memoria

- Enlaces estáticos
- Displays

## Nueva architect.

- Memoria
- Registros de activación

## Organización de la traducción

- Esquema de la traducción de subprogramas
- Inicio
- Prellamada
- Prólogo
- Epílogo
- Postllamada

## Paso de parámetros

- Por valor
- Por variable(referencia)
- Amplificación de la TS

## Traducción

- Inicio
- Declaraciones
- Procedimientos
- Bloque de código del proc.
- Invocaciones
- Modificación en la traducción
- Acceso a variables y parámetros

?

# Organización de la memoria

- Es posible acceder a los datos globales

# Organización de la memoria

- ▶ Es posible acceder a los datos globales
- ▶ Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)

# Organización de la memoria

- ▶ Es posible acceder a los datos globales
- ▶ Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- ▶ Dos Posibles organizaciones:

# Organización de la memoria

- ▶ Es posible acceder a los datos globales
- ▶ Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- ▶ Dos Posibles organizaciones:
  1. Enlaces estáticos

# Organización de la memoria

- ▶ Es posible acceder a los datos globales
- ▶ Desde cualquier registro de activación es necesario referir al registro de activación asociado con el bloque padre (que no tiene porque ser necesariamente el registro de activación anterior)
- ▶ Dos Posibles organizaciones:
  1. Enlaces estáticos
  2. Displays

# Enlaces estáticos

- ▶ En el registro de activación se incluye un enlace al registro de activación del bloque padre (enlace estático)
- ▶ La memoria se organiza en forma de pila de registros de activación, enlazados a través de los enlaces estáticos



# Enlaces estáticos: Ejemplo

```

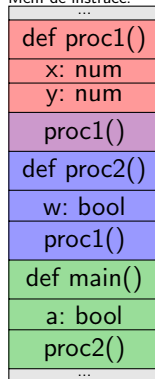
proc proc1(){
    x: num;
    y: num;
    proc1();
}

proc proc2(){
    w: bool;
    proc1();
}

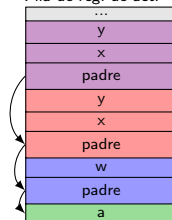
main(){
    a: bool;
    proc2();
}

```

Mem de instrucc.



Pila de reg. de act.



# Enlaces estáticos: Problemas

¿Qué problemas hay?

# Enlaces estáticos: Problemas

¿Qué problemas hay?

1. La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado  $k$  niveles por encima, es necesario realizar  $k$  indirecciones hasta llegar al correspondiente registro de activación

# Enlaces estáticos: Problemas

¿Qué problemas hay?

1. La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado  $k$  niveles por encima, es necesario realizar  $k$  indirecciones hasta llegar al correspondiente registro de activación
2. Hay que considerar la complejidad de generar código que gestione de manera adecuada los enlaces estáticos

# Enlaces estáticos: Problemas

¿Qué problemas hay?

1. La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado  $k$  niveles por encima, es necesario realizar  $k$  indirecciones hasta llegar al correspondiente registro de activación
2. Hay que considerar la complejidad de generar código que gestione de manera adecuada los enlaces estáticos

Solución:

# Enlaces estáticos: Problemas

¿Qué problemas hay?

1. La recuperación del enlace de un identificador global supone seguir toda la cadena de enlaces estáticos. Si el identificador ha sido declarado  $k$  niveles por encima, es necesario realizar  $k$  indirecciones hasta llegar al correspondiente registro de activación
2. Hay que considerar la complejidad de generar código que gestione de manera adecuada los enlaces estáticos

Solución:

Almacenar los enlaces estáticos *fuera* de los registros de activación. La estructura que los almacena se llama **display**.

# Display

- Secuencia de celdas consecutivas que apuntan a registros de activación

# Display

- ▶ Secuencia de celdas consecutivas que apuntan a registros de activación
- ▶ La celda  $i$  apunta al registro de activación que está siendo utilizado en el nivel de anidamiento  $i$



# Display

- ▶ Secuencia de celdas consecutivas que apuntan a registros de activación
- ▶ La celda  $i$  apunta al registro de activación que está siendo utilizado en el nivel de anidamiento  $i$
- ▶ Esta estructura facilita el acceso a los datos globales: el enlace para un identificador declarado en un bloque que se encuentra a profundidad  $i$  estará en el registro de activación referido por la celda  $i$  del display (el *display  $i$*  a partir de ahora)

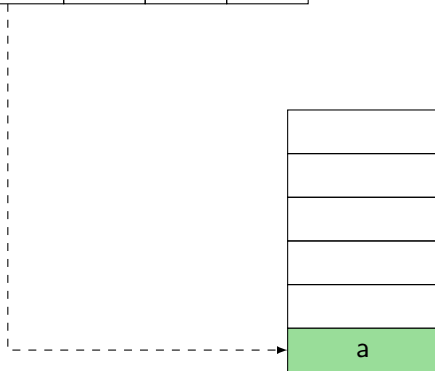
# Display: Ejemplo

```

proc proc1(){
    x: num;
    y: num;
    proc1();
}
proc proc2(){
    w: bool;
    proc1();
}
main(){
    a: bool;
    proc2();
}

```

| Disp0 | Disp1 | Disp2 | Disp3 |
|-------|-------|-------|-------|
| 0x09  | null  | null  | null  |

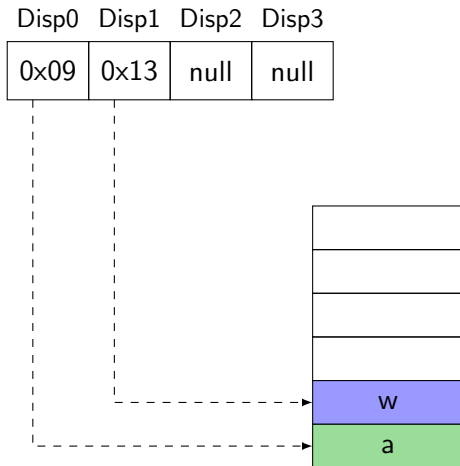


# Display: Ejemplo

```

proc proc1(){
    x: num;
    y: num;
    proc1();
}
proc proc2(){
    w: bool;
    proc1();
}
main(){
    a: bool;
    proc2();
}

```

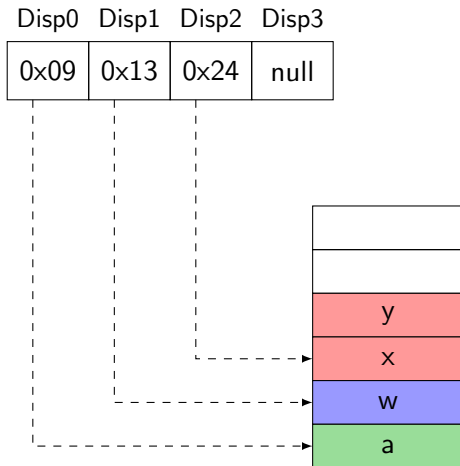


# Display: Ejemplo

```

proc proc1(){
    x: num;
    y: num;
    proc1();
}
proc proc2(){
    w: bool;
    proc1();
}
main(){
    a: bool;
    proc2();
}

```

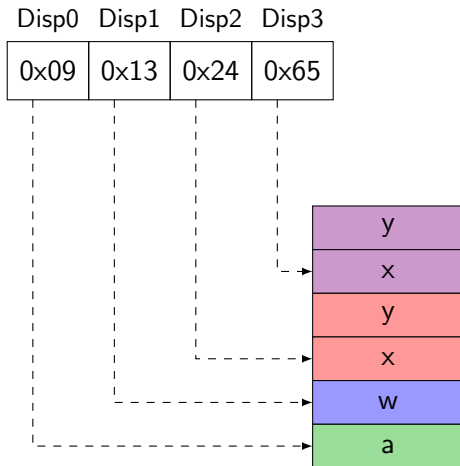


# Display: Ejemplo

```

proc proc1(){
    x: num;
    y: num;
    proc1();
}
proc proc2(){
    w: bool;
    proc1();
}
main(){
    a: bool;
    proc2();
}

```

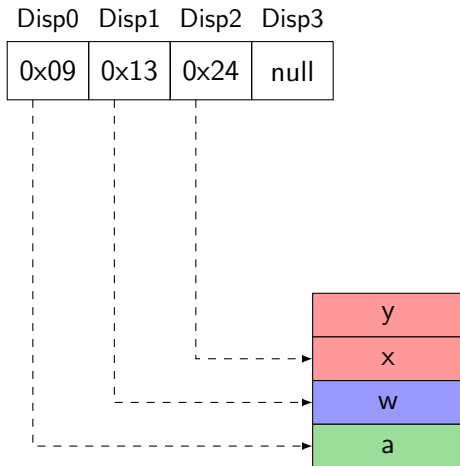


# Display: Ejemplo

```

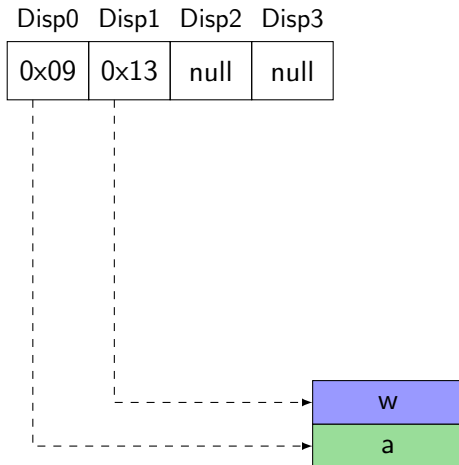
proc proc1(){
    x: num;
    y: num;
    proc1();
}
proc proc2(){
    w: bool;
    proc1();
}
main(){
    a: bool;
    proc2();
}

```



# Display: Ejemplo

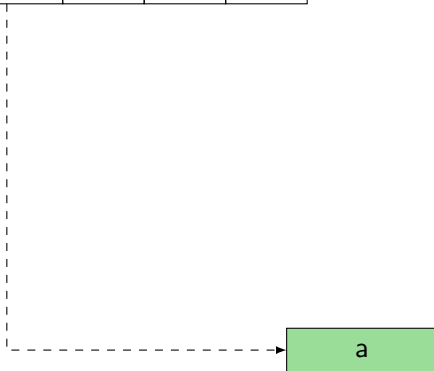
```
proc proc1(){  
    x: num;  
    y: num;  
    proc1();  
}  
proc proc2(){  
    w: bool;  
    proc1();  
}  
main(){  
    a: bool;  
    proc2();  
}
```



# Display: Ejemplo

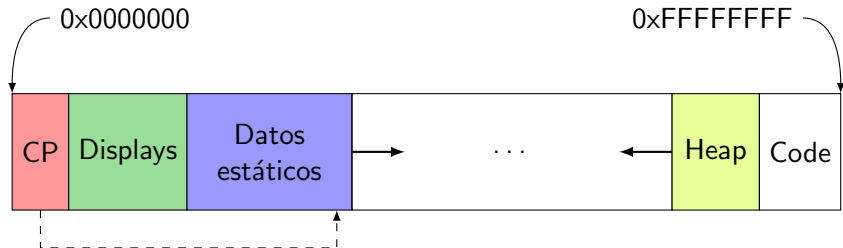
```
proc proc1(){  
    x: num;  
    y: num;  
    proc1();  
}  
proc proc2(){  
    w: bool;  
    proc1();  
}  
main(){  
    a: bool;  
    proc2();  
}
```

| Disp0 | Disp1 | Disp2 | Disp3 |
|-------|-------|-------|-------|
| 0x09  | null  | null  | null  |





# Memoria de un programa I



# Memoria de un programa II

Las primeras celdas de la memoria se destinarán a mantener la información de estado necesaria para gestionar adecuadamente la pila de registros de activación:

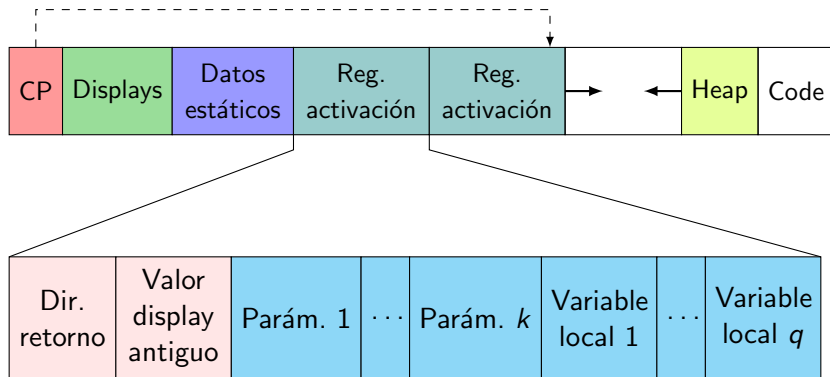
- ▶ Registro *CP*: Contendrá siempre la dirección de la **última celda ocupada** por la pila de registros de activación (cuando la pila esté vacía, el valor de *CP* será la dirección de la celda anterior: la última celda ocupada por los datos estáticos)

# Memoria de un programa II

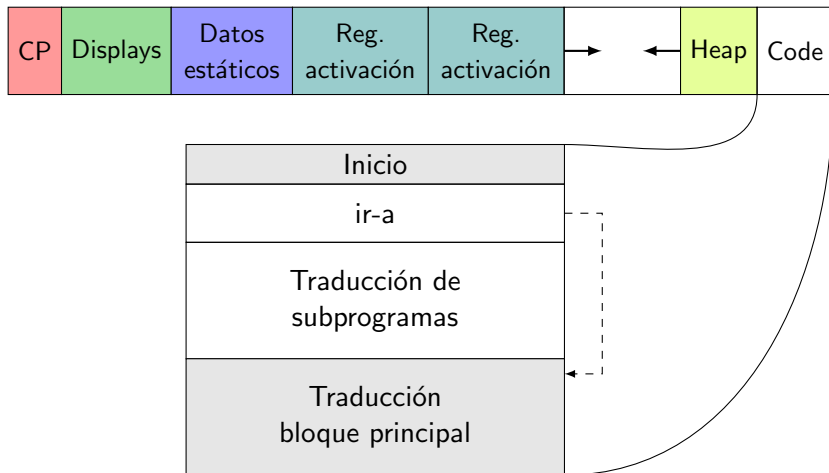
Las primeras celdas de la memoria se destinarán a mantener la información de estado necesaria para gestionar adecuadamente la pila de registros de activación:

- ▶ Registro *CP*: Contendrá siempre la dirección de la **última celda ocupada** por la pila de registros de activación (cuando la pila esté vacía, el valor de *CP* será la dirección de la celda anterior: la última celda ocupada por los datos estáticos)
- ▶ *Display*: Secuencia de celdas ocupadas por los displays.

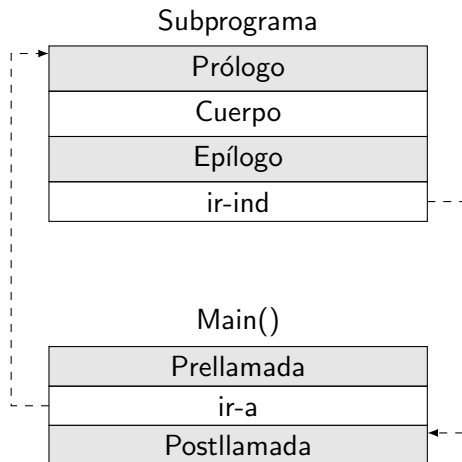
# Estructura de los Registros de activación



# Esquema de la traducción



# Manejo de la activación y desactivación I

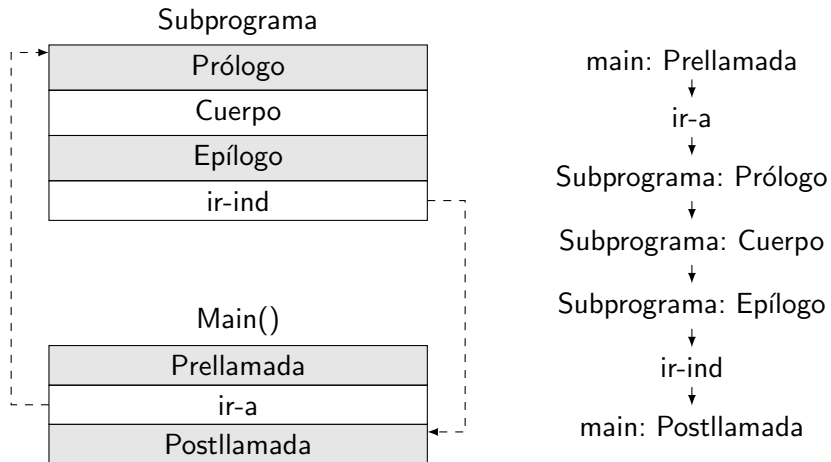


ir-ind salta a la dirección indicada en la cima de la pila de evaluación, consumiendo dicha cima.

$PC \leftarrow Pila[cima]$

$cima \leftarrow cima - 1$

# Manejo de la activación y desactivación II



# Ejemplo de invocación

```
tipo tpar= rec x:num; y:num;
proc distanciaEuclidea(p1:tpar, p2:tpar, var res
: num)
  a: num; b: num;
  proc sumacuadrado(a:num, b:num, var r:num)
    a:=a*a;
    b:=b*b
    r:=a+b;
  proc raizcuadrada(var n:num)
    ...
  &
  a:=p1.x-p2.x;
  b:=p1.y-p2.y;
  sumacuadrado(a, b, res);
  raizcuadrada(res);
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5;
par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,resultado);
```

¿Cual es el máximo nivel de anidamiento para éste programa?

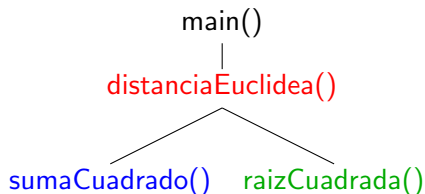


# Ejemplo de invocación

```

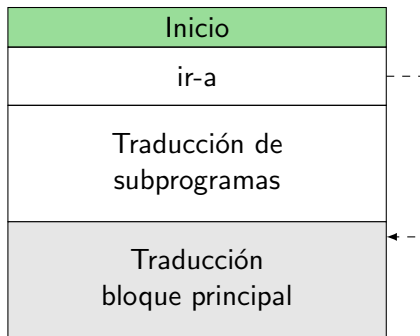
tipo tpar= rec x:num; y:num;
proc distanciaEuclidea(p1:tpar, p2:tpar, var res
:num)
  a: num; b: num;
  proc sumacuadrado(a:num, b:num, var r:num)
    a:=a*a;
    b:=b*b;
    r:=a+b;
  proc raizcuadrada(var n:num)
    ...
  &
  a:=p1.x-p2.x;
  b:=p1.y-p2.y;
  sumacuadrado(a, b, res);
  raizcuadrada(res);
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5;
par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,resultado);

```



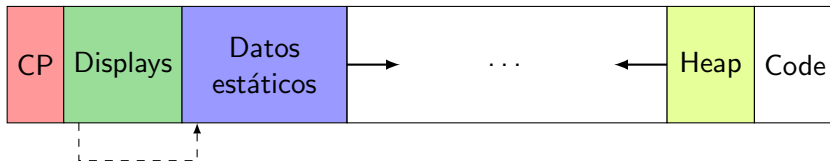
¿Cual es el máximo nivel de anidamiento para éste programa? 2

# Inicio: Recapitulación



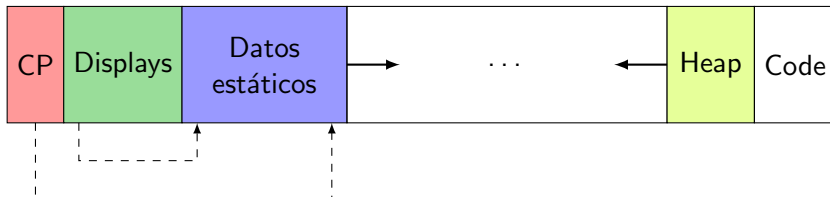
# Inicio I

- Se fija el *display 0* a la primera celda de datos estáticos



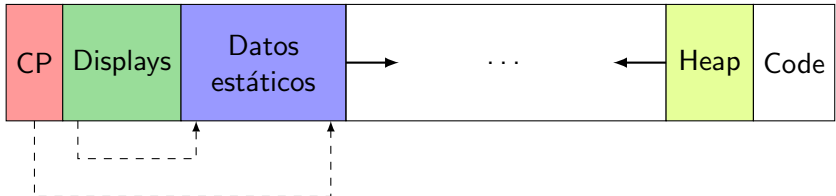
# Inicio I

- ▶ Se fija el *display 0* a la primera celda de datos estáticos
- ▶ Se fija el *CP* a la posición de la última celda ocupada por los datos estáticos.



# Inicio I

- ▶ Se fija el *display 0* a la primera celda de datos estáticos
- ▶ Se fija el *CP* a la posición de la última celda ocupada por los datos estáticos.
- ▶ Con ello se consigue un esquema homogéneo de direccionamiento de datos estáticos y de datos en los registros de activación



# Inicio II

```

1 fun inicio(numNiveles,tamDatos) devuelve
2   // fijamos display 0 a la 1a celda de datos estaticos:
3   apila(numNiveles+2)           ||// +2: CP, display 0
4   desapila-dir(0x1)             ||
5   // fijamos CP a la ultima celda de datos estaticos:
6   apila(1+numNiveles+tamDatos) ||// +1: display 0
7   desapila-dir(0x0)
8 ffun
9 cons longInicio = 4

```

Ejemplo: **inicio(2,5)**

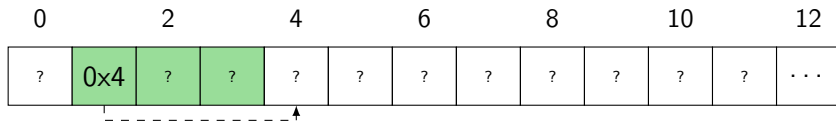
| 0 | 2 | 4 | 6 | 8 | 10 | 12  |
|---|---|---|---|---|----|-----|
| ? | ? | ? | ? | ? | ?  | ... |

# Inicio II

```

1  fun inicio(numNiveles,tamDatos) devuelve
2      // fijamos display 0 a la 1a celda de datos estaticos:
3      apila(numNiveles+2)           ||// +2: CP, display 0
4      desapila-dir(0x1)             ||
5      // fijamos CP a la ultima celda de datos estaticos:
6      apila(1+numNiveles+tamDatos) ||// +1: display 0
7      desapila-dir(0x0)
8  ffun
9  cons longInicio = 4
    
```

Ejemplo: **inicio(2,5)**



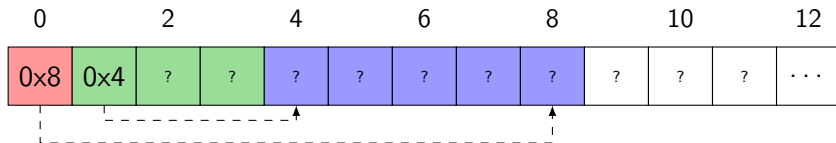
# Inicio II

```

1 fun inicio(numNiveles,tamDatos) devuelve
2   // fijamos display 0 a la 1a celda de datos estaticos:
3   apila(numNiveles+2)           ||// +2: CP, display 0
4   desapila-dir(0x1)            ||
5   // fijamos CP a la ultima celda de datos estaticos:
6   apila(1+numNiveles+tamDatos) ||// +1: display 0
7   desapila-dir(0x0)
8 ffun
9 cons longInicio = 4

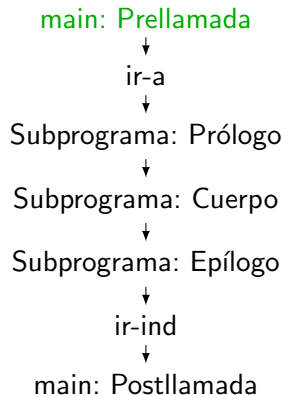
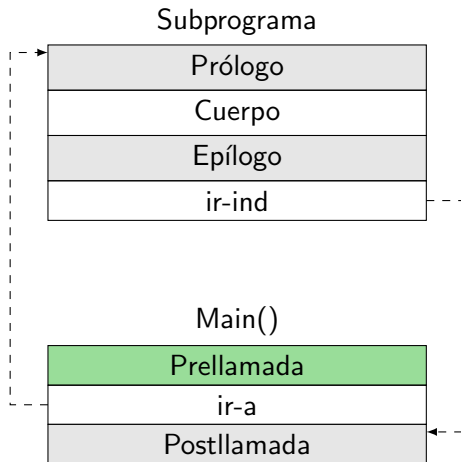
```

Ejemplo: **inicio(2,5)**





# Prellamada: Recapitulación



# Prellamada

Asociada con la invocación  $p(e_1, \dots, e_k)$ :

1. Guardar en memoria la direccion de retorno

# Prellamada

Asociada con la invocación  $p(e_1, \dots, e_k)$ :

1. Guardar en memoria la direccion de retorno
2. Evaluar y almacenar parámetros de ejecución del procedimiento (por ahora nos olvidamos)

# Prellamada

Asociada con la invocación  $p(e_1, \dots, e_k)$ :

1. Guardar en memoria la direccion de retorno
2. Evaluar y almacenar parámetros de ejecución del procedimiento (por ahora nos olvidamos)
3. Saltar a la dirección de inicio del procedimiento (*ir-a*)

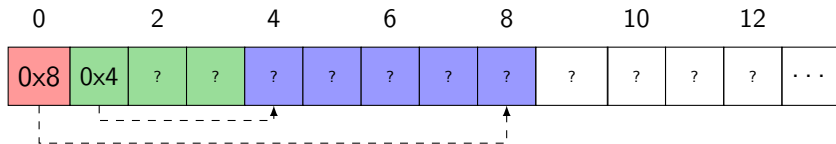
# Prellamada: Ejemplo

```

1 fun apila-ret(ret) devuelve
2   // calcular CP+1:
3   apila-dir(0x0)      ||
4   apila(1)            ||
5   suma                ||
6   // guardar dir retorno:
7   apila(ret)          ||
8   desapila-ind        ||
9 ffun
10 cons longApilaRet = 5

```

Ejemplo: **apila-ret(0x87)** , siendo 0x87 el n° de instrucción.



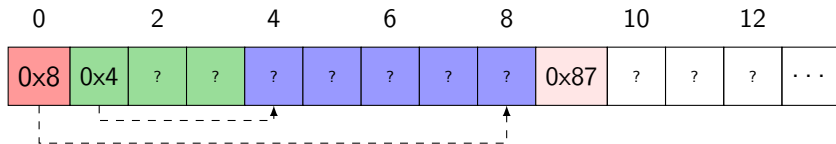
# Prellamada: Ejemplo

```

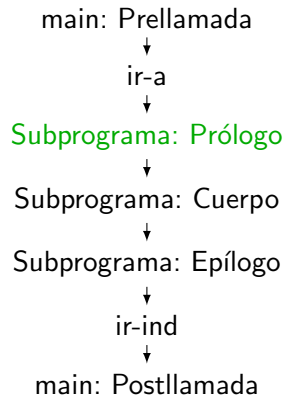
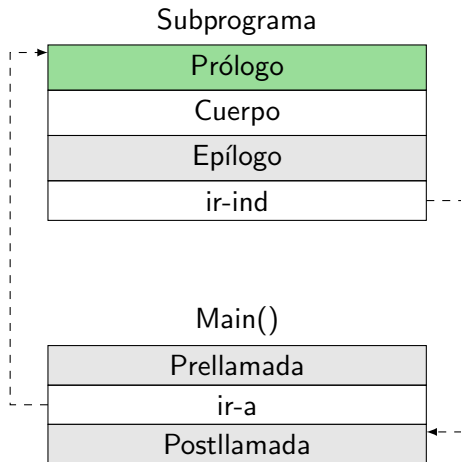
1  fun apila-ret(ret) devuelve
2    // calcular CP+1:
3    apila-dir(0x0)          ||
4    apila(1)                ||
5    suma                    ||
6    // guardar dir retorno:
7    apila(ret)              ||
8    desapila-ind            ||
9  ffun
10 cons longApilaRet = 5

```

Ejemplo: **apila-ret(0x87)** , siendo 0x87 el n° de instrucción.



# Prólogo: Recapitulación



# Prólogo I

Asociado con el prodecimiento *proc*  $p(\dots)$ :

1. Guardar el antiguo valor del display



# Prólogo I

Asociado con el prodecimiento *proc*  $p(\dots)$ :

1. Guardar el antiguo valor del display
2. Actualizar el valor nuevo del display

# Prólogo I

Asociado con el prodecimiento *proc*  $p(\dots)$ :

1. Guardar el antiguo valor del display
2. Actualizar el valor nuevo del display
3. Reservar espacio para las variables locales

# Prólogo II

```

1  fun prologo(nivel,tamlocales) devuelve
2      // salvar display antiguo:
3      apila-dir(0x0)           ||
4      apila(2)                 ||
5      suma                     ||
6      apila-dir(0x1+nivel)     ||
7      desapila-ind             ||
8      // fijar el display actual:
9      apila-dir(0x0)           ||
10     apila(3)                  ||
11     suma                      ||
12     desapila-dir(0x1+nivel)   ||
13     // reservar espacio para datos locales:
14     apila-dir(0x0)            ||
15     apila(tamlocales+2)       ||
16     suma                      ||
17     desapila-dir(0x0)
18  ffun
19  cons longPrologo = 13

```

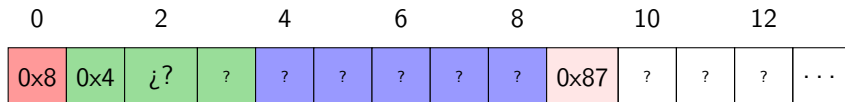
# Prólogo: Ejemplo I

```

1 fun prologo(nivel,tamlocales) devuelve
2   // salvar display antiguo:
3   apila-dir(0x0)           ||
4   apila(2)                 || // dir. retorno, antiguo display
5   suma                     ||
6   desapila-dir(0x1+nivel)  || // +1: saltar al display 0
7   desapila-ind             ||
8   // fijar el display actual:
9   apila-dir(0x0)           ||
10  apila(3)                 ||
11  suma                     ||
12  desapila-dir(0x1+nivel)  ||
13  // reservar espacio para datos locales:
14  apila-dir(0x0)           ||
15  apila(tamlocales+2)      ||
16  suma                     ||
17  desapila-dir(0x0)
18 ffun

```

Ejemplo: **prologo(1,7)**



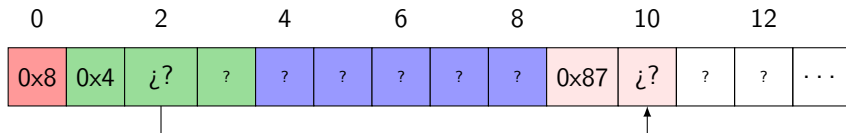
# Prólogo: Ejemplo I

```

1 fun prologo(nivel,tamlocales) devuelve
2   // salvar display antiguo:
3   apila-dir(0x0)           ||
4   apila(2)                 || // dir. retorno, antiguo display
5   suma                     ||
6   apila-dir(0x1+nivel)     || // +1: saltar al display 0
7   desapila-ind             ||
8   // fijar el display actual:
9   apila-dir(0x0)           ||
10  apila(3)                 ||
11  suma                     ||
12  desapila-dir(0x1+nivel)   ||
13  // reservar espacio para datos locales:
14  apila-dir(0x0)           ||
15  apila(tamlocales+2)       ||
16  suma                     ||
17  desapila-dir(0x0)
18 ffun

```

Ejemplo: **prologo(1,7)**



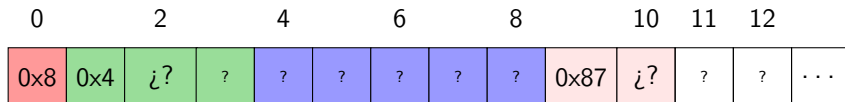
# Prólogo: Ejemplo II

```

1 fun prologo(nivel,tamlocales) devuelve
2   // salvar display antiguo:
3   apila-dir(0x0)           ||
4   apila(2)                 ||
5   suma                     ||
6   apila-dir(0x1+nivel)     ||
7   desapila-ind             ||
8   // fijar el display actual:
9   apila-dir(0x0)           ||
10  apila(3)                  ||
11  suma                      ||
12  desapila-dir(0x1+nivel) ||
13  // reservar espacio para datos locales:
14  apila-dir(0x0)            ||
15  apila(tamlocales+2)       ||
16  suma                      ||
17  desapila-dir(0x0)
18 ffun

```

Ejemplo: **prologo(1,7)**



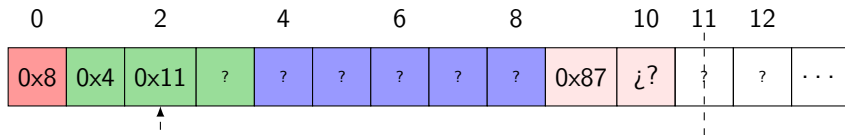
# Prólogo: Ejemplo II

```

1 fun prologo(nivel,tamlocales) devuelve
2   // salvar display antiguo:
3   apila-dir(0x0)           ||
4   apila(2)                 ||
5   suma                     ||
6   apila-dir(0x1+nivel)     ||
7   desapila-ind             ||
8   // fijar el display actual:
9   apila-dir(0x0)           ||
10  apila(3)                  ||
11  suma                      ||
12  desapila-dir(0x1+nivel) ||
13  // reservar espacio para datos locales:
14  apila-dir(0x0)           ||
15  apila(tamlocales+2)      ||
16  suma                     ||
17  desapila-dir(0x0)
18 ffun

```

Ejemplo: **prologo(1,7)**



## Prólogo: Ejemplo III

```

1  fun prologo(nivel,tamlocales) devuelve
2      // salvar display antiguo:
3      apila-dir(0x0)      ||
4      apila(2)            ||
5      suma                ||
6      apila-dir(0x1+nivel) ||
7      desapila-ind        ||
8      // fijar el display actual:
9      apila-dir(0x0)      ||
10     apila(3)            ||
11     suma                ||
12     desapila-dir(0x1+nivel) ||
13     // reservar espacio para datos locales:
14     apila-dir(0x0)      || // Mem[1+nivel] = Pila[Cima]
15     apila(tamlocales+2) || // +2: dir. retorno, antiguo display
16     suma                ||
17     desapila-dir(0x0)
18  ffun

```

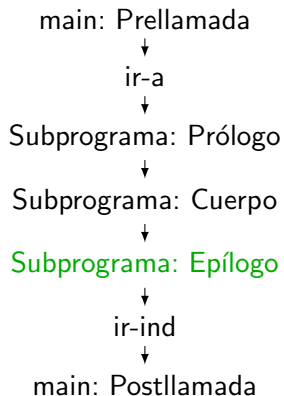
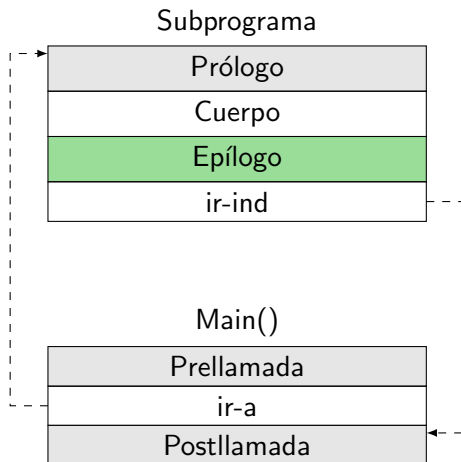
### Ejemplo: **prologo(1,7)**

| 0   | 2   | 4    | 6 | 8 | 10 | 12 | 14 | 16   | 18 |   |   |   |   |   |   |     |
|-----|-----|------|---|---|----|----|----|------|----|---|---|---|---|---|---|-----|
| 0x8 | 0x4 | 0x11 | ? | ? | ?  | ?  | ?  | 0x87 | ¿? | ? | ? | ? | ? | ? | ? | ... |





# Epílogo: Recapitulación



# Epílogo I

Asociado con el prodecimiento *proc*  $p(\dots)$ :

- ▶ Almacenar el valor devuelto por la función (en nuestro caso no se hace, no tenemos funciones)

# Epílogo I

Asociado con el prodecimiento *proc*  $p(\dots)$ :

- ▶ Almacenar el valor devuelto por la función (en nuestro caso no se hace, no tenemos funciones)
- 1. Liberar el espacio utilizado por las variables locales (mover hacia atrás el CP)

# Epílogo I

Asociado con el prodecimiento *proc p(...)*:

- ▶ Almacenar el valor devuelto por la función (en nuestro caso no se hace, no tenemos funciones)
- 1. Liberar el espacio utilizado por las variables locales (mover hacia atrás el CP)
- 2. Restaurar el antiguo display

# Epílogo I

Asociado con el prodecimiento *proc p(...)*:

- ▶ Almacenar el valor devuelto por la función (en nuestro caso no se hace, no tenemos funciones)
- 1. Liberar el espacio utilizado por las variables locales (mover hacia atrás el CP)
- 2. Restaurar el antiguo display
- 3. Apilar la dirección de retorno y saltar usando *ir-ind*

## Epílogo II

```

1  fun epilogo(nivel) devuelve
2      // apilar la dir. retorno:
3      apila-dir(0x1+nivel)    ||
4      apila(2)                ||
5      resta                   ||
6      apila-ind               ||
7      // liberar espacio (mover CP):
8      apila-dir(0x1+nivel)    ||
9      apila(3)                ||
10     resta                   ||
11     copia                   ||
12     desapila-dir(0x0)       ||
13     // recuperar antiguo display:
14     apila(2)                ||
15     suma                    ||
16     apila-ind               ||
17     desapila-dir(0x1+nivel)
18  ffun
19  cons longEpilogo = 13

```

# Epílogo: Ejemplo I

```

1  fun epilogo(nivel) devuelve
2      // apilar la dir. retorno:
3      apila-dir(0x1+nivel)  ||
4      apila(2)              ||
5      resta                 ||
6      apila-ind             ||
7      // liberar espacio (mover CP):
8      apila-dir(0x1+nivel)  ||
9      apila(3)              ||
10     resta                 ||
11     copia                 ||
12     desapila-dir(0x0)     ||
13     // recuperar antiguo display:
14     apila(2)              ||
15     suma                  ||
16     apila-ind             ||
17     desapila-dir(0x1+nivel)
18  ffun

```

## Ejemplo: epilogo(1)

|      |     |      |   |   |    |    |    |    |      |    |   |   |   |   |   |   |     |
|------|-----|------|---|---|----|----|----|----|------|----|---|---|---|---|---|---|-----|
| 0    | 2   | 4    | 6 | 8 | 10 | 12 | 14 | 16 | 18   |    |   |   |   |   |   |   |     |
| 0x17 | 0x4 | 0x11 | ? | ? | ?  | ?  | ?  | ?  | 0x87 | ¿? | ? | ? | ? | ? | ? | ? | ... |



# Epílogo: Ejemplo I

```

1 fun epilogo(nivel) devuelve
2   // apilar la dir. retorno:
3   apila-dir(0x1+nivel)  ||
4   apila(2)              ||
5   resta                 ||
6   apila-ind             ||
7   // liberar espacio (mover CP):
8   apila-dir(0x1+nivel)  ||
9   apila(3)              ||
10  resta                 ||
11  copia                 ||
12  desapila-dir(0x0)     ||
13  // recuperar antiguo display:
14  apila(2)              ||
15  suma                  ||
16  apila-ind             ||
17  desapila-dir(0x1+nivel)
18 ffun

```

## Ejemplo: epilogo(1)



→ Pila de evaluación

## Epílogo: Ejemplo II

```

1  fun epilogo(nivel) devuelve
2      // apilar la dir. retorno:
3      apila-dir(0x1+nivel)    ||
4      apila(2)                ||
5      resta                   ||
6      apila-ind               ||
7      // liberar espacio (mover CP):
8      apila-dir(0x1+nivel)    ||
9      apila(3)                ||
10     resta                   ||
11     copia                   || // si no se usa copia, hay que mover CP lo ultimo
12     desapila-dir(0x0)        ||
13     // recuperar antiguo display:
14     apila(2)                 ||
15     suma                     ||
16     apila-ind                ||
17     desapila-dir(0x1+nivel)
18  ffun

```

¿Es esta la mejor forma de implementar el epílogo? ¿Moviendo el CP en 2º lugar?

## Epílogo: Ejemplo II

```

1  fun epilogo(nivel) devuelve
2      // apilar la dir. retorno:
3      apila-dir(0x1+nivel)    ||
4      apila(2)                ||
5      resta                   ||
6      apila-ind               ||
7      // liberar espacio (mover CP):
8      apila-dir(0x1+nivel)    ||
9      apila(3)                ||
10     resta                   ||
11     copia                   || // si no se usa copia, hay que mover CP lo ultimo
12     desapila-dir(0x0)       ||
13     // recuperar antiguo display:
14     apila(2)                ||
15     suma                    ||
16     apila-ind               ||
17     desapila-dir(0x1+nivel)
18  ffun

```

### Ejemplo: **epilogo(1)**

| 0    | 2   | 4    | 6 | 8 | 10 | 12 | 14 | 16   | 18 |   |   |   |   |   |   |     |
|------|-----|------|---|---|----|----|----|------|----|---|---|---|---|---|---|-----|
| 0x17 | 0x4 | 0x11 | ? | ? | ?  | ?  | ?  | 0x87 | ¿? | ? | ? | ? | ? | ? | ? | ... |



## Epílogo: Ejemplo III

```

1 fun epilogo(nivel) devuelve
2   // apilar la dir. retorno:
3   apila-dir(0x1+nivel)    ||
4   apila(2)                ||
5   resta                   ||
6   apila-ind               ||
7   // liberar espacio (mover CP):
8   apila-dir(0x1+nivel)    ||
9   apila(3)                ||
10  resta                   ||
11  copia                   ||
12  desapila-dir(0x0)       ||
13  // recuperar antiguo display:
14  apila(2)                 ||
15  suma                     ||
16  apila-ind                ||
17  desapila-dir(0x1+nivel)
18 ffun

```

### Ejemplo: epilogo(1)

| 0   | 2   | 4    | 6 | 8 | 10 | 12 | 14 | 16   | 18 |   |   |   |   |   |   |     |
|-----|-----|------|---|---|----|----|----|------|----|---|---|---|---|---|---|-----|
| 0x8 | 0x4 | 0x11 | ? | ? | ?  | ?  | ?  | 0x87 | ¿? | ? | ? | ? | ? | ? | ? | ... |

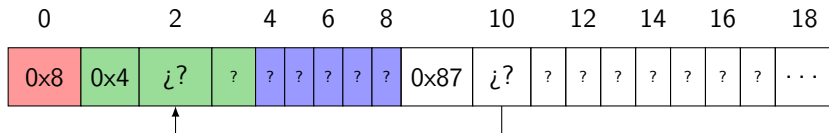
## Epílogo: Ejemplo III

```

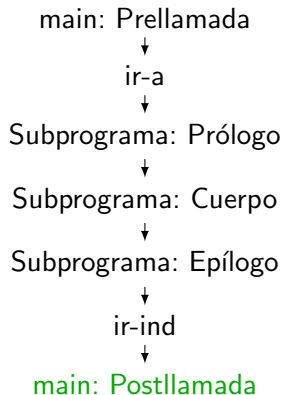
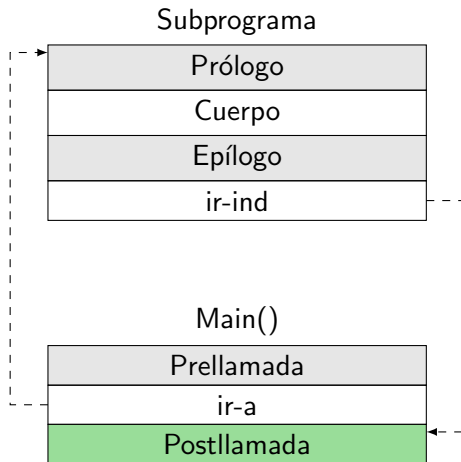
1 fun epilogo(nivel) devuelve
2   // apilar la dir. retorno:
3   apila-dir(0x1+nivel)      ||
4   apila(2)                  ||
5   resta                     ||
6   apila-ind                  ||
7   // liberar espacio (mover CP):
8   apila-dir(0x1+nivel)      ||
9   apila(3)                  ||
10  resta                      ||
11  copia                      ||
12  desapila-dir(0x0)          ||
13  // recuperar antiguo display:
14  apila(2)                   ||
15  suma                       ||
16  apila-ind                  ||
17  desapila-dir(0x1+nivel)
18 ffun

```

Ejemplo: **epilogo(1)**



# Postllamada: Recapitulación



# Postllamada

No es necesaria en nuestra implementación:

- El epílogo recupera el estado anterior a la invocación



# Postllamada

No es necesaria en nuestra implementación:

- ▶ El epílogo recupera el estado anterior a la invocación
- ▶ El epílogo termina y deja la dirección (indirecta) de retorno en la cima de la pila

# Postllamada

No es necesaria en nuestra implementación:

- ▶ El epílogo recupera el estado anterior a la invocación
- ▶ El epílogo termina y deja la dirección (indirecta) de retorno en la cima de la pila

En otras arquitecturas:

# Postllamada

No es necesaria en nuestra implementación:

- ▶ El epílogo recupera el estado anterior a la invocación
- ▶ El epílogo termina y deja la dirección (indirecta) de retorno en la cima de la pila

En otras arquitecturas:

- ▶ Soportan funciones con retorno de valor: copiar el valor devuelto por la función donde sea necesario

# Postllamada

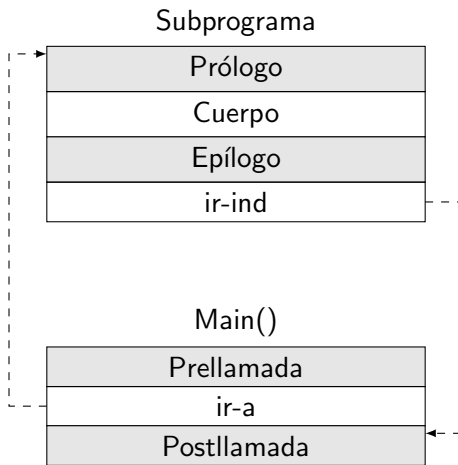
No es necesaria en nuestra implementación:

- ▶ El epílogo recupera el estado anterior a la invocación
- ▶ El epílogo termina y deja la dirección (indirecta) de retorno en la cima de la pila

En otras arquitecturas:

- ▶ Soportan funciones con retorno de valor: copiar el valor devuelto por la función donde sea necesario
- ▶ *x86, x86\_64, ARM . . .*: restaurar el estado (registros, diferentes punteros). . .

# Resumen



## Paso de parámetros (durante la prellamada)

- Las posiciones de los parámetros en el reg. de activación son relativas al *CP*

## Paso de parámetros (durante la prellamada)

- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al *CP*
- ▶ Durante la prellamada, el *CP* apunta a la celda anterior a la primera del reg. de activación

## Paso de parámetros (durante la prellamada)

- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al  $CP$
- ▶ Durante la prellamada, el  $CP$  apunta a la celda anterior a la primera del reg. de activación
- ▶ Por lo tanto, los parámetros y variables locales empezarán a partir de  $CP + 3$



## Paso de parámetros (durante la prellamada)

- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al  $CP$
- ▶ Durante la prellamada, el  $CP$  apunta a la celda anterior a la primera del reg. de activación
- ▶ Por lo tanto, los parámetros y variables locales empezarán a partir de  $CP + 3$
- ▶ Sus direcciones deben precalcularse en ejecución (antes de ejecutar el método) y guardarse en la TS

## Paso de parámetros (durante la prellamada)

- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al  $CP$
- ▶ Durante la prellamada, el  $CP$  apunta a la celda anterior a la primera del reg. de activación
- ▶ Por lo tanto, los parámetros y variables locales empezarán a partir de  $CP + 3$
- ▶ Sus direcciones deben precalcularse en ejecución (antes de ejecutar el método) y guardarse en la TS

¿Por qué deben precalcularse en ejecución?

## Paso de parámetros (durante la prellamada)

- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al  $CP$
- ▶ Durante la prellamada, el  $CP$  apunta a la celda anterior a la primera del reg. de activación
- ▶ Por lo tanto, los parámetros y variables locales empezarán a partir de  $CP + 3$
- ▶ Sus direcciones deben precalcularse en ejecución (antes de ejecutar el método) y guardarse en la TS

¿Por qué deben precalcularse en ejecución?

Por que el  $CP$  se va moviendo y dejan de ser accesibles

## Paso de parámetros (durante la prellamada)

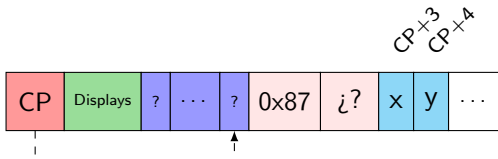
- ▶ Las posiciones de los parámetros en el reg. de activación son relativas al  $CP$
- ▶ Durante la prellamada, el  $CP$  apunta a la celda anterior a la primera del reg. de activación
- ▶ Por lo tanto, los parámetros y variables locales empezarán a partir de  $CP + 3$
- ▶ Sus direcciones deben precalcularse en ejecución (antes de ejecutar el método) y guardarse en la TS

¿Por qué deben precalcularse en ejecución?

Por que el  $CP$  se va moviendo y dejan de ser accesibles

Ejemplo: **proc**

**suma(x:num, y:num);**



# Paso de parámetros por valor

Dos formas:

1. modo **var**: `proc1(x);`

# Paso de parámetros por valor

Dos formas:

1. modo **var**: `proc1(x)`;
2. modo **val**: `proc1(3)`; ó `proc1(x+1)`; ó `proc1(3+4)`;  
Es decir, si pasa por la pila de evaluación, es de tipo **val**

## Paso de parámetros por valor

Dos formas:

1. modo **var**: `proc1(x)`;
2. modo **val**: `proc1(3)`; ó `proc1(x+1)`; ó `proc1(3+4)`;

Es decir, si pasa por la pila de evaluación, es de tipo **val**

¿Que se hace en nuestra arquitectura?

## Paso de parámetros por valor

Dos formas:

1. modo **var**: `proc1(x)`;
2. modo **val**: `proc1(3)`; ó `proc1(x+1)`; ó `proc1(3+4)`;  
Es decir, si pasa por la pila de evaluación, es de tipo **val**

¿Que se hace en nuestra arquitectura?

1. modo **var**: se copia el valor en el registro de activación (instrucción mueve)



## Paso de parámetros por valor

Dos formas:

1. modo **var**: `proc1(x)`;
2. modo **val**: `proc1(3)`; ó `proc1(x+1)`; ó `proc1(3+4)`;  
Es decir, si pasa por la pila de evaluación, es de tipo **val**

¿Que se hace en nuestra arquitectura?

1. modo **var**: se copia el valor en el registro de activación (instrucción mueve)
2. modo **val**: la cima de la pila de evaluación contendrá el valor de la expresión. Debemos desapilar el valor en el registro de activación

## Paso de parámetros por valor: Gramática

**var** es sólo para Mem. Las reglas que tienen operadores su modo es **val** ya que devuelven un tipo fijo (*true, false, number*)

|                          |                               |
|--------------------------|-------------------------------|
| Fact ::= Mem             | Term ::= Fact                 |
| Fact.mod0 = <b>var</b>   | Term . mod0 = Fact.mod0       |
| Fact ::= num             | ExpS ::= ExpS OpAd Term       |
| Fact.mod0 = <b>val</b>   | ExpS.mod0 = <b>val</b>        |
| Fact ::= true            | ExpS ::= ExpS <i>or</i> Term  |
| Fact.mod0 = <b>val</b>   | ExpS.mod0 = <b>val</b>        |
| Fact ::= false           | ExpS ::= Term                 |
| Fact.mod0 = <b>val</b>   | ExpS.mod0 = Term.mod0         |
| Fact ::= ( Exp )         | Exp ::= ExpS OpComp ExpS      |
| Fact.mod0 = Exp.mod0     | Exp.mod0 = <b>val</b>         |
| Term ::= Term OpMul Fact | Exp ::= ExpS                  |
| Term0.mod0 = <b>val</b>  | Exp.mod0 = ExpS.mod0          |
|                          | Term ::= Term <i>and</i> Fact |
|                          | Term0.mod0 = <b>val</b>       |

## Paso de parámetros por valor: var

- modo **var**: se copia el valor en el registro de activación (instrucción mueve, que copia un trozo de código a otras posiciones de memoria)

```

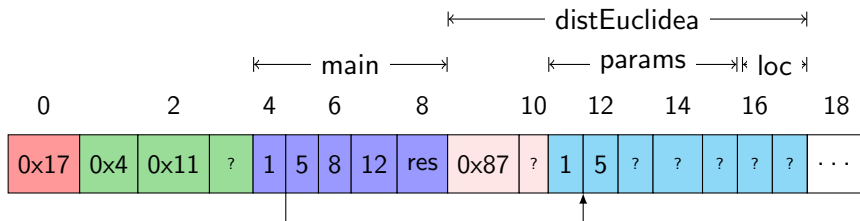
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,& resultado);
    
```

|      |     |      |   |   |    |    |    |     |      |   |   |   |   |   |   |   |     |
|------|-----|------|---|---|----|----|----|-----|------|---|---|---|---|---|---|---|-----|
| 0    | 2   | 4    | 6 | 8 | 10 | 12 | 14 | 16  | 18   |   |   |   |   |   |   |   |     |
| 0x17 | 0x4 | 0x11 | ? | 1 | 5  | 8  | 12 | res | 0x87 | ? | ? | ? | ? | ? | ? | ? | ... |

## Paso de parámetros por valor: var

- modo **var**: se copia el valor en el registro de activación (instrucción mueve, que copia un trozo de código a otras posiciones de memoria)

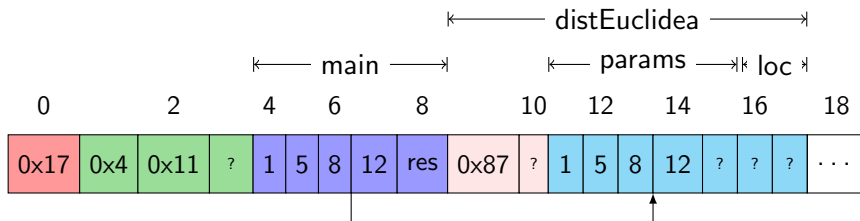
```
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,& resultado);
```



## Paso de parámetros por valor: var

- modo **var**: se copia el valor en el registro de activación (instrucción mueve, que copia un trozo de código a otras posiciones de memoria)

```
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,& resultado);
```



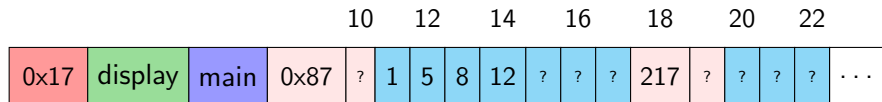
## Paso de parámetros por valor: val

- modo **val**: la cima de la pila de evaluación contendrá el valor de la expresión. Debemos desapilar el valor en el registro de activación

```

proc distanciaEuclidea(p1:tpar, p2:tpar, & res:num)
  ...
  proc sumacuadrado(a:num, b:num, & r:num)
    ...
  &
  sumacuadrado(3*2,7,& res);

```

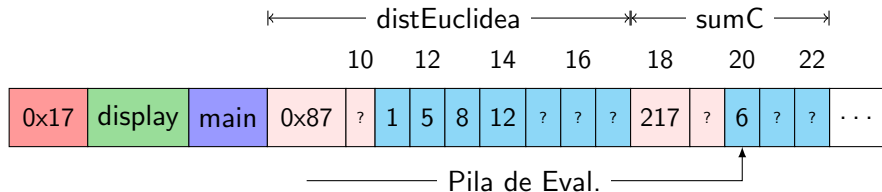


## Paso de parámetros por valor: val

- modo **val**: la cima de la pila de evaluación contendrá el valor de la expresión. Debemos desapilar el valor en el registro de activación

```

proc distanciaEuclidea(p1:tpar, p2:tpar, & res:num)
  ...
  proc sumacuadrado(a:num, b:num, & r:num)
    ...
  &
  sumacuadrado(3*2,7,& res);
  
```

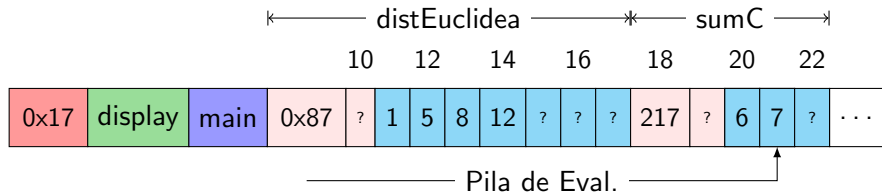


## Paso de parámetros por valor: val

- modo **val**: la cima de la pila de evaluación contendrá el valor de la expresión. Debemos desapilar el valor en el registro de activación

```

proc distanciaEuclidea(p1:tpar, p2:tpar, & res:num)
  ...
  proc sumacuadrado(a:num, b:num, & r:num)
    ...
  &
  sumacuadrado(3*2,7,& res);
  
```





## Paso de parámetros por variable(referencia) I

- En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación

## Paso de parámetros por variable(referencia) I

- ▶ En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación
- ▶ En la evaluación de las expresiones debe retardarse el apilado de los valores de las direcciones para Mem

## Paso de parámetros por variable(referencia) I

- ▶ En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación
- ▶ En la evaluación de las expresiones debe retardarse el apilado de los valores de las direcciones para Mem

¿Cómo sabe el procemiento que ha sido por **variable (&)**, y lo que hay en el registro de activación es una dirección de mem.?

# Paso de parámetros por variable(referencia) I

- ▶ En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación
- ▶ En la evaluación de las expresiones debe retardarse el apilado de los valores de las direcciones para Mem

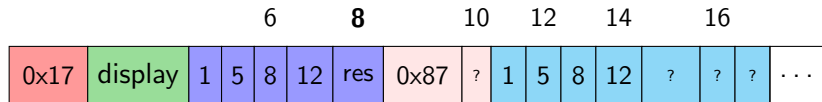
¿Cómo sabe el procemiento que ha sido por **variable (&)**, y lo que hay en el registro de activación es una dirección de mem.?

Se accede con indirección: Display + dir. de la variable de la TS

## Paso de parámetros por variable(referencia) II

- ▶ En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación

```
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,& resultado);
```

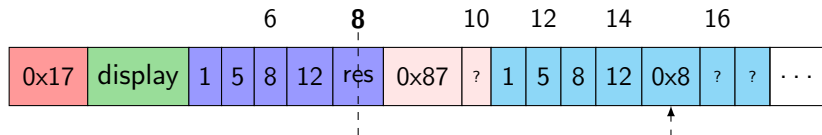


## Paso de parámetros por variable(referencia) II

- En la cima de la pila debe estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación

```

par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,& resultado);
    
```



## Paso de parámetros: ampliacion de la TS I

- ▶ Deben asignarse direcciones a los parámetros de los procedimientos:

```

LFPParams ::= LFPParams, FParam
    LFPParams0.ts = aniaideID(LFPParams1.ts, FParam.id,
    FParam.props  $\oplus$  <dir:LFPParams1.dir>)
    LFPParams0.dir = LFPParams1.dir + FParam.tam
    FParam.dirh = LFPParams1.dir
LFPParam ::= FParam
    LFPParam.ts = aniaideID(LFPParam.tsph, FParam.id, FParam.props  $\oplus$  <dir:0>
    )
    LFPParam.dir = FParam.tam
    FParam.dirh = 0
FParam ::= & iden: Tipo // por referencia
    FParam.tam = 1 // es una direccion: integer.tam = 1
    Fparam.param = <modo: variable, tipo: Tipo.tipo, dir: Fparam.dirh>
FParam ::= id: Tipo // por valor
    FParam.tam = Tipo.tipo.tam
    Fparam.param = <modo: valor, tipo: Tipo.tipo, dir: Fparam.dirh>
    
```

## Paso de parámetros: ampliación de la TS II

- La dirección de comienzo de un bloque de declaraciones no tiene que ser ya necesariamente cero, sino la dirección de fin del bloque de arámetros: las declaraciones reciben una dirección de comienzo heredada

```

Prog ::= Decs && Is
    Decs.dirh = 0
DecProc ::= proc iden FParams Bloque fproc
    Bloque.dirh = FParams.dir
FParams ::= ( LParams )
    FParams.dir = LParams.dir
FParams ::= λ
    FParams.dir = 0
Bloque ::= Decs && I
    Decs.dirh = Bloque.dirh // las decs reciben dirheredada
Decs ::= Decs ; Dec
    Decs1.dirh = Decs0.dirh
Decs ::= Dec
    Decs.dir = Decs.dirh + Dec.tam
    Dec.dirh = Decs.dirh

```



# Paso de parámetros: ampliacion de la TS III

| nivel | id                | clase             | otras   |
|-------|-------------------|-------------------|---|
| 1     | p1                | var //valor       | < tipo :< t : ref, id : tpar >, dir : 0 >   |
| 1     | p2                | var //valor       | < tipo :< t : ref, id : tpar >, dir : 2 >   |
| 1     | res               | pvar //referencia | < tipo :< t : num >, dir : 4 >  |
| 1     | distanciaEuclidea | prodedimiento     | < tipo :< t : proc, params : [modo : valor], tipo :< tipo :< t : red, id : tpar > ][modo : valor, tipo :< t : ref, id : tpar > ][modo : variable, tipo :< t : num > ] >>> |
| 1     | a                 | var //local       | < tipo :< t : num >, dir : 5 >  |
| 1     | b                 | var //local       | < tipo :< t : num >, dir : 6 >  |
|       |                   | :                 |   |
|       |                   | :                 |   |
| 1     | sumaCuadrado      | procedimiento     | < tipo :< t : proc, params : [modo : valor], tipo :< tipo :< t : num > ][modo : valor, tipo :< t : num > ][modo : variable, tipo :< t : num > ] >>>                       |
| 1     | raizCuadrada      | procedimiento     | < tipo :< t : proc, params : [modo : valor], tipo :< t : num > ] >>   |

|      |     |      |   |   |   |   |    |     |      |    |        |        |        |        |        |        |        |
|------|-----|------|---|---|---|---|----|-----|------|----|--------|--------|--------|--------|--------|--------|--------|
| 0    | 1   | 2    | 3 | 4 | 5 | 6 | 7  | 8   | 9    | 10 | Disp+0 | Disp+1 | Disp+2 | Disp+3 | Disp+4 | Disp+5 | Disp+6 |
| 0x17 | 0x4 | 0x11 | ? | 1 | 5 | 8 | 12 | res | 0x87 | ?  | 1      | 5      | 8      | 12     | 0x8    | a      | b      |

# Acceso a las variables y parámetros desde el procedimiento

- ▶ Desplazamiento relativo al valor del display del bloque en el que reside la variable
- ▶ Además, si el parámetro es por variable(referencia) (tipo *pvar*), debe realizarse un apilado indirecto para acceder al enlace en sí

```

1  fun accesoVar(id) devuelve
2      apila-dir(0x1+id.nivel)    || // dir del display
3      apila(id.dir)              || // desplazamiento respecto al display
4      suma                      // traer dir o valor a Cima
5      // aqui nos falta un apila_ind !!!
6      ( si id.clase = pvar entonces apila-ind
7        si no λ )
8  ffun
9
10 fun longAccesoVar(id)
11     si id.clase = pvar entonces 4
12     si no 3
13 ffun

```

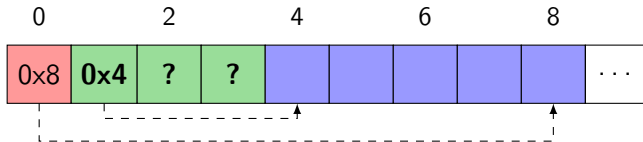
# Traducción: Inicio I

- ▶ Al iniciar la traducción, necesitamos reservar espacio para los displays
- ▶ Para ello, usamos la función *inicio()*

```

1 fun inicio(numNiveles,tamDatos)
2   apila(numNiveles+2)
3   desapila-dir(0x1)
4   apila(1+numNiveles+tamDatos)
5   desapila-dir(0x0)
6 ffun
7
8 cons longInicio = 4
    
```

Ejemplo: **inicio(2,5)**



# Traducción: Inicio II

- ▶ Para reservar el espacio, es necesario conocer el nivel de anidamiento, que se sintetiza en las declaraciones:

```

Prog ::= Decs && Is
      Decs.nh = 0
Decs ::= Decs ; Dec
      Decs1.nh = Decs0.nh
      Decs0.n = max(Decs1.n, Dec.n)
Decs ::= Dec
      Dec.nh = Decs.nh
      Decs.n = Dec.n
Dec ::= DecVar
      Dec.n = Dec.nh
Dec ::= DecTipo
      Dec.n = Dec.nh
Dec ::= DecProc
      DecProc.nh = Dec.nh
      Dec.n = DecProc.n

```

```

DecProc ::= proc iden FParams Bloque
          fproc
          FParams.nh = Bloque.nh = DecProc
            .nh + 1
          DecProc.n = Bloque.n
Bloque ::= Decs && I
          Decs.nh = Bloque.nh
          Bloque.n = Decs.n
Bloque ::= I
          Bloque.n = Bloque.nh

```

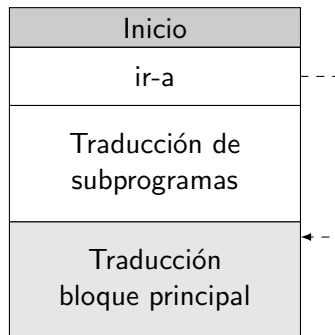
# Traducción: Inicio III

- Conocido el tamaño de los displays, la traducción del programa se especifica como:

```

Prog ::= Decs && Is //Subprogs ∈ Decs
Proc.cod = inicio(Decs.n,Decs.dir) ||
    // Decs.dir: tam del main
ir-a(Decs.etq) ||
Decs.cod || // aqui van los Subprogs
Is.cod ||
stop
Decs.etqh = longInicio +1 // +1: ir-a
Is.etqh = Decs.etq

```

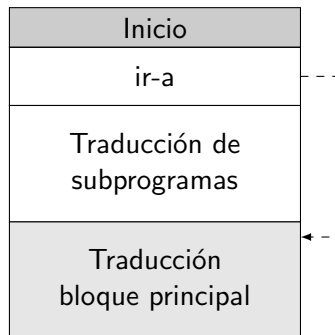


# Traducción: Inicio III

- Conocido el tamaño de los displays, la traducción del programa se especifica como:

```

Prog ::= Decs && Is //Subprogs ∈ Decs
Proc.cod = inicio(Decs.n,Decs.dir) ||
    // Decs.dir: tam del main
    ir-a(Decs.etq) ||
Decs.cod || // aqui van los Subprogs
Is.cod ||
stop
Decs.etqh = longInicio +1 // +1: ir-a
Is.etqh = Decs.etq
  
```



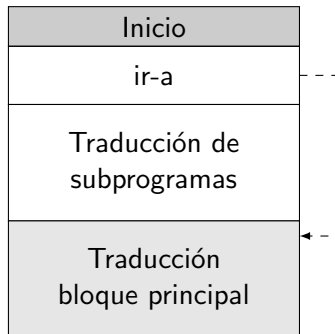
¿Hay que parchear?

## Traducción: Inicio III

- Conocido el tamaño de los displays, la traducción del programa se especifica como:

```

Prog ::= Decs && Is //Subprogs ∈ Decs
Proc.cod = inicio(Decs.n,Decs.dir) ||
    // Decs.dir: tam del main
ir-a(Decs.etq) ||
Decs.cod || // aqui van los Subprogs
Is.cod ||
stop
Decs.etqh = longInicio +1 // +1: ir-a
Is.etqh = Decs.etq
    
```



¿Hay que parchear?

Sí, el ir-a para apuntar al main, del cual no sabemos su dir todavía

# Traducción: Declaraciones

```

Prog ::= Decs && Is
Decs ::= Decs ; Dec
    Decs1.etqh = Decs0.etqh
    Dec.etqh = Decs1.etq
    Decso.etq = Dec.etq
    Decs0.cod = Decs1.cod || Dec.cod
Decs ::= Dec
    Dec.etqh = Decs.etqh
    Decs.etq = Dec.etq
    Decs.cod = Dec.cod
Dec ::= DecVar
    Dec.cod =  $\lambda$  // solo afecta a la TS
    Dec.etq = Dec.etqh
Dec ::= DecTipo
    Dec.cod =  $\lambda$  // solo afecta a la TS
Dec.etq = Dec.etqh

```



# Traducción: Procedimientos I

- ▶ Al traducir un procedimiento añadimos a la TS su dirección de inicio (para luego realizar el salto en la invocación)

```

Dec ::= DecProc
    DecProc.etqh = Dec.etqh
    Dec.etq = DecProc.etq
    Dec.cod = DecProc.cod
    Dec.propsop = DecProc.propsop // hay que parchear

DecProc ::= proc iden FParams Bloque fproc
    DecProc.cod = Bloque.cod
    DecProc.propsop = <inicio:Bloque.inicio> // Bloque.inicio: dir donde empieza el proc.
    Bloque.etqh = DecProc.etqh
    DecProc.etq = Bloque.etq
    Bloque.tsph = aniaideID(FParams.ts, DecProp.id,
        DecProc.props  $\oplus$  {nivel:DecsProp.nh +1}
         $\oplus$  DecProc.propsop)
    
```

## Traducción: Procedimientos II

| nivel | id                | clase             | otras  |
|-------|-------------------|-------------------|--|
| 0     | tpar              | tipo              | < tipo :< t : rec, campos : [id : x, tipo :< t : num > , desp : 0][id : y, tipo :< t : num > , desp : 1] >, tam : 2 >  |
| 0     | par1              | var //valor       | < tipo :< t : ref, id : tpar >>  |
| 0     | par2              | var //valor       | < tipo :< t : ref, id : tpar >>  |
| 1     | resultado         | pvar //referencia | < tipo :< t : num >>   |
| 1     | distanciaEuclidea | prodedimiento     | < tipo :< t : proc, params : [modo : valor, tipo :< t : ref, id : tpar >][modo : valor, tipo :< t : ref, id : tpar >][modo : variable, tipo :< t : num >] > inicio : 6 > |

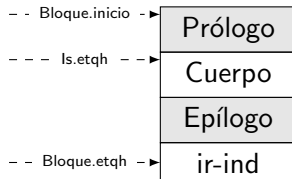
inicio: Dir de la 1ª instrucción (despues de Decs)

# Traducción: Bloque de código del proc.

```

Bloque ::= Decs && I
  Decs.etqh = Bloque.etqh
  Bloque.inicio = Decs.etq
  I.etqh = Decs.etq + longPrologo
  Bloque.etq = I.etq + longEpilogo + 1
  Bloque.cod = Decs.cod ||
                                prologo(Bloque.nh,
                                Decs.dir) ||
                                I.cod ||
                                epilogo(Bloque.nh ) ||
                                ir-ind

```



```

Bloque ::= I
  Bloque.cod = prologo(Bloque.nh, Bloque.dirh ) ||
                I.cod ||
                epilogo(Bloque.nh ) ||
                ir-ind
  I.etqh = Bloque.etqh + longPrologo
  Bloque.inicio = Bloque.etqh
  Bloque.etq = I.etq + longEpilogo + 1

```

# Traducción: Invocaciones I

```

I ::= ICall
    ICall.etqh = I.etqh
    I.etq = ICall.etq
    I.cod = ICall.cod
ICall ::= iden AParams
    ICall.cod = // Prellamada:
                apila-ret(ICall.etq) || // parchear a despues de proc
                AParams.cod           || // cod de '3+a'
                ir-a(ICall.tsh [iden.lex].inicio)
    AParams.etqh = ICall.etqh + longApilaRet
    ICall.etq = AParams.etq + 1 // +1: ir-a
AParams ::= LParams
    AParams.cod = inicio-paso || LParams.cod || fin-paso
    LParams.etqh = AParams.etqh + longInicioPaso
    AParams.etq = LParams.etq + longFinPaso

```

## Traducción: Invocaciones II (paso parámetros)

- El inicio del paso de parámetros apila la dirección de comienzo de los parámetros en el display:  $CP + 3$

```

1  cons inicio-paso = apila-dir(0x0) || // apila el CP
2                      apila(3)      || // salta ret, display antiguo
3                      suma
4  cons longInicioPaso = 3

```

## Traducción: Invocaciones II (paso parámetros)

- ▶ El inicio del paso de parámetros apila la dirección de comienzo de los parámetros en el display:  $CP + 3$

```

1  cons inicio-paso = apila-dir(0x0) || // apila el CP
2                      apila(3)      || // salta ret, display antiguo
3                      suma
4  cons longInicioPaso = 3

```

- ▶ Este valor permanece en la pila para ser utilizado por cada parámetro (que lo copiará)
- ▶ Por lo tanto, el fin del paso de parámetros simplemente desapila la dirección de comienzo de los parámetros

```

1  cons fin-paso = desapila
2  cons longFinPaso = 1

```

## Traducción: Invocaciones III (paso parám., func. auxiliares)

- Función para obtener la dirección de un parámetro en el registro de activación

```

1 fun direccionParFormal(pformal) devuelve
2   apila(pformal.dir) || // dir relativa al procedimiento
3   suma
4 ffun
5 fun longdireccionParFormal(pformal) devuelve 2 ffun

```

- Función para realizar el paso de los parámetros

```

1 fun pasoParametro(tipoParamPorValor,pformal) devuelve
2   si pformal.modo = val ^ tipoParamPorValor = var
3     // hay que apilar dir de origen del parametro. Se hace mas
4     adelante en accesoVar()
5     mueve(pformal.tipo.tam) // copia del valor en el caso de expr.
6     mem
7   si no desapila-ind          // copia del valor,0bien de la direcc.
8 ffun
9 fun longPasoParametro(tipoParamPorValor,pformal) devuelve1ffun

```

## Traducción: Invocaciones IV

```

AParams ::= λ
    AParams.cod = λ
    AParams.etq = AParams.etqh
LAParams ::= LAParams, Exp
    LAParams0.cod = LAParams1.cod      ||
                                copia    ||
                                direccionParFormal(
                                    LAParams0.fparams[LAParams0.nparams]
                                    )      ||
    Exp.cod      ||
    pasoParametro(Exp.modos,
                    LAParams0.fparams[LAParams0.nparams])
LAParams1.etqh = LAParams0.etqh
Exp.etqh = LAParams1.etq+1
LAParams0.etq = Exp.etq +1+
    longDireccionParFormal(
        LAParams0.fparams[LAParams0.nparams])+
    longPasoParametro(Exp.modos, LAParams0.fparams[
        LAParams0.nparams])

```



# Traducción: Invocaciones V

```

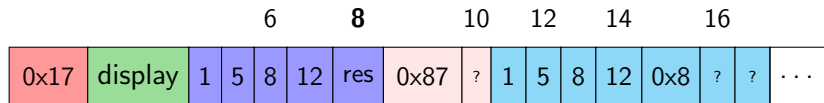
LAParams ::= Exp
    LAParams.cod = copia    ||
                        Exp.cod ||
                        pasoParametro(Exp.modulo, LAParams.fparams[1])
Exp.etqh = LAParams.etq+1
LAParams.etq = Exp.etq +1+
                longPasoParametro(Exp.modulo, LAParams.fparams[1])
    
```

# Traducción: Modificación en la trad. de expresiones I

- ▶ Problema con el paso de parámetros por variable(referencia):
  - ▶ En la cima de la pila debe de estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación
- ▶ Sin embargo, en las expresiones *Mem* hay un *apila-ind* que deja su valor (no la dirección) en la cima

```

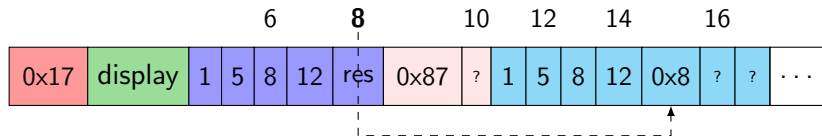
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,resultado);
    
```



# Traducción: Modificación en la trad. de expresiones I

- ▶ Problema con el paso de parámetros por variable(referencia):
  - ▶ En la cima de la pila debe de estar la dirección de comienzo de la variable pasada como parámetro. Dicha dirección se copia en el registro de activación
- ▶ Sin embargo, en las expresiones *Mem* hay un *apila-ind* que deja su valor (no la dirección) en la cima

```
par1:tpar; par2:tpar; resultado:num;
&
par1.x:=1; par1.y:=5; par2.x:=8; par2.y:=12;
distanciaEuclidea(par1,par2,resultado);
```



# Traducción: Modificación en la trad. de expresiones II

## Solución:

- ▶ Debe evitarse cargar adelantadamente el valor de las posiciones de memoria cuando éstas aparecen como parámetros reales
- ▶ A fin de no modificar el esquema de generación de código de las expresiones y asignación, la solución menos intrusiva es determinar si el contexto de ocurrencia de un designador (derivado de *Mem*) es como parámetro real: atributo heredado *parh*

# Traducción: Modificación en la trad. de expresiones III

```

Exp ::= ExpS OpComp ExpS
    ExpS0.parh = ExpS1.parh =
        false
Exp ::= ExpS
    ExpS.parh = Exp.parh
ExpS ::= ExpS OpAd Term
    ExpS1.parh = Term.parh =
        false
ExpS ::= ExpS or Term
    ExpS1.parh = Term.parh =
        false
    
```

```

ExpS ::= Term
    Term.parh = ExpS.parh
Term.parh = ExpS.parh
Term ::= Term OpMul Fact
    Term1.parh = Fact.parh =
        false
Term ::= Term and Fact
    Term1.parh = Fact.parh =
        false
Term ::= Fact
    Fact.parh = Term.parh
Fact ::= (Exp)
    Exp.parh = Fact.parh
Fact ::= OpUn Fact
    Fact1.parh = false
    
```

## Traducción: Modificación en la trad. de expresiones IV

- Los valores iniciales deben fijarse allí donde pueda aparecer una expresión:

```

IASig ::= Mem := Exp
    Exp.parh = false
IIf ::= if Exp then I PElse
    Exp.parh = false
IWhile ::= while Exp do I PElse
    Exp.parh = false
LAParams ::= LAParams, Exp
    Exp.parh = (LAParams0.fparamsh[LAParams0.nparams].modo == var)
LAParams ::= Exp
    Exp.parh = (LAParams0.fparamsh[1].modo == var)
    
```

# Traducción: Modificación en la trad. de expresiones V

- Ahora la generación de código para *Mem* puede modificarse como sigue:

```
Fact ::= Mem
    Fact.cod = si compatible(Mem.tipo,<t:num>,Fact.tsh ) ∨
                compatible(Mem.tipo,<t:bool>,Fact.tsh ) ∧
                ¬ Fact.parh
            entonces
                Mem.cod || apila-ind // Por valor
                si no Mem.cod          // Por variable
Fact.etq = si compatible(Mem.tipo,<t:num>,Fact.tsh ) ∨
            compatible(Mem.tipo,<t:bool>,Fact.tsh ) ∧
            ¬ Fact.parh
        entonces
            Mem.etq + 1
        si no Mem.etq
```

# Traducción: Modificación en la trad. de expresiones VI

- El otro aspecto a modificar es el acceso a los identificadores: ahora debe computarse adecuadamente el enlace a los mismos:

```
Mem ::= id  
    Mem.cod = accesoVar(Mem.tsh [id.lex])  
    Mem.etq = Mem.etqh + longAccesoVar(Mem.tsh [id.lex])
```



# Traducción: Acceso a las variables y parámetros desde el proc. I

- ▶ Desplazamiento relativo al valor del display del bloque en el que reside la variable
- ▶ Además, si el parámetro es por variable(referencia) (tipo *pvar*), debe realizarse un apilado indirecto para acceder al enlace en sí

```

1  fun accesoVar(id) devuelve
2      apila-dir(0x1+id.nivel) ||
3      apila(id.dir)           ||
4      suma
5      ( si id.clase = pvar entonces apila-ind
6        si no λ )
7  ffun
8
9  fun longAccesoVar(id) devuelve
10     si id.clase = pvar entonces 4
11     si no 3
12  ffun

```

# Traducción: Acceso a las variables y parámetros desde el proc. II

| nivel | id                | clase             | otras  |
|-------|-------------------|-------------------|--|
| 0     | par1              | var //valor       | < tipo :< t : ref, id : tpar >>  |
| 0     | par2              | var //valor       | < tipo :< t : ref, id : tpar >>  |
| 1     | resultado         | pvar //referencia | < tipo :< t : num >>   |
| 1     | distanciaEuclidea | prodedimiento     | < tipo :< t : proc, params : [modo : valor, tipo :< t : ref, id : tpar >][modo : valor, tipo :< t : ref, id : tpar >][modo : variable, tipo :< t : num >] > inicio : 6 > |
| 1     | a                 | pvar //referencia | < tipo :< t : num >>   |
| 1     | b                 | pvar //referencia | < tipo :< t : num >>   |
| 1     | sumaCuadrado      | prodedimiento     | < tipo :< t : proc, params : [modo : valor, tipo :< t : num >][modo : valor, tipo :< t : numr >][modo : variable, tipo :< t : num >] >>                                  |
| 1     | raizCuadrada      | prodedimiento     | < tipo :< t : proc, params : [modo : variable, tipo :< t : num >] >>   |

|      |     |      |   |   |   |   |    |     |      |    |        |        |        |        |        |        |        |
|------|-----|------|---|---|---|---|----|-----|------|----|--------|--------|--------|--------|--------|--------|--------|
| 0    | 1   | 2    | 3 | 4 | 5 | 6 | 7  | 8   | 9    | 10 | Disp+0 | Disp+1 | Disp+2 | Disp+3 | Disp+4 | Disp+5 | Disp+6 |
| 0x17 | 0x4 | 0x11 | ? | 1 | 5 | 8 | 12 | res | 0x87 | ?  | 1      | 5      | 8      | 12     | 0x8    | a      | b      |

# Preguntas

