

Lenguajes de Programación

Tipos compuestos

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

Registros (producto cartesiano) - Variantes (uniones)

Este constructor de tipo relaciona datos de tipos heterogéneos, permitiendo que se almacenen y manipulen juntos.

Detalles

- Algunos lenguajes (Algol 68n C, Lisp) utilizan `struct` en lugar de `record`

en C

```
struct elemento {  
    char nombre[2];  
    int num-atómico;  
    double peso-atómico;  
    _Bool metalico;  
}
```

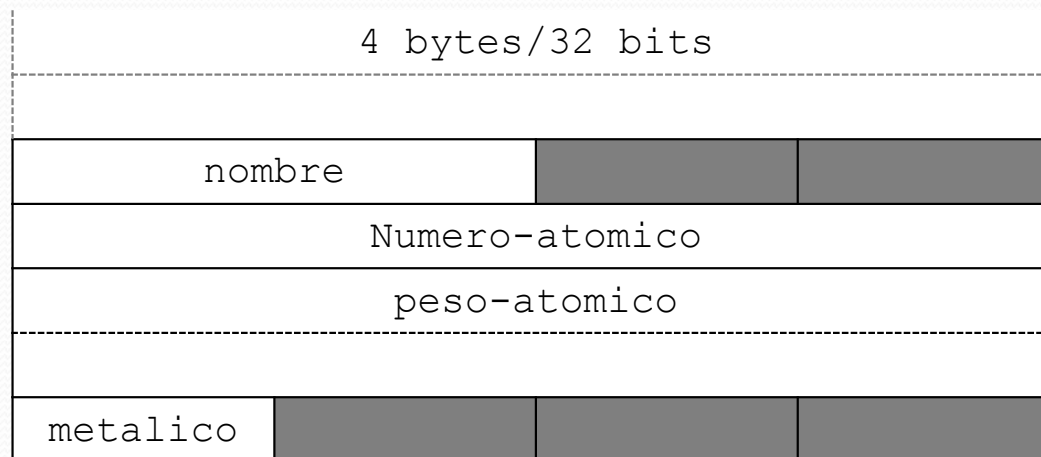
en Pascal

```
type elemento = record  
    nombre: two_chars;  
    num-atómico: integer;  
    peso-atómico: real;  
    metalico: Boolean  
end;
```

- Se utiliza la notación de punto para acceder a las componentes

Registros (producto cartesiano) - Variantes (uniones)

Los campos usualmente se almacenan en locaciones adyacentes de memoria.



Registros (producto cartesiano) - Variantes (uniones)

Para mejorar el uso de memoria algunos lenguajes permiten empaquetar los registros

4 bytes/32 bits	
nombre	numero-
atomico	
peso-atomico	
	metalico

Respetando el orden

4 bytes/32 bits		
nombre	metalico	
numero-atomico		
peso-atomico		

Sin respetar el orden
(minimizar agujeros)

Registros (producto cartesiano) - Variantes (uniones)

La mayoría de los lenguajes permite la anidación de estos constructores.

Si se tienen estructuras complejas, el acceso a las componentes se vuelve bastante extraño y difícil.

Para simplificar estas situaciones algunos lenguajes proveen una operación adicional. En el caso de pascal es la sentencia `with`

Registros Variantes (uniones)

Esta idea se popularizó en los lenguajes de programación en un momento donde las limitaciones de memoria eran muy fuertes

Esta estructura permitía utilizar diferentes variantes en el mismo espacio de memoria. La clave estaba en identificar que opciones no eran necesarias al mismo tiempo

en C

```
unión {  
    int i;  
    double d;  
    _Bool b;  
};
```

Arreglos (mapeo)

Es el tipo de dato mas común e importante. Ha sido parte fundamental de los primeros lenguajes de alto nivel y sigue manteniendo su vigencia.

Detalles

- Definen colecciones de datos homogéneas (con respecto al tipo)
- La sintaxis varia de lenguaje en lenguaje:
 - Incluye la definición del rango o
 - Define la cantidad de componentes

Arreglos (mapeo)

Operaciones

- [] para el acceso a las componentes
- Slices

Dimensiones y tamaño

- ¿Pueden definirse arreglos de arreglos?
- El tamaño ... ¿es fijo o puede modificarse?

Implementación de arreglos

Alcance estático – Alocación en pila

Program Principal

Procedure Q

var i:integer

a:array[1..10]of real

begin

a[i]:= 1;

end Q

End Principal

Registro del esquema de alocación

	Pila
0	
1	
2	
3	
4	límite inf.
5	límite sup.
6	
7	
...	
15	

DESCRIPTOR

Implementación de arreglos

Alcance estático – Estruct. bloques anidados - Alocación en pila

Program Principal

Procedure Q

var i:integer

a:array[1..10] of real

begin

a[i]:= 1;

end Q

End Principal

Registro de activación de Q

	Pila	
0		
1		
2		
3	i	
4	1 (límite inf.)	a
5	10 (límite sup.)	
6		
7		
...		
15		

¿Cómo se resuelve el acceso a una componente?

Implementación de arreglos

Alcance estático - Alocación en pila (tamaño no conocido en compilación)

- Estructuras de datos de tamaño conocido al momento de activarse la unidad.
- El tamaño del registro de activación no se conoce estáticamente, pero si en el momento de la creación del mismo.
- Se mantiene una estructura de pila para la memoria.
- El acceso a las variables semi-dinámicas, requiere un nivel de indirección.

Implementación de arreglos

Alcance estático – Alocación semi-dinámica

Program Principal

Procedure Q(n,m:integer)

var i:integer

a: array[1..n] of real;

b: array[1..m] of real

begin

a[i]:= 1;

end Q

End Principal

¿Cómo se resuelve el acceso a una componente?

Objetos de datos del sistema

i

LI = 1

LS = n

Ptero. al comienzo de a

LI = 1

LS = m

Ptero. al comienzo de b

Descriptor de a

Descriptor de b

a

b

12

Implementación de arreglos

Alcance estático – Alocación heap

- Estructuras de datos de tamaño variable.
- Se mantiene la estructura de pila y se agrega un **HEAP**.

Program Principal

Procedure Q

var i:integer

a: array[1..n] of real;

b: array[1..m] of real

begin

a[i]:= 1;

end Q

End Principal

Al trabajar con arreglos dinámicos, el tamaño del arreglo no queda determinado en forma fija, ni siquiera en ejecución.

13

Implementación de arreglos

Alcance estático – Alocación heap

- El acceso a los objetos de dato dinámicos se realiza con un nivel de indirección.
- Con la dirección del registro de activación y el desplazamiento del arreglo, se obtiene la dirección efectiva del arreglo, es decir la dirección del descriptor.
- Desplazándose en el descriptor se obtiene la dirección en el **heap**.
- Con la dirección en el **heap** y el subíndice se calcula el desplazamiento de la componente referenciada.

Implementación de arreglos

Alcance estático – Alocación heap

Program Principal

Procedure Q

var i:integer

a: array[1..n] of real;

b: array[1..m] of real

begin

a[i]:= 1;

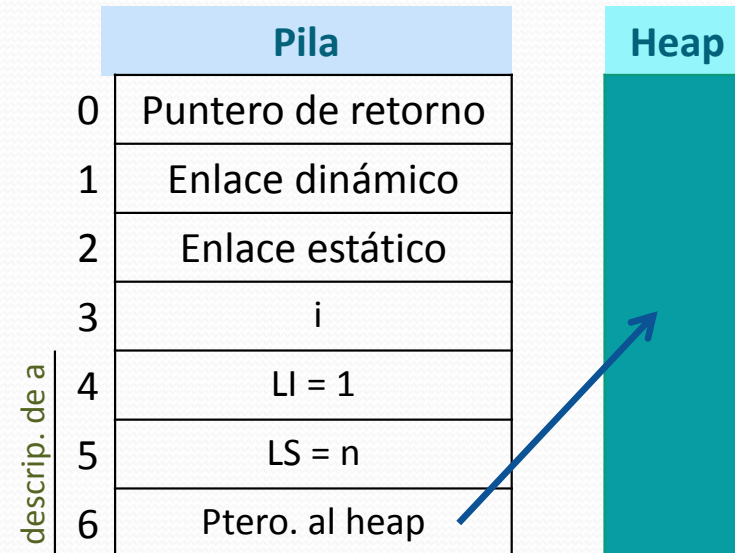
end Q

End Principal

Registro del esquema de alocación

	Pila	Heap
0	Puntero de retorno	
1	Enlace dinámico	
2	Enlace estático	
3	i	
4	LI = 1	
5	LS = n	
6	Ptero. al heap	

descrip. de a



¿Cómo se resuelve el acceso a una componente?

Arreglos (mapeo)

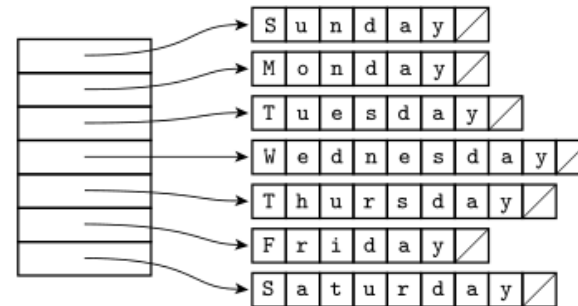
Layout

- El layout de los arreglos en memoria es similar al problema presentado para los registros empaquetados.
- Es necesario hacer un tradeoff entre diseño e implementación.

```
char days[][10] = {  
    "Sunday", "Monday", "Tuesday",  
    "Wednesday", "Thursday",  
    "Friday", "Saturday"  
};  
...  
days[2][3] == 's'; /* in Tuesday */
```

S	u	n	d	a	y	/			
M	o	n	d	a	y	/			
T	u	e	s	d	a	y	/		
W	e	d	n	e	s	d	a	y	/
T	h	u	r	s	d	a	y	/	
F	r	i	d	a	y	/			
S	a	t	u	r	d	a	y	/	

```
char *days[] = {  
    "Sunday", "Monday", "Tuesday",  
    "Wednesday", "Thursday",  
    "Friday", "Saturday"  
};  
...  
days[2][3] == 's'; /* in Tuesday */
```



Tomado del libro de Scott, sección 8,2

16

Strings

En muchos lenguajes es un arreglo de caracteres.

En otros tiene un estatus especial, con operaciones propias (que no están disponibles para arreglos de otros tipos)

- Operaciones de pattern-matching
- Facilidades basadas en expresiones regulares

Conjuntos

Colección no ordenada de un conjunto de valores del mismo tipo.

- Colección no ordenada de un conjunto de valores del mismo tipo.
- Suelen implementarse con otras estructuras, tabla hash, arreglos, arboles.
- Para tipos discretos con un número modesto de componentes se suele apelar al vector de bits.
- Las operaciones sobre conjuntos representados con bit-vectors permiten el uso de instrucciones máquina mas veloces.

Punteros y tipos recursivos

Un tipo recursivo es aquel cuyos objetos pueden contener una o mas referencias a otros objetos del tipo.

Detalles

- ¿ puede referenciar a cualquier objeto o solo los ubicados en el heap?
- ¿Cómo se reclama la memoria de los objetos ubicados en el heap una vez que pierden la referencia?

Sintaxis y operaciones

- Almacenamiento y liberación de objetos en memoria
- Dereferenciación y asignación

Puntero y dirección son dos conceptos diferentes

Lista

Esta definida de manera recursiva como la lista vacia o un par (que consiste de una objeto y otra lista.

Objeto= lista o átomo

Detalles

- Es un constructor típico de los lenguajes funcionales. Esto se debe a las facilidad que provee combinado con el pattern matching y las funciones de alto orden.

Archivos I/O

Permiten la comunicación con el “mundo exterior”

Detalles

- Es uno de los aspectos de los lenguajes de programación con mas aspectos difíciles desde el punto de vista del diseño.
- Se pueden incorporar de manera:
 - Built-in
 - librerías