**UNIDAD 9: ESTRUCTURAS Y UNIONES EN C Y CAMPOS DE BIT**

REPASO

Los int, char, float con todos sus calificadores, junto con el tipo void y los punteros son conocidos como tipos escalares

Los arreglos, vistos en la Unidad 6, forman parte de los tipos agregados

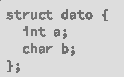
Los arreglos sirven para almacenar datos relacionados del mismo tipo bajo un mismo nombre. Existe otra forma de datos de tipo agregado…

ESTRUCTURAS

Las estructuras son tipos de datos derivados que agrupan datos relacionados que pueden ser de distinto tipo

Ej. una estructura que tenga un entero y un char

ESTRUCTURAS: DEFINICIÓN



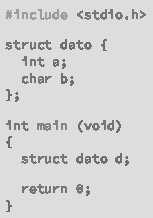
| * struct es la palabra reservada para indicar que se define una estructura * dato es la etiqueta de la estructura * entre llaves se definen los miembros de la estructura (la cantidad que se quiera) * los miembros son variables de cualquier tipo, se definen con tipo y nombre, terminan en punto y coma (;) * los miembros pueden ser de cualquier tipo, incluso arreglos, punteros u otras estructuras. * la definición de la estructura termina con punto y coma (;) |
| --- |

La definición de una estructura no asigna memoria, solo crea un nuevo tipo de datos que puede ser usado para definir variables

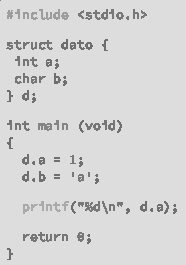
Para definir una variable con el nuevo tipo, se antepone al nombre de la variable la palabra struct y el nombre de la etiqueta de la estructura



en este ejemplo se define una variable d de tipo struct dato definido anteriormente



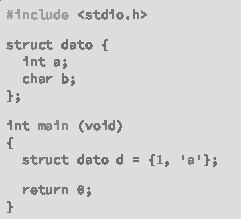
La definición de las variables de tipo struct dato deberán hacerse, luego de la definición de la estructura



Se pueden definir variables en la misma definición de la estructura

En este caso, d es una variable global

INICIALIZACIÓN

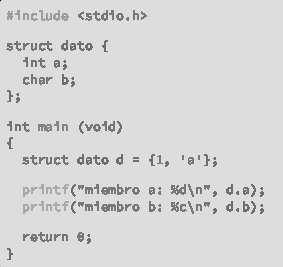


| * Al igual que en los arreglos se inicializan entre llaves, donde los elementos se asignan en el orden que están definidos dentro de la estructura * Si se desea hacer todos los elementos iguales a cero, se coloca entre las llaves un cero * Si hay menos inicializadores que miembros, los que faltan son puestos en cero * Si hay más inicializadores que miembros, se genera una advertencia (o error con -pedantic-errors) |
| --- |

OPERADOR PUNTO

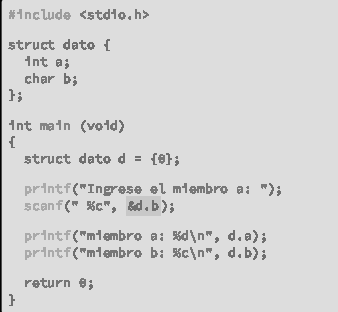
En el caso de los arreglos se usaban los [] (corchetes) para identificar cada elemento que componía el arreglo

En el caso de las estructuras, se utiliza el operador punto y el nombre del miembro al que se quiere acceder



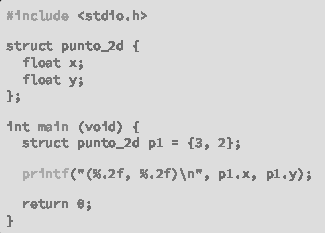
La variable d, al ser de tipo struct dato tiene miembros llamados a y b a los que se accede mediante el operador punto (ver los printf)

d.a es un entero y se la puede usar usar como a cualquier entero, al igual que d.b es usado como cualquier carácter

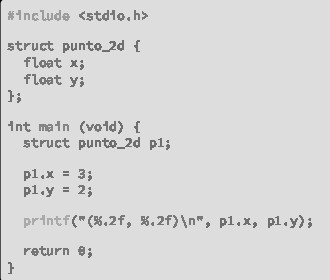


Es importante actualizar la tabla de precedencia de operadores (ver scanf (“ %c”, &d.b); )

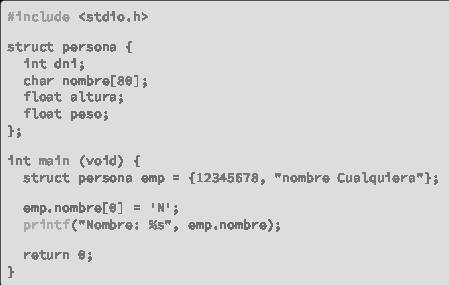
ACCESO A LOS MIEMBROS DE UNA ESTRUCTURA



Para imprimir en pantalla



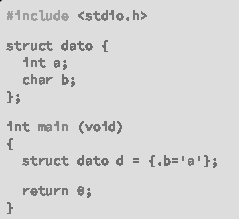
Asignar valores a los miembros de la estructura



Arreglo de caracteres en una estructura

A igual precedencia, con asociatividad desde la izq. primero opera el punto

INICIALIZACIÓN (EXTRA)

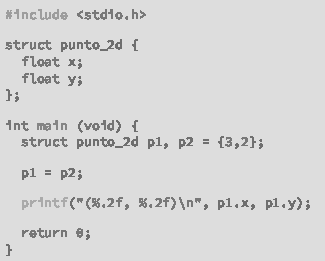


| * También se puede inicializar un elemento usando el operador punto y el miembro que corresponde. Se conocen como inicializadores designados. |
| --- |

OPERACIONES PERMITIDAS

| Asignación | Tomar dirección de memoria con & | Desreferenciar con \* | Acceder a miembros con . o -> | Operador sizeof |
| --- | --- | --- | --- | --- |

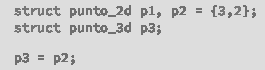
OPERACIONES PERMITIDAS: ASIGNACIÓN





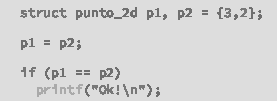


Se puede hacer asignación si y sólo si se trata del mismo tipo de estructuras, de lo contrario hay error de compilación



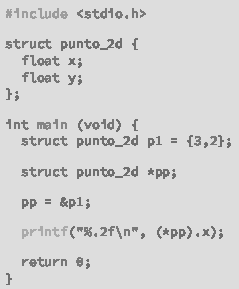
OPERACIONES NO PERMITIDAS

No se permite el uso de los operadores de relación (==, !=, >, <, etc)





PUNTEROS A ESTRUCTURAS

De la misma manera que en variables de tipo escalar se usa el asterisco entre el nombre y el tipo, en estructuras se usa el asterisco, pero recordando que el tipo incluye la palabra reservada struct

En la expresión (\*pp).x deben usarse los paréntesis para que la desreferencia sea correcta

Debido al orden de precedencia del operador punto deben usarse paréntesis para que la desreferencia del puntero se realice primero.

De lo contrario el compilador da un error, ya que el operador punto espera una estructura y un miembro al que acceder, no un puntero.

Para simplificar la notación y disminuir la posibilidad de errores se usa el operador flecha (->)



En lugar de se puede usar

El operador flecha espera un puntero a una estructura a la izquierda y un miembro de esa estructura a la derecha

**ESTRUCTURAS Y UNIONES EN C (SEGUNDA PARTE)**

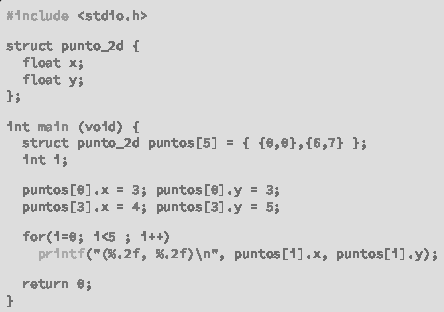
ARREGLOS DE ESTRUCTURAS

Para definir un arreglo de estructuras simplemente se usa el corchete en el nombre de la variable



En el ejemplo se crea un arreglo de 10 elementos, cada uno de tipo struct punto\_2d

Se accede a los miembros de cada estructura, primero a través del uso del índice que corresponde en el arreglo



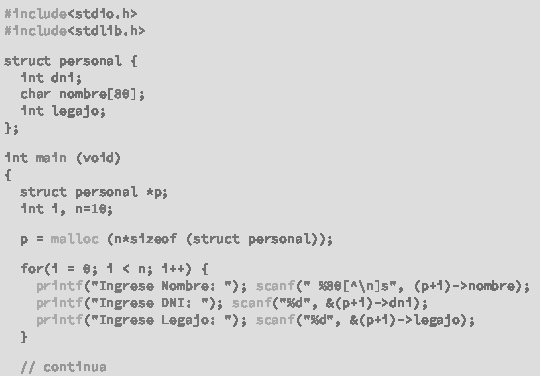


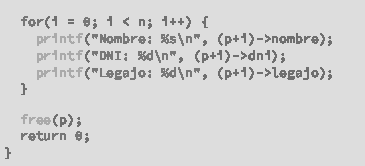
En el caso de se opera primero el corchete por estar más a la izq.

La variable puntos antes que nada es un arreglo por lo que el elemento se accede con los corchetes.

Luego, el elemento accedido con los corchetes es de tipo struct puntos\_2d, por lo que entonces se puede acceder al miembro de la estructura con el operador punto.

ARREGLOS ASIGNADOS DINÁMICAMENTE





FUNCIONES Y ESTRUCTURAS

A las funciones se pueden pasar:

* miembros de la estructura,
* la estructura completa, o
* un puntero a una estructura (o arreglo de estructuras)

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

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

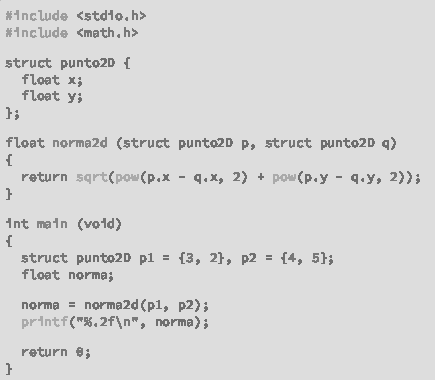
Por ejemplo, si la función espera enteros



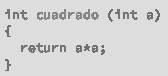
el llamado se puede hacer



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



*(antes de seguir, repaso de Unidad 6)*

Supongamos la función 

puede ser llamada con una variable

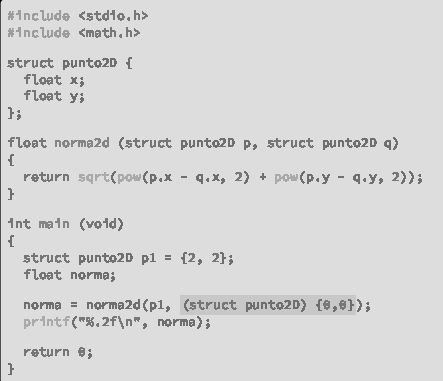
o un literal

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



Ver norma = norma2d(p1, (struct punto2D) {0,0});

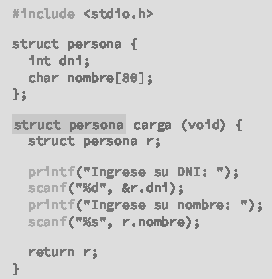
Cuando se pasan arreglos de estructuras a funciones, son automáticamente por referencia como todos los arreglos…

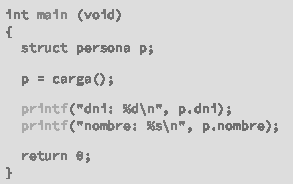
Pero si una estructura tiene arreglos estos se pasan por copia, como todos los elementos de la estructura.

Siempre se dijo que las funciones solo pueden devolver un solo valor...esto todavía es así, pero se puede devolver una estructura

Entonces de esta manera se pueden devolver varios valores, siempre que haya coincidencia entre el tipo de estructura devuelto y la variable que recibe la estructura

FUNCIONES QUE DEVUELVEN ESTRUCTURAS





TYPEDEF

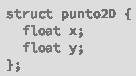
La palabra clave typedef prevé un mecanismo para generar sinónimos o alias.



| * typedef es la palabra clave que indica que se va a definir un alias * a continuación se coloca el tipo de datos del que se quiere generar un sinónimo (el “uint”) * se finaliza con el alias (y el punto y coma) |
| --- |

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

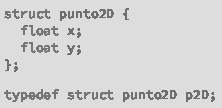
Se usa mucho en estructuras para simplificar notación...

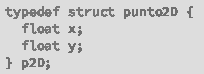
supongamos la definición de una estructura

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



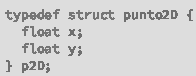
Se puede usar typedef de varias formas:





o

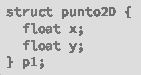
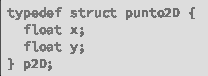
| Entonces teniendo: | el prototipo: |
| --- | --- |



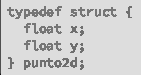
puede pasar a:



Notar que la definición + el alias con typedef tiene una sintaxis parecida a la declaración de una variable global p1



Hay que tener cuidado de no confundir estos casos



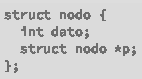
Las estructuras pueden ser definidas sin etiquetas si se definen usando typedef

o se usan para declarar una variable global



Cuidado con la diferencia: punto2d puede servir para definir nuevas variables, p2d es una variable, y no puede haber otra igual

ESTRUCTURAS AUTOREFERENCIADAS

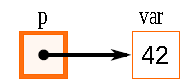


Se usan para generar estructuras de datos complejos en C.

Se llaman autoreferenciadas porque uno (o más) de sus miembros es un puntero a una estructura del mismo tipo que se está definiendo.

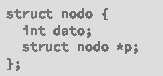


Si se representa un int como un bloque...



...se puede representar un puntero con otro bloque del mismo

color (del mismo tipo) y hacer que el apunte a la variable

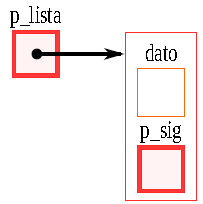
De la misma forma, si se tiene definida una estructura autoreferenciada de tipo struct nodo, se puede asignar memoria suficiente para contenerla…

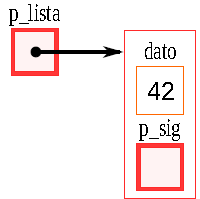


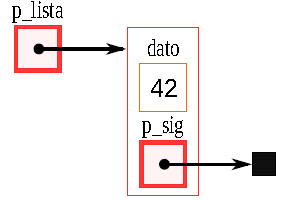
Haciendo la comparación del principio:



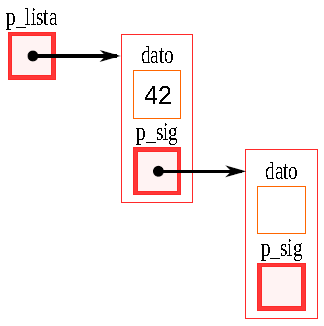


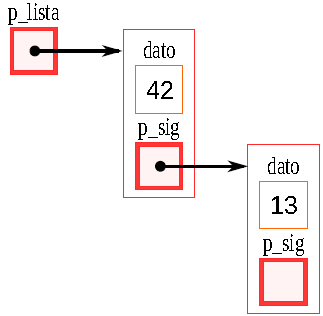


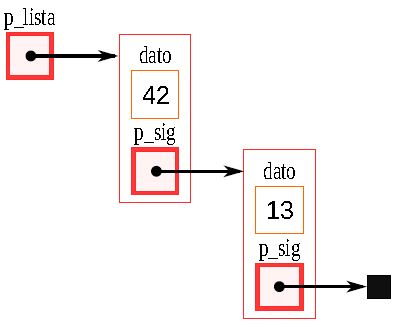






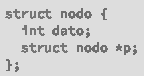






La parte práctica de estructuras autoreferenciadas NO entra en el parcial o final

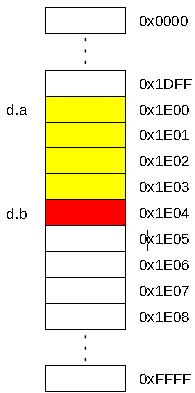
Se ve en Informática 2



¿Qué hay que saber?

Que las estructuras autoreferenciadas tienen un miembro que es un puntero al mismo tipo de la estructura que se está definiendo.

*(para fijar concepto antes de seguir)*

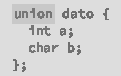


Cada miembro de la estructura se coloca después del otro, en la medida que se pueda mantener a todos los datos alineados



UNIONES

Las Uniones son tipos derivados como las estructuras y de similar definición.

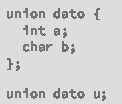


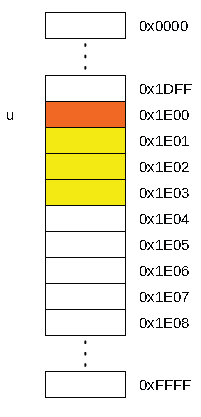
La palabra clave para definir uniones es union

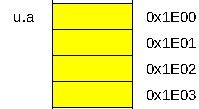
Tiene miembros como las estructuras y se acceden con el operador punto (.) o flecha (->)

La gran diferencia...

Cada miembro de la unión comparte la posición memoria con los otros





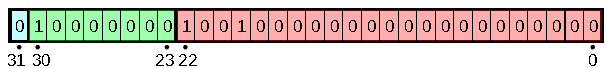


El tamaño en memoria de la unión es suficiente para almacenar el miembro que ocupe mayor espacio

Pero compartir memoria trae problemas si se accesa de forma equivocada

*(repaso de Unidad 2)*

Si se tiene un número real, por ej. 3.125 almacenado en formato IEEE754 de simple precisión, en bits sería



pero si se interpretan esos mismos bits



como un entero... se lee 1078460416 ya que:

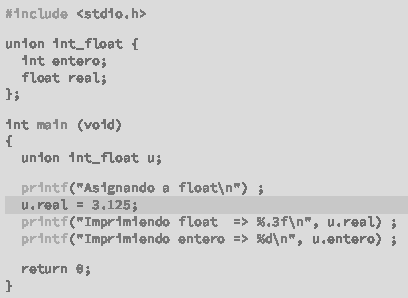
2^30+2^22+2^19 = 1078460416

Entonces, para definir una unión se sigue el mismo formato que la estructura pero con la palabra clave union

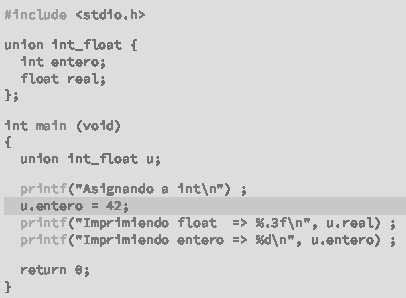
Puede tener cualquier tipo y cantidad de miembros, los cuales se definen cada uno con su tipo y nombre finalizando cada línea con punto y coma (;)

La etiqueta y los nombres de los miembros deben respetar las reglas de los identificadores

Toda la definición debe terminar con punto y coma (;)



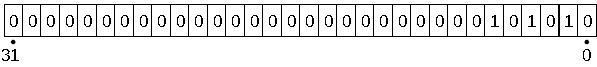




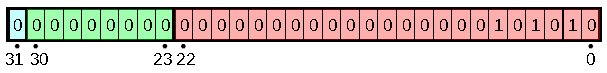


¿Por qué imprimir el float da 0?

El entero 42 se almacena como



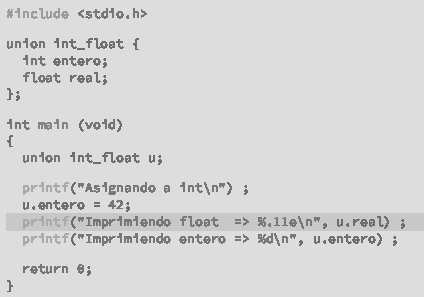
Si se lee como un float



al convertir este número se obtiene 5.88545355016e-44

y en notación decimal se escribe 0. y cuarenta y tres ceros antes del 5

Se puede ver el valor almacenado real con %e





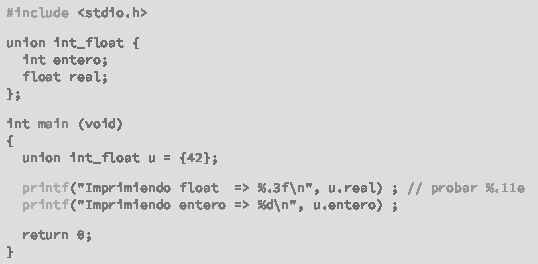
En resumen, si se asigna un valor a un miembro de la unión, este se guarda siguiendo las reglas de ese tipo de datos

Pero siempre recordar que todos los valores, del valor que sean se guardan en bits

Si se lee desde un miembro de un tipo diferente, se interpretan los bits almacenados con las reglas de este último tipo de miembro

UNIONES: INICIALIZACIÓN

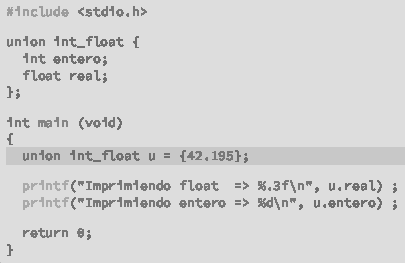
Las uniones se inicializan en la declaración con llaves





Solo se puede inicializar un valor, ya que es compartido por todos los miembros

El formato de carga depende de cual sea el valor que esté primero en la definición de la unión



Aunque se inicialice con un número real, como el primer miembro es entero, se produce un truncamiento en la inicialización, y solo se inicializa la variable entera con un 42



Se puede inicializar un miembro que no sea el primero en la definición usando los inicializadores designados

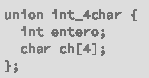
UNIONES: OPERACIONES PERMITIDAS

* Se permite asignar uniones a otras uniones del mismo tipo
* Se puede usar el operador dirección de memoria (&) para devolver la dirección de la estructura (que es la misma que la de cualquiera de sus miembros)
* El operador punto (.) y el operador flecha (->) se usan de la misma manera que en las estructuras

UNIONES: OPERACIONES NO PERMITIDAS

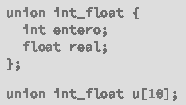
* No se permiten las comparaciones (==, !=, <, > etc.)
* No se permiten operaciones aritméticas (+, -, \*, /)

UNIONES: POSIBLES USOS

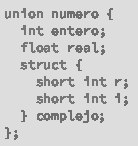
Las uniones pueden tener arreglos como miembros (los cuales son pasados por valor a las funciones)

No es necesario que el arreglo tenga el mismo tamaño que el otro u otros miembros

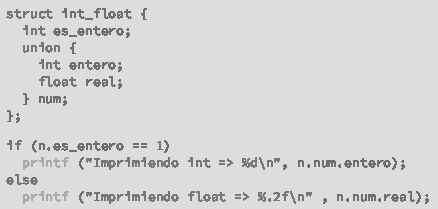
En cualquier caso el tamaño de la unión será igual al tamaño en memoria de miembro que ocupe más espacio



Se pueden hacer arreglos de uniones

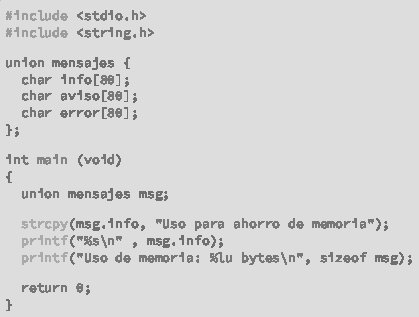


Las uniones pueden tener estructuras como miembros



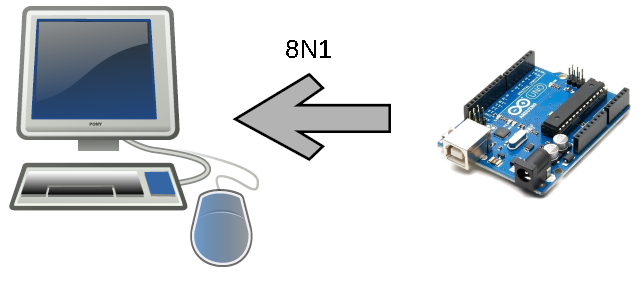
Las estructuras pueden tener uniones como miembros

UNIONES: USOS - AHORRO DE MEMORIA



UNIONES: USOS - EJEMPLO: COMUNICACIÓN SERIE

Visualizar en la PC datos provenientes de un sistema embebido con sensores de magnitudes reales (temperatura, humedad, aceleración, etc.) enviando los valores mediante serie configurado como 8N1



El problema radica en que el número real almacenado como IEEE754 de simple precisión tiene 4 bytes

Sin embargo el bus de comunicación tiene solo 1 byte de ancho

En este caso se puede usar una unión entre un float y un arreglo de cuatro char

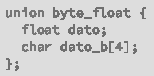
El procedimiento sería: cargar el valor real en el miembro float y enviar byte por byte del arreglo por el puerto serie

Del otro lado, en la PC, se recibe byte por byte y se los va colocando en el arreglo char

Luego, para visualizar, se accede al miembro float

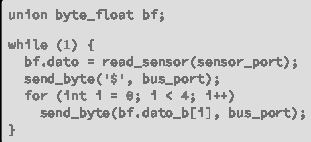
Los valores de cada char no tendrían sentido, sin embargo al leerlos como float se recupera el valor original

A ambos lados del bus se usará la misma unión

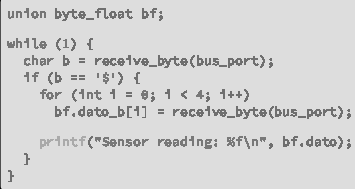


y una función para enviar bytes llamada send\_byte y para recibir bytes llamada receive\_byte

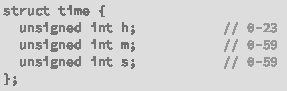
Del lado de los sensores se hace el envío suponiendo que hay una función que lee el sensor llamada read\_sensor



Entonces, la forma más sencilla sería, del lado del sensor…



y del lado del receptor

CAMPOS DE BIT

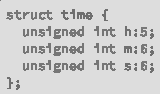
Las variables en general, y las estructuras en particular, pueden ser capaces de contener valores más grandes de lo que realmente necesitan

El rango de unsigned int es 0 a 4294967295





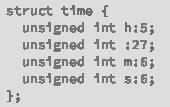
DEFINICIÓN



| * se usan (:) para definir cuántos bits se deben usar * el tamaño en bits no debe exceder el tamaño del tipo usado * solo se pueden declarar miembros int, unsigned int, char y unsigned char * el tamaño de bits puede ser cero * no se puede usar el operador & para tomar la dirección de los bits * la organización de un campo de bits en la memoria depende del compilador |
| --- |

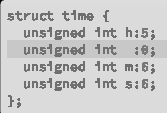






También existen los campos de bit sin nombre que se utilizan para agregar padding (completar lugares para alinear datos en la memoria).

Se utiliza un miembro de la estructura sin nombre, y luego de los dos puntos (:) el padding

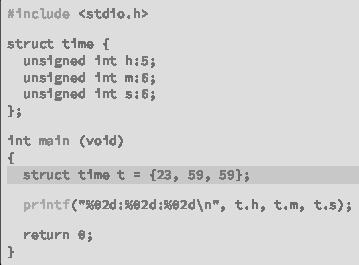


También se puede poner ancho cero a un campo de bit sin nombre.

Esto genera el padding necesario para alinear el siguiente campo de bit en la siguiente unidad de almacenamiento.

INICIALIZACIÓN Y ACCESO A UN CAMPO DE BITS

Los campos de bit se inicializan igual que las estructuras usando llaves.



Incluso usando inicializadores designados.

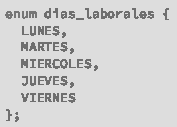
El acceso se realiza mediante el operador punto (o flecha si es

un puntero)

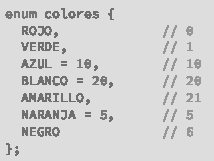
CONSTANTES DE ENUMERACIÓN

Son un conjunto de constantes enteras representadas por identificadores.

Su definición es semejante a las estructuras a menos que se explicite lo contrario, se enumeran las constantes desde el 0 (cero)

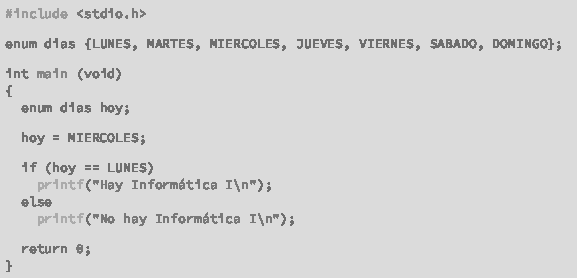


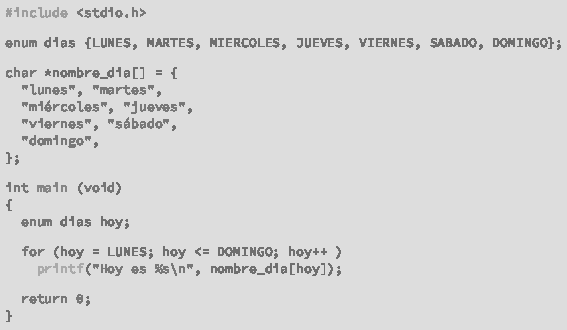
| * la palabra clave es enum * a continuación se coloca la etiqueta (opcional) * las constantes van separadas de comas (,) * se recomienda el uso de mayúsculas para las constantes |
| --- |



En la definición se puede asignar cualquier valor entero usando el igual (=), a partir de ahí, los siguientes se incrementan de a 1

USOS





## TABLA DE PRECEDENCIA DE OPERADORES (MÁS RECIENTE ENCONTRADA)

## 