1. Tipos Compuestos o estructurados

1. **Introducción**
2. **Definición de tipo estructurado en C**

Bibliografía:

- ***Introducción a la programación. Tomo 2*. Clavel y Biondi**
- ***Fundamentos de programación : algoritmos y estructura de datos y objetos*. Joyanes**
- ***El lenguaje de programación C.* Kernighan y Ritchie.**

-

1. Introducción

Tipos escalares vs tipos compuestos

- **escalares**: enteros, reales, carácter, lógico
- **compuestos**:
  - *arrays* (tabla o vector): los arrays son un conjunto de **elementos del mismo tipo** que están lógica o conceptualmente **relacionados**
  - Tipos **estructurados** o registros
    - **Definidos por el programador** a partir de otros tipos escalares o compuestos
    - Conjunto de datos **conceptualmente relacionados**
    - **Cada dato del conjunto puede ser de un tipo distinto**

1. Introducción

Sintaxis

<decl_objeto_compuesto> ::=

<decl_tipo_compuesto> | <decl_vble_compuesta>

<decl_tipo_compuesto> ::=

<tipo_compuesto> = REGISTRO DE

<decl_objeto>

{; <decl_objeto>}

FIN REGISTRO

<Variable>:<tipo_compuesto> ;

<decl_vble_compuesta> ::=

<nombre_vble> : registro de <tipo_compuesto> fin registro;

1. Introducción

- **Semántica**
  - Campo: cada objeto de un tipo estructurado, es una variable de un cierto tipo (escalar o compuesta)
  - Para acceder a los campos de un registro:

    <variable_estructurada>.<nombre_campo>

  Ejemplo:

```
persona = registro de
    nombre, apellido1, apellido2: cadena;
    sexo, estado_civil: carácter;
    edad, dni: numérico;
fin registro
```

1. Introducción

{Declaración de variables}

perso1: persona;

nombre	apellido1	apellido2	sexo	estado_civil	edad	dni
“ramón”	“suárez”	“garcía”	‘v’	‘c’	28	12345678

{Acceso a los campos}

perso1.nombre → variable de tipo cadena

perso1.sexo → variable de tipo carácter

perso1.edad → variable de tipo numérico {entera}

2. Definición de tipo estructurado en C

- Los tipos estructurados en C se definen mediante ***struct.***

struct [nombre_estructura] **{**

campo_1;

campo_2;

…

} [variable_estructura];

- Las variables de tipo estructurado deben ir precedidas del nombre del tipo, que en C lleva la palabra **struct**.

Ejemplo:

```
struct  persona {
 int edad, dni;
 char nombre[30], apellido1[40], apellido2[40];
 char sexo, estado_civil;
};
struct persona perso1;
```

- Para acceder a un campo se utiliza el operador . (perso1.edad)

2. Definición de tipo estructurado en C

- Además de los tipos **struct**, se pueden crear registros con la sentencia **union**. Permite utilizar la misma memoria para distintos tipos.
- El formato general de una declaración union es:

union [nombre_union] **{**

campo_1;

campo_2;

…

} [variable_union];

- La diferencia entre una variable tipo **union** y un registro **struct** es que todos los campos de una **union** ocupan la misma posición de memoria (que será del tamaño del mayor de ellos), mientras que cada campo de una **struct** ocupa posiciones de memoria distintas.
- Se utiliza para que una misma variable pueda contener datos de distinto tipo, o cuando se quiere utilizar una misma variable desde distintos puntos de vista.

2. Definición de tipo estructurado en C

- Utilizando **struct** se pueden definir los tipos y las variables de ese tipo al mismo tiempo:

```
struct dir {
  char nombre[30], calle[40], ciudad[20], prov[20];
  unsigned long int DP;
} ainfo, binfo, cinfo;
```

- También se pueden definir variables de tipo registro, sin definir un tipo estructurado:

```
struct {
 char nombre[30], calle[40], ciudad[20], prov[20];
  unsigned long int DP;
} dinfo;
```

- Sólo debemos usar esta última opción cuando no vaya a volver a utilizarse el tipo de dato.

2. Tipos estructurados en C (cont.)

- Dado que el tamaño de los distintos tipos de datos variará de unas máquinas a otras, se dispone de la función **sizeof()** que devuelve el tamaño de la variable que se pase como argumento, sea esta del tipo que sea:

```
char ca;
int i;
float f;
printf ("%d\n", sizeof(ca));
printf ("%d\n", sizeof(i));
printf ("%d\n", sizeof(f));
```

Renombrar tipos de datos con typedef

- Podemos utilizar la sentencia **typedef** para renombrar tipos de datos y que nos sea más sencilla la programación:

```
typedef int logico;
logico ok;

typedef float balance;
balance deuda;

struct dir {
 char nombre[30], calle[40], ciudad[20], prov[20];
  unsigned long int DP;
};
typedef struct dir direccion;
direccion dire;
```

Renombrar tipos de datos con typedef

```
typedef struct {
  float credito;
  int deuda;
  char nombre[40];
} cliente;
cliente clist[NUMCLIENTES];
/* Define un array de estructuras del tipo cliente, que es una estructura */
```

- Con este método la programación es más sencilla, pero no se están definiendo realmente nuevos tipos de datos.

Ejemplo

{Temperaturas diarias, durante un año: máxima, mínima y media}
{DECLARACIÓN DE TIPOS}
TEMP = REGISTRO DE
maxima, minima, media: numérico;
FIN REGISTRO;
TEMP_ANUAL = vector[1..12,1..31] de TEMP;
{DECLARACIÓN DE VARIABLES}
T: TEMP_ANUAL;

¿T(1,10)?
¿T(3,17).max?
¿T(7,31).media?