Tipos estructurados y definidos por el usuario: struct y union

Además de los tipos de datos básicos, se pueden crear registros con las sentencias **struct** y **union**.

El formato general de una declaración **struct** es:

```
struct [nombre_estructura] {
    campo_1;
    campo_2;
    ...
} [variable_estructura];
```

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 la primera implica que todos los campos ocupan la misma posición de memoria (que será del tamaño del mayor de ellos).

Se utiliza cuando se quiere que una misma variable pueda contener datos de distinto tipo, o cuando se quiere utilizar una misma variable desde distintos puntos de vista.

También se pueden renombrar tipos de datos existentes utilizando *typedef*.

Struct

Definición de un tipo de dato dirección: **struct dir**

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

Si queremos definir una variable:

```
struct dir ainfo;
```

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];
   char calle[40];
   char ciudad[20];
   char prov[20];
   unsigned long int DP;
} dinfo;
```

Sólo debemos usar esta última opción cuando no vaya a reutilizarse el tipo de dato.

Para referirnos a los elementos de una estructura o unión se utiliza la notación:

nombre_estructura.nombre_campo

```
ainfo.DP = 33180;
printf ("%d\n",ainfo.DP);
```

Si lo que queremos es tener un vector de estructuras, tendremos que definir:

```
struct dir ainfo[100];

printf("%d\n",ainfo[i].DP);

struct x {
    int a[10][10];
    float b;
} y;
```

Union

Cuando se crea la unión se reserva memoria para almacenar la mayor cantidad posible de información.

```
union u {
    int i;
    char ca;
};
union u cnvt;
```

El tamaño de la variable vendrá dado por el máximo de las longitudes de:

cnvt.i ó cnvt.ca

La función sizeof

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

Si lo que queremos es utilizar la sentencia **typedef** para renombrar tipos de datos y que nos sea más sencilla la programación:

```
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;
```

```
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.