Estructuras

- Una estructura, conocida en otros lenguajes como registro, permite agrupar en una unidad un conjunto de variables relacionadas.
- Para poder declarar una variable de tipo estructura primero debemos describir el “molde” o “patrón” de la estructura y luego declarar la variable de este “tipo de estructura”

```
struct empleado {//declaro el tipo
        int legajo;
        char nombre[30];
};

struct empleado e1, e2; //defino variables
```

Declaración y definición

- Puedo declarar la estructura y definir las variables en un solo paso

```
struct punto {
        double x;
        double y;
} pto1, pto2;
```

- Puedo definir una variable de una estructura anónima

```
struct { //anónima (falta nombre del struc
        int i, j; //posible pero desaconsejado
} anon1;
```

Inicialización

- Puedo incializar valores al declarar la variable

```
struct punto pto3 = { 1.0, 2.0};
struct empleado e3 = { 525, "Pablo"};
```

- Puedo hacer todo junto (declarar e inicializar)

```
struct ejemplo {
        int a;
        int b;
} vejemplo = {1, 5};
```

- A partir de C99 se puede incializar en cualquier orden si “designo” el campo al que asigno.

```
struct punto pto = {.y = 2.0, .x = 1.0};
/* idéntico a pto = {1.0, 2.0} */
```

Uso

- Para acceder a un miembro de una estructura uso el operador . (punto) si lo que tengo es una variable

```
e1.legajo = 123;
strcpy(e1.nombre, "Juan");
```

- Puedo asignar para copiar TODOS los campos

```
e2 = e1;
```

- Puedo pasar una estructura como parámetro o devolverla como resultado de una función

Punteros

- Si declaro un puntero a una estructura y quiero acceder a un campo, por tema de precedencias debo usar paréntesis

```
struct punto *p;
p = &pto;
(*p).x = 5.0;
```

- Como esto es engorroso se definió el operador -> que es lo que se acostumbra utilizar.

```
p->x = 5.0; //equivalente a (*p).x = 5.0;
```

typedef y autoreferencias

- Se puede declarar una estructura dentro de otra y es posible tener un miembro de tipo puntero a la estructura que lo contiene

```
struct enlazada {
    int clave;
    struct {
        double medida;
        char *descrip;
    } datos;
    struct enlazada *siguiente;
};
typedef struct enlazada *ptr2en;
```

```
struct enlazada enl01, enl02;
enl01.clave = 1;
enl01.datos.medida = 25.6;
enl01.siguiente = &enl02;

ptr2en primero = &enl01;
primero->siguiente->clave = 2;
/*equivale a: enl02.clave = 2 */

struct enlazada enl03 = {1, {3.0}};
/*descrip y siguiente puestos en
NULL */
```

Uniones

- Al declarar una estructura se asigna memoria para todos sus miembros, uno detrás de otro, en el orden en que fueron declarados, con posibles espacios intermedios por cuestiones de alineación
- Una unión es similar pero asigna espacio para el miembro más grande ya que todos se alinean al principio. Es decir, en la unión solo uno de los miembros puede estar almacenado en un momento dado.
- Salvo por el modo de almacenaje, en todos los demás aspectos se comprota igual que una estructura

Uniones

```
enum  modos_venta
{XUNIDAD, XPESO, OTROS};

struct pedido {
    int tipo_pedido;
    union {
        int unidades;
        double peso;
        char descrip[20];
    } dato;
} vped;
```

```
/* Supongamos que ya leímos y
cargamos vped.tipo_pedido */

switch (vped.tipo_pedido) {
case XUNIDAD:
    scanf("%d"
          , &vped.dato.unidades);
    break;
case XPESO:
    scanf("%lf"
          , &vped.dato.peso);
    break;
case OTROS:
    gets(vped.dato.descrip);
}
```

Licencia