Estructuras Compuestas.

Introducción.

Una estructura es una colección de variables, posiblemente de diferente tipo, agrupadas bajo un único nombre .
Las estructuras permiten tratar variables relacionadas como una unidad y no como entidades separadas (Kernighan- Ritchie).

Declaración de estructuras.

Consideremos como que entidad básica es un punto.

Asumiremos que tiene una coordenada *x* y una coordenada *y*, ambas enteras.

Las dos componentes pueden ubicarse en una estructura declarada como sigue:

La palabra reservada `struct` introduce la declaración de una estructura que es una lista de declaraciones encerrada entre llaves.

Etiqueta de la estructura.

```
struct punto
{
    int x;
    int y;
};
```

Declaración de la estructura.

Miembros de la Estructura.

```
struct punto {
    int x;
    int y;
};
```

Declaración de la estructura: *no reserva almacenamiento*; es solamente una *plantilla* o *forma* de una estructura.

Otro ejemplo de declaraciones de estructuras:

```
struct elemento_quimico {
    int num_atomico;
    char nombre[10];
    char simbolo[3];
    float masa;
    char *p1;
};
```

Cadena de caracteres.

Variable tipo Puntero como miembro de una estructura.

Definición de una variable tipo `struct`:

La *etiqueta* puede usarse luego de una declaración en definiciones de *instancias de la estructura*.

Por ejemplo, dada la declaración precedente de `punto`,

```
struct punto pt;
```

Define una variable `pt` que es una estructura de tipo `struct punto`.

Otra manera de **definir** una variable tipo **`struct`.**

```
struct punto {
    int x;
    int y;
} pt ;
```

Define una variable `pt` que es una estructura de tipo `struct punto`.

Autorreferencias en Estructuras:

Una estructura *no* puede contener una instancia de sí misma entre sus miembros. Sin embargo, puede contener un *puntero* a una estructura del mismo tipo.

```
struct empleado {
    char nombre[20];
    char apellido[20];
    int edad;
    char sexo;
    double salario_por_hora;
    struct empleado e;
    struct empleado *ptr;
};
```

/***ERROR*****/**

Puntero a una `struct empleado`.

Inicialización de estructuras al momento de declararlas.

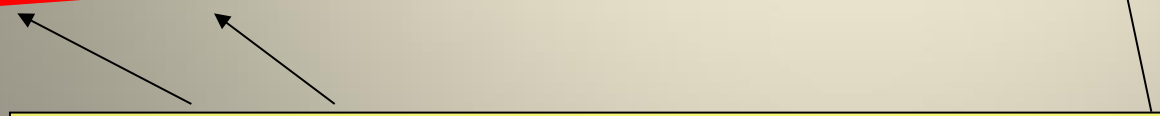
Una estructura puede inicializarse a continuación de su definición con una lista de inicialización.

```
struct punto pt = { 1, 3 };
```

Inicialización de todos los miembros de la estructura `pt`.

Otro ejemplo de inicialización de estructuras.

```
struct ficha_personal{/*Declaración de la estructura.*/
    char nombre[20];
    char apellido[20];
    int edad;
    char sexo;
    float salario_por_hora; /*en pesos*/
};          .
            .
            .
/* Inicialización de una estructura tipo: struct ficha_personal*/
struct ficha_personal  empleado_1 = {“Roger", “Waters", 15,
’M’, 15.35};
```



Inicialización de todos los miembros de la estructura `empleado_1`

Acceso a miembros de una estructura.

Se puede hacer acceder a un miembro particular de una estructura por medio de una construcción de la forma:

```
etiqueta_estructura.nombre_miembro
```

El operador punto conecta el nombre de la estructura y el nombre del miembro al que se desea acceder.

Ejemplo: Podemos imprimir las coordenadas del punto **pt** como sigue.

```
struct punto{ /*declaración */
    int x;
    int y;
};

/*definición e inicialización*/
struct punto pt = { 1, 3 };
```

```
/*se imprimen los miembros de la estructura pt */
printf("%d,%d", pt.x, pt.y);
```

Ejemplo: Podemos ingresar por teclado las coordenadas del punto **pt**.

```
struct punto{ /*declaración */
   int x;
   int y;
};
struct punto pt; /*definición*/
```

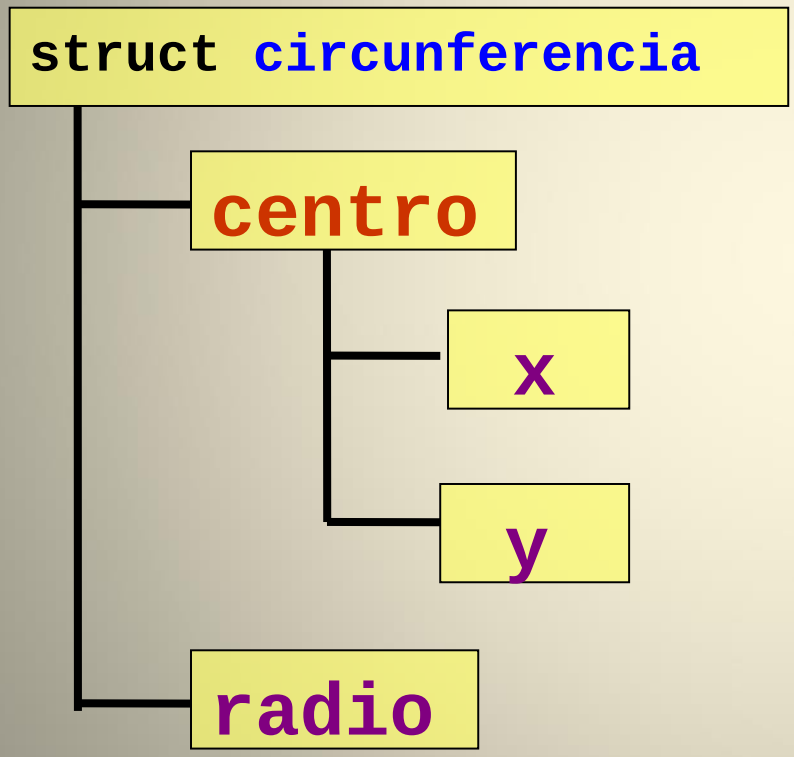
```
/*Podemos ingresar por teclado las coordenadas de pt*/
printf("Ingrese el valor de la coordenada x:");
scanf("%d", &pt.x);
printf("Ingrese el valor de la coordenada y:");
scanf("%d", &pt.y);
            ↑   ↑

fflush(stdin);
```

Estructuras anidadas.

```
struct punto{ /*declaración */
    int x;
    int y;
};
. . .
struct circunferencia{
    struct punto centro;
    int radio;
};
```

Estructura como un miembro de una estructura.

Miembros de la estructura `struct circunferencia`



Acceso a los miembros de `centro`**:**

```
circunferencia.centro.x
circunferencia.centro.y
```

Acceso a `radio`**:**

```
circunferencia.radio
```

Punteros a Estructuras.

También podemos acceder a los miembros de una estructura usando punteros a estructuras.

Los punteros a estructuras son como punteros a variables ordinarias `char`, `int`, etc.

Declaración de un puntero a una Estructura.

```
struct punto *pp;
```

`pp` es un puntero a una estructura de tipo `struct punto`.

Acceso a los miembros de una estructura con punteros.

Por medio del puntero `pp` accedemos a los miembros de la estructura como sigue:

```
(*pp).x;
(*pp).y;
```

Los paréntesis en `(*pp)` son necesarios porque el operador miembro " . " tiene precedencia superior a la del operador indirección "*" .

Ejemplo:

```
struct punto{ /*declaración */
  int x;
  int y;
};
. . .
struct punto par_ordenado, *pp;
par_ordenado.x=8;
par_ordenado.y=23;

pp = &par_ordenado;

(*pp).x=12;
(*pp).y=3;

printf("El par ordenado es:(%d, %d)\n", (*pp).x,
(*pp).y);
```

Asignación de una dirección a un puntero.
La dirección de una estructura es la misma que la dirección de su primer miembro. Por ejemplo:
&par_ordenado es &par_ordenado.x

Acceso a los miembros de una estructura usando un puntero.

Notación alternativa para punteros a estructuras.

Si p es un puntero a una estructura, entonces es posible hacer referencia a un miembro en particular de la siguiente manera:

```
p -> miembro_de_la_estructura
```

Consiste de un "signo menos" seguido de un signo "mayor que". No se permiten espacios en blanco entre " - " y " >".

Formas equivalentes de acceso a los miembros de una estructura.

```
(*pp).x=12;
 pp->x=12;
```

Tienen el mismo significado.

```
. . .
printf("El par ordenado es:(%d, %d)\n", pp->x, pp->y);
```

Imprime los miembros de `par_ordenado`.