

## Repaso de programación en C: tipos de datos definidos por el usuario 2ª parte

*Uniones:* compartir espacio de memoria y extraer partes de una variable mayor

- 1) Mediante una unión acceder a los bytes individuales de una variable de tipo float con un vector de 4 char.

*Campos de bits:* permiten definir el tamaño en bits de campos de una estructura

- La longitud de un campo de bits no podrá exceder el tamaño de su dato base: unsigned char a : 9 -> no es posible, unsigned short int a:9 -> ok
  - Los campos de bits se alinean en bytes, es decir, su tamaño total será múltiplos de 8 bits
  - Un campo de bits de longitud cero genera que el próximo campo inicie en el próximo byte
  - Cada campo se ubica uno a continuación del otro
  - Se pueden incluir campos sin nombre los cuales servirán para acomodar los campos siguientes en la posición deseada
- 2) Definir una estructura con campos de bits para almacenar una fecha en formato dd/mm/aa ocupando la menor cantidad de memoria posible
  - 3) Mediante una unión de estructuras acceder a los bytes, bits y nibbles de una variable de 16 bits

Ejemplo de una posible implementación:

```
typedef union
{
    uint8_t val;
    struct {
```

```

        uint8_t _0:1;
        uint8_t _1:1;
        uint8_t _2:1;
        uint8_t _3:1;
        uint8_t _4:1;
        uint8_t _5:1;
        uint8_t _6:1;
        uint8_t _7:1;
    } bits;
    struct
    {
        uint8_t _0:4;
        uint8_t _1:4;
    } nibbles;
    } byte8_t;
typedef union
{
    unsigned short int ent;
    uint16_t val;
    struct {
        byte8_t _1;
        byte8_t _0;
    } bytes;
} byte16_t;

```

4) Ejecute el siguiente programa y explique su salida:

```

struct A
{
    unsigned int a:10;
    unsigned int b:10;
    unsigned int c:10;
    unsigned int d:10;
    unsigned int e:10;
};

int main(void) {
    struct A a1;
    printf("%d\n", sizeof(a1));

    unsigned int a2[5];
    printf("%d", sizeof(a2));

    return 0;
}

```

- 5) A continuación se muestra una tabla que representa dos registros de 32 bits de configuración de un microcontrolador PIC32 (<http://ww1.microchip.com/downloads/en/devicedoc/61105f.pdf>, pag. 7):







**Table 14-2: Timers SFR Summary**

Register Name <sup>(1)</sup>	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
T1CON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	SIDL	TWDIS	TWIP	—	—	—
	7:0	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—
TxCON	31:24	—	—	—	—	—	—	—	—
	23:16	—	—	—	—	—	—	—	—
	15:8	ON	—	SIDL	—	—	—	—	—
	7:0	TGATE	TCKPS<2:0> <sup>(2)</sup>			T32 <sup>(3)</sup>	—	TCS	—

Implemente las estructuras T1CON y TxCON con campos de bits para representar ambos registros y cada uno de sus campos en la posición adecuada. Verificar las posiciones de cada campo asignando valores a cada uno y mostrando el contenido con la función para mostrar números binarios al final de este documento.

## Operadores a nivel de bits:

### Operadores C que trabajan a nivel de bit:

-  **&** AND (“y” a nivel de bit)
-  **|** OR (“ó” a nivel de bit)
-  **^** XOR (“ó exclusiva” a nivel de bit)
-  **~** NOT (“no” a nivel de bit)
-  **>>** desplazamiento de bits a la derecha
-  **<<** desplazamiento de bits a la izquierda

### Tabla de verdad de las operaciones AND, OR y XOR:

1	&	1	=	1
1	&	0	=	0
0	&	1	=	0
0	&	0	=	0

1		1	=	1
1		0	=	1
0		1	=	1
0		0	=	0

1	^	1	=	0
1	^	0	=	1
0	^	1	=	1
0	^	0	=	0

### Ejemplos:

Operación en C (hex.)	Resultado (hex.)	Operación en binario	Resultado (binario)
0xF5 & 0x5A →	0x50	11110101b AND 01011010b →	01010000b
0xF5   0x5A →	0xFF	11110101b OR 01011010b →	11111111b
0xF5 ^ 0x5A →	0xAF	11110101b XOR 01011010b →	10101111b
~ 0xF5 →	0x0A	NOT 11110101b →	00001010b
0x75 >> 2 →	0x3D	01110101b >> 2 →	00011101b
0xF5 << 2 →	0xD4	11110101b << 2 →	11010100b

### Código de ejemplo:

```
uint8_t bit = 0; // b0,b1,b2,b3,b4,b5,b6,b7
//poner a 1 el bit 5
bit |= 1<<5;
if(bit & (1<<5)) //chequeo si el bit 5 esta en 1
    printf("bit 5 en 1\n");
bit = 0xFF;
//poner a 0 el bit 5
```

```

bit &= ~(1<<5);
number_to_binary(bit);
if(~(bit & (1<<5)))//chequeo si el bit 5 esta en 0
    printf("\nbit 5 en 0\n");

```

- 6) Mediante una máscara poner los bits 2, 5 y 6 de una variable a cero (AND)

```

//poner a cero los bits 2 5 y 6
uint8_t dato = 0xFF;
uint8_t mascara = (1<<2)+(1<<5)+(1<<6);
dato &= ~mascara;

```

- 7) Mediante una máscara, poner los 3 bits más altos de una variable char a uno

- 8) Mediante una máscara, invertir los bits 3 8 y 12 de una variable de 16 bits

- 9) Mediante una máscara determinar si los bits 8 9 y 12 de una variable de 16 bits están en uno

- 10) En los bits 10, 9 , 8 y 7 de una variable *short int* se encuentra un valor que se desea extraer y almacenar en una variable *unsigned char*. Implementar el código necesario para extraer el valor y mostrarlo en pantalla con printf.

- 11) Mediante una máscara determinar si los bits 8 9 y 12 de una variable de 16 bits están en uno

- 12) Se tiene almacenado en una variable de 16bits la información de temperatura, presión y humedad de un sensor ocupando 5 bits cada una en el orden indicado, mediante desplazamientos extraer la información en 3 variables de tipo int.

```

uint16_t dato=0xFFFF;
//mascaras:
uint16_t m_temp = (0b11111);
uint16_t m_presion = (0b11111)<<5;
uint16_t m_humedad = (0b11111)<<10;

uint16_t temp, presion, humedad;
temp = dato & m_temp;
presion = (dato & m_presion)>>5;
humedad = (dato & m_humedad)>>10;

```

- 13) Implementar el mismo problema anterior utilizando una estructura de 16 bits con 3 campos de 5 bits para cada dato.
- 14) La siguiente funcion permite mostrar en pantalla una variable de 8 bits (uint8\_t) como una cadena de caracteres 0 y 1. Explicar como funciona:

```
void number_to_binary(uint8_t x)
{
    char b[9];
    b[8] = '\0';

    uint8_t z;
    int w = 0;
    for (z = 1; w < 8; z <= 1, ++w)
    {
        b[w] = ((x & z) == z) ? '1' : '0';
    }

    printf("%s",b);
}
```

Ejemplo de uso:

```
int main(void)
{
    uint8_t c = 0xA5;
    printf("binario: "); number_to_binary(c);
    return 0;
}
```

Salida:

binario: 10100101

15) El siguiente programa permite comprender mejor como se almacenan los bits en una variable. Su salida depende exclusivamente del procesador en el que se ejecute.

```
int main(void)
{
    int a;
    typedef struct {
        int b7 : 1;
        int b6 : 1;
        int b5 : 1;
        int b4 : 1;
        int b3 : 1;
        int b2 : 1;
        int b1 : 1;
        int b0 : 1;
    } byte;

    byte ab0 = {0,0,0,0,0,0,0,1};
    a = *(int*)&ab0;
    printf("ab0 is %x ",a); number_to_binary(a); printf("\n");

    byte ab1 = {0,0,0,0,0,0,1,0};
    a = *(int*)&ab1;
    printf("ab1 is %x ",a); number_to_binary(a); printf("\n");

    byte ab2 = {0,0,0,0,0,1,0,0};
    a = *(int*)&ab2;
    printf("ab2 is %x ",a); number_to_binary(a); printf("\n");

    byte ab3 = {0,0,0,0,1,0,0,0};
    a = *(int*)&ab3;
    printf("ab3 is %x ",a); number_to_binary(a); printf("\n");

    byte ab4 = {0,0,0,1,0,0,0,0};
    a = *(int*)&ab4;
    printf("ab4 is %x ",a); number_to_binary(a); printf("\n");

    byte ab5 = {0,0,1,0,0,0,0,0};
    a = *(int*)&ab5;
    printf("ab5 is %x ",a); number_to_binary(a); printf("\n");

    byte ab6 = {0,1,0,0,0,0,0,0};
    a = *(int*)&ab6;
    printf("ab6 is %x ",a); number_to_binary(a); printf("\n");

    byte ab7 = {1,0,0,0,0,0,0,0};
    a = *(int*)&ab7;
```

```
printf("ab7 is %x ",a); number_to_binary(a); printf("\n");

return 0;}
```

- 16) Interpretar la salida del siguiente programa, la manera en que se muestra el campo d de la estructura está relacionada con el concepto de Endianness (<https://es.wikipedia.org/wiki/Endianness>)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct tagT{

    int a:4;
    int b:4;
    int c:8;
    int d:16;
}T;

int main()
{
    char data[]={0x12,0x34,0x56,0x78};
    T *t = (T*)data;
    printf("a =0x%x\n" ,t->a);
    printf("b =0x%x\n" ,t->b);
    printf("c =0x%x\n" ,t->c);
    printf("d =0x%x\n" ,t->d);
    int i;
    for(i = 0; i<4; i++)
        printf("0x%X, ",data[i]);

    return 0;
}
```