Informática

Ingeniería en Electrónica y Automática Industrial

Tipos de datos definibles en lenguaje C

Tipos de datos definibles en lenguaje C

- Estructuras
- Uniones
- Campos de bits
- Definición de tipos con `typedef`

Estructuras (I)

- Una estructura es un tipo de variable especial que permite almacenar datos de tipos diferentes con un identificador común.
- Las variables que forman parte de la estructura se llaman *elementos de la estructura*.
- **Definir una estructura** consiste en crear el tipo de estructura:

```
struct nombretipoestructura
{
        tipodato1 nombrelemento1;
        tipodato2 nombrelemento2;
        …
        tipodatoN nombrelementoN;
};
```

- La definición de un tipo de estructura no crea ninguna variable ni ocupa memoria
- **Declarar una estructura** consiste en crear una variable de un tipo determinado de estructura:

```
struct nombretipoestructura variablestructura;
```


Estructuras (II)

- Se pueden declarar variables de estructura en la misma sentencia que la declaración:

```
struct nombretipoestructura
{
        tipodato1 nombrelemento1;
        tipodato2 nombrelemento2;
        ...
        tipodatoN nombrelementoN;
} listavariablestructura;
```

  - `nombretipoestructura` es el identificador del tipo de estructura que se está definiendo
  - `tipodatoX` representa al diferente tipo de dato de cada elemento
  - `nombrelementoX` son los identificadores propios de los elementos de la estructura
  - `listavariablestructura` representa al identificador (o una lista de estos) de la variable de estructura que se va a crear.

Estructuras (III)

- La definición de las estructuras se suele hacer fuera de la función `main()` en los archivos cabecera `.h`
- Los elementos de una estructura se almacenan en posiciones consecutivas de memoria (¡no del todo cierto!)
- La cantidad de memoria ocupada por una variable estructura (más o menos las suma de la ocupada por sus elementos), puede obtenerse con el operador `sizeof`)
- Ejemplos

```
struct Militar   /* Tipo de estructura */
{
         char nombre[40];
         char apellidos[80];
         unsigned edad;
         float estatura;
         unsigned long teléfono;
} cabo, sargento, teniente;
struct Militar capitan; /* cabo, sargento, teniente y */
                        /* capitan son variables      */
                        /* estructuras del tipo militar */
```


Estructuras (IV)

- Salvo la copia de una variable estructura en otra del mismo tipo (mediante el operador de asignación) **no se pueden realizar operaciones con estructuras**, deben realizarse con sus elementos y por separado
- La referencia a un elemento de una estructura se hace mediante las etiquetas de la estructura y del elemento unidas por el operador punto «`.`»

```
variablestructura.nombrelemento
```

  - `variablestructura` es el nombre de la variable de estructura en la que se encuentra el elemento que se quiere referenciar.
  - `nombrelemento` es el nombre del elemento de la estructura
  - El operador punto «`.`» une los dos identificadores en uno solo
  - En caso de anidación de estructuras, el operador punto aparecerá entre identificadores sucesivos

Estructuras (V)

- La dirección en memoria de un elemento de una estructura se obtiene aplicando el operador de dirección a la referencia a ese elemento
  `&variableestructura.nombrelemento`
- Una variable de estructura se asimila a una ficha de una base de datos; los elementos de la estructura son los campos de la ficha.
- Ejemplos:

```
/* Inicialización de algunos campos
   de la variable cabo de tipo
   estructura militar */
gets(cabo.nombre)
cabo.telefono = 916830106;
scanf("%f", &cabo.estatura);
```


Uniones (I)

- Constituyen una porción de memoria compartida por variables de diferentes tipos
- Es una forma de interpretar los mismos datos de diferente manera.
- Se definen y declaran **igual que las estructuras** cambiando la palabra reservada `struct` por `union`
- La referencia a los elementos de una unión se hace con el operador punto «`.`», del mismo modo que en las estructuras

Uniones (II)

- Definición de un tipo unión

```
union nombretipounion
{
        tipodato1 nombrelemento1;
        tipodato2 nombrelemento2;
        ...
        tipodatoN nombrelementoN;
};
```

- Creación de una variable unión de un tipo previamente definido:

```
union nombretipounion variableunion;
```


Uniones (III)

- La cantidad de memoria necesaria para almacenar una unión es la misma que la que ocupa el elemento de mayor tamaño
- Es responsabilidad del programador conocer el dato almacenado en la variable de tipo unión
- Ejemplos

```
union Talla
{
        int numero;             /* 38, 40, 42 */
        char letra;             /* P, M, G */
        char letras[4];         /* L, XL, XXL */
} camiseta, camisa, jersey;
camiseta.numero = 44;
scanf("%c",&camiseta.letra);
gets(camiseta.letras);
/* Primero se almacena el entero 44, después se almacena la
   letra leída con scanf() y al final se guarda una cadena de
   hasta 3 caracteres (más el nulo). Los anteriores se pierden */
/* Podría utilizarse para guardar la talla en un formato
   diferente en cada variable */
```

Campos de bits (I)

- Un campo de bits es un tipo especial de elemento de una estructura en el que se puede definir su tamaño en bits.
- Definición de un campo de bits

```
tipodato nombrecampo:longitud;
```

  - `tipodato` el el tipo de dato, que sólo puede ser entero (`int`, `signed`, `unsigned`, `char`, `short`, `long`, ...)
  - `nombrecampo` representa el nombre del elemento que va a ser un campo de bits
  - `longitud` representa a un entero positivo que indica el número de bits de ese campo
- En una estructura se pueden declarar elementos ordinarios o campos de bits, indistintamente

Campos de bits (II)

- Restricciones en los campos de bits:
  - Su almacenamiento en memoria, depende de la máquina y del compilador.
  - No se puede obtener la dirección en memoria de un campo de bits.
  - Su tamaño no debe exceder del tamaño de un entero
- Ejemplo

```
struct Campobit{
                    int entero;
                    unsigned sietebits:7;
                    char letra
                } trescampos;
trescampos.sietebits = dato;
```

- Los campos de bits
  - Facilitan las operaciones a nivel de bits
  - Facilitan el almacenamiento de variables lógicas (tipo booleano)
  - Aumenta el número de operaciones de la CPU
  - No suponen un ahorro de memoria importante

Definición de tipos con `typedef`

- Ya vimos que la expresión `typedef` permite dar un nombre particular a cualquier tipo de dato válido. Se puede usar en estructuras para más comodidad.
- Sintaxis:

  `typedef tipodatovalido` *`nuevonombre`*`;`
  - `tipodatovalido` representa un tipo de dato válido
  - `nuevonombre` es el nuevo identificador para ese tipo de dato
- Ejemplo:

```
typedef struct Militar {
                          ...
                       } Midato;
Midato soldado, cabo, sargento;
```

V1.2 © Autores 13