Tema 5:
Estructuras de Datos

Índice

- Estructuras de datos en Memoria Principal
  - Vectores
  - Matrices
  - Cadenas de caracteres
  - **Estructuras**
- Estructuras de datos en Memoria Externa
  - Ficheros

Estructuras

- Una **estructura** es un tipo de datos que permite agrupar bajo un mismo nombre elementos, denominados **miembros (o componentes o elementos o campos)**, que pueden ser de distintos tipos de datos y que están relacionados entre sí.

- Una estructura puede tener **cualquier número de miembros**, cada uno de los cuales tiene un **nombre único.**

- Una estructura es un tipo de datos **definido por el programador**, que se debe declarar antes de utilizar.

Estructuras. Declaración

- La forma de **declarar** una estructura es:

```
struct <nom_estructura>
{
    [tipo_dato_miembro1] [nom_miembro1];
    [tipo_dato_miembro2] [nom_miembro2];
    ...
    [tipo_dato_miembron] [nom_miembron];
};
```

- En la declaración se indica el **nombre** y el **formato** de la estructura de datos. **No** se han **creado variables**.
- Es decir, únicamente, **se declara un nuevo tipo de datos** (una plantilla para futuras variables).

Estructuras. Declaración

- Ejemplo de declaración de estructuras:

```
struct amigo{
    char nombre[10];
    char apellido[20];
    char telefono[9];
};

struct fecha{
   int dia, mes, anyo;
};
```

```
struct empleado{
        char nombre[10];
        char apellido1[10];
        char apellido2[10];
         float sueldo;
};

struct venta{
        int referencia;
        int cantidad;
};
```

Estructuras. Definición

- Para **declarar** variables de una estructura definida existen dos formas:
  - nombrándolas después de la llave de cierre de la declaración:

```
struct <nom_estructura>
{
    [tipo_dato_miembro1] [nom_miembro1];
    …
    [tipo_dato_miembron] [nom_miembron];
} lista _nomb_variable;
```

  - indicando el tipo de la estructura creada seguida por las variables correspondientes:

```
struct   nom_estructura   lista_nomb_variable;
```

Estructuras. Definición

- Ejemplo:
  - Se puede poner:

```
struct venta{
  int referencia;
  int cantidad;
} mayor_venta, ventaHoy;
```

  - O bien, por separado:

```
struct venta{
  int referencia;
  int cantidad;
};
. . .
struct  venta  mayor_venta, ventaHoy;
```

venta define la forma de la estructura, su tipo.

mayor_venta, ventaHoy son variables de la estructura.

Estructuras. Acceso a los campos

- Para **acceder** a los campos o los **miembros** de una estructura la **sintaxis** es:

nomb_var_estructura . nomb_miembro

- El **punto** proporciona el **camino directo al miembro** correspondiente.

- Un miembro de un struct, debe ser **tratados según al tipo de datos al que pertenezcan.**

Estructuras. Acceso a los campos

- Ejemplo:
  - Si tenemos declaradas las variables:

```
struct venta{
  int referencia;
  int cantidad;
} mayor_venta, ventaHoy;
```

  - Ejemplo de como se puede acceder a los miembros:

```
mayor_venta.referencia = 35;
mayor_venta.cantidad = 300;
scanf("%d", &ventaHoy.referencia);
printf("La cantidad es: %d", mayor_venta.cantidad);
```

Estructuras. Inicialización

- Se puede **inicializar** una estructura de dos formas:
  - como **parte de la definición**:

```
struct   nom_estructura   nom_variable={
  valor miembro1,
  …
  valor miembron};
```

Ejemplo:
```
struct venta ventaHoy = {
                        145,        /*referencia*/
                        3 };        /*cantidad*/
```

  - en la sección de **código del programa**, por ejemplo:

```
printf("Introd. la referencia y la cantidad vendida: ");
scanf("%d%d", &ventaHoy.referencia,  &ventaHoy.cantidad);
```

Estructuras. Ejemplo

Para **inicializar** una variable del siguiente tipo:

```
struct ejem{
  int x[3];
  int y[5];
  char nombre[10];
};
```

Se puede poner, en la declaración:

```
struct ejem ejemplo1={{2,3,4}, {1,3,5,7,9}, “una cosa”};
```

O bien mediante lecturas, por ejemplo:

```
for (i=0;i<3;i++)
   scanf(“%d”, &ejemplo1.x[i]);
for (i=0;i<5;i++)
   scanf(“%d”, &ejemplo1.y[i]);
gets(ejemplo1.nombre);
```

Ejercicio

- Dada la siguiente declaración de estructura en un programa C, que representa los datos de un empleado de una empresa:

```
struct empleado{
  char nombre[10];
  char apellido1[10];
  char apellido2[10];
   float sueldo;
};
```

Inicializar mediante lecturas una variable **emp1** de tipo **struct empleado**, se introduce un dato en cada línea. A continuación mostrar en pantalla los datos almacenados en la estructura.

Estructuras. Asignación de datos

- Una estructura **se puede asignar a otra**, exactamente igual que cualquier tipo de datos simple de los que hemos visto (int, float, char)

Ejemplo:

```
struct venta{
  int referencia;
  int cantidad;
}
struct  venta  mayor_venta, venta1, venta2;

mayor_venta = venta1;
venta1 = venta2;
```

Estructuras Anidadas

- Un **miembro** de una estructura puede ser de un tipo de datos básico (char, int …), pero también **puede ser un array o cualquier otro dato estructurado**.

- Cuando un miembro de un struct es de tipo struct, se dice que tenemos una **estructura anidada.**

- Para **acceder a un miembro anidado** se usa el operador miembro, el punto (.), tantas veces como sea necesario para alcanzarlo.

Estructuras. Ejemplo

Ejemplo de estructuras anidadas:

```
struct fecha{
  int dia, mes, anyo;
};
struct amigo{
    char nombre[10];
    char apellido[20];
    char telefono[9];
   struct fecha fechaNac;
} amigoClase;
```

Cómo acceder a:

- Inicial del nombre: `amigoClase.nombre[0]`
- Mes de la fecha de nacimiento: `amigoClase.fechaNac.mes`

Estructuras. Ejemplo

Dadas las siguientes definiciones de estructuras, indicar como se accede a los miembros de libro1:

```
struct fecha{
   int dia;
   int mes;
   int anio;
};

struct Ficha{
   char titulo[40];
   char autor[30];
   struct fecha fcompra;
   int prestado;     //1=está prestado, 0=no está prestado
} libro1;

 …
```

a) Asignarle 5 al mes de compra

b) Escribir el primer carácter del título

c) Si el libro está prestado que escriba el autor

Estructuras. Arrays de estructuras

Es bastante frecuente **usar conjuntamente estructuras y arrays**, Por ejemplo un array de libros de una biblioteca:

```
struct fecha{
 int dia;
 int mes;
 int anio;
};
struct Ficha{
 char titulo[40];
 char autor[30];
 struct fecha fcompra;
 int prestado;
};
struct Ficha biblio[100];
```

Reserva espacio de memoria para 100 variables de tipo estructura Ficha

Estructuras. Arrays de estructuras

Por ejemplo un array de libros de una biblioteca:

```
struct fecha{
   int dia;
   int mes;
   int anio;
};
struct Ficha{
   char titulo[40];
   char autor[30];
   struct fecha fcompra;
   int prestado;
} libro1;
struct Ficha biblio[100];
```

Ejemplo:
para escribir el título del segundo libro:

O la inicial del autor del tercer libro:

y el año de compra del primer libro:

Escribir todos los títulos:

Asignar libro1 al primer libro de la biblioteca:

Ejercicio

- Diseñar los tipos de datos necesarios para trabajar con una **lista** de ventas (struct venta) de un empleado de una tienda. El máximo número de ventas que puede realizar es de 60. Para cada venta se guarda la referencia del producto y la cantidad vendida.

Estructuras. Funciones

- Paso de estructuras como parámetros a funciones.
  - **Parámetros de entrada** (no se quiere modificar los valores de la estructura dentro de la función),
    - hay que definir el **parámetro formal** como en la declaración de una variable struct
    - En la **llamada a la función**, hay que indicar como parámetro real el nombre de la variable estructura.
  - **Parámetros de salida o de entrada/salida** (la función pueda modificar la estructura),
    - hay que definir el **parámetro formal** como un puntero a la estructura `(struct nomb_estructura *nomb_parametro)`. En este caso, en el código de la función, **hay que utilizar el operador ->, en lugar del operado . para acceder a los miembros de la estructura.**
    - En la **llamada a la función**, hay que indicar como parámetro real una dirección de memoria a la estructura que se desea modificar.

Es decir casi de la misma forma que con las variables simples

Estructuras. Funciones. Ejemplo

- **Paso de parámetros de entrada**

```
#include <stdio.h>
struct listin{
    int codigo;
   char nombre[25];
   char telefono[9];
};
void escribir (struct listin uno);
void main()
{
   struct listin amigo;
   scanf("%d %s", &amigo.codigo, &amigo.nombre);
   gets(amigo.telefono);      //de otra forma
   escribir(amigo);
}
void escribir (struct listin uno)
{
   printf("\n%d %s %s", uno.codigo, uno.nombre, uno.telefono);
}
```

Se puede observar que:

la declaración de la estructura tiene que ser **global** para poder declarar parámetros estructura en funciones

Estructuras. Funciones. Ejemplo

- **Paso de parámetros de salida o de entrada/salida**

```
#include <stdio.h>
struct listin{
    int codigo;
   char nombre[25];
   char telefono[9];
};
void leer (struct listin *otro);
void main()
{
   struct listin amigo;
   leer(&amigo);
   printf("\n%d %s %s", amigo.codigo, amigo.nombre, amigo.telefono);
}
void leer (struct listin *otro)
{
   scanf("%d %s", &otro->codigo, otro->nombre);
   gets(otro->telefono);
}
```

Se puede observar que:

la declaración de la estructura tiene que ser **global** para poder declarar parámetros estructura en funciones

typedef
Sinónimo de un tipo de datos

- No es específico de struct
- Se utiliza para dar nuevos nombres a tipos de datos ya existentes.
- La finalidad es conseguir un programa más documentado, más fácil de leer y por lo tanto de modificar.
- Facilita también la portabilidad de programas.
- Sintaxis:

```
typedef  tipo_datos_existente  nuevo_nombre_tipo
```

typedef Ejemplos

- Sintaxis: **typedef tipo_datos_existente nuevo_nombre_tipo**

Ejemplo1:

```
typedef int entero;  //redefinimos el tipo int
entero n1, n2;   //declaramos dos variables int
```

Ejemplo2:

```
struct fecha{
    int dia;
    int mes;
    int anio;
  };
typedef struct fecha unaFecha;
```

Para declarar variables del tipo struct, podríamos poner:

```
unaFecha  fech;
```

O bien:

```
struct fecha fech;
```

typedef Ejemplos

- Sintaxis: **typedef tipo_datos_existente nuevo_nombre_tipo**

Ejemplo 3, otra forma de hacerlo con los struct:

```
typedef struct{
     int dia;
     int mes;
     int anio;
   } Tfecha;
```

En este caso, para declarar variables del tipo struct, pondríamos:

```
Tfecha  fech;   //fech es la variable del tipo struct
```

Ejemplo 4:

```
typedef char Tcadena [10]
```

En este caso, para declarar variables del tipo cadena de 10 caracteres, pondríamos:

```
Tcadena cad;   //cad es la variable del tipo cadena
               //de 10 caracteres
```

Estructuras. Resumen

- Con una estructura se puede trabajar a dos niveles:
  - Con la **estructura completa**: en **asignaciones** y en paso de **parámetros** a las funciones.
  - y con sus **miembros** de forma independiente: se utiliza el operador de miembro (**.**): variable_estructura**.**nom_miembro

    en el caso de punteros a estructuras, es decir, en parámetros por referencia, se utiliza **->** en lugar de **.**

- Un **miembro de una estructura** puede ser de un tipo de **datos básico** (char, int …), pero también puede ser un **array u otra estructura**.
- En el caso de tener **estructuras anidadas**, para acceder a un miembro anidado se usa el operador miembro (.) tantas veces como sea necesario para alcanzarlo.