

PROGRAMACIÓN PROCEDURAL

Tipos de Datos

Unidad 2 Segunda Parte



Tipos de Datos Estructurados

Un Objeto de datos estructurado o estructura de datos está constituido por un agregado de otros objetos de datos llamados **componentes**.

Definidos por el programador

Registros, Arreglos, etc.

Definidos por el sistema durante la ejecución del programa

La pila que usa el sistema para trabajar con funciones

Tipos de Datos Estructurados

Especificación

- **Número de componentes**

Estructuras de tamaño fijo: número de componentes invariable durante su tiempo de vida (Arreglos y Registros).

Estructura de tamaño variable: número de componentes cambia durante su tiempo de vida . Usan generalmente un apuntador para vincular componentes (Listas, Archivos).

- **Tipo de componentes**

Estructuras homogéneas: componentes del mismo tipo (Arreglos).

Estructuras heterogéneas: componentes de distinto tipo (Registros).

Tipos de Datos Estructurados

Especificación

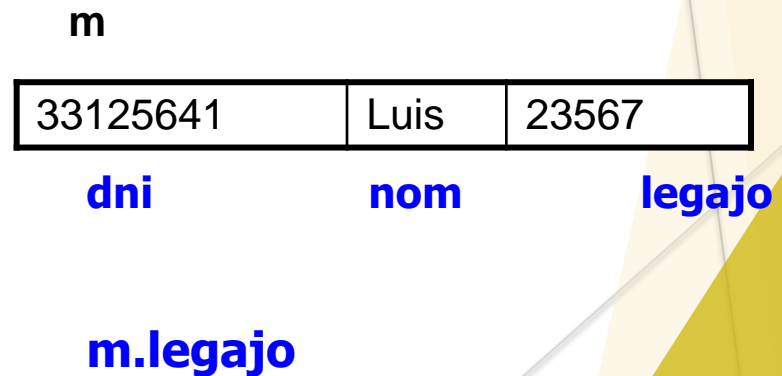
- Mecanismo de selección de una componente

```
int Arre[20];
```

Arre[i] selección de la *i*ésima componente

```
struct alumno  
{ int dni;  
  char nom[20];  
  int legajo;  
};
```

```
struct alumno m;
```



Tipos de Datos Estructurados

Especificación

- **Organización de las componentes**

Unidimensional o según una serie lineal (arreglo unidimensional, registro, cadena de caracteres, listas.)

Multidimensional (arreglo multidimensional, registro cuyos componentes son registros, listas cuyos componentes son listas, etc.)

Tipos de Datos Estructurados

Operaciones sobre estructuras de datos

1- Operaciones particulares sobre las componentes de una estructura de datos (selección directa o secuencial)

```
int Arre[20];
```

Acceso a una componente

Acceso a cada una de ellas

2 - Operaciones sobre una estructura de datos completa

conjunto limitado de operaciones, dependiendo del lenguaje.

```
int Arre1[20], Arre2[20];
```

```
Arre2 = Arre1; ??
```

```
struct alumno
```

```
{ int dni;
```

```
  char nom[20];
```

```
  int legajo;
```

```
};
```

```
alumno m1, m2
```

```
m1 = m2 ??
```


Tipos de Datos Estructurados

Representación de almacenamiento

a. Representación Secuencial: usa un solo bloque de memoria contigua, incluye tanto componentes como descriptor.

Operación de Selección directa de un componente

Dirección de la componente = Dirección base + Desplazamiento



Dirección inicial del
bloque completo



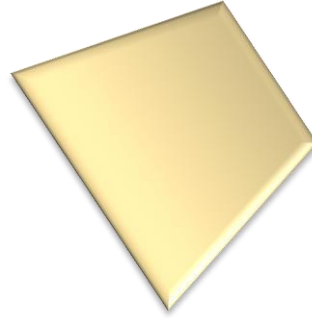
Dirección relativa de
**componente dentro del
bloque** secuencial

b. Representación Vinculada: usa varios bloques de memoria no contiguos vinculados a través de punteros.

Operación de Selección de un componente

Sigue una cadena de apuntadores desde el primer bloque de almacenamiento de la estructura hasta la componente deseada.

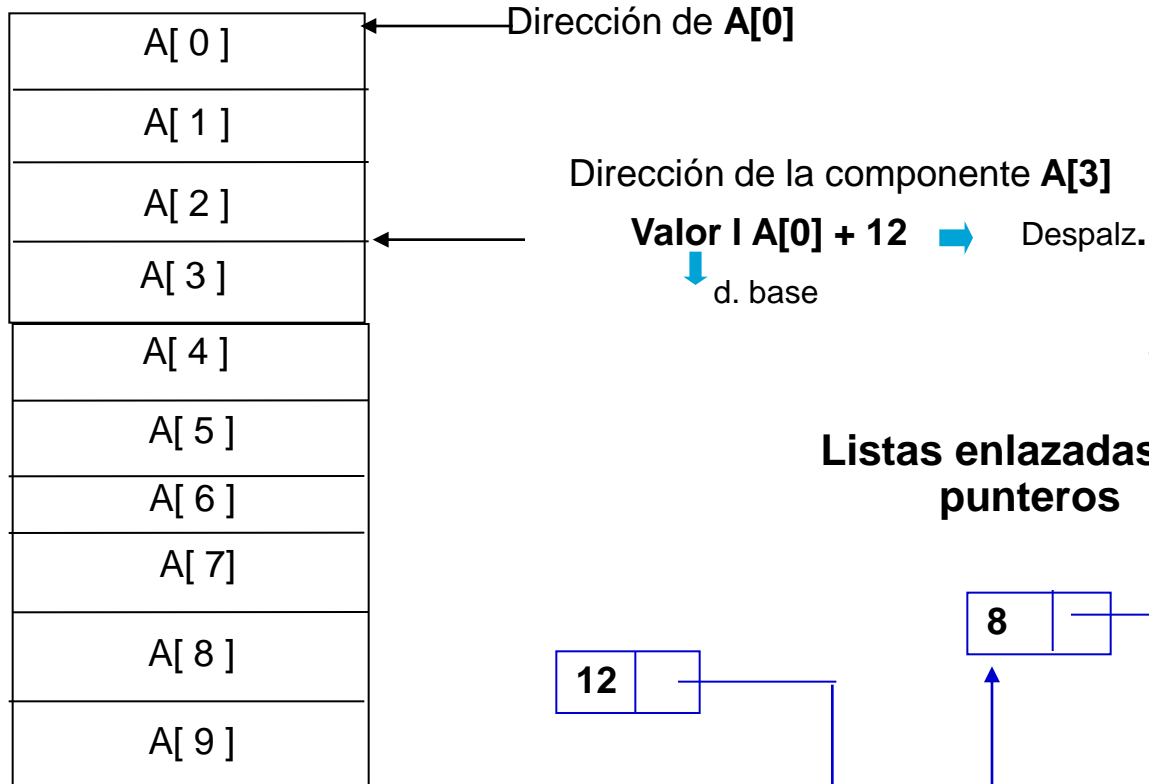
Distintos tipos de representaciones



Secuencial

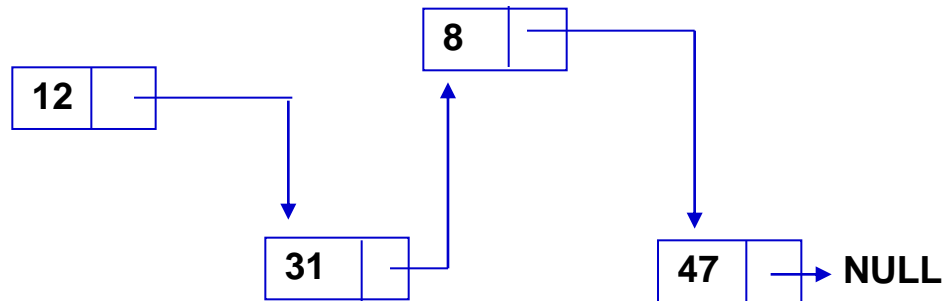
Arreglos unidimensionales

desplazamiento



Vinculada

Listas enlazadas por punteros



Tipos de Datos Estructurados

Arreglos Unidimensionales - Especificación

Estructura homogénea de tamaño fijo organizada como una serie lineal simple

Atributos

- ✓ Número de componentes
- ✓ Tipo de dato
- ✓ Subíndice para seleccionar una componente

Operaciones sobre arreglos unidimensionales

Selección de una componente: **subindización**

En C:

```
int arre[10];
```

```
arre[2]=16;
```

La selección de `arre[2]` incluye dos pasos

- Operación de **Referenciamiento**
- Operación de **Selección**

En Pascal:

```
Var arre: array [-5..5] of integer;
```

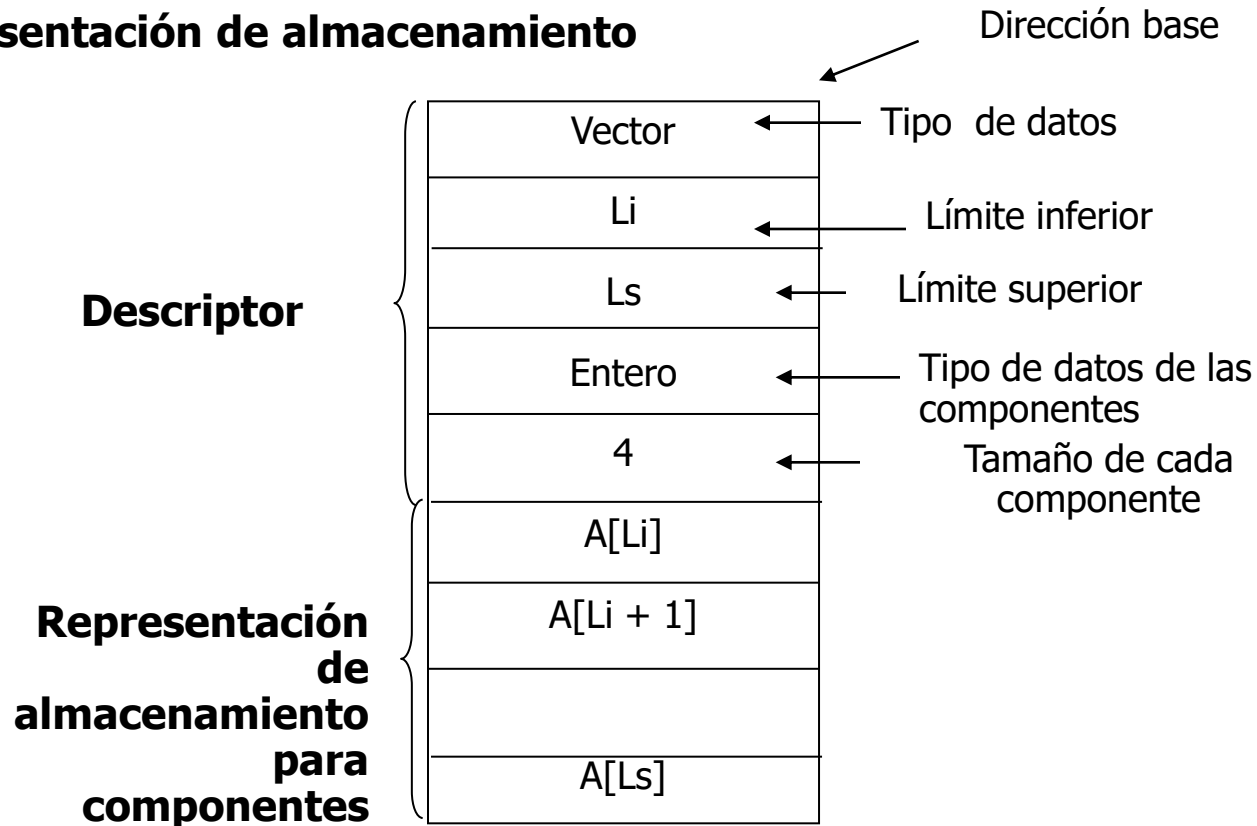
```
arre[-3]=16;
```

Existen otras operaciones sobre las componentes de los arreglos se verán en cursos superiores

Tipos de Datos Estructurados

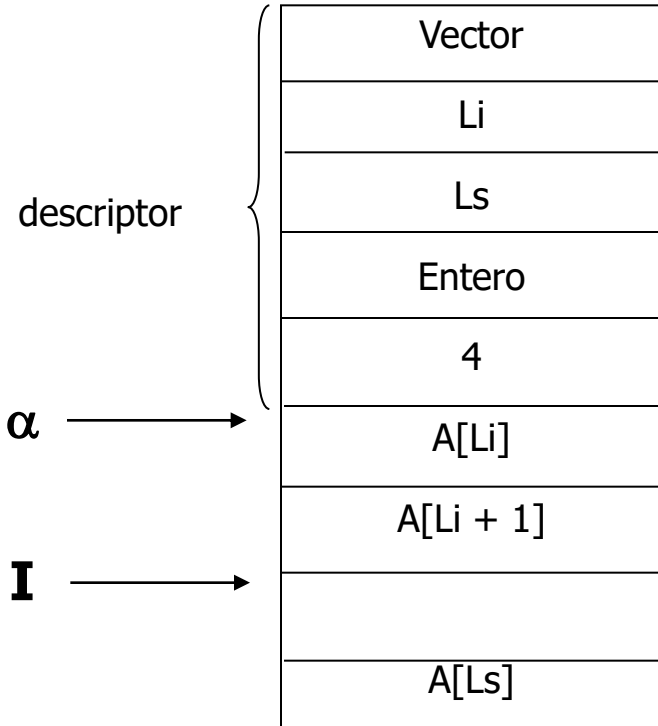
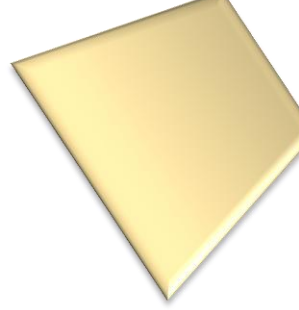
Arreglos Unidimensionales

Representación de almacenamiento



- Representación secuencial de almacenamiento.
- Se puede incluir un descriptor con algunos o todos los atributos, esta información no se conoce hasta el tiempo de ejecución.

Cálculo de la dirección de la componente A[I]



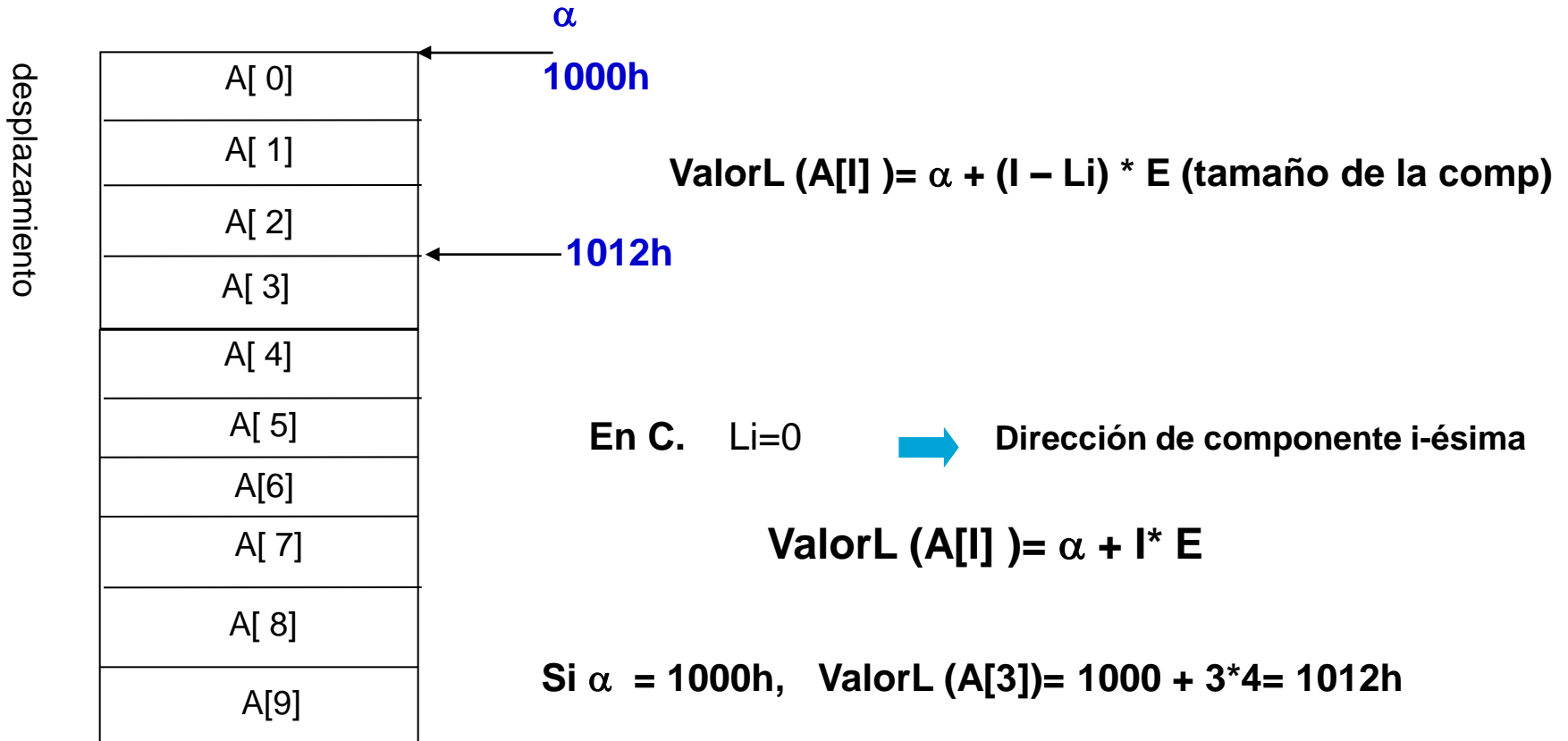
$$\text{Dir}(A[I]) = \text{ValorL}(A[I])$$

$$\text{ValorL}(A[I]) = \alpha + (I - \text{Li}) * \text{Tam. componente}$$

$$\text{ValorL}(A[I]) = \alpha + (I - \text{Li}) * E$$

C no necesita descriptor: información conocida en tiempo de compilación

Cálculo de la dirección de la componente A[I] en C



Valor L de componentes cte., puede calcularse en tiempo de compilación implica acceso rápido

Arreglos Bidimensionales (Matriz)

Especificación

En este caso para cada dimensión debe especificarse un intervalo de subíndices.

| | 0 | 1 | 2 | | ... | | 5 |
|---|---|---|---|--|-----|--|---|
| 0 | 5 | 3 | 4 | | | | 2 |
| 1 | 1 | 6 | | | | | 8 |
| | | | | | | | |
| | | | | | | | |
| 8 | 3 | | | | | | 5 |
| 9 | 2 | | | | | | 6 |

En C

int A[10][6];

arreglo bidimensional de 10 filas y 6 columnas.

En Pascal

Var A: array[3..12, -5..5] of integer;

arreglo bidimensional de 10 filas y 10 columnas

Arreglos Bidimensionales (Matriz)

Ejemplo: Arreglo bidimensional para almacenar total de votos escrutados para 10 partidos políticos en 30 distritos.

Partidos



Distritos



| | 0 | 1 | 2 | | ... | | 29 |
|---|-----|-----|-----|--|-----|--|-----|
| 0 | 100 | 300 | 345 | | | | 25 |
| 1 | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| 8 | 811 | | | | | | 125 |
| 9 | 230 | | | | | | 63 |

int T[10,30]

T[0]



T[1]



T[8]



T[9]



| | 0 | 1 | 2 | | ... | | 29 |
|---|-----|-----|-----|--|-----|--|-----|
| 0 | 100 | 300 | 345 | | | | 25 |
| 1 | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| 8 | 811 | | | | | | 125 |
| 9 | 230 | | | | 14 | | 63 |

Arreglos Bidimensionales - Matriz

Representación

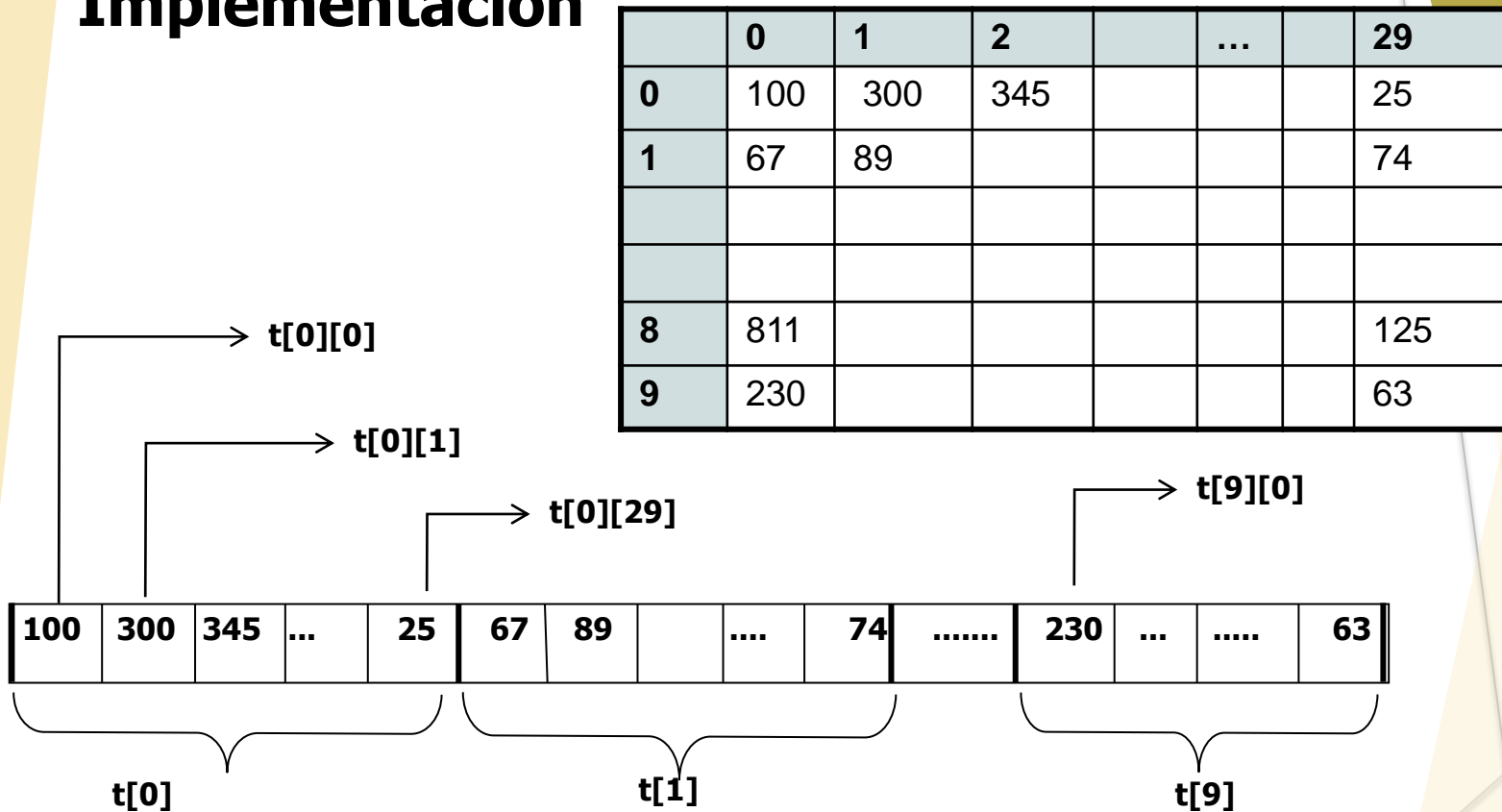
En general, un arreglo de cualquier dimensión tiene representación en orden **por filas**.

- Esta representación implica que el arreglo se divide en subarreglos para cada elemento del intervalo del primer subíndice. Cada uno de estos subarreglos es un arreglo.
- Para el caso de un arreglo bidimensional `int t[10][30]`, el arreglo es un arreglo que tiene 10 subarreglos de 30 componentes cada uno de ellos.

Arreglo Bidimensional es un arreglo de arreglos.

Arreglos Bidimensionales - Matriz

Implementación



Tamaño del arreglo en memoria: número de elementos por tamaño de cada uno de ellos.

En `int t[10][30];` el tamaño es
bytes

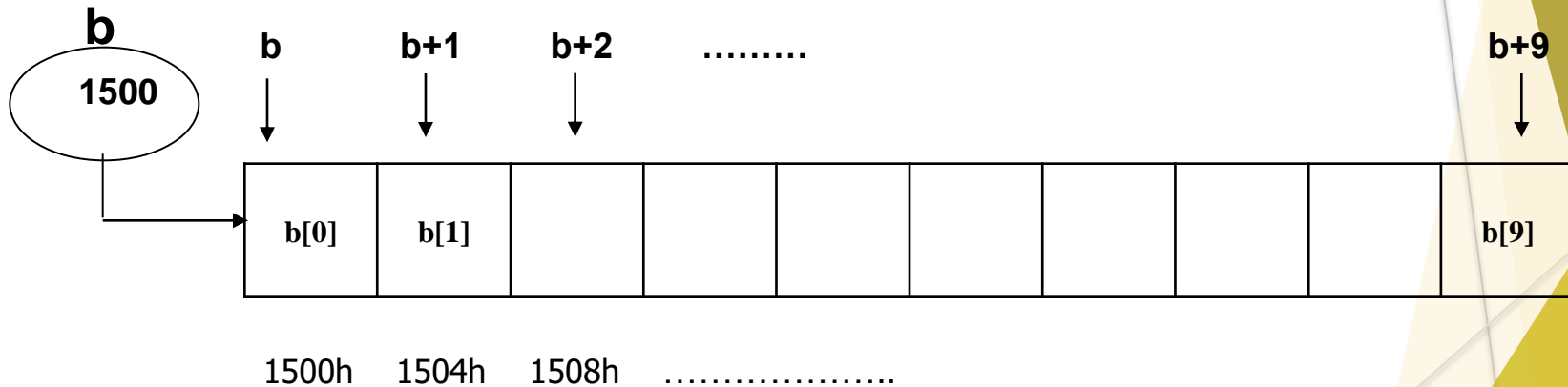
4 byte * 300 elementos = 1200

Arreglos y Punteros en lenguaje C

```
int b[10];
```



1500h 1504h 1508h



b ⇔ **&b[0]**
b+1 ⇔ **&b[1]**
***(b+2)** ⇔ **???**

Arreglos y Punteros en lenguaje C

Existe una estrecha relación entre indexación en un arreglo y aritmética de punteros.

Acceso a la dirección de un elemento de un arreglo:

- 1) Nombre del elemento precedido por un **&**.
&b[i].
- 2) Nombre del arreglo más una cantidad entera **i**.
b+i.

Acceso a un elemento o contenido de esas direcciones

b[i] o ***(b+i)**

Por lo tanto, las siguientes expresiones son equivalentes:

$$b+i \iff \&b[i]$$

$$*(b+i) \iff b[i]$$

Arreglos y Punteros en lenguaje C

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
const int N=5;
```

```
void carga (int x[], int lim)
```

```
{ int i;
```

```
    printf("\nCarga el arreglo con %d elementos\n ", lim );
```

```
    for(i=0; i < lim; i++)
```

```
        scanf("%d",&x[i]); // equivalente scanf("%d",x+i);
```

```
}
```

```
void lista(int x[], int lim)
```

```
{int i;
```

```
    printf("\n Componente Valor Dirección \n" );
```

```
    for(i=0; i<N;i++)
```

```
        printf("\n x[%2d] %10d %10x ", i , x[i], x+i);
```

```
}
```

```
int main(void)
```

```
{ int a[N],i;
```

```
    carga(a, N);
```

```
    lista (a, N);
```

```
    for(i=0; i<N; i++)
```

```
        *(a+i)=*(a+i)*3;
```

```
    lista (a, N);
```

```
    printf("\n Valor del puntero %0x ",a );
```

```
    printf("\n La direccion del primer elemento del arreglo es %0 x", &a[0] );
```

```
    getch();
```

```
}
```

```
Componente Valor Direccion
```

```
x[ 0]          3      28ff20
```

```
x[ 1]          4      28ff24
```

```
x[ 2]         -6      28ff28
```

```
x[ 3]          8      28ff2c
```

```
x[ 4]         12      28ff30
```

```
Componente Valor Direccion
```

```
x[ 0]          9      28ff20
```

```
x[ 1]         12      28ff24
```

```
x[ 2]        -18      28ff28
```

```
x[ 3]         24      28ff2c
```

```
x[ 4]         36      28ff30
```

```
Valor del puntero 28ff20
```

```
La direccion del primer elemento del
```

```
arreglo es 28ff20_
```

CADENAS DE CARACTERES

Objeto de datos compuesto de caracteres.

Especificación y sintaxis

Cadena de longitud fija declarada en el programa. `char nom[]="FIN"`

Cadena de longitud variable hasta un límite máximo declarado en el programa. `char nom[20]`

Cadena de longitud no determinada. `char * nom`

CADENAS DE CARACTERES EN C

- C no provee tipo de datos cadena, utiliza arreglos de caracteres.
- Las funciones de manejo de cadena estan en la librería <string.h>
- El carácter nulo ‘\0’ sigue al último carácter de la cadena, es anexado por el traductor al almacenar la cadena.

| | | | | | | | | | | | |
|-----|---|---|---|---|---|---|---|----|---|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 11 | 12 |
| L | U | N | E | S | | 3 | 1 | \0 | | | |
| cad | | | | | | | | | | | |

```
char cad[13];  
{gets(cad);  
 puts(cad);  
 ---
```

Ejemplo

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
int main(void)
{char cad[11] ; int i,l, cont=0;
 printf("\n Ingrese cadena  minúsculas( máximo 10 car.: " );
 gets(cad);
 l=strlen(cad); Qué pasa si uso scanf("%s" cad) ?
 printf("\n la cadena tiene longitud %d: ",l );
 for ( i=strlen(cad);i>=0;i--)
     if (toupper(cad[i] )=='M')          cont ++;
 printf("\n Total de letras M es : % d", cont );
 printf("\n cadena en mayusculas: " );
 puts(strupr(cad));getch();
}
```

REGISTROS

Especificación

Estructura de datos compuesta por un número fijo de componentes de igual o distinto tipo, de longitud fija.

Operación básica: selección.

| | |
|------|----------|
| Edad | Nombre |
| | Regidtro |

Atributos de un registro

- ✓ Número de componentes, campos o miembros
- ✓ Tipo de datos de cada componente,
- ✓ Identificador para nombrar cada componente

REGISTROS

En lenguaje C:

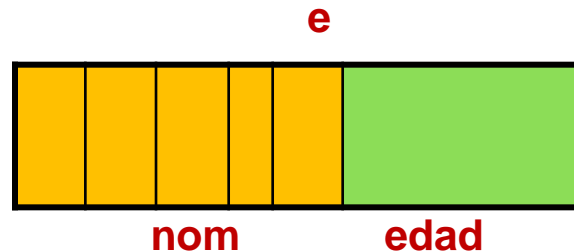
```
struct empleados  
{  
  char nom[10];  
  int  edad;  
}
```

struct empleados e;

En lenguaje Pascal:

```
Type empleados =  
  record  
    nom: string[10];  
    edad: integer;  
  end;
```

Var e:empleados;



e.edad operación de selección del campo edad

↑
Operador punto

e.edad=23;

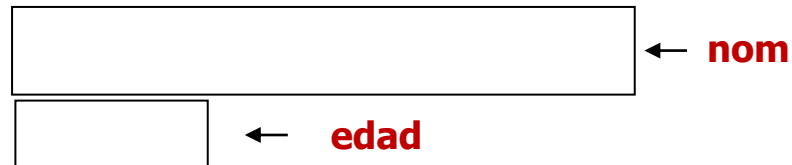
REGISTROS

Implementación

Se usa una representación secuencial de almacenamiento.

La selección de una componente se implementa con facilidad pues los nombres de los campos se conocen en tiempo de traducción.

```
struct empleados  
{  
  char nom[10];  
  int  edad;  
} e;
```



Registros en Lenguaje C – Struct

Declaración de variable :

struct <identificador> < var 1>, ..., <var n>;

Acceso a Componente:

<identif_struct>. <identif_campo>

```
struct alumno  
{  
  int dni;  
  char nom[20];  
  int legajo;  
  char carrera;  
};
```

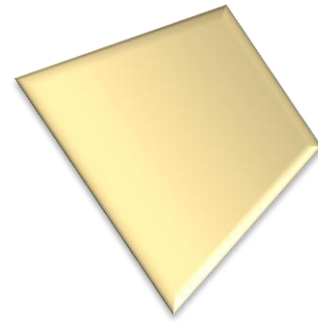
```
struct alumno  
{  
  int dni;  
  char nom[20];  
  int legajo;  
  char carrera;  
} al;
```

struct alumno al;

al

| | | | |
|-----|-----|--------|---------|
| dni | nom | legajo | carrera |
|-----|-----|--------|---------|

Asignación de valores a campos o atributos de una estructura



Inicialización

```
struct empleados  
{ char nom[10];  
  int  edad;  
} a = {"mario",23} ;
```

Asignación Directa

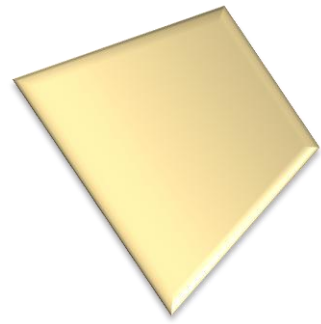
```
main()  
strcpy (a.nom, "carlos");  
a.edad=27;
```

Lectura desde teclado

```
main()  
{  
  empleados a;  
  printf(" Ingrese nombre y edad");  
  gets(a.nom);  
  scanf("%d",&(a.edad));
```

Asignación de estructuras

Situación 1



a=b Si a y b son estructuras del mismo tipo

```
struct alumno
{ int dni;
  char nom[20];
  int legajo;
};
```

m

| | | |
|----------|------|-------|
| 33125641 | Luis | 50000 |
|----------|------|-------|

```
struct alumno m={33125641,"luis",23567}, n;
```

n

| | | |
|----------|------|-------|
| 33125641 | Luis | 23567 |
|----------|------|-------|

```
main()
{
  n=m;
  m. legajo= 50000
  printf("Registro m  nombre %s legajo %d", m.nom, m. legajo );
  printf("Registro n nombre %s legajo %d", n.nom, n. legajo);
}
```

Asignación de estructuras (2)

Situación 2

a=b

Si a y b son estructuras del mismo tipo

```
struct alumno
{ int dni;
  char nom[20];
  int * legajo;
};
```

```
struct alumno m={33125641,"luis",23567}, n;
```

```
main()
```

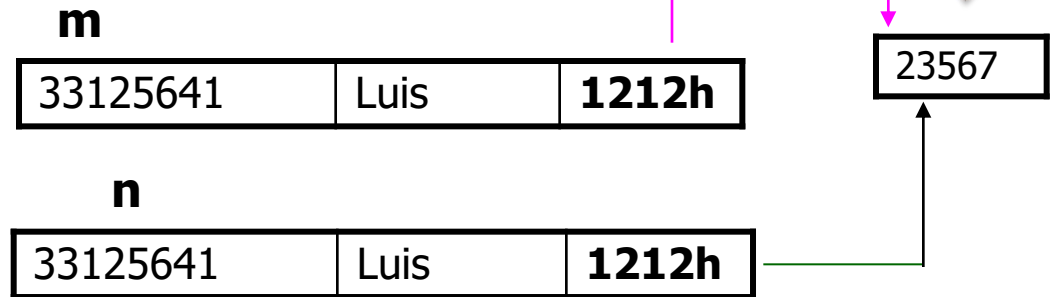
```
{n=m;
```

```
*(m.legajo)= 50000    ///// que sucede???
```

```
printf("Registro m  nombre %s legajo %d", m.nom,*(m.legajo));
```

```
printf("Registro n nombre %s legajo %d", n.nom, *(n.legajo));
```

```
}
```



Arreglos de struct

```
struct alumno
{ int dni;
  char nom[20];
  int registro;
  char carrera;
};
alumno alu[3];
```

| alu | | | | | | | |
|----------|-------------|------|---|--|---------|-----------|--------|
| 26123901 | Juan García | 8700 | C | | 9814567 | Ana Pérez | 9800 S |
| alu[0] | | | | | | | |

Actividad Áulica 3:

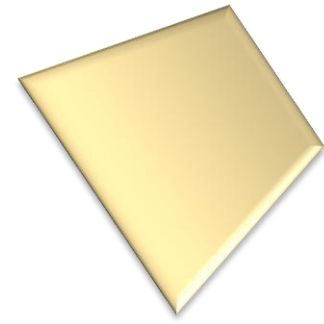
Dada la declaración,

```
alumno c[ ] = {
    { 255699, " PEDRO MANZANO ", 12124, 'C'},
    { 29334666 , "PAULA MAREIRA", 150068, 'S'}
    { 30562289, " NICOLAS ORTIZ", 14445, 'S'},
};
```

Escribir las sentencias que permitan:

- Mostrar el número de registro del tercer alumno.
- Cambiar por una O, el octavo carácter del nombre del segundo alumno
- Indicar la cantidad de memoria necesaria para almacenar la variable c.

Punteros a struct



```
#include<string.h>
```

```
struct alumno
```

```
{ char nom[20];
```

```
  int legajo;
```

```
  char carrera; };
```

```
void main(void)
```

```
{ alumno al, *p;
```

```
  printf("ingrese nombre"); gets ( al.nom );
```

```
  printf("ingrese legajo"); scanf ( "%d",&al.legajo );
```

```
  printf("ingrese carrera"); al. carrera = getche();
```

```
  printf("\n %s %d %c",al. nom, al. legajo, al. carrera);
```

```
p=&al; // apunto a la struct a través de p
```

```
p->carrera = 'P'; // (*p).carrera ='P';
```

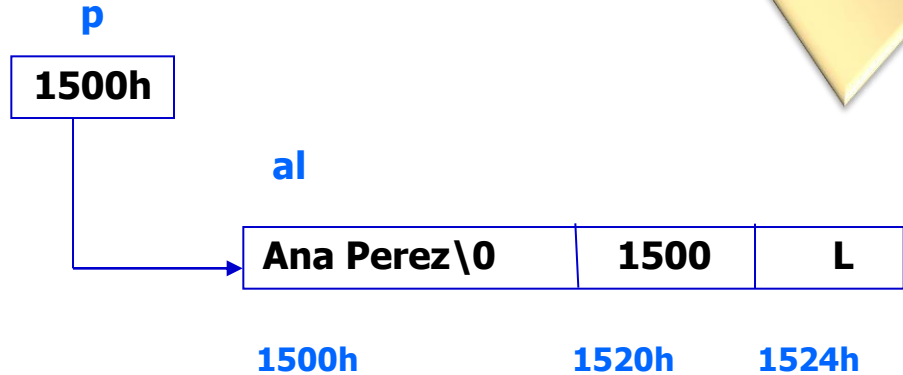
```
p->legajo = 1000; // (*p).legajo='1000';
```

```
strcpy(p->nom,"Mario Perez");
```

```
printf("\n %s %d %c",al. nom, al. legajo, al. carrera);
```

```
printf("\n ..... ", p, &(al.nom), &(al.legajo), &p);
```

```
}
```



Valor r(p)= Valor l(al)

Acceso a campos

puntero->miembro

(*puntero).miembro

REGISTROS Variantes en C – union

En un registro ordinario, cada componente existe a lo largo del tiempo de vida del registro.

En un registro variante existen algunos componentes, que dependiendo del valor marca, pueden estar o no presentes.

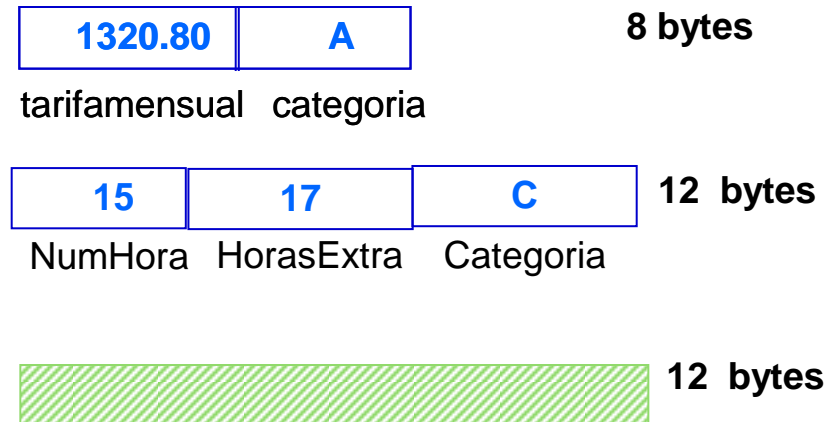
```
typedef struct
{ float
  tarifamensual;
  char  categoria;
} asalariados;
```

```
typedef struct
{ int NumHora;
  int HorasExtra;
  char categoria;
} porHoras;
```

```
typedef union
{
  asalariados as;
  porHoras ph;
} empleado;
```

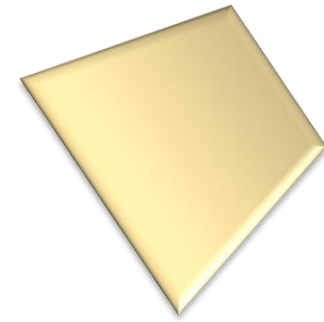
```
void main(void)
{ empleado a;
  int i;

}
```



En lenguaje C los registros variantes se denominan **union libre**, porque no admiten marcas

Diferencias entre Struct y Union



union U

```
{
int A;
double B;
char C;
}
```

U

| 1000h | | A / B / C |
|-----------------|-----------|---------------|
| 1001h | | |
| : | | |
| 1005 : | | |
| 1007 | | |
| Dir. de Memoria | Contenido | Identificador |

Valor I(U)= Valor I(A)= Valor I(B)= Valor I(C)

struct T

```
{
int A;
double B;
char C;
}
```

T

| 1000h | | A |
|-----------------|-----------|---------------|
| : | | |
| 1003h | | |
| 1004h | | B |
| : | | |
| | | |
| | | |
| 1011 | | |
| 1012 | | C |
| Dir. de memoria | Contenido | Identificador |

Valor I(T)= Valor I(A)

Almacenamiento la union U necesita solo 8 byte, mientras que la struct T necesita de 16 bytes