

Tema 8: Representación de información en memoria

1. La tabla de símbolos (TDS)
 1. *Requisitos de la TDS*
 2. *Implementación de la TDS*
 3. *La TDS en un lenguaje con estructura de bloques*
2. Gestión de memoria en tiempo de ejecución
 1. *Conceptos básicos*
 2. *Asignación estática*
 3. *Gestión del montículo (heap)*
 4. *Gestión de la pila (stack) . Registros de activación*
3. Ejemplo de asignación de memoria

1. La tabla de símbolos (TDS)

La Tabla de Símbolos

- La TDS es una estructura de datos que usa el PdL para almacenar información sobre los **símbolos** que aparecen en el **lenguaje fuente**.
- Durante las distintas **fases** de un compilador se **almacena** o **lee** información de la TDS.
- La información que **contiene** depende del propósito del PdL.
- Ej. Compilador: nombre de variable, tipo, posición de memoria,...

1.1. Requisitos de la TDS

- Operaciones básicas: Inserción, búsqueda y borrado.
- Ejemplo de información a incluir:

Tdsímbolos	Una entrada por cada objeto definido por el usuario.
Nombre	Lexema
Objeto	Categoría del objeto: <i>variable, parámetro, función...</i>
Tipo	Tipo del objeto: <i>tinteger, tarray, trecord, tvacio o terror...</i>
Des	Desplazamiento relativo en el segmento de datos
Niv	Nivel de anidamiento del objeto.
ref	Referencia a la tabla auxiliar: TdVectores , TdRegistros TdDominios...

TdVectores	Una entrada para cada dimensión del array, y el tipo de los elementos del array . Cada entrada contiene el número de elementos de la dimensión (o el límite inferior (min) y superior (max) según el lenguaje).
-------------------	---

TdRegistros	Una entrada por cada campo de los registros.
--------------------	--

Nombre	Lexema
---------------	--------

Tipo	Tipo del campo.
-------------	-----------------

Des	Desplazamiento relativo del campo en el segmento correspondiente.
------------	---

TdDominios	Una entrada por cada dominio definido en los bloques.
-------------------	---

Tratamiento de palabras reservadas:

- Por el **analizador léxico**: Un token específico para cada palabra reservada
- Como identificadores y **tabla de palabras reservadas** para diferenciar si se trata de una palabra reservada o de un identificados

Almacenamiento de los lexemas:

- Tabla de lexemas:
 - Mejor aprovechamiento de memoria
 - Elimina límite en la longitud de los lexemas

1.2. Implementación de la TDS

- Array de registros
- Listas enlazada ordenadas. Listas ordenadas doblemente enlazadas
- Árboles equilibrados ordenados
- Tablas de dispersión (hash)

1.3. TDS en lenguaje con estructura de bloques (LEB)

```
int  a, b;
void function f1 (int p1, p2){
    int  c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```


Problemas a resolver:

- Control del alcance de cada declaración
- Varios objetos con el mismo nombre accesibles

Posibles soluciones:

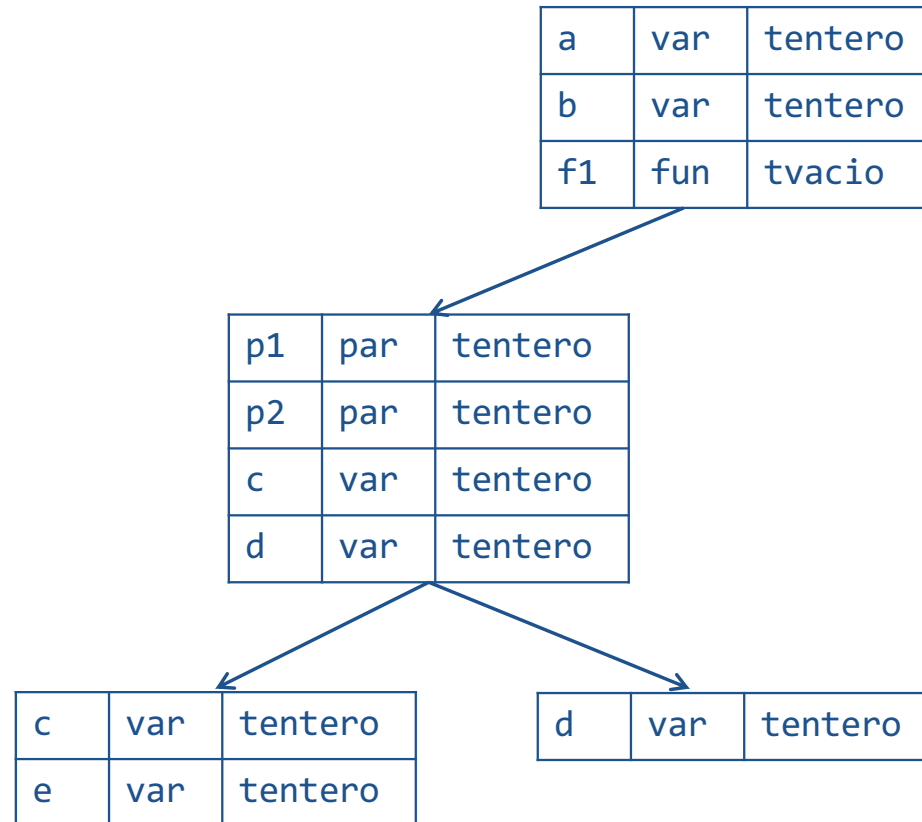
- a) Una subtabla para cada ámbito de declaraciones. Solución adecuada para compiladores de varias pasadas.
- b) Gestión como pila apoyada en una tabla de ámbitos. Solución no adecuada para compiladores de varias pasadas.

a) Gestión con una subtabla de símbolos por bloque

- Para cada *bloque* con declaraciones (por cada **ámbito** de declaraciones) se prepara una **subtabla** de símbolos.
- Cada subtabla de símbolos tendrá una entrada para cada símbolo **declarado en el bloque**.
- Las subtabla estarán **enlazadas** para representar el **anidamiento** de bloques.
- La **búsqueda** de símbolos en la TDS comienza por la subtabla del **bloque más interior** y continuará por la subtabla en la que ésta está definida.

Ejemplo

```
int  a, b;  
void function f1 (int p1, p2){  
    int  c, d ;  
    if (p1 != 0) {  
        int c, e ;  
        ...  
    }  
    if (p2 > 3) {  
        int d ;  
        ...  
    }  
}
```



b) Gestión LIFO de TDS en LEB

- Al comenzar a compilar cada bloque iniciamos un nuevo ámbito. Se irán insertando en la TDS sus objetos locales.
- Al finalizar la compilación de un bloque se sacarán de la TDS todos sus objetos locales.
- Con esta gestión se garantiza que al compilar un bloque están en la TDS todos los objetos accesibles desde él, y solo los objetos accesibles desde él.

La Tabla de Ámbitos (TdA)

- Para hacer más eficiente la operación de desapilar todos los objetos locales a un bloque, podemos apoyarnos en una Tabla de Ámbitos (TdA).
- TdA: Estructura con una entrada por cada nivel de anidamiento de los bloques que aparecen en el programa fuente.
- TdA[n] apuntará al primer objeto definido en un bloque de nivel de anidamiento n insertado en la TDS.

Para sacar (desapilar) de la TDS objetos de nivel de anidamiento n: $\text{Top_TDS} = \text{TdA}[n]$

Ejemplo

c	var	tipo	2
e	var	tipo	2
p5	par	tipo	2
p4	par	tipo	2
p3	par	tipo	2
f2	fun	tipo	1
d	var	tipo	1
c	var	tipo	1
p2	par	tipo	1
p1	par	tipo	1
f1	fun	tvacio	0
b	var	tipo	0
a	var	tipo	0

TdA

	2
	1
	0

```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo

p7	par	tipo	2
p6	par	tipo	2
f3	fun	tvació	1
f2	fun	tipo	1
d	var	tipo	1
c	var	tipo	1
p2	par	tipo	1
p1	par	tipo	1
f1	fun	tvació	0
b	var	tipo	0
a	var	tipo	0

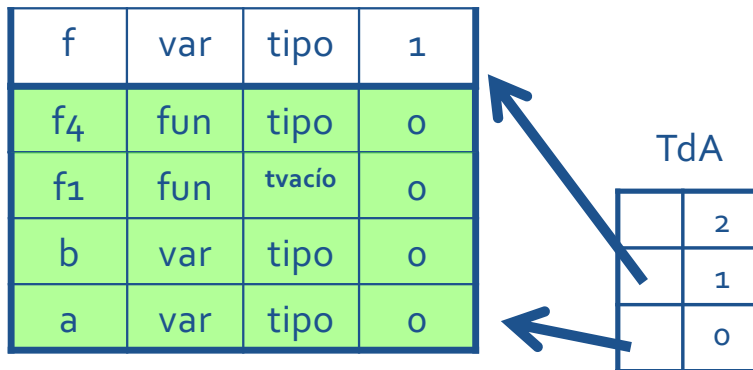
TdA

	2
	1
	0

```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo

```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```



2. Gestión de memoria en tiempo de ejecución

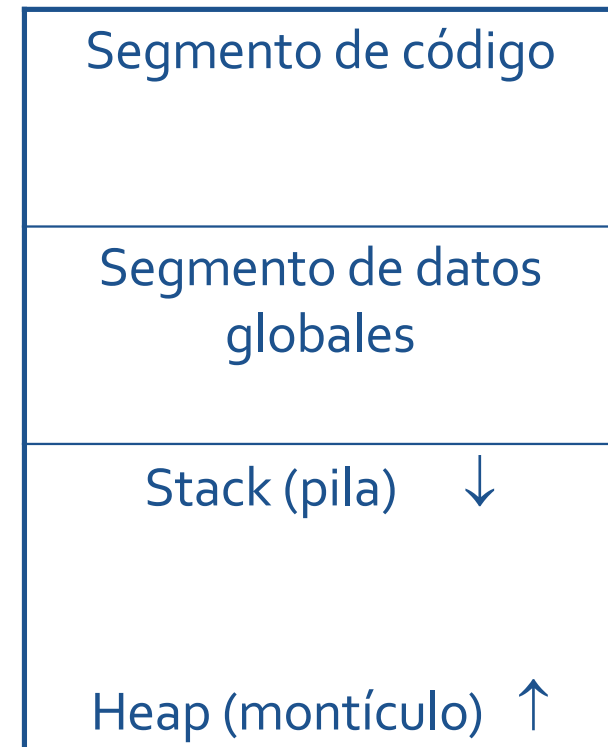
2.1. Conceptos básicos

Asignación dinámica:

- La localización del objeto solo se conocerá en tiempo de ejecución.

Asignación estática:

- La localización del objeto se conoce en tiempo de compilación.
- Requiere:
 - Conocer en tiempo de compilación el tamaño del objeto.
 - Conocer en tiempo de compilación el número de instancias simultaneas del objeto en ejecución.



2.1. Conceptos básicos

Ejemplo de esquema de memoria de programa en C

- **Segmento de código (o segmento de "texto"):**
Contiene la instrucciones ejecutables
- **Segmento de datos inicializados (o segmento de datos):**
Contienen variables globales y estáticas inicializadas por el programador.
Puede dividirse en zona de solo lectura (para constantes) y de lectura-escritura para el resto.
- **Segmento de datos sin inicializar (segmento .bss)**
- **Stack o pila de programa.**
La cima es apuntada por el stack pointer.
- **Heap:**
Usado para asignar memoria dinámica (malloc,...)

Posiciones altas



2.2. Asignación estática

A los objetos se les asigna una dirección absoluta que se mantiene durante la ejecución del programa.

Ejemplo:

- Variable global
 - Variables locales estáticas (que mantienen su valor entre llamadas a la función),
 - Cadenas de caracteres,...
-
- Si el lenguaje no dispone de recursión, se puede emplear asignación estática para las variables locales.

Ej. Fortran 90.

Ejemplo de asignación estática

$P \rightarrow \{ \text{NIVEL} = 0 ; \text{DESP} = 0 ; \}$

L_Decla

$L_Decla \rightarrow Decla \mid L_Decla \quad Decla$

$Decla \rightarrow DV$

***** Declaración de variables *****

$DV \rightarrow T \text{ id} ; \quad \{ \text{InsertarTds} (id.nom, \text{"variable"}, T.tipo, \text{NIVEL}, \text{DESP}) ;$

$\text{DESP} := \text{DESP} + T.talla ; \}$

$T \rightarrow \text{int} \quad \{ T.tipo = Tentero ; \quad T.talla = TALLA_ENTERO ; \}$

$\mid \text{float} \quad \{ T.tipo = Treal ; \quad T.talla = TALLA_REAL ; \}$

$\mid \text{bool} \quad \{ T.tipo = Tlogico ; \quad T.talla = TALLA_LOGICO ; \}$

$\mid \text{struct } \{ C \} \quad \{ T.tipo := testructura (C.tipo) ; T.talla = C.talla \}$

$C \rightarrow T \text{ id} \quad \{ C.tipo := (id.nom \times T.tipo) ; \quad C.talla = T.talla ; \}$

$\mid C_1 ; T \text{ id} \quad \{ C.tipo := C_1.tipo \times (id.nom \times T.tipo) ; \quad C.talla = C_1.talla + T.talla ; \}$

2.3. Gestión el montículo (heap)

Montículo: Región de memoria en la que los subbloques pueden ser asignados y liberados en cualquier orden.

- Se debe usar siempre que un objeto pueda cambiar de tamaño.
- Liberación de bloques:
 - **Explícita:** Indicada por el programador.
 - **Implícita:** El bloque asignado a un objeto debe liberarse automáticamente cuando se detecte que no se va a usar más el objeto. Requiere mecanismo recolector de basura (*garbage collector*) en tiempo de ejecución.
- Asignación de bloques: Problemas de fragmentación
 - **Fragmentación interna:**

El algoritmo de asignación asigna un bloque mayor del requerido para almacenar el objeto.
 - **Fragmentación externa:**

Los bloques asignados se van dispersando: Puede haber espacio disponible pero repartido en trozos tan pequeños que puede no ser suficiente para almacenar un objeto entero.

Asignación de memoria del montículo

1. Usando **lista de bloques** de memoria libres. Inicialmente hay un único bloque (todo el montículo).
 2. Ante petición de memoria para objeto de tamaño t :
 - Algoritmo **first-fit**: Asigna primer bloque de la lista de tamaño $\geq t$
 - Algoritmo **best-fit**: Asigna bloque más pequeño de la lista de tamaño $\geq t$
 3. El bloque asignado se **divide en dos** para asignar solo un tamaño t . El resto se mete en la lista de bloques libres.
 4. Al liberar un bloque se comprueba si puede **fusionarse** con los bloques adyacentes (si están libres).
- Mejora: Mantener **varias listas** de bloques libres en función del tamaño de éstos.
 - Para *eliminar la fragmentación externa*: **Compactar** (moviendo bloques asignados)

2.4. Gestión de la pila

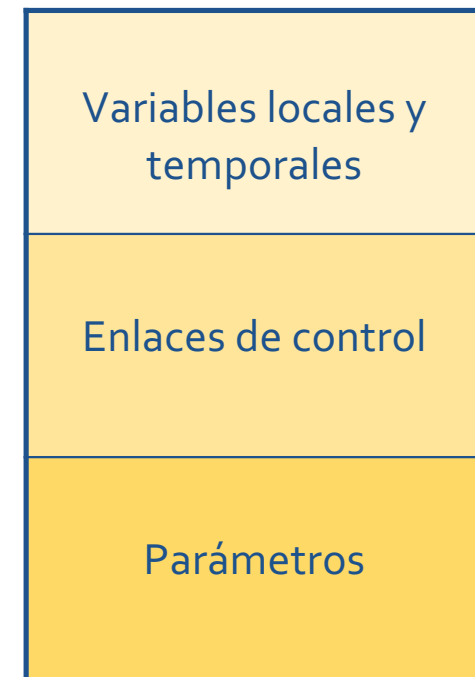
Los bloques de memoria se asignan y liberan en orden **LIFO**

- Llamamos **activación** de un bloque a cada una de sus ejecuciones.
- **Tiempo de vida** de una activación es la **secuencia de pasos** entre el primer y último paso de la ejecución.
 - Los tiempos de vida de dos activaciones o no se solapan, o uno incluye completamente al otro.
- **Árbol de activación**: Representación del tiempo de vida de las activaciones de un programa:
 - Cada **nodo** representa una activación.
 - A es **padre de** B si B se activa durante el tiempo de vida de A
 - A está **a la izquierda de** B si el tiempo de vida de A es anterior a B (A finaliza antes de comenzar B)
- El **flujo de control** del programa coincide con un recorrido en profundidad del árbol de activación

Registro de activación

- Espacio de la pila de ejecución asignado a una activación para almacenar sus datos locales.
- En algunos lenguajes se llama **marco** (*frame*) de pila.
- **Display**: Bloque de punteros con una entrada por cada nivel de anidamiento de los objetos, donde **Display[n]** apunta al último RA con objetos de nivel de anidamiento n cargado. En algunos lenguajes es un solo registro

Estructura general



Ejemplo de RA y acceso (ej. Pascal)

Acceso a Vble. local (niv , desp) :

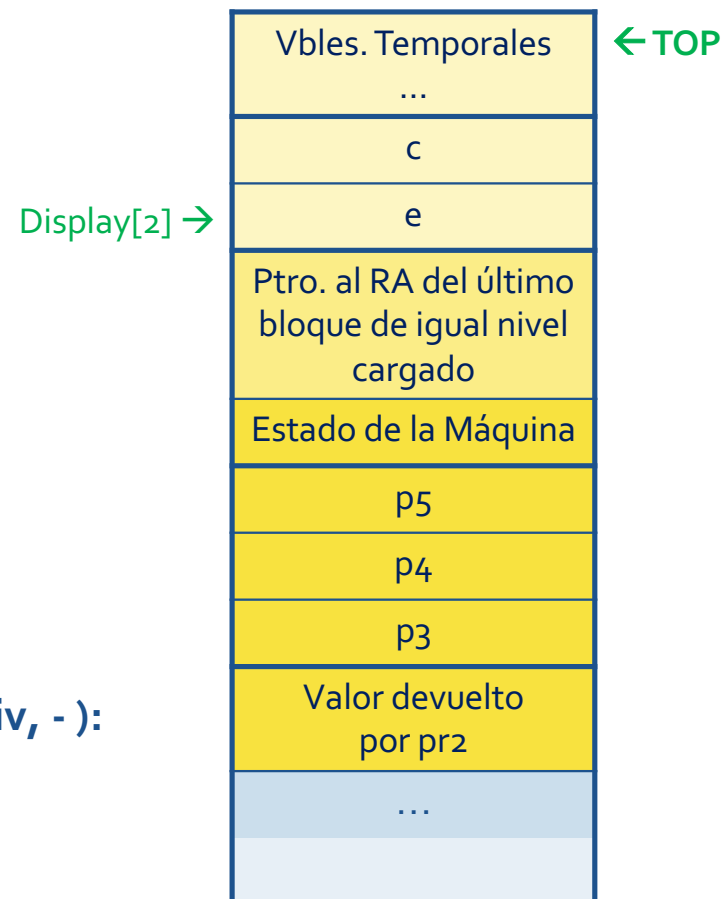
vble.pos := display[niv] + desp

Acceso a Parámetro (niv , desp) :

param.pos := display[niv] + desp
- (TallaTotalParametros + TallaEnlaces
+ TallaEstadoMaquina)

Depositar valor devuelto por una función de nombre (niv , -):

display[niv+1]
- (TallaTotalParametros + TallaEnlaces
+ TallaEstadoMaquina + TallaValorDevuelto)



Ejemplo de Frame y acceso en C

- A los registros de activación se les suele denominar *Frames*
- No hay anidamiento de funciones: Todos los objetos serán globales o locales.

Display solo necesita 1 nivel -> *FramePointer* (FP)

Nivel 0 = Global

Nivel 1 = Local (apuntado por FramePointer)

Ejemplo de Frame y acceso en C

Acceso a Vble. local (desp) :

$\text{vble.pos} := \text{frame_pointer} + \text{desp}$

Acceso a Parámetro (desp) :

(apilados en orden inverso a declaración)

$\text{param.pos} := \text{frame_pointer}$
 $- (\text{desp} + \text{TallaEnlaces} + \text{TallaEstadoMaquina}$
 $+ \text{TallaDelParametro})$

Depositar valor devuelto por una función de nombre (niv, -):

$\text{frame_pointer} - (\text{TallaParámetros} + \text{TallaEnlaces}$
 $+ \text{TallaEstadoMaquina} + \text{TallaValorDevuelto})$

Frame Pointer →

Vbles. Temporales ...
c
e
Ptro. al RA del último bloque de igual nivel cargado
Estado de la Máquina
p3
p4
p5
Valor devuelto por pr2
...

← Stack
pointer

Secuencia de carga RA

Bloque llamador

1. Evaluar parámetros actuales
2. Si la función devuelve un valor: Crear variable temporal para el valor devuelto
3. Apilar parámetros actuales
4. Apilar Estado de la Máquina (incluida la dirección de retorno) y saltar al código del bloque llamado (CALL)

Bloque llamado

1. Apilar enlace de control (Apilar Display[n])
2. Actualizar display (display[n] = TOP)
3. Reservar espacio para área de datos locales

Secuencia de descarga RA

Bloque llamado


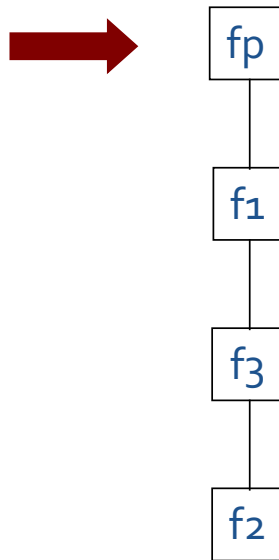
1. Desapilar área de datos (TOP= Display[n])
2. Desapilar enlace de control y restaurar valor de display (display[n]= POP)
3. Desapilar Estado de la máquina, incluida la dirección de retorno y saltar a ella (RETURN)

Bloque llamador

1. Desapilar parámetros actuales
2. Si la función devuelve un valor: Desapilar valor de retorno

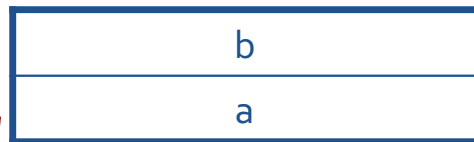
Ejemplo (I)

El árbol de activación es:



```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
    int function f4 {
        int f ;
        ... }
void function principal {
    f1(1,2) ; }
```

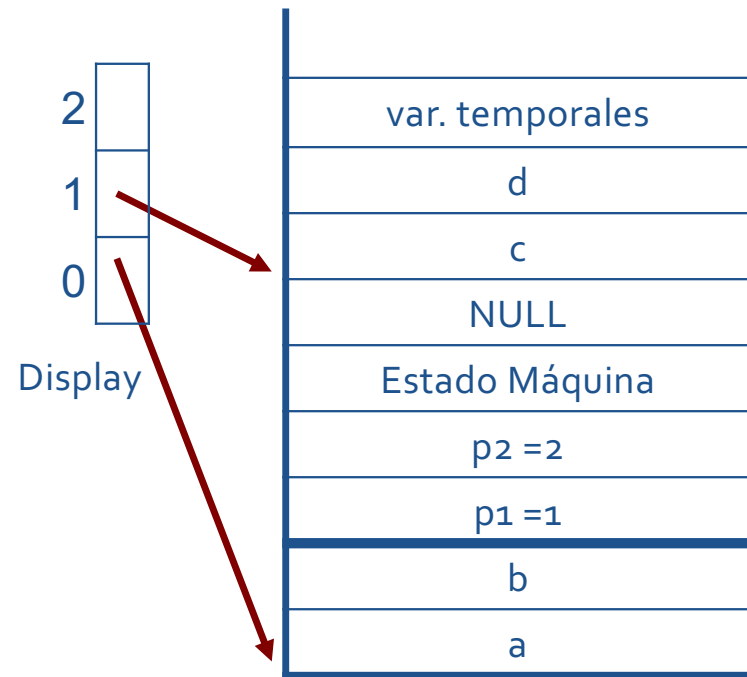
Ejemplo (II)



Memoria global

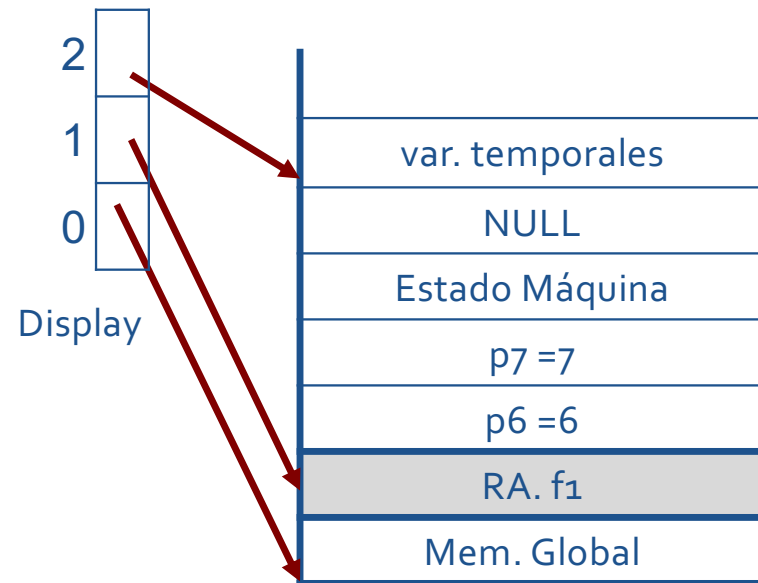
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (III)



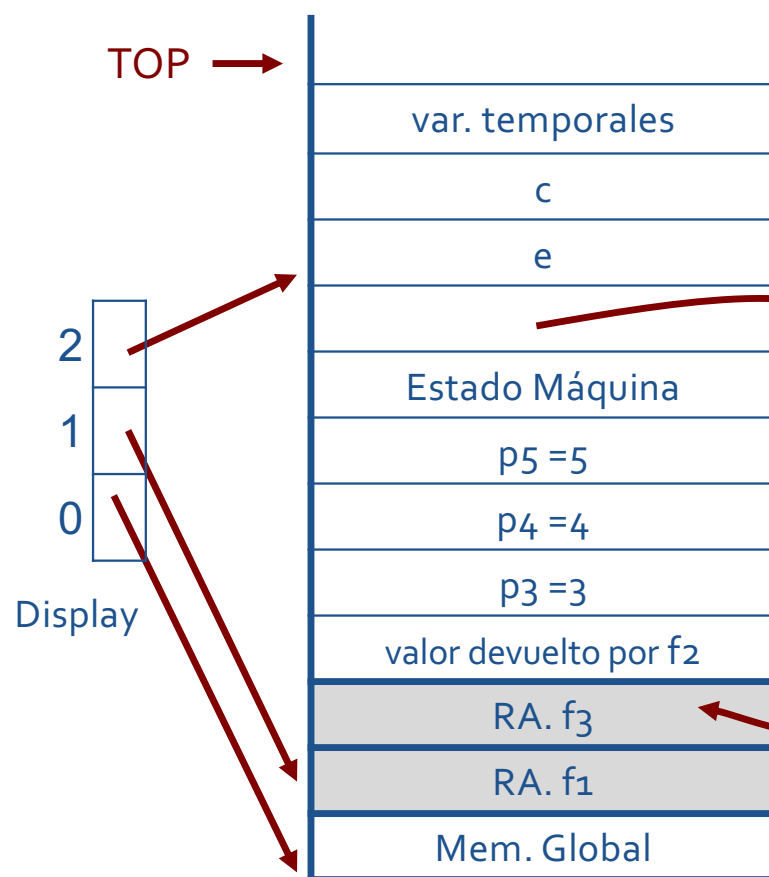
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```


Ejemplo (IV)



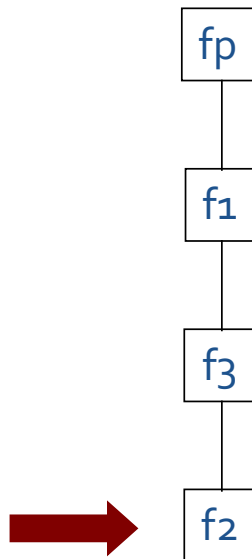
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (V)



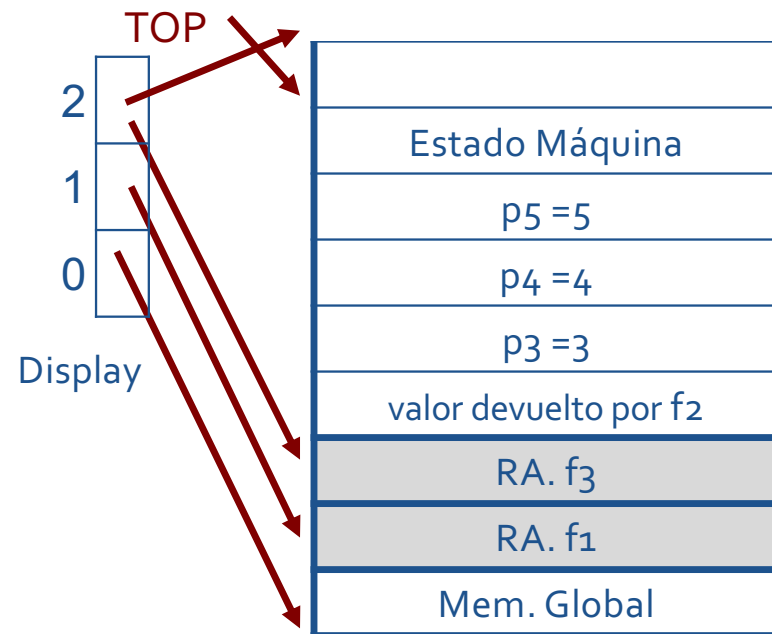
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (VI)



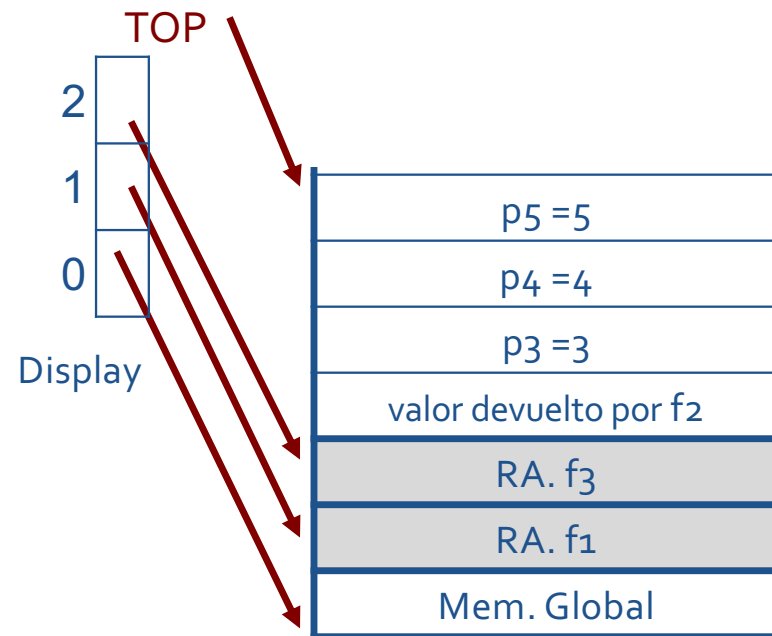
```
int  a, b;
void function f1 (int p1, p2){
    int  c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
    int function f4 {
        int f ;
        ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (VII)



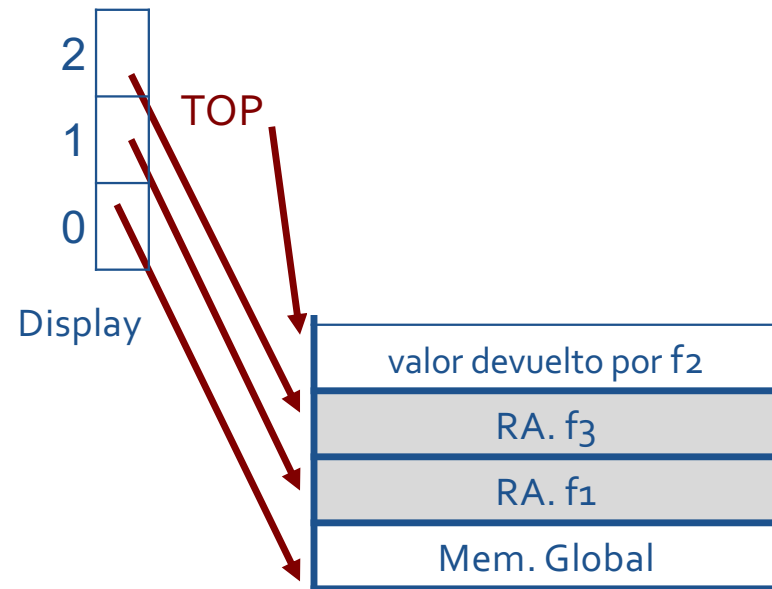
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (VIII)



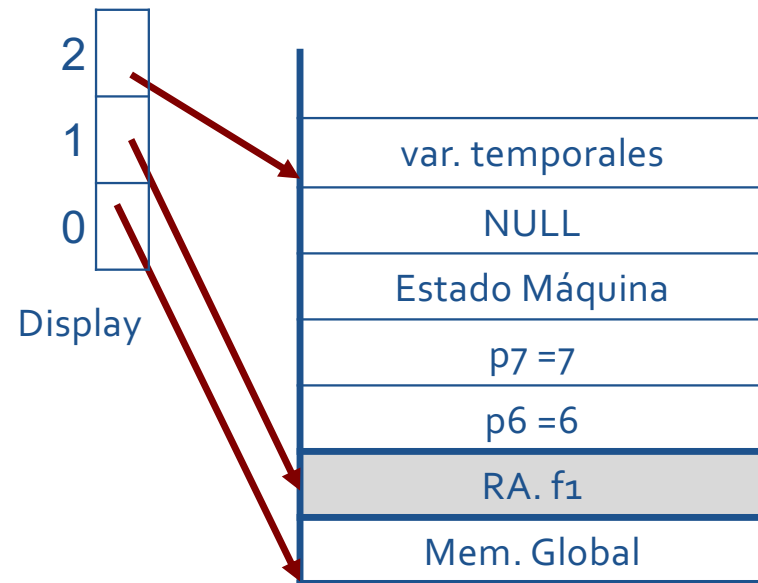
```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
    int function f4 {
        int f ;
        ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (IX)



```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo (X)



```
int a, b;
void function f1 (int p1, p2){
    int c, d ;
    int function f2 (int p3, p4, p5){
        int e, c ;
        ... }
    void function f3 (int p6, p7){
        f2(3,4,5) ; }
    f3(6,7); ... }
int function f4 {
    int f ;
    ... }
void function principal {
    f1(1,2) ; }
```

Ejemplo de asignación de memoria

$P \rightarrow L_Decla$

$L_Decla \rightarrow Decla$

| L_Decla $Decla$

$Decla \rightarrow DV$

***** Declaración de variables *****

$DV \rightarrow T \text{ id } ;$

| $T \rightarrow \text{int}$

| float

| bool

| $\text{struct } \{ C \}$

***** Miembros de estructuras *****

$C \rightarrow T \text{ id}$

| $C_1 ; T \text{ id}$

3. Asignación de memoria

Ejemplo de asignación de memoria (1/4)

$P \rightarrow L_Decla$

$L_Decla \rightarrow Decla \mid L_Decla \ Decla$

$Decla \rightarrow DV \mid DF$

***** Declaración de variables *****

$DV \rightarrow T \ id \ ; \mid T \ id \ LI \ ;$

$T \rightarrow int$

$\mid float$

$\mid bool$

$\mid struct \ \{ C \}$

***** Miembros de estructuras *****

$C \rightarrow T \ id$

$\mid C_1 \ ; T \ id$

***** Índices de declaración de array *****

$L_I \rightarrow [cte] \mid L_I_1 [cte]$

***** Declaración de función *****

$DF \rightarrow Cab_F \ Bloque$

$Cab_F \rightarrow T \ id \ (P_F)$

$Bloque \rightarrow \{ DVL \ L_Inst \}$

$DVL \rightarrow DVL \ DV \mid \epsilon$

***** Parámetros formales *****

$P_F \rightarrow L_PF \mid \epsilon$

$L_PF \rightarrow T \ id \mid T \ id, \ L_PF_1$

Ejemplo de asignación de memoria (2/4)

$P \rightarrow \{ \text{NIVEL} = 0 ; \text{DESP} = 0 ; \}$

L_Decla

$L_Decla \rightarrow Decla \mid L_Decla \quad Decla$

$Decla \rightarrow DV \mid DF$

/ ** Declaración de variables ** */*

$DV \rightarrow T \text{ id } ; \quad \{ \text{InsertarTds}(\text{id.nom}, \text{"variable"}, T.tipo, \text{NIVEL}, \text{DESP});$
 $\quad \text{DESP} := \text{DESP} + T.talla ; \}$

$DV \rightarrow T \text{ id } L_l ; \{ \text{InsertarTds}(\text{id.nom}, \text{"variable"}, \text{vector}(T.tipo), \text{NIVEL}, \text{DESP});$
 $\quad \text{DESP} := \text{DESP} + (L_l.talla * T.talla) ; \}$

$T \rightarrow \text{int} \quad \{ T.tipo = Tentero; T.talla = TALLA_ENTERO; \}$
 $\mid \text{float} \quad \{ T.tipo = Treal; T.talla = TALLA_REAL; \}$
 $\mid \text{bool} \quad \{ T.tipo = Tlogico; T.talla = TALLA_LOGICO; \}$
 $\mid \text{struct } \{ C \} \quad \{ T.tipo := testructura(C.tipo); T.talla = C.talla \}$

Ejemplo de asignación de memoria (3/4)

***** Índices de declaración de array *****

$L_l \rightarrow [cte] \quad \{L.l.talla := cte.lexval;\}$
 $| L_l [cte] \quad \{L.l.talla := cte.lexval * L_l.talla;\}$

$C \rightarrow T \text{ id} \quad \{C.tipo := (id.nom \times T.tipo); \quad C.talla = T.talla\}$
 $| C_1 ; T \text{ id} \quad \{C.tipo := C_1.tipo \times (id.nom \times T.tipo); \quad C.talla = C_1.talla + T.talla\}$

Ejemplo de asignación de memoria (4/4)

***** Declaración de función *****

DF → T id (P_F) { DVL L_Inst }	{ NIVEL++ ; CreaAmbito() ; DF.old_desp = DESP ; DESP := 0 ; } { InsertarTds (id.nom, "funcion", P_F.tipo -> T.tipo, NIVEL-1 , 0) ; } { EliminaAmbito(); NIVEL-- ; DESP :=DF.old_desp; }
DVL → DVL DV ε	

***** Parámetros formales *****

P_F → L_PF	{ L_PF.tallapar := TALLA_EC + TALLA_EM ; } { P_F.tipo := L_PF.tipo }
ε	{ P_F.tipo := tvacio }
L_PF → T id	{ InsertarTds (id.nom, "parametro", T.tipo, NIVEL , - (L_PF.tallapar + T.talla)) ; L_PF.tipo := T.tipo ; }
T id , L_PF ₁	{ L_PF ₁ .tallapar := L_PF.tallapar + T.talla ; InsertarTds (id.nom, "parametro", T.tipo, NIVEL , - L_PF ₁ .tallapar) } { L_PF.tipo := L_PF ₁ .tipo x T.tipo }

Ejercicio 1

```
int x= 0 ;
float f1 (int p1, float p2) {
    int i=1 ; ← TDS, PILA
    if (p1 > p2) return f1(p1-i, p2);
    else return p1*p2;
}
int main () {
    int x=1, i=1 ; float y = 1.0;
    { int i=2; ← TDS, PC
      x= x * i;
    }
    x = f1(x, y);
}
```

Suponiendo que la talla de los enteros es 2, de los reales y enlaces de control 4, y del “estado de la máquina” 10:

- Muestra el contenido de la TDS en los puntos marcados como TDS
- Indica el desplazamiento relativo de i y x en el punto de control PC.
- Muestra el contenido de la pila de ejecución cada vez que se pasa por el punto PILA.

Ejercicio 2

```
float prod( float a, float b) {  
    return a*b }  
  
float sum (int a, int b) {  
    float t  
    if (a <= 5) { t= prod(a,b) ; return t ; }      /* ← TDS    */  
    else {                                          /* ← PILA    */  
        return suma(a-1, b-1);}                  /* ← PC1     */  
  
int main () {  
    int a; int b;  
    a = 7;                                         /* ← PC2     */  
    b = 9;                                         /* ← TDS     */  
    printf( "Resultado %f\n", sum(a, b)); }  
}
```

- a) Muestra contenido de TDS en puntos TDS (supón que la talla de enteros es 2 y de reales 4)
- b) Indica el desplazamiento relativo de a y b en los puntos de control: PC1 y PC2.
- c) Muestra contenido de la pila de ejecución cada vez que se pasa por el punto PILA.