Funciones y Estructuras

Funciones y Estructuras

A las funciones se pueden pasar:

Funciones y Estructuras

A las funciones se pueden pasar:

- miembros de la estructura,

Funciones y Estructuras

A las funciones se pueden pasar:

- miembros de la estructura,
- la estructura completa, o

Funciones y Estructuras

A las funciones se pueden pasar:

- miembros de la estructura,
- la estructura completa, o
- un puntero a una estructura

Funciones y Estructuras

A las funciones se pueden pasar:

- miembros de la estructura,
- la estructura completa, o
- un puntero a una estructura

En el caso de un miembro de la estructura o la estructura completa, es pasaje es *por valor*, o sea que la estructura original con la que se hace el llamado no se modifica dentro de la función

Funciones y Estructuras

Funciones y Estructuras

Pasar miembros de una estructura es igual al paso de variables...

Funciones y Estructuras

Pasar miembros de una estructura es igual al paso de variables...

Por ejemplo, si la función espera enteros

```
int suma (int a, int b) {
  return a+b;
}
```

Funciones y Estructuras

Pasar miembros de una estructura es igual al paso de variables...

Por ejemplo, si la función espera enteros

```
int suma (int a, int b) {
  return a+b;
}
```

el llamado se puede hacer

```
printf("%d\n", suma(p.x, p.y));
```

Funciones y Estructuras

Pasar miembros de una estructura es igual al paso de variables...

Por ejemplo, si la función espera enteros

```
int suma (int a, int b) {
  return a+b;
}
```

el llamado se puede hacer

```
printf("%d\n", suma(p.x, p.y));
```

Funciones y Estructuras

Pasar miembros de una estructura es igual al paso de variables...

Por ejemplo, si la función espera enteros

```
int suma (int a, int b) {
  return a+b;
}
```

el llamado se puede hacer

```
printf("%d\n", suma(p.x, p.y));
```

Funciones y Estructuras

Funciones y Estructuras

Pasar estructuras completas también es igual a cualquier variable...

Funciones y Estructuras

Pasar estructuras completas también es igual a cualquier variable...

...teniendo cuidado de no olvidar el tipo completo en la lista de parámetros del encabezado de la función

Funciones y Estructuras

Funciones y Estructuras

```c
#include <stdio.h>
#include <math.h>

struct punto2D {
  float x;
  float y;
};

float norma2d (struct punto2D p, struct punto2D q)
{
  return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}

int main (void)
{
  struct punto2D p1 = {3, 2}, p2 = {4, 5};
  float norma;

  norma = norma2d(p1, p2);
  printf("%.2f\n", norma);

  return 0;
}
```

Funciones y Estructuras

```c
#include <stdio.h>
#include <math.h>

struct punto2D {
  float x;
  float y;
};

float norma2d (struct punto2D p, struct punto2D q)
{
  return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}

int main (void)
{
  struct punto2D p1 = {3, 2}, p2 = {4, 5};
  float norma;

  norma = norma2d(p1, p2);
  printf("%.2f\n", norma);

  return 0;
}
```

Funciones y Estructuras

Funciones y Estructuras

(antes de seguir, repaso de Unidad 7)

Funciones y Estructuras
(antes de seguir, repaso de Unidad 7)

supongamos la función

```
int cuadrado (int a)
{
  return a*a;
}
```

Funciones y Estructuras
(antes de seguir, repaso de Unidad 7)

supongamos la función

```c
int cuadrado (int a)
{
  return a*a;
}
```

puede ser llamada con una variable

```c
int var = 5;
printf("%d\n", cuadrado(var));
```

Funciones y Estructuras
(antes de seguir, repaso de Unidad 7)

supongamos la función

```
int cuadrado (int a)
{
  return a*a;
}
```

Funciones y Estructuras
(antes de seguir, repaso de Unidad 7)

supongamos la función

```c
int cuadrado (int a)
{
  return a*a;
}
```

o con un *literal*

```c
printf("%d\n", cuadrado(5));
```

Funciones y Estructuras

Funciones y Estructuras

Las estructuras también pueden ser pasadas a funciones como literales...

Funciones y Estructuras

Las estructuras también pueden ser pasadas a funciones como literales...

...así se conocen como *literales compuestos*.

Funciones y Estructuras

Las estructuras también pueden ser pasadas a funciones como literales...

...así se conocen como *literales compuestos*.

Entre llaves se enumeran los valores de los miembros de la estructura y se agregan adelante paréntesis con el tipo de dato.

Funciones y Estructuras

Las estructuras también pueden ser pasadas a funciones como literales...

...así se conocen como *literales compuestos*.

Entre llaves se enumeran los valores de los miembros de la estructura y se agregan adelante paréntesis con el tipo de dato.

Se pueden usar los *inicializadores designados*.

Funciones y Estructuras

Funciones y Estructuras

```c
#include <stdio.h>
#include <math.h>

struct punto2D {
  float x;
  float y;
};

float norma2d (struct punto2D p, struct punto2D q)
{
  return sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
}

int main (void)
{
  struct punto2D p1 = {2, 2};
  float norma;

  norma = norma2d(p1, (struct punto2D) {0,0});
  printf("%.2f\n", norma);

  return 0;
}
```

Funciones y Estructuras

Funciones y Estructuras

Cuando se pasan arreglos de estructuras a funciones, son automáticamente por *referencia* como todos los arreglos...

Funciones y Estructuras

Cuando se pasan arreglos de estructuras a funciones, son automáticamente por *referencia* como todos los arreglos...

Pero una estructura tiene arreglos estos se pasan por copia, como todos los elementos de la estructura.

Funciones y Estructuras

Funciones y Estructuras

ejemplo de función que devuelve struct

Typedef

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

- `typedef` es la palabra clave que indica que se va a definir un *alias*

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

- a continuación se coloca el tipo de datos del que se quiere generar un sinónimo.

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

- se finaliza con el *alias* (y el punto y coma)

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

A partir de ese punto se puede usar indistintamente el alias o el tipo completo

Typedef

La palabra clave `typedef` prevé un mecanismo para generar sinónimos o *alias*.

```
typedef unsigned int uint;
```

A partir de ese punto se puede usar indistintamente el alias o el tipo completo

```
unsigned int valor1;
uint valor2;
```

Typedef

Typedef

Se usa mucho en estructuras para simplificar notación...

Typedef

Se usa mucho en estructuras para simplificar notación...

supongamos la definición de una estructura

```
struct punto2D {
  float x;
  float y;
};
```

Typedef

Se usa mucho en estructuras para simplificar notación...

supongamos la definición de una estructura

```
struct punto2D {
  float x;
  float y;
};
```

El prototipo de funciones que reciben estructuras con este nombre podrían ser extensas, por ejemplo

Typedef

Se usa mucho en estructuras para simplificar notación...

supongamos la definición de una estructura

```
struct punto2D {
  float x;
  float y;
};
```

El prototipo de funciones que reciben estructuras con este nombre podrían ser extensas, por ejemplo

```
struct punto2D suma (struct punto2D p, struct punto2D q);
```

Typedef

Typedef

Se puede usar `typedef` de varias formas

Typedef

Se puede usar `typedef` de varias formas

```
struct punto2D {
  float x;
  float y;
};
```

Typedef

Se puede usar `typedef` de varias formas

```
struct punto2D {
  float x;
  float y;
};

typedef struct punto2D p2D;
```

Typedef

Se puede usar `typedef` de varias formas

```
struct punto2D {
  float x;
  float y;
};

typedef struct punto2D p2D;
```

o

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

Typedef

Typedef

Entonces teniendo

Typedef

Entonces teniendo

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

Typedef

Entonces teniendo

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

el prototipo

Typedef

Entonces teniendo

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

el prototipo

```
struct punto2D suma (struct punto2D p, struct punto2D q);
```

Typedef

Entonces teniendo

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

el prototipo

```
struct punto2D suma (struct punto2D p, struct punto2D q);
```

puede pasar a

Typedef

Entonces teniendo

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

el prototipo

```
struct punto2D suma (struct punto2D p, struct punto2D q);
```

puede pasar a

```
p2D suma (p2D p, p2D q);
```

Typedef

Typedef

Notar que la definición + el *alias* con `typedef`

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

Typedef

Notar que la definición + el *alias* con `typedef`

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

tiene una sintaxis *parecida* a la declaración de una variable global `p1`

Typedef

Notar que la definición + el *alias* con `typedef`

```
typedef struct punto2D {
  float x;
  float y;
} p2D;
```

tiene una sintaxis *parecida* a la declaración de una variable global `p1`

```
struct punto2D {
  float x;
  float y;
} p1;
```