

Capt. 10 - Estructuras, uniones, manipulaciones de bits y enumeraciones¹

Esquema

10.1 Introducción

10.2 Definiciones de la estructura

10.3 Inicialización de estructuras

10.4 Acceso a los miembros de las estructuras

10.5 Uso de estructuras con funciones

10.6 Typedef

10.7 Ejemplo:

Simulación de barajado y reparto de cartas de alto rendimiento

10.8 Uniones

10.9 Operadores de Bitwise

10.10 Campos de bits

10.11 Constantes de enumeración



10.1 Introducción

- Estructuras
 - Colección de variables relacionadas (agregadas) bajo un mismo nombre
 - Puede contener variables de diferentes tipo de datos
 - Comúnmente usado para definir registros a ser guardadas en archivos
 - Combinado con punteros, puede crear:
 - listas enlazadas, pilas, colas, y árboles



10.2 Definición de Estructuras (Struct)

- Ejemplo

```
struct carta {  
    char * cara;  
    char * palo;  
};
```

- **struct** introduce la definición para estructura **carta**
- **carta** es el *nombre de la estructura* y es utilizado para declarar variables de *tipo se estructura*.
- **carta** contiene dos miembros de tipo **char *** – **cara** y **palo**



10.2 Definición de Estructuras (struct) (II)

- Información de Estructura (Struct)

- Un struct no puede contener una instancia de sí mismo
- Puede contener un miembro que es un puntero al mismo tipo de struct
- La definición de una estructura no reserva espacio en memoria
- Crea un nuevo tipo de datos que es usado para declarar variables de estructuras

- Declaraciones

- Declarado como cualquier otra variable:

```
carta oneCard, deck[ 52 ], *cPtr;
```

- Puede usarse una lista separada por coma:

```
struct carta {  
    char * cara;  
    char * palo;  
} unaCarta, mazo[ 52 ], *ptrCarta;
```



10.2 Definición de Estructuras (`struct`) (III)

- Operaciones Válidas

- Asigna una estructura a una estructura del mismo tipo (`=`)
- Toma la dirección (`&`) de una estructura
- Accede los miembros de una estructura (`" . " ; "=>"`)
- Utiliza el operador **`sizeof`** para determinara el tamaño de una estructura



10.3 Inicializando Estructuras

- Lista de inicialización

- Ejemplo:

```
carta unaCarta = { "Tres", "Corazones" };
```

- Declaración de asignación

- Ejemplo:

```
carta tresCorazones = unaCarta;
```

- O:

```
carta tresCorazones;
```

```
tresCorazones.cara = "Tres";
```

```
tresCorazones.palo = "Corazones";
```



10.4 Acceso a miembros de estructuras

- Accediendo a miembros de estructuras
 - Operador punto (.) - utilizado con el nombre de variables estructura
- Operador fecha (->) - utilizado con punteros a variables estructuras

```
carta miCarta;  
printf( "%s", miCarta.palo );
```

```
carta * miPtrCarta = &miCarta;  
printf( "%s", miPtrCarta -> palo );
```

`miPtrCarta => palo`
equivalente a
`(*miPtrCarta).palo`



10.5 Usando Estructuras con Funciones

- Paso de estructuras a funciones
 - Paso de estructura entera
 - O, paso de miembros individuales
 - Ambos paso son llamada por valor
- Para pasar estructura con llamada por referencia
 - Pasar su dirección
 - Pase la referencia a él
- Para pasar arreglos llamada por valor
 - Crear una estructura con el arreglo como miembro
 - Pasar la estructura



10.6 Typedef

- **typedef**

- Crea sinonimos (alias) para tipos de datos previamente definidos
- Use **typedef** para crear nombres de tipos cortos.
- Ejemplo:

```
typedef Carta * PtrCarta;
```

- Define un nuevo tipo de nombre **PtrCarta** como sinónimo para tipo **Carta ***
- **typedef** no crea nuevo tipo de dato
 - Solo crea un alias



10.7 Ejemplo: Simulación de alto rendimiento de cartas y de reparto de cartas.

- Pseudocódigo:
 - Crea un arreglo de estructuras **carta**
 - Ubica las cartas en el mazo
 - Mezcla la carta
 - Reparte las cartas



```

1  /* Fig. 10.3: fig10_03.c
2  Programa para barajar y repartir con el uso de estructuras */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  struct carta {
8      const char * cara;
9      const char * palo;
10 };
11
12 typedef struct carta Carta;
13
14 void llenaCarta( Carta * const, const char *[],
15                 const char *[] );
16 void barajar( Carta * const );
17 void repartir( const Carta * const );
18
19 int main()
20 {
21     Carta mazo[ 52 ];
22     const char *cara[] = { "As", "Dos", "Tres",
23                             "Cuatro", "Cinco",
24                             "Sies", "Siete", "Ocho",
25                             "Nueve", "Diez",
26                             "Joto", "Quina", "Rey"};
27     const char *palo[] = { "Coranones", "Diamentes",
28                             "Treboles", "Espadas"};
29
30     srand( time( NULL ) );

```



Outline

1. Carga **headers**

1.1 Define un **struct**

1.2 Prototipo de Función

1.3 Inicializa mazo[] y cara[]

1.4 Inicializa palo[]



2. Aleatorizar

2.1 llenarMazo

```
31
32     llenarMazo( mazo, cara, palo );
33     barajar( mazo );
34     repartir( mazo );
35     return 0;
36 }
37
38 void llenarMazo( Carta * const wMazo, const char * wCara[],
39                 const char * wPalo[] )
40 {
41     int i;
42
43     for ( i = 0; i <= 51; i++ ) {
44         wMazo[ i ].cara = wCara[ i % 13 ];
45         wMazo[ i ].palo = wPalo[ i / 13 ];
46     }
47 }
48
49 void barajar( Carta * const wMazo )
50 {
51     int i, j;
52     Carta temp;
53
54     for ( i = 0; i <= 51; i++ ) {
55         j = rand() % 52;
56         temp = wMazo[ i ];
57         wMazo[ i ] = wMazo[ j ];
58         wMazo[ j ] = temp;
59     }
60 }
```

Coloca las 52 cartas en la baraja.
Cara y palo determinados por el
resto (módulo).

2.1 barajar

Selecciona un número al azar entre 0 y 51.
Intercambia el elemento i con ese elemento.

```
61
62 void repartir( const Carta * const wMazo )
63 {
64     int i;
65
66     for ( i = 0; i <= 51; i++ )
67         printf( "%5s of %-8s%c", wMazo[ i ].cara,
68                 wMazo[ i ].palo,
69                 ( i + 1 ) % 2 ? '\t' : '\n' );
70 }
```



2.2 repartir

Recorrer la arreglo e
imprimir los datos.



Salida del Programa

Tres de Diamantes	Ocho de Espadas
Siete de Corazones	Nueve de Diamantes
Joto de Espadas	Rey de Treboles
Cuatro de Treboles	Diez de Corazones
Rey de Diamantes	Joto de Treboles
Joto de Corazones	Seis de Diamantes
Nueve de Corazones	Cuatro de Diamante
Ocho de Diamantes	Quina de Diamantes
Ocho de Treboles	Diez de Diamantes
As de Diamantes	As de Espadas
Cinco de Treboles	Nueve de Espadas
Diez de Treboles	Rey de Espadas
Rey de Corazones	Seis de Treboles
Dos de Diamantes	As de Treboles
As de Corazones	Dos de Espadas
Cuatro de Espadas	Nueve de Treboles
Quina de Corazones	Cinco de Diamantes
Tres de Treboles	Siete de Treboles
Siete de Espadas	Cinco de Corazones
Joto de Diamantes	Cinco de Espadas
Quina de Treboles	Ocho de Corazones
Dos de Treboles	Tres de Espadas
Seis de Espadas	Seis de Corazones
Tres de Corazones	Siete de Diamantes

10.8 Uniones (union)

- **union**
 - Memoria que contiene una variedad de objetos sobre el tiempo
 - Solo contiene un miembro dato a la vez
 - Miembros de una unión de espacio compartido
 - Conserva el almacenamiento
 - Solo el último miembro definido puede ser accedido
- **union** - declaraciones
 - Igual a **struct**

```
union Number {  
    int x;  
    float y;  
};  
Union miObjeto;
```



10.8 Uniones (II)

- Operadores de **union** validas
 - Asignar a unión del mismo tipo: `"="`
 - Tomar dirección: `"&"`
 - Acceso a los miembros de unión: `"."`
 - Accessing members using pointers: `"=>"`




```

1  /* Fig. 10.5: fig10_05.c
2     Un ejemplo de unión*/
3  #include <stdio.h>
4
5  union numero{
6      int x;
7      double y;
8  };
9
10 int main()
11 {
12     union numero valor;
13
14     valor.x = 100;
15     printf( "%s\n%s\n%s%d\n%s%f\n\n",
16             "Coloca un valor en el miembro entero",
17             "e imprime ambos miembros.",
18             "int:  ", valor.x,
19             "double:\n", valor.y );
20
21     valor.y = 100.0;
22     printf( "%s\n%s\n%s%d\n%s%f\n",
23             "Coloca un valor en el miembro flotante",
24             "e imprime ambos miembros.",
25             "int:  ", valor.x,
26             "double:\n", valor.y );
27     return 0;
28 }

```



1. Define union

1.1 Inicializa variables

2. Fija variables

3. Imprime

10.9 Operadores a nivel de bits

- Todos los datos representados internamente como secuencia de bits
 - Cada bit puede ser o **0** o **1**
 - Secuencia de 8 bits forma un *byte*

Operador	Nombre	Descripción
&	AND a nivel de bits	Los bits del resultado se establecen en 1, si los bits correspondientes a los dos operandos son 1.
	OR a nivel de bits	Los bits del resultado se establecen en 1, si al menos uno de los bits correspondientes a los dos operandos es 1.
^	OR exclusivo a nivel de bits	Los bits del resultado se establecen en 1, si exactamente uno de los bits correspondientes a los dos operandos es 1.
<<	corrimiento izquierdo	Desplaza hacia la izquierda los bits del primer operando, el número de bits especificados por el segundo operando; desde la derecha llena con bits en 0.
>>	corrimiento derecho	Desplaza hacia la derecha los bits del primer operando, el número de bits especificados por el segundo operando; el método de llenado desde la izquierda depende de la máquina.
~	Complemento a uno	Todos los bits en 0 se establecen en 1, y todos los bits en 1 se establecen en 0.

```

1  /* Fig. 10.9: fig10_09.c
2      Uso de los operadores de bits AND, OR incluyente,
3      OR excluyente a nivel de bits y complemento */
4  #include <stdio.h>
5
6  void despliegaBits( unsigned );
7
8  int main()
9  {
10     unsigned numero1, numero2, mascara, estableceBits;
11
12     numero1 = 65535;
13     mascara = 1;
14     printf( "El resultado de combinar los siguientes valores\n"
15     despliegaBits( numero1 );
16     despliegaBits( mascara);
17     printf( "con el uso del operador de bits AND (&) es\n" );
18     despliegaBits( numero1 & mascara );
19
20     numero1 = 15;
21     estableceBits = 241;
22     printf("\nEl resultado de combinar los siguientes valores\n" );
23     despliegaBits( numero1 );
24     despliegaBits( estableceBits );
25     printf( "using the bitwise inclusive OR operator | is\n" );
26     despliegaBits( numero1 | estableceBits);
27
28     numero1 = 139;
29     numero2 = 199;
30     printf("\nEl resultado de combinar los siguientes valores\n" );

```



Outline

1. Prototipado de Función

1.1 Inicializa variables

2. Llamada a función

2.1 Imprime

```
31  despliegaBits( numero1 );
32  despliegaBits( numero2 );
33  printf( "con el uso del operador de bits OR excluyente (^) es\"
34  despliegaBits( numero1 ^ numero2 );
35
36  numero1 = 21845;
37  printf( "\\nThe one's complement of\\n" );
38  despliegaBits( numero01 );
39  printf( "is\\n" );
40  despliegaBits( ~numero1 );
41
42  return 0;
43 }
44
45 void despliegaBits( unsigned valor )
46 {
47     unsigned c, despliegaMascara = 1 << 31;
48
49     printf( "%7u = ", valor);
50
51     for ( c = 1; c <= 32; c++ ) {
52         putchar( despliegaMascara & valor ? '1' : '0' );
53         valor <<= 1;
54
55         if ( c % 8 == 0 )
56             putchar( ' ' );
57     }
58
59     putchar( '\\n' );
60 }
```

2.1 Imprime

MASCARA creado con solo un
establecimiento de bit
i.e. (10000000 00000000)

3. Definición de la función

La **MASCARA** es constantemente aplicado el
operador **AND** con **valor**.

MASCARA solo contiene un bit, así si **AND**
retorna verdadero esto significa que **value**
debe tener ese bit.

value es entonces corrido para verificar el
siguiente bit.

El resultado de combinar los siguientes valores
65535 = 00000000 00000000 11111111 11111111
1 = 00000000 00000000 00000000 00000001
con el uso del operador de bits AND & es
1 = 00000000 00000000 00000000 00000001

```
12      numero1 = 65535;  
13      mascara = 1;  
  
18      despliegaBits( numero1 & mascara );
```

El resultado de combinar los siguientes valores
15 = 00000000 00000000 00000000 00001111
241 = 00000000 00000000 00000000 11110001
con el uso del operador de bits OR incluyente | es
255 = 00000000 00000000 00000000 11111111

El resultado de combinar los siguientes valores
139 = 00000000 00000000 00000000 10001011
199 = 00000000 00000000 00000000 11000111
con el uso del operador de bits OR excluyente (^) es
76 = 00000000 00000000 00000000 01001100

El complemento a uno de
21845 = 00000000 00000000 01010101 01010101
es
4294945450 = 11111111 11111111 10101010 10101010

Salida del Programa

El resultado de combinar los siguientes valores
65535 = 00000000 00000000 11111111 11111111
1 = 00000000 00000000 00000000 00000001
con el uso del operador de bits AND & es
1 = 00000000 00000000 00000000 00000001

El resultado de combinar los siguientes valores
15 = 00000000 00000000 00000000 00001111
241 = 00000000 00000000 00000000 11110001
con el uso del operador de bits OR incluyente | es
255 = 00000000 00000000 00000000 11111111

```
20 numero1 = 15;  
21 estableceBits = 241;  
  
26 despliegaBits( numero1 | estableceBits);
```

El resultado de combinar los siguientes valores
139 = 00000000 00000000 00000000 10001011
199 = 00000000 00000000 00000000 11000111
con el uso del operador de bits OR excluyente (^) es
76 = 00000000 00000000 00000000 01001100

El complemento a uno de
21845 = 00000000 00000000 01010101 01010101
es
4294945450 = 11111111 11111111 10101010 10101010



Salida del Programa

El resultado de combinar los siguientes valores
65535 = 00000000 00000000 11111111 11111111
1 = 00000000 00000000 00000000 00000001
con el uso del operador de bits AND & es
1 = 00000000 00000000 00000000 00000001

El resultado de combinar los siguientes valores
15 = 00000000 00000000 00000000 00001111
241 = 00000000 00000000 00000000 11110001
con el uso del operador de bits OR incluyente | es
255 = 00000000 00000000 00000000 11111111

El resultado de combinar los siguientes valores
139 = 00000000 00000000 00000000 10001011
199 = 00000000 00000000 00000000 11000111
con el uso del operador de bits OR excluyente (^) es
76 = 00000000 00000000 00000000 01001100

El complemento a uno de
21845 = 00000000 00000000 01010101 01010101
es
4294945450 = 11111111 11111111 10101010 10101010

```
28     numero1 = 139;  
29     numero2 = 199;  
  
34     despliegaBits( numero1 ^ numero2 );
```


Salida del Programa

El resultado de combinar los siguientes valores

65535 = 00000000 00000000 11111111 11111111

1 = 00000000 00000000 00000000 00000001

con el uso del operador de bits AND & es

1 = 00000000 00000000 00000000 00000001

El resultado de combinar los siguientes valores

15 = 00000000 00000000 00000000 00001111

241 = 00000000 00000000 00000000 11110001

con el uso del operador de bits OR incluyente | es

255 = 00000000 00000000 00000000 11111111

El resultado de combinar los siguientes valores

139 = 00000000 00000000 00000000 10001011

199 = 00000000 00000000 00000000 11000111

con el uso del operador de bits OR excluyente (^) es

76 = 00000000 00000000 00000000 01001100

El complemento a uno de

21845 = 00000000 00000000 01010101 01010101

es

4294945450 = 11111111 11111111 10101010 10101010

36 numero1 = 21845;

40 despliegaBits(~numero1);

10.10 Campos de Bit

- Campo de Bit

- Miembro de una estructura cuyo tamaño (en bits) ha sido especificado
- Habilita una mejor utilización de memoria
- *Debe ser* declarado como **int** o **unsigned**
- No puede acceder a campos individuales

- Declarando los campos de bits

- Siguiendo miembros de **unsigned** o **int** con doble punto (:) y un entero constante representando el *ancho* del campo
- Ejemplo:

```
struct BitCard {  
    unsigned cara: 4;  
    unsigned palo : 2;  
    unsigned color : 1;  
};
```



10.10 Campos de bit (II)

- Campos de bit sin nombre

- El campo utilizado como **relleno en la estructura**
- No se puede almacenar nada en los bits

```
struct Ejemplo {  
    unsigned a : 13;  
    unsigned   : 3;  
    unsigned b : 4;  
}
```

- Un *campo de bits sin nombre con un ancho de 0* se utiliza para alinear el siguiente campo de bits en un nuevo límite de la unidad de almacenamiento.

```
struct Ejemplo {  
    unsigned a : 13;  
    unsigned   : 0;  
    unsigned b : 4;  
}
```



10.11 Ejemplo: Un juego de azar e introducción de `enum`

- Enumeration
 - Conjunto de enteros representado por identificadores
 - Constantes de enumeración, como constantes simbólicas cuyos valores se fijan automáticamente
 - Valores inician en **0** y son incrementados por **1**
 - Valores pueden ser fijados explícitamente con **=**
 - Necesita nombres constantes únicos
 - Declarar las variables como normales
 - Las variables de enumeración sólo pueden asumir sus valores constantes de enumeración (no las representaciones enteras)



10.11 Ejemplo: Un juego de azar e introducción de `enum` (II)

- Ejemplo:

```
enum Meses { ENE=1, FEB, MAR, ABR, MAY, JUN,  
             JUL, AGO, SEP, OCT, NOV, DIC};
```

- Inicia a **1**, incrementado por **1**



```

1  /* Fig. 10.18: fig10_18.c
2      Uso de un tipo de enumeración */
3  #include <stdio.h>
4
5  enum meses { ENE = 1, FEB, MAR, ABR, MAY, JUN,
6              JUL, AGO, SEP, OCT, NOV, DIC };
7
8  int main()
9  {
10     enum meses mes; /* puede contener cualquiera de los 12
11     const char *nombreMes [] = { "", "Enero", "Febrero",
12                                "Marzo", "Abril", "Mayo",
13                                "Junio", "Julio", "Agosto",
14                                "Septiembre", "Octubre",
15                                "Noviembre", "Diciembre" };
16     /* ciclo a través de los meses */
17     for ( mes = ENE; mes <= DIC; mes++ )
18         printf( "%2d%11s\n", mes, nombreMes[ mes ] );
19
20     return 0; /* indica terminación exitosa */
21 } /* fin de main */

```



Outline



1. Define enumeration

1.1 Inicializa variable

2. Ciclo

2.1 Imprime



Outline



Salida de Programa

01 Enero
02 Febrero
03 Marzo
04 Abril
05 Mayo
06 Junio
07 Julio
08 Agosto
09 Septiembre
10 Octubre
11 Noviembre
12 Diciembre

Capt. 10 - Estructuras, uniones, manipulaciones de bits y enumeraciones

Aguije !!!!

