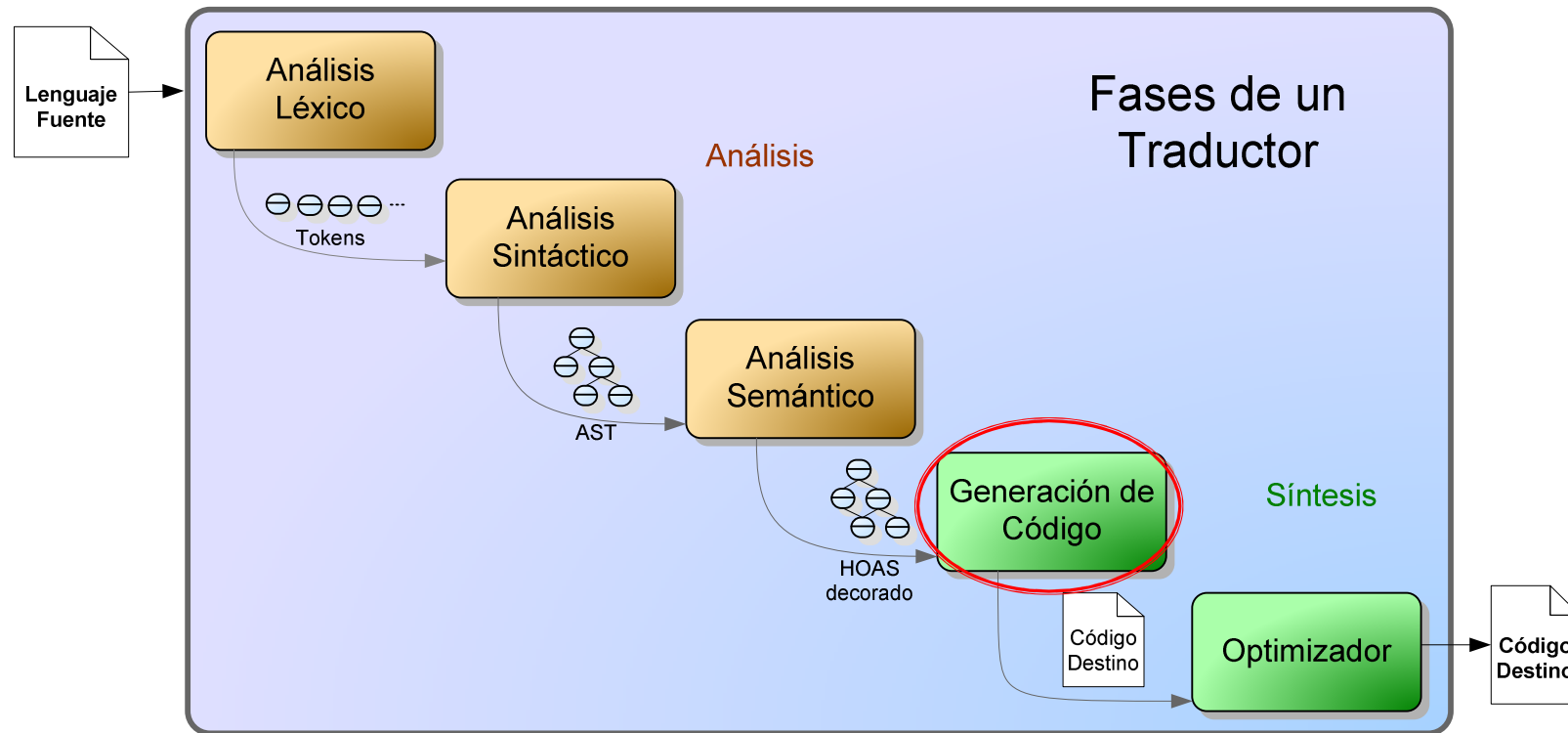

Generación de Código (I)

Diseño de Lenguajes de Programación
Ingeniería Informática
Universidad de Oviedo
(v1.9)

Raúl Izquierdo Castanedo

Usted está aquí...



Cuál es la situación

Alto Nivel	
Datos	Código
Arrays	Funciones
Estructuras	Sobrecarga
Referencias	Enlace dinámico
Clases	Funciones de Primer Orden
Genericidad	Excepciones
...	...

Bajo nivel	
Datos	Código
Enteros	Mover bytes
Reales	Operar
	Saltar

Qué hay que hacer

La fase de Generación de Código se encarga de determinar...

- Cómo se representan los Datos
 - Representar los tipos abstractos de alto nivel con las limitaciones de la plataforma de destino
- Cómo se traduce el Código
 - Traducir las sentencias de alto nivel a secuencias de instrucciones de la plataforma de destino

personas[2].edad = 5



pusha 38
push 5
store

Subfases:

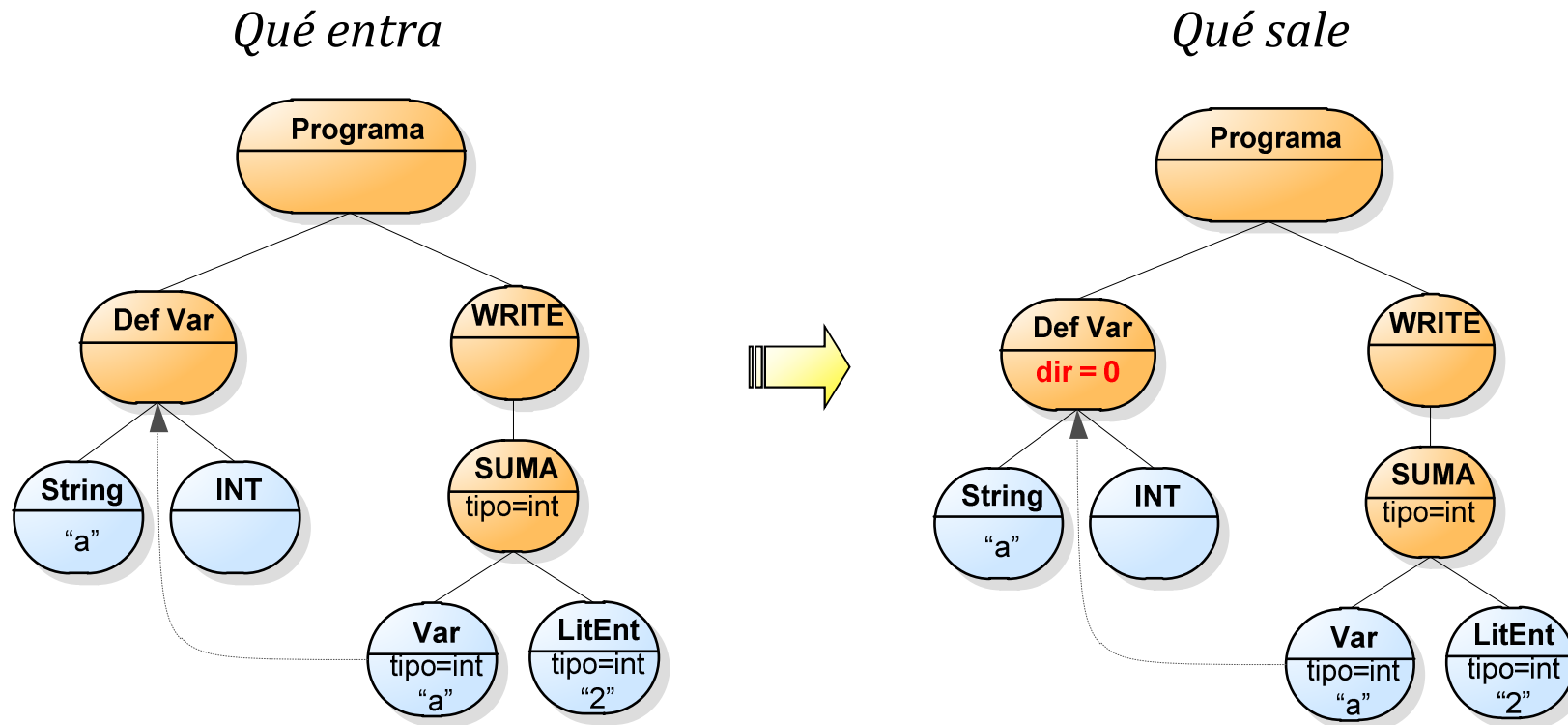
- Gestión de Memoria
- Selección de Instrucciones

Gestión de Memoria

Gestión de Memoria

Fase que se encarga de

- Determinar **cómo** y **dónde** representar todas las abstracciones de alto nivel en la plataforma de destino



Gestión de Memoria

Para determinar *cómo* ubicar cada símbolo se necesita saber

- ¿De qué se dispone en la plataforma destino?
 - Normalmente
 - Segmento de Código
 - Segmento de Datos
 - Segmento de Pila

Segmento de Pila: Control de Flujo

`call <etiqueta>` \Rightarrow
 {

`push (IP+1)`

`push BP`

`BP = SP`

`jmp <etiqueta>`

Instrucciones	Memoria Estática	Vista de la Pila
0000 push 6	00 00	1011 00
0001 call f	01 00	1012 00
0002 out	02 00	1013 00
0003 halt	03 00	1014 00
f:	04 00	1015 00
0004 push 1492	05 00	1016 00
0005 out	06 00	1017 00
0006 ret	07 00	1018 00
	08 00	1019 00
	09 00	1020 00
	10 00	1021 00
	11 00	SP 06
	12 00	1023 00 6 push

Instrucciones	Memoria Estática	Vista de la Pila
0000 push 6	00 00	1011 00
0001 call f	01 00	1012 00
0002 out	02 00	1013 00
0003 halt	03 00	1014 00
f:	04 00	1015 00
0004 push 1492	05 00	1016 00
0005 out	06 00	1017 00
0006 ret	07 00	1018 00
	08 00	1019 00
	09 00	1020 00
	10 00	1021 00
	11 00	SP BP 00
	12 00	1023 00 6 push

`ret` \Rightarrow
 {

`BP = pop`

`IP = pop`

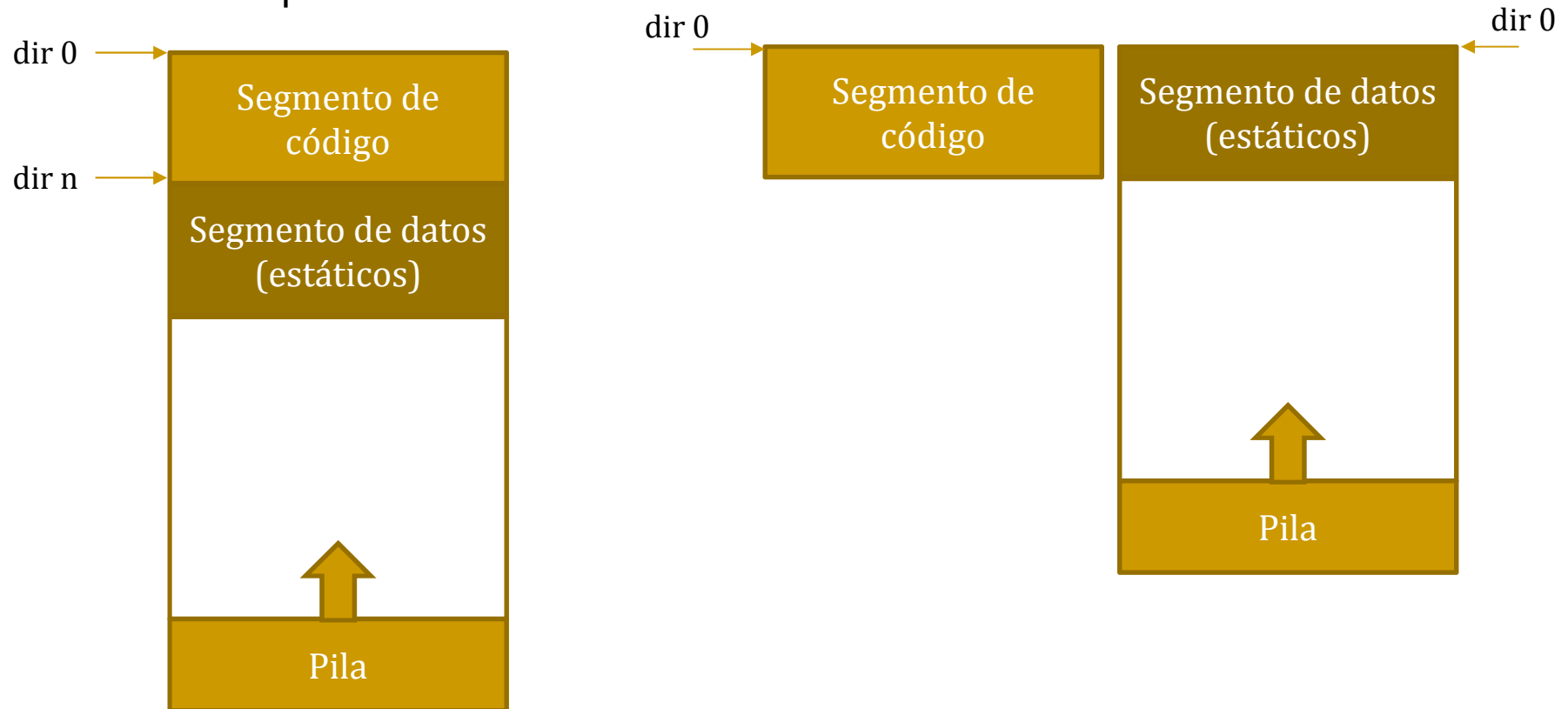
Instrucciones	Memoria Estática	Vista de la Pila
0000 push 6	00 00	1011 00
0001 call f	01 00	1012 00
0002 out	02 00	1013 00
0003 halt	03 00	1014 00
f:	04 00	1015 00
0004 push 1492	05 00	1016 212
0005 out	06 00	1017 05
0006 ret	07 00	SP BP 00
	08 00	1019 04 BP anterior 1024
	09 00	1020 02 Dirección de retorno 2
	10 00	1021 00
	11 00	1022 06
	12 00	1023 00 6 push

Instrucciones	Memoria Estática	Vista de la Pila
0000 push 6	00 00	1011 00
0001 call f	01 00	1012 00
0002 out	02 00	1013 00
0003 halt	03 00	1014 00
f:	04 00	1015 00
0004 push 1492	05 00	1016 212
0005 out	06 00	1017 05
0006 ret	07 00	1018 00
	08 00	1019 04
	09 00	1020 02
	10 00	1021 00
	11 00	SP 06
	12 00	1023 00 6 push

Ubicación de los segmentos

Los tres segmentos anteriores pueden aparecer:

- En mismo espacio de direcciones
- En espacios separados
- Esquema mixto



Representación de Datos

Representación de Datos

La representación de un dato se determina en función de:

- Su ámbito
- Su tipo

¿Qué determina cada uno?

Representación de Datos por Ámbito

Representación por ámbito

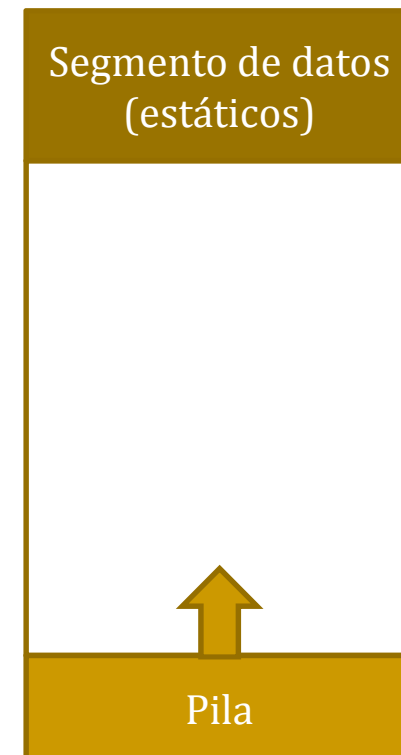
El ámbito de una dato indica DÓNDE se representa

Los datos más habituales son:

- Variables globales
- Literales
- Variables locales
- Parámetros
- Valores de retorno

Para cada uno de ellos hay que determinar

- En qué segmento se ubican
- En qué dirección dentro del mismo



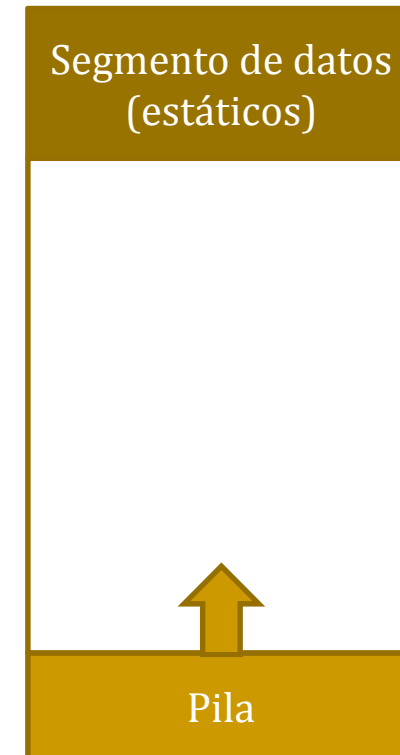
Variables Globales

Representación de variables globales

- ¿En qué segmento se ubican?
 - ¿En qué dirección?

```
int a;  
real f;  
char c;  
real g;
```

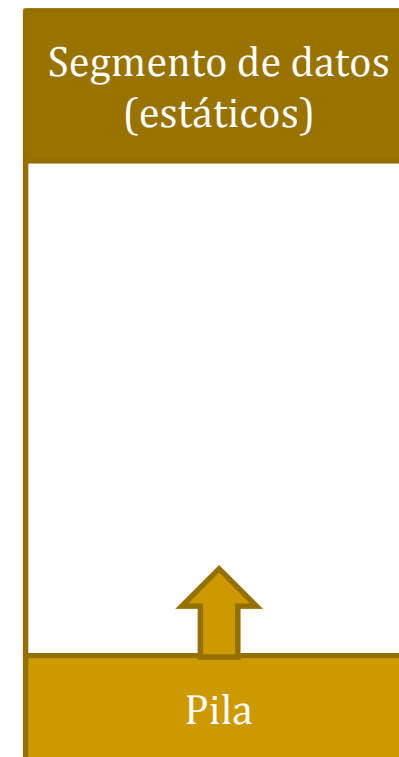
- ¿Qué código se generaría?
f = 1,618;



Variables Locales

Representación de variables locales

- ¿En qué segmento se ubican?
 - Depende



Variables locales en Segmento de Datos

Representación de variables locales

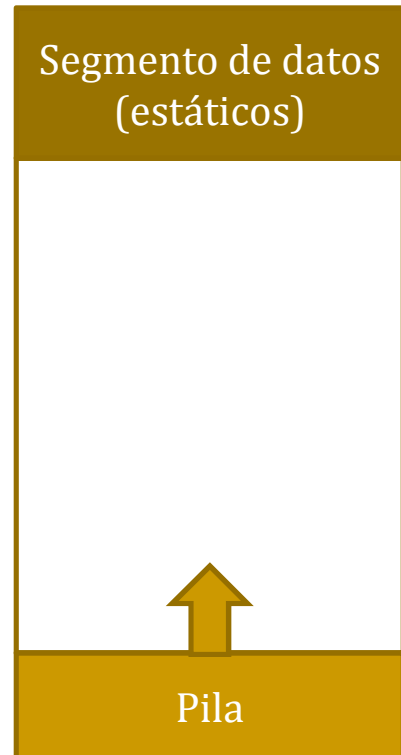
- En segmento de datos

```
int a;  
int b;  
real c;
```

```
void f() {  
    int local_f1;  
    int local_f2;  
    g();  
}
```

```
void g() {  
    real local_g1;  
    char local_g2;  
    g();  
}
```

```
void main() {  
    int local_main;  
    f();  
}
```



globales

f

g

main

Memoria Estática

00	00	a	(sin inicializar)
01	00		
02	00	b	(sin inicializar)
03	00		
04	00	c	(sin inicializar)
05	00		
06	00	local_f1	(sin inicializar)
07	00		
08	00	local_f2	(sin inicializar)
09	00		
10	00		
11	00	local_g1	(sin inicializar)
12	00		
13	00		
14	00	local_g2	(sin inicializar)
15	00		
16	00	local_main	(sin inicializar)
17	00		
18	00		
19	00		

- ¿Cómo se asignan direcciones?
- ¿Qué no se puede hacer en estos lenguajes?

Variables locales en la Pila

Representación de variables locales

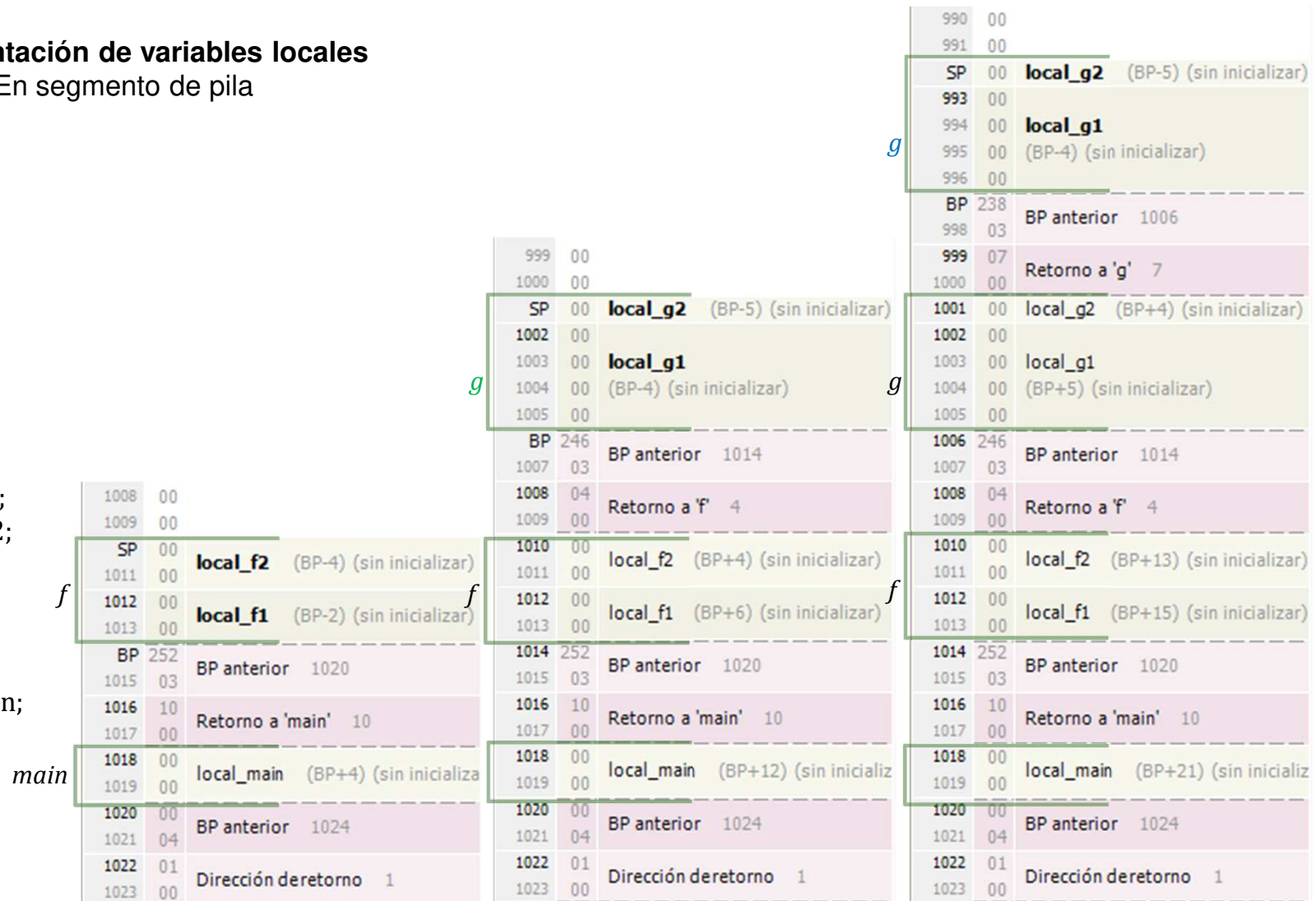
- En segmento de pila

```
int a;
int b;
```

```
void f() {
    int local_f1;
    int local_f2;
    g0;
}
```

```
void g() {
    real local_g1;
    char local_g2;
    g0;
}
```

```
void main() {
    int local_main;
    f();
}
```



Variables locales. Creación (I)

Representación de variables locales

- ¿Cómo se reserva el espacio para las locales de cada función dinámicamente?

<pre>void f() { int local_f1; int local_f2; 0002 enter 4 g(); 0003 call g [0005] } 0004 ret 0, 4, 0 void g() { real local_g1; char local_g2; 0005 enter 5 g(); 0006 call g [0005] } 0007 ret 0, 5, 0 void main() { int local_main; 0008 enter 2 f(); 0009 call f [0002] } 0010 ret 0, 2, 0</pre>		$\text{enter } \langle n \rangle \Rightarrow \text{SP} = \text{SP} - \langle n \rangle$	
		1008	00
		1009	00
		1010	00
		1011	00
		1012	00
		1013	00
	SP BP	252	
	1015	03	BP anterior 1020
	1016	10	Retorno a 'main' 10
	1017	00	
	1018	00	local_main (BP+4) (sin ini
	1019	00	
	1020	00	BP anterior 1024
	1021	04	
	1022	01	Dirección deretorno 1
	1023	00	

<pre>void f() { int local_f1; int local_f2; 0002 enter 4 g(); 0003 call g [0005] } 0004 ret 0, 4, 0 void g() { real local_g1; char local_g2; 0005 enter 5 g(); 0006 call g [0005] } 0007 ret 0, 5, 0 void main() { int local_main; 0008 enter 2 f(); 0009 call f [0002] } 0010 ret 0, 2, 0</pre>			
		1008	00
		1009	00
		1010	00
		1011	00
		1012	00
		1013	00
	SP	00	local_f2 (BP-4) (sin ini
	1011	00	
	1012	00	local_f1 (BP-2) (sin ini
	1013	00	
	BP	252	
	1015	03	BP anterior 1020
	1016	10	Retorno a 'main' 10
	1017	00	
	1018	00	local_main (BP+4) (sin i
	1019	00	
	1020	00	BP anterior 1024
	1021	04	
	1022	01	Dirección deretorno 1
	1023	00	

Variables locales. Direcciones

Representación de variables locales en el segmento de pila

- ¿Cuál es su dirección?

$$dir(local_i) = BP - \sum_{j=0}^i tamaño(local_j)$$

```
void f() {
  int f1;
  int f2;

  f1 = 0;
}
```

f1 = 0; → pusha ¿?
push 0
store

```
void g() {
  real g1;
  f();
}
```

```
void main() {
  char c1;
  char c2;
  f();
  g();
}
```

999	00		
1000	00		
1001	00		
1002	00		
1003	00		
1004	00		
1005	00		
1006	00		
1007	00		
1008	00		
1009	00		
SP	00	f2	(BP-4) (sin inicializar)
1011	00		
1012	00	f1	(BP-2) (sin inicializar)
1013	00		
BP	252	BP anterior	1020
1015	03		
1016	09	Retorno a 'main'	9
1017	00		
1018	00	c2	(BP+4) (sin inicializar)
1019	00	c1	(BP+5) (sin inicializar)
1020	00		
1021	04	BP anterior	1024
1022	01	Dirección deretorno	1
1023	00		

999	00		
1000	00		
1001	00		
SP	00	f2	(BP-4) (sin inicializar)
1003	00		
1004	00	f1	(BP-2) (sin inicializar)
1005	00		
BP	246	BP anterior	1014
1007	03		
1008	06	Retorno a 'g'	6
1009	00		
1010	00		
1011	00	g1	(BP+4) (sin inicializar)
1012	00		
1013	00		
1014	252	BP anterior	1020
1015	03		
1016	10	Retorno a 'main'	10
1017	00		
1018	00	c2	(BP+12) (sin inicializar)
1019	00	c1	(BP+13) (sin inicializar)
1020	00		
1021	04	BP anterior	1024
1022	01	Dirección deretorno	1
1023	00		

¿Y la solución es?

Variable locales. Liberación

¿Cómo debería quedar la pila al salir de una función?

	void f() {		
	int local_f1;		
	int local_f2;		
0002	enter 4		
	print 1111;		
0003	push 1111		
0004	out		
	}		
→ 0005	ret 0, 4, 0		
	void g() {		
	real local_g1;		
	char local_g2;		
0006	enter 5		
	print 2222;		
0007	push 2222		
0008	out		
	}		
0009	ret 0, 5, 0		
	void main() {		
	int local_main;		
0010	enter 2		
	f();		
0011	call f [0002]		
	g();		
0012	call g [0006]		
	}		
0013	ret 0, 2, 0		

1008	87	
1009	04	
SP	00	
1011	00	local_f2 (BP-4) (sin inici
1012	00	
1013	00	local_f1 (BP-2) (sin inici
BP	252	
1015	03	BP anterior 1020
1016	12	
1017	00	Retorno a 'main' 12
1018	00	
1019	00	local_main (BP+4) (sin
1020	00	
1021	04	BP anterior 1024
1022	01	
1023	00	Dirección deretorno 1

	void f() {		
	int local_f1;		
	int local_f2;		
0002	enter 4		
	print 1111;		
0003	push 1111		
0004	out		
	}		
0005	ret 0, 4, 0		
	void g() {		
	real local_g1;		
	char local_g2;		
0006	enter 5	1008	87
	print 2222;	1009	04
0007	push 2222	1010	00
0008	out	1011	00
	}	1012	00
0009	ret 0, 5, 0	1013	00
	void main() {	1014	252
	int local_main;	1015	03
0010	enter 2	1016	12
	f();	1017	00
0011	call f [0002]	SP	00
	g();	1019	00
0012	call g [0006]	BP	00
	}	1021	04
0013	ret 0, 2, 0	1022	01
		1023	00

	local_main (BP-2) (sin i
BP anterior	1024
Dirección deretorno	1

Variables locales. Ejemplo E1

Mostrar

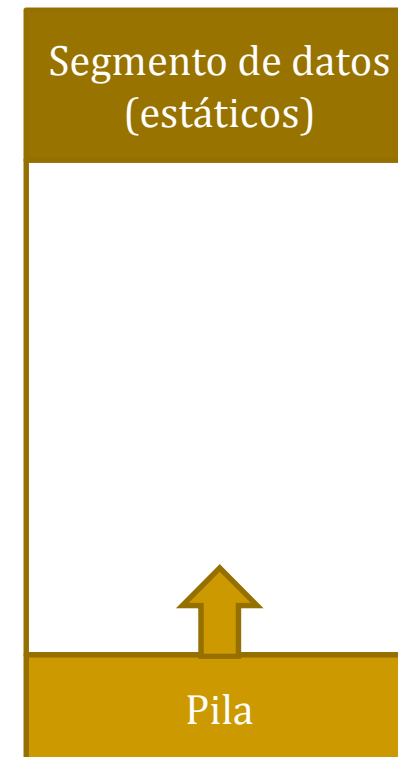
- Direcciones de todas las variables del programa en el momento de entrar en la función ejemplo
- Escribir el programa equivalente en MAPL

```
void main() {  
    char c;  
    int a;  
    int b;  
  
    a = 37;  
    read b;  
}
```

Parámetros

La forma más habitual de ubicar los parámetros son:

- Registros
- Memoria
 - Segmento de Datos
 - Segmento de Pila



Parámetros en Segmento de Datos

Parámetros en segmento de datos

```
int a;  
int b;  
real c;
```

```
void f(real par1) {  
    int local_f1;  
    char local_f2;  
    g(2222);  
}
```

```
void g(int par2) {  
    real local_g1;  
    char local_g2;  
}
```

```
void main() {  
    int local_main;  
    f(10);  
}
```

Memoria Estática			
globales	00	00	a (sin inicializar)
	01	00	
	02	00	b (sin inicializar)
	03	00	
	04	00	c (sin inicializar)
	05	00	
f	06	00	par1 (sin inicializar)
	07	00	
	08	00	
	09	00	
	10	00	local_f1 (sin inicializar)
	11	00	
g	12	00	local_f2 (sin inicializar)
	13	00	par2 (sin inicializar)
	14	00	
	15	00	local_g1 (sin inicializar)
	16	00	
	17	00	
main	18	00	local_g2 (sin inicializar)
	19	00	
	20	00	local_main (sin inicializar)
	21	00	
	22	00	
	23	00	

```
int a;  
int b;  
real c;
```

```
real par1  
int local_f1;  
char local_f2;
```

```
int par2;  
real local_g1;  
char local_g2;
```

```
int local_main;
```

```
void f() {  
    par2 = 2222;  
    g();  
}
```

```
void g() {  
}
```

```
void main() {  
    par1 = 10;  
    f();  
}
```


Parámetros en Segmento de Pila

<pre> void f(int param1, real param2) { print param1; 0002 pusha BP 0003 push 8 0004 add 0005 load 0006 out } 0007 ret 0, 0, 6 void main() { f(10, 1.62); 0008 push 10 0009 pushf 1,62 → 0010 call f [0002] } 0011 ret 0, 0, 0 </pre>		
1009	00	
1010	00	
1011	00	
1012	00	
1013	00	
SP	41	
1015	92	1,62
1016	207	pushf
1017	63	
1018	10	10 push
1019	00	
BP	00	BP anterior 1024
1021	04	
1022	01	Dirección deretorno 1
1023	00	

<pre> void f(int param1, real param2) { print param1; → 0002 pusha BP 0003 push 8 0004 add 0005 load 0006 out } 0007 ret 0, 0, 6 void main() { f(10, 1.62); 0008 push 10 0009 pushf 1,62 0010 call f [0002] } 0011 ret 0, 0, 0 </pre>		
1009	00	
SP BP	252	BP anterior 1020
1011	03	
1012	11	Retorno a 'main' 11
1013	00	
1014	41	
1015	92	param2
1016	207	(BP+4) 1,62
1017	63	
1018	10	param1 (BP+8) 10
1019	00	
1020	00	BP anterior 1024
1021	04	
1022	01	Dirección deretorno 1
1023	00	

$$dir(param_i) = BP + 4 + \sum_{j=i+1}^n tamaño(param_j)$$

¿Cómo imprimiría *f* su segundo parámetro?

print param2;

Parámetros. Liberación

¿Cómo se borran los parámetros?

- ¿Cómo debería quedar la pila?

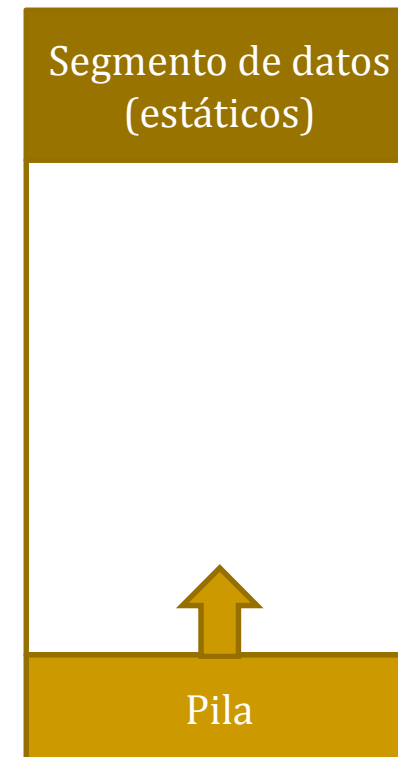
void f(int param1, real param2) {			1009	00	
print param1;					
0002	pusha BP		SP BP	252	
0003	push 8		1011	03	BP anterior 1020
0004	add				
0005	load		1012	11	
0006	out		1013	00	Retorno a 'main' 11
}					
→ 0007	ret 0, 0, 6		1014	41	
void main() {			1015	92	param2
f(10, 1.62);			1016	207	(BP+4) 1,62
0008	push 10		1017	63	
0009	pushf 1,62		1018	10	param1 (BP+8) 10
0010	call f [0002]		1019	00	
}			1020	00	BP anterior 1024
0011	ret 0, 0, 0		1021	04	
			1022	01	Dirección deretorno 1
			1023	00	

void f(int param1, real param2) {			1009	00	
print param1;					
0002	pusha BP		1010	252	
0003	push 8		1011	03	
0004	add		1012	11	
0005	load		1013	00	
0006	out		1014	41	
}					
0007	ret 0, 0, 6		1015	92	
void main() {			1016	207	
f(10, 1.62);			1017	63	
0008	push 10		1018	10	
0009	pushf 1,62		1019	00	
0010	call f [0002]		SP BP	00	BP anterior 1024
}			1021	04	
→ 0011	ret 0, 0, 0		1022	01	Dirección deretorno 1
			1023	00	

Valores de Retorno de las funciones

Representación de valores de retorno

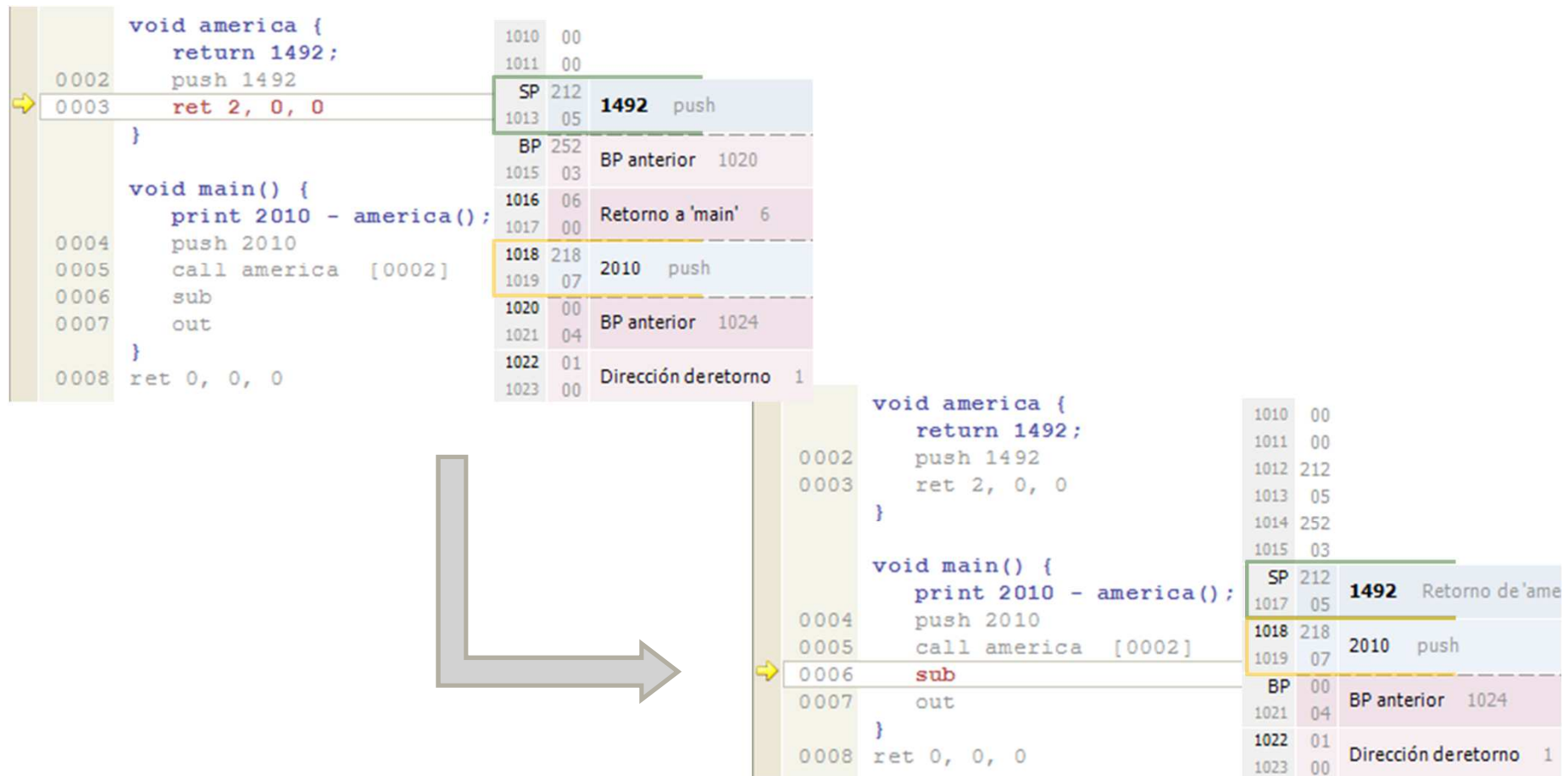
- Lo más habitual
 - Registros
 - Memoria
 - Segmento de Datos
 - Segmento de Pila



Valores de Retorno en la Pila

Valores de retorno en la pila

- Se dejan encima del stack frame
- ¿Cómo debería quedar la pila al salir de una función?



Instrucción *ret*. Resumen

Formato:

ret <cte1>, <cte2>, <cte3> (*ret* \Rightarrow *ret* 0, 0, 0)

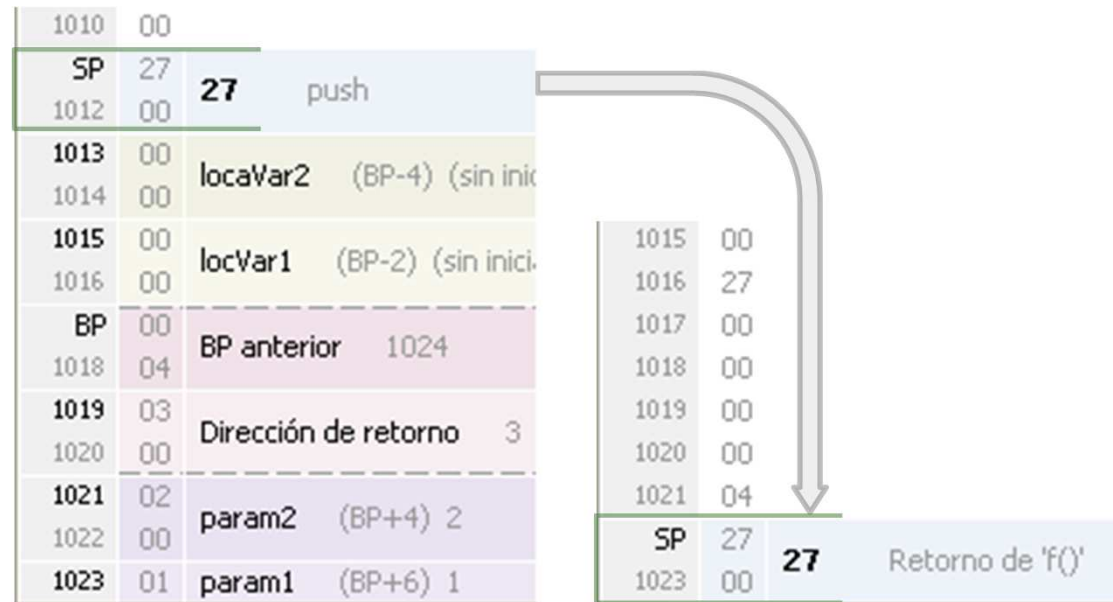
- cte1 = tamaño del valor de retorno (0 si no hay)
- cte2 = tamaño de las variables locales de la función
- cte3 = tamaño de los parámetros de la función

```
int f(byte param1, int param2)
{
    int locVar1, locVar2;

    return 27;
}
```

↓
Código MAPL generado

f: enter 4
push 27
→ **ret 2, 4, 3**



Ejercicio E2

Mostrar

- Direcciones de todas las variables del programa cuando se esté dentro de la función *ejemplo*
- Escribir el programa equivalente en MAPL

```
int a;  
float b;  
char c;
```

```
int ejemplo(float par1, int par2) {  
    int local1;  
    float local2;  
  
    b = 3.4;  
    par1 = 3.4;  
    local2 = 3.4;  
  
    return par2 + a + local1;  
}
```

```
void main() {  
    print a + ejemplo(1.5, 27);  
}
```

Representación de Datos por Tipo

Representación de datos por tipo

El tipo de un dato indica CÓMO se representa

- Su tamaño
- Su representación interna

Tipos Simples

En esta fase hay que determinar cómo se representa

- Cada tipo simple del alto nivel
 - char
 - enteros
 - reales
 - boolean

Estructuras

Estructuras

- Determinar
 - Alineamiento
 - Orden de los campos

```
struct S {  
    char c1;  
    int i1;  
    char c2;  
    int i2;  
}
```

- Influyen en
 - Cálculo del tamaño
 - Direcciones de un campo de una variable

Estructuras. Ejemplo E3

Mostrar

- Direcciones de todas las variables del programa
- Escribir el programa equivalente en MAPL

```
struct Persona {  
    float sueldo;  
    int edad;  
    char inicial;  
}
```

dir $[[s.c]]$ = dirección de s + desplazamiento de c

```
byte a;  
struct Persona carlos;  
struct Persona javi;
```

```
void main()  
{  
    carlos.edad = 20;  
    javi.edad = 20;  
}
```

Vectores (arrays)

Vectores (arrays)

- Determinar
 - Orden de las dimensiones
 - Por filas o por columnas
 - Alineamiento
- Influyen en
 - Cálculo del tamaño
 - Direcciones de cada elemento
 - Para una dimensión
$$\text{dir}(v[i]) = \text{dir}(v) + (i * \text{tamaño}(v[0]))$$

int v[3][3];

v	0	1	2	3
0	v[0][0]	v[0][1]	v[0][2]	v[0][3]
1	v[1][0]	v[1][1]	v[1][2]	v[1][3]
2	v[2][0]	v[2][1]	v[2][2]	v[2][3]

v[0][0]	v[0][0]
v[0][1]	v[1][0]
v[0][2]	v[2][0]
v[0][3]	v[0][1]
v[1][0]	v[1][1]
v[1][1]	v[2][1]
v[1][2]	v[0][2]
v[1][3]	v[1][2]
v[2][0]	v[1][2]
v[2][1]	v[0][3]
v[2][2]	v[1][3]
v[2][3]	v[2][3]

Ejemplo E4

Mostrar

- Direcciones de todas las variables del programa
- Escribir el programa equivalente en MAPL

```
char b;  
int i;  
int v[10];
```

```
void main()  
{  
    i = 7;  
    v[i] = 1111;  
}
```

$$\text{dir}(v[i]) = \text{dir}(v) + (i * \text{tamaño}(v[0]))$$

Paso por Referencia

Paso por Referencia (I). Argumentos

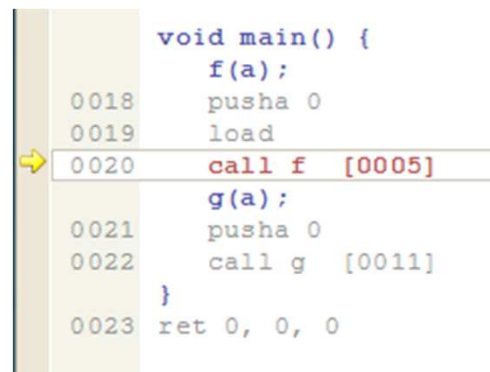
Ejemplo

```
int a = 10;
```

```
void f(int param_f) {  
    print param_f;  
}
```

```
void g(ref int param_g) {  
    print param_g;  
}
```

```
void main() {  
    f(a);  
    g(a);  
}
```



```
void main() {  
    f(a);  
0018    pusha 0  
0019    load  
→ 0020    call f    [0005]  
    g(a);  
0021    pusha 0  
0022    call g    [0011]  
    }  
0023    ret 0, 0, 0
```

Paso por Referencia (II). Parámetros

Ejemplo

```
int a = 10;
```

```
void f(int param_f) {  
    print param_f;  
}
```

```
0005    pusha BP  
0006    push 4  
0007    add  
0008    load  
0009    out  
}
```

1010	04	
1011	00	
SP	10	10 valor de param_f
1013	00	
BP	252	
1015	03	BP anterior 1020
1016	21	Retorno a 'main' 21
1017	00	
1018	10	param_f (BP+4) 10
1019	00	
1020	00	
1021	04	BP anterior 1024
1022	04	Dirección deretorno 4
1023	00	

```
void g(ref int param_g) {  
    print param_g;  
}
```

```
0011    pusha BP  
0012    push 4  
0013    add  
0014    load  
0015    load  
0016    out  
}
```

1010	04	
1011	00	
SP	00	0 &a valor de param_g
1013	00	
BP	252	
1015	03	BP anterior 1020
1016	23	Retorno a 'main' 23
1017	00	
1018	00	param_g (BP+4) 0 &a
1019	00	
1020	00	
1021	04	BP anterior 1024
1022	04	Dirección deretorno 4
1023	00	

```
void main() {  
    f(a);  
    g(a);  
}
```

Tiempo de Ubicación

Ubicación Estática

Hasta ahora hemos determinado para cada dato

- **Dónde** se ubica
 - Segmento de datos o pila
- **Cómo** se ubica
 - Estructura interna del dato

Falta el **cuándo se ubica el dato en memoria**

Hasta ahora se ha hablado de datos estáticos

- ¿Cuándo se ubican éstos en memoria?

```
int v[i+2];
```

¿Qué ocurre si no se sabe hasta tiempo de ejecución?

- Nodos de un AST
 - ¿Cómo se haría?

Ubicación Dinámica

Memoria dinámica

- ¿Qué permite un lenguaje con memoria dinámica?
 - Ofrece primitivas de gestión de la memoria
 - Explícitas o implícitas
- ¿Qué tipo de datos requiere?

Implementación

- Ubicación
- Componentes
 - Bloque de memoria
 - Registro de asignaciones

Código máquina
(solo lectura)

Variables globales y
literales

Memoria asignada en
tiempo de ejecución

Stack frames, locales,
parámetros y operandos

Segmento de código

Segmento de datos
(estáticos)

Memoria dinámica
(heap)



Pila

Resumen

Gestión de Memoria

- Qué entra
 - Un AST con un programa válido

- Qué hace
 - Calcula las direcciones de todos los símbolos del programa en función de su ámbito y su tipo

- Qué sale
 - Un AST con:
 - una propiedad *dirección* en las definiciones de variables
 - una propiedad *tamaño* en los tipos

Tarea Obligatoria

Antes de la siguiente clase de *prácticas*:

- 1) Bajar MAPL del campus
- 2) Leer “*Manual MAPL.pdf*”
 - ☐ Se debería haber hecho ya
- 3) Seguir todos los ejemplos de la carpeta:
“*Tutorial\2 Gestión de Memoria*”

OBLIGATORIO

Ejercicio E5

Mostrar

- Direcciones de todas las variables del programa
- Escribir el programa equivalente en MAPL

```
struct Circulo {  
    x:int;  
    y:int;  
    radio:real;  
}  
  
real v[4];  
int i;  
Circulo diana;  
  
void main() {  
    i = 0;  
    diana.y = 7;  
    diana.x = diana.y + i;  
  
    v[0] = 1.618033;  
    diana.radio = v[i] + 2.0;  
    print v[i + 2];  
    print v[diana.y - 7];  
}
```

Soluciones

Solución E1

```
void main() {
    char c;
    real a;
    int b;

    a = 37;
    read b;
}
```

1010	00	
1011	00	
1012	00	
SP	00	b (BP-7) (sin inicializar)
1014	00	
1015	00	
1016	00	a
1017	00	(BP-5) (sin inicializar)
1018	00	
1019	00	c (BP-1) (sin inicializar)
BP	00	BP anterior 1024
1021	04	
1022	01	Dirección deretorno 1
1023	00	

main:

enter 7

push BP

push -5

add

pushf 37

storef

push BP

push -7

add

in

store

ret 0, 7, 0

1005	00	
1006	00	
SP	00	
1008	00	37
1009	20	pushf
1010	66	
1011	247	1015 &a BP + -5
1012	03	
1013	00	b (BP-7) (sin inicializar)
1014	00	
1015	00	
1016	00	a
1017	00	(BP-5) (sin inicializar)
1018	00	
1019	00	c (BP-1) (sin inicializar)
BP	00	BP anterior 1024
1021	04	
1022	01	Dirección deretorno 1
1023	00	

Solución E2

```

int a;
float b;
char c;

int ejemplo(float par1, int par2) {
    int local1;
    float local2;
    0002    enter 6

    b = 3.4;
    0003    pusha 2
    0004    pushf 3,4
    0005    storef
    par1 = 3.4;
    0006    pusha BP
    0007    push 6
    0008    add
    0009    pushf 3,4
    0010    storef
    local2 = 3.4;
    0011    pusha BP
    0012    push -6
    0013    add
    0014    pushf 3,4
    0015    storef

    0016    pusha BP
    0017    push 4
    0018    add
    0019    load
    0020    pusha 0
    0021    load
    0022    add
    0023    pusha BP
    0024    push -2
    0025    add
    0026    load
    0027    add
    }
    0028    ret 2, 6, 6

void main() {
    print a + ejemplo(1.5, 27);
    0029    pusha 0
    0030    load
    0031    pushf 1,5
    0032    push 27
    0033    call ejemplo [0002]
    0034    add
    0035    out
    }
    0036    ret 0, 0, 0

```

Memoria Estática			
00	00	a	(sin inicializar)
01	00		
02	00	b	(sin inicializar)
03	00		
04	00		
05	00		
06	00	c	(sin inicializar)
07	00		
08	00		
09	00		
999	00		
1000	00		
1001	00		
SP	00		
1003	00	local2	(BP-6) (sin inicializar)
1004	00		
1005	00		
1006	00	local1	(BP-2) (sin inicializar)
1007	00		
BP	252		
1009	03	BP anterior	1020
1010	34	Retorno a 'main'	34
1011	00		
1012	27	par2	(BP+4) 27
1013	00		
1014	00	par1	(BP+6) 1,5
1015	00		
1016	192		
1017	63		
1018	00	0	valor de a
1019	00		
1020	00	BP anterior	1024
1021	04		
1022	01	Dirección de retorno	1
1023	00		

Solución E3

```

struct Persona {
    float sueldo;
    int edad;
    char inicial;
}

byte a;
struct Persona carlos;
struct Persona javi;

void main()
{
    carlos.edad = 20;
0002    pusha 1
0003    push 4
0004    add
0005    push 20
→ 0006    store
        javi.edad = 20;
0007    pusha 8
0008    push 4
0009    add
0010    push 20
0011    store
    }
0012    ret 0, 0, 0

```

Memoria Estática		
00	00	a (sin inicializar)
01	00	carlos.sueldo (sin inicializar)
02	00	
03	00	
04	00	
05	00	carlos.edad (sin inicializar)
06	00	
07	00	carlos.inicial (sin inicializar)
08	00	javi.sueldo (sin inicializar)
09	00	
10	00	
11	00	
12	00	javi.edad (sin inicializar)
13	00	
14	00	javi.inicial (sin inicializar)
15	00	
16	00	
17	00	

1013	00	
1014	00	
1015	00	
SP	20	20 push
1017	00	
1018	05	5 &carlos.edad 1 + 4
1019	00	
BP	00	BP anterior 1024
1021	04	
1022	01	Dirección deretorno 1
1023	00	

Solución E4

```

char b;
int i;
int v[10];

void main()
{
    i = 7;
    0002    pusha 1
    0003    push 7
    0004    store
           v[i] = 1111;
    0005    pusha 3
    0006    pusha 1
    0007    load
    0008    push 2
    0009    mul
    0010    add
    0011    push 1111
    0012    store
    }
    0013    ret 0, 0, 0

```

Memoria Estática			
00	00	b	(sin inicializar)
01	07	i	7
02	00		
03	00	v[0]	(sin inicializar)
04	00		
05	00	v[1]	(sin inicializar)
06	00		
07	00	v[2]	(sin inicializar)
08	00		
09	00	v[3]	(sin inicializar)
10	00		
11	00	v[4]	(sin inicializar)
12	00		
13	00	v[5]	(sin inicializar)
14	00		
15	00	v[6]	(sin inicializar)
16	00		
17	00	v[7]	(sin inicializar)
18	00		
19	00	v[8]	(sin inicializar)
20	00		
21	00	v[9]	(sin inicializar)
22	00		
23	00		
24	00		

1013	00		
1014	02		
1015	00		
SP	87		
1017	04	1111	push
1018	17		
1019	00	17 &v[7]	3 + i * 2
BP	00		
1021	04	BP anterior	1024
1022	01		
1023	00	Dirección deretorno	1

Solución E5

Memoria Estática		
00	00	
01	00	v[0]
02	00	(sin inicializar)
03	00	
04	00	
05	00	v[1]
06	00	(sin inicializar)
07	00	
08	00	
09	00	v[2]
10	00	(sin inicializar)
11	00	
12	00	
13	00	v[3]
14	00	(sin inicializar)
15	00	
16	00	
17	00	i (sin inicializar)
18	00	diana.x (sin inicializar)
19	00	
20	00	diana.y (sin inicializar)
21	00	
22	00	
23	00	diana.radio
24	00	(sin inicializar)
25	00	
26	00	
27	00	
28	00	

```

struct Circulo {
    x:int;
    y:int;
    radio:real;
}

```

```

real v[4];
int i;
Circulo diana;

```

```

main() {
    i = 0;
    0002 pusha 16
    0003 push 0
    0004 store
    diana.y = 7;
    0005 pusha 18
    0006 push 2
    0007 add
    0008 push 7
    0009 store
    diana.x = diana.y + i;

```

```

    0010 pusha 18
    0011 push 0
    0012 add
    0013 pusha 18
    0014 push 2
    0015 add
    0016 load
    0017 pusha 16
    0018 load
    0019 add
    0020 store

```

```

    v[0] = 1.618033;
    0021 pusha 0
    0022 push 0
    0023 push 4
    0024 mul
    0025 add
    0026 pushf 1,618033
    0027 storef

```

```

    diana.radio = v[i] + 2.0;
    0028 pusha 18
    0029 push 4
    0030 add
    0031 pusha 0
    0032 pusha 16
    0033 load
    0034 push 4
    0035 mul
    0036 add
    0037 loadf
    0038 pushf 2
    0039 addf
    0040 storef
    print v[i + 2];
    0041 pusha 0
    0042 pusha 16
    0043 load
    0044 push 2
    0045 add
    0046 push 4
    0047 mul
    0048 add
    0049 loadf
    0050 outf
    print v[diana.y - 7];
    0051 pusha 0
    0052 pusha 18
    0053 push 2
    0054 add
    0055 load
    0056 push 7
    0057 sub
    0058 push 4
    0059 mul
    0060 add
    0061 loadf
    0062 outf
    }
    0063 ret 0, 0, 0

```