

UNT – FACEyT - DEEC



# Programación I

Año 2016

# Unidad 4: Registros en Lenguaje C

## Introducción

### Estructura de la información:

- entidad
- atributo
- valor

### Registro como tipo de dato:

*"Dato estructurado estático que contiene un conjunto ordenado de datos, generalmente heterogéneos, referenciados por un único identificador"*

# Unidad 4: Registros en Lenguaje C

## Introducción

Estructura de la Información



entidad

atributo

valor



Registro en lenguaje C



registro(struct)

campo o atributo

valor

# Unidad 4: Registros en Lenguaje C

## Definición

### Definición de registro (struct) en lenguaje C

"Es una colección de una o más variables, de tipos posiblemente diferentes, agrupadas bajo un solo nombre para manejo conveniente" (Kernighan & Ritchie).

"Es una colección de variables que se referencian bajo un único nombre, proporcionando un medio conveniente de mantener junta una información relacionada" (Herbert Schildt)

# Unidad 4: Registros en Lenguaje C

## Definición de tipo de dato

Los registros o estructuras no son un dato estándar de C, deben ser definidos por el usuario.

Para utilizar un dato del tipo `struct` debe:

- 1° Definirse el tipo de dato,
- 2° declarar la/s variable/s correspondientes

Existen dos formas de definir un dato tipo `struct`

# Unidad 4: Registros en Lenguaje C

## Definición de tipo de dato

Caso a):

```
struct <etiqueta> {  
    tipo_de_dato1    variable_1_1 [,variable_1_2, ...];  
    tipo_de_dato2    variable_2_1 [,variable_2_2, ...];  
    ...  
    tipo_de_datoN    variable_N_1 [,variable_N_2, ...];  
};
```

- ***tipo\_de\_dato***: cualquier tipo de dato primitivo o definido por el usuario

# Unidad 4: Registros en Lenguaje C

## Definición de tipo de datos

Caso b):

```
typedef struct {  
    tipo_de_dato1    variable_1_1 [,variable_1_2, . . .];  
    tipo_de_dato2    variable_2_1 [,variable_2_2, . . .];  
    . . .  
    tipo_de_datoN    variable_N_1 [,variable_N_2, . . .];  
} <etiqueta> ;
```

- ***tipo\_de\_dato***: cualquier tipo de dato primitivo o definido por el usuario.

# Unidad 4: Registros en Lenguaje C

## Definición de tipo de dato

### Ejemplos de definición de registros

Caso a)

```
struct nroComplejo {  
    float    a;  
    float    b;  
};
```

```
struct parOrdenado {  
    float    a;  
    float    b;  
};
```

Caso b)

```
typedef struct {  
    float    x;  
    int      y;  
} coordenadas1;
```

```
typedef struct {  
    float    x;  
    int      y;  
} coordenadas2;
```



# Unidad 4: Registros en Lenguaje C

## Declaración de variables

Simultáneamente con la definición sólo puede realizarse en el Caso a)

```
struct <etiqueta> {  
    tipo_de_dato1        variable_1_1 [,variable_1_2, ...];  
    tipo_de_dato2        variable_2_1 [,variable_2_2, ...];  
    .  
    .  
    .  
    tipo_de_datoN        variable_N_1 [,variable_N_2, ...];  
} var1 [, var2 , ...];
```

# Unidad 4: Registros en Lenguaje C

## Introducción

### Ejemplo

```
struct nroComplejo {  
    float a;  
    float b;  
} z1, z2;  
  
struct parOrdenado {  
    float a;  
    float b;  
} par1, par2;
```

# Unidad 4: Registros en Lenguaje C

## Declaración de variables

Otra forma, la más usada, de declarar las variables es luego de la definición de la etiqueta o tipo de dato.

Caso a)

```
struct <etiqueta>      var1_1 [, var1_2 , ...];
```

Caso b)

```
<etiqueta>            var1_1 [, var1_2 , ...];
```

# Unidad 4: Registros en Lenguaje C

## Declaración de variables

### Ejemplos

Caso a): La declaración debe estar precedida de la palabra reservada `struct`:

```
struct nroComplejo      z1, z2;  
struct parOrdenado      par1, par2;
```

Caso b): El uso la palabra reservada ***typedef*** define un nuevo "tipo de dato" y no es necesario especificar que se trata de una estructura:

```
coordenadas1            punto1, punto2;  
coordenadas2            puntoA, puntoB;
```

# Unidad 4: Registros en Lenguaje C

## Asignación de valores a los atributos

- Para asignar valores a un registro debe hacerse a cada campo o atributo en particular.
- Para acceder a un atributo de un registro se utiliza el operador punto o selector de campo (•).

`<variable_tipo_struct>.<atributo>`

# Unidad 4: Registros en Lenguaje C

## Asignación de valores a los atributos

- Como toda asignación puede realizarse:
  - a) En el momento de la declaración de la variable (inicialización).
  - b) En el cuerpo del programa con el operador de asignación.
  - c) En el cuerpo del programa utilizando funciones de lectura de datos.

# Unidad 4: Registros en Lenguaje C

## Asignación de valores a los atributos

### a) Asignación en la declaración o inicialización

```
struct ntroComplejo    z1 = { 11.22, 33.44 }, z2;
```

```
struct parOrdenado     par1, par2 = { 11.33, -22.44 };
```

```
coordenadas1          punto1 = { 3.14, 5 }, punto2{0.707, -3};
```

```
coordenadas2          puntoB = { 8.88, 49 }, puntoA;
```

# Unidad 4: Registros en Lenguaje C

## Asignación de valores a los atributos

b) Asignación directa con el operador de asignación:

```
z1.a = 55.66;
```

```
z1.b = 77.88;
```

```
punto1.x = -9.01;
```

```
punto1.y = -2;
```

```
puntoB.x = 2.435;
```

```
puntoB.y = -6;
```



# Unidad 4: Registros en Lenguaje C

## Asignación de valores a los atributos

### c) Usando funciones de lectura de datos

```
scanf("<cadena de control>", <dirección de memoria>);
```

```
fscanf(FILE*, "<cadena de control>", <dirección de memoria>);
```

El especificador de tipo de dato en la *<cadena de control>* y *<dirección de memoria>* deben ser concordantes con el tipo de dato del campo o atributo. Por ejemplo:

```
printf("\n Ingrese la parte real: ");  
scanf("%f", &z1.a);  
printf("\n Ingrese punto1.y: ");  
fscanf(stdin, "%d", &punto1.y);
```

# Unidad 4: Registros en Lenguaje C

## Acceso a los atributos o campos de un registro

### Salida por monitor

```
printf("\n Parte real: %f", z1.a);  
printf("\n Parte imaginaria: %f", z1.b);  
  
printf("\n x : %f", punto1.x);  
fprintf(stdout, "\n y: %d", punto1.y);  
  
fprintf(stdout, "\n x: %f", puntoB.x);  
fprintf(stdout, "\n y: %d", puntoB.y);
```

# Unidad 4: Registros en Lenguaje C

## Operaciones con los atributos de un registro

Con los atributos de un registro se pueden realizar todo tipo de operaciones correspondientes al tipo de dato

```
struct nroComplejo    z1, z2;
coordenadas2          puntoB , puntoA;
z1.a = 55.66 , z1.b = 77.88;
puntoB.x = 2.345 , puntoB.y = -6;
// copia de los valores de los campos
z2.a = z1.a;
z2.b = z1.b;
puntoA.x = 2 * puntoB.x;
puntoA.y = puntoB.y / 3;
```

# Unidad 4: Registros en Lenguaje C

## Operaciones con los atributos de un registro

```
coordenadas1      punto1, punto2;  
coordenadas2      puntoA;
```

```
punto1.x = 12.34;  
punto1.y = 56;
```

```
// copia de registros
```

```
punto2 = punto1;  
//asignación válida
```

```
puntoA = punto1;  
//asignación inválida. Tipos de datos diferentes
```

# Unidad 4: Registros en Lenguaje C

## Definición de registros de forma global

La definición de registros dentro de la función `main()` dificulta el uso de funciones para el tratamiento de este tipo de datos.

Esta dificultad se subsana definiendo los registros de manera global o en archivos cabeceras que luego se incluyen en la compilación.

a) Definición global de datos tipos `struct`

Unidad-4-Ej-10.c

b) Definición en un archivo cabecera

registros.h

Unidad-4-Ej-11.c

# Unidad 4: Registros en Lenguaje C

## Funciones para el tratamiento de registros

### 1.- Funciones que devuelven registros

Unidad-4-Ej-12.c

### 2.- Paso como parámetros

a) de los atributos o campos por valor

Unidad-4-Ej-13.c

b) de los atributos por referencia

Unidad-4-Ej-14.c

c) del registro por valor

Unidad-4-Ej-15.c

d) de la registro por referencia

Unidad-4-Ej-16.c

Para acceder a un atributo de un puntero a `struct` debe utilizarse el *operador flecha* ( $\rightarrow$ ) en lugar del **operador punto** ( $\bullet$ )

### 3.- Operaciones con números complejos

Unidad-4-Ej-17.c

# Unidad 4: Registros en Lenguaje C

## Registros anidados

Para la presentación del tema se dieron ejemplos simples.

Ahora se tratará ejemplo típico de un registro:

```
typedef struct {  
    char nombres[18];  
    char apellidos[15];  
    char fechaNac[20];  
    char domic[20];  
} datos;
```

# Unidad 4: Registros en Lenguaje C

## Registros anidados

Cuando se estudió "estructura de la información", era posible que esta estructura se jerarquizara. Esto ocurría cuando un atributo que se desglosaba en sub-atributos. Por ejemplo:

Entidad	Atributos		Valor
datos	apellidos		Maza
	nombres		Elena Isabel
	fechaNac	dia	12
		mes	mayo
		año	2016
	domicilio	calle	Bolívar
		número	215
		localidad	Trancas



# Unidad 4: Registros en Lenguaje C

## Registros anidados

De forma análoga a la estructura de la información jerarquizada se pueden "anidar" registros en lenguaje C. Un registro puede tener atributos que, también son registros.

Hay dos formas de anidar registros en C:

1. Definiendo los registros de menor jerarquía y declarando las variables correspondientes dentro de la definición del registro correspondiente a la entidad. Es este caso los registros internos no pueden definirse con la sentencia *typedef*
2. Definiendo los registros de menor jerarquía fuera del registro principal. De esta manera es posible usar la definición del registro interno con *typedef*

# Unidad 4: Registros en Lenguaje C

## Registros anidados

```
struct datos {  
    char        nombres[18];  
    char        apellidos[15];  
  
    struct fecha {  
        int      dia;  
        char      mes[10];  
        int      anio;  
    } fechaNac;  
  
    struct domicilio {  
        char      calle[15];  
        int        num;  
        char      localidad[20];  
    } domic;  
};
```

```
typedef struct {  
    int      dia;  
    char      mes[10];  
    int      anio;  
} fecha;  
  
typedef struct {  
    char      calle[15];  
    int        num;  
    char      localidad[20];  
} domicilio;  
  
typedef struct {  
    char      nombres[18];  
    char      apellidos[15];  
    fecha      fechaNac;  
    domicilio  domic;  
} datos ;
```

# Unidad 4: Registros en Lenguaje C

## Registros anidados

Declaración de variables:

```
int main(int argc, char *argv[])
{
    // primer caso
    struct datos persona;
    ...
return 0;
};
```

```
int main(int argc, char *argv[])
{
    // segundo caso
    datos persona;
    ...
return 0;
};
```

# Unidad 4: Registros en Lenguaje C

## Registros anidados

El acceso a los atributos de la variable será:

```
// declaración de variable
    datos persona;
    . . .
// asignación de valores
    strcpy(persona.nombres, "Elena Isabel");
    strcpy(persona.apellidos, "Maza");

    persona.fechaNac.dia = 12;
    strcpy(persona.fechaNac.mes, "mayo");
    persona.fechaNac.anio = 1996;

    strcpy(persona.domic.calle, "Bolívar");
    persona.domic.num = 215;
    strcpy(persona.domic.localidad, "Trancas");
```

# Unidad 4: Registros en Lenguaje C

## Ejemplos de funciones definidas por el usuario

1. Primer caracter de *localidad* en mayúscula y el resto en minúsculas.

```
// llamada de la función
...
    primeraMayusc(persona.domic.localidad);
...
// función
void primeraMayusc(char pal[])
{
    int i;

    pal[0] = toupper(pal[0]);
    for (i=1; i<strlen(pal); i++)
        pal[i] = tolower(pal[i]);
    return;
}
```

# Unidad 4: Registros en Lenguaje C

## Ejemplos de funciones definidas por el usuario

2. Cambiar a mayúscula el primer caracter de una segunda cadena.

```
//llamada de la función
...
    primerasegundoNombMay (persona.apellidos);
...
//función
void segundoNombMay(char s[])
{
    int i;
    for (i=1; i<strlen(s); i++)
        if (s[i] == ' ')
        {
            s[i+1] = toupper(s[i+1]);
            for(j=i+2; j < strlen(s) ; j++)
                s[j] = tolower(s[j]);
            break;
        }

    return;
}
```

# Unidad 4: Registros en Lenguaje C

## Ejemplos de funciones definidas por el usuario

### 3. Control de lectura de una fecha.

```
fecha leerFecha()  
{  
    fecha f;  
    printf("\n Ingresar Fecha de Nacimiento: ");  
    do{  
        printf("\n\t Dia ");  
        f.dia = entero();  
    }while (f.dia < 1 || f.dia > 31);  
    printf("\n\t Mes ");  
    controlAlfabetico(f.mes);  
    do{  
        printf("\n\t Anio ");  
        f.anio = entero();  
    }while (f.anio < 1916 || f.anio > 2016);  
    return f;  
}
```

# Unidad 4: Registros en Lenguaje C

## Ejemplos de funciones definidas por el usuario

### 4. Control de lectura de un domicilio.

```
domicilio leerDomicilio()
{
    domicilio d;
    printf("\n Ingresar Domicilio: ");
    printf("\n\t Calle: ");
    controlAlfabetico(d.calle);
    primeraMayusc(d.calle);
    segundoNombMay(d.calle);
    do{
        printf("\n\t Numero ");
        d.num = entero();
    }while (d.num < 0 || d.num > 10000);
    printf("\n\t Localidad: ");
    controlAlfabetico(d.localidad);
    primeraMayusc(d.localidad);
    segundoNombMay(d.localidad);
    return d;
}
```



# Unidad 4: Registros en Lenguaje C

## Ejemplos de funciones definidas por el usuario

5. Ingreso de valores para los atributos de un registro utilizando funciones definidas por el usuario,.

```
datos leerRegistro()  
{  
    datos reg;  
    int aux;  
    printf("\n\t\t Ingresar datos");  
    printf("\n\t Apellidos");  
    controlAlfabetico(reg.apellidos);  
    primeraMayusc(reg.apellidos);  
    segundoNombMay(reg.apellidos);  
    printf("\t Ingrese Nombres");  
    controlAlfabetico(reg.nombres);  
    primeraMayusc(reg.nombres);  
    segundoNombMay(reg.nombres);  
    reg.fechaNac = leerFecha();  
    reg.domic = leerDomicilio();  
    return reg;  
}
```

# Unidad 4: Registros en Lenguaje C

## Arreglos de registros

Los elementos de un arreglo también pueden ser registros. Aunque el registro es un conjunto de datos heterogéneos el arreglo continúa siendo de datos homogéneos puesto que todos sus elementos son del mismo tipo de registro.

Una declaración de un arreglo de registros podría ser:

```
...  
    datos listaReg[10];  
...
```

y el elemento genérico será:

```
listaReg[i];
```

# Unidad 4: Registros en Lenguaje C

## Arreglos de registros

### Acceso a cada atributo de un registro

Tomando las definiciones y declaraciones de ejemplos anteriores:

...

```
listaReg[1].domicilio.num = 215;  
strcpy(listaReg[1].domicilio.calle, "Bolivar");
```

...

# Unidad 4: Registros en Lenguaje C

## Punteros a registros

Cuando se trabaja con un solo registro hay dos formas de asociar al puntero con una dirección de memoria.

1. Declarando una variable del mismo tipo del puntero y asignarle la dirección de memoria.

Por ejemplo:

```
...  
    datos *ptrStruct;  
    datos persona;  
  
    ptrStruct = &persona;  
...
```

# Unidad 4: Registros en Lenguaje C

## Punteros a registros

2. Asignarle memoria dinámica al puntero. Para esto se utiliza la función:

*(tipo\_de\_dato\*) malloc(sizeof(tipo\_de\_dato))*

esta función asigna un bloque de memoria de tamaño de *tipo\_de\_dato*. Si no hay memoria disponible devuelve **null**.

Cuando se deja de utilizar la variable a la que se asignó memoria dinámica debe liberarse la memoria con la función:

**free**(*puntero*)

# Unidad 4: Registros en Lenguaje C

## Punteros a registros

### Arreglos de registros tratados con punteros a registros

Cuando se asigna memoria dinámica a un puntero a registro toma la dirección del inicio del bloque de memoria. Por lo tanto es posible asignar, en tiempo de ejecución, un bloque de memoria correspondiente a  $n$  registros. Como este conjunto de bloques están ubicados en direcciones contiguas de memoria se puede trabajar como si fuese un arreglo convencional.

```
...  
    datos *ptrStruct;  
    int    n;  
    ptrStruct =(datos*) malloc(n*sizeof(datos));  
...
```

# Unidad 4: Registros en Lenguaje C

## Punteros a registros

### Arreglos de registros tratados con punteros a registros

Las funciones definidas para los arreglos de registros convencionales son válidas para los punteros a registros. Esto es posible porque el nombre de un arreglo en lenguaje C es, en definitiva, la dirección del inicio de un bloque de memoria asignado en tiempo de compilación. Cuando se invoca una función que lleva como argumento un puntero a registro (`struct`) la dirección de este es tomada por el correspondiente parámetro de la función invocada y no diferencia si esa dirección proviene de una constante (nombre del arreglo) o de una variable a la que se asignó memoria dinámica.

# Unidad 4: Registros en Lenguaje C

## Arreglos de registros

### Arreglos como atributos de un registro

Los atributos de un registro pueden ser arreglos, para ello se debe tener en cuenta :

1. Como todo arreglo el tamaño debe determinarse en el momento de la declaración.
2. Debe prestarse especial atención en no confundir los índices del arreglo de registros con los índices de los atributos que son arreglos.