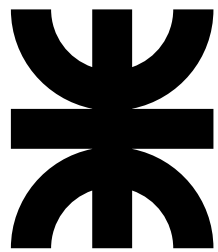
Informática II

Uniones y manipulación de bits

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2018 –

Uniones

Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Uniones
Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

Uniones
Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

Uniones
Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande

Uniones
Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande
- En general contienen dos o más tipos de datos

Uniones
Definición

Es un tipo de dato derivado —como lo es una estructura— cuyos miembros comparten el mismo espacio de almacenamiento.

Para ciertas situaciones en un programa, algunas variables pudieran ser importantes y otras no. Las uniones comparten el espacio, en vez de desperdiciar almacenamiento en variables que no están siendo utilizadas.

```
union numero {
  int x;
  float y;
};
```

- Ocupa en memoria lo suficiente para contener el miembro más grande
- En general contienen dos o más tipos de datos
- En cada momento se puede referenciar un tipo de dato

Uniones

Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de un unión

Uniones

Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de un unión

Se pueden comparar las uniones?

Uniones

Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de un unión

Se pueden comparar las uniones? NO

Uniones

Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de un unión

Se pueden comparar las uniones? NO (y las estructuras?)

Uniones
Operaciones permitidas

- Tener acceso a los miembros de una unión utilizando el operador de miembro de estructura y el operador de apuntador de estructura
- Asignar una unión a otra unión del mismo tipo
- Tomar la dirección (`&`) de un unión

Se pueden comparar las uniones? NO (y las estructuras?)

Inicialización: Se puede inicializar en la declaración con un valor del mismo tipo que el primer miembro de la union

```
union numero {
  int x;
  float y;
};

union numero u1 = {10};
union numero u1 = {0.02} /* Qué hace? */;
```

Uniones. Ejercicio 1

- Definir union formada por un `int` y un `float`
- Solicitar valor entero e imprimir ambos campos (lo mismo para el valor real)

Uniones. Ejercicio 1

- Definir union formada por un `int` y un `float`
- Solicitar valor entero e imprimir ambos campos (lo mismo para el valor real)

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};

void imprimir_union_int_float(union int_float);

int main(void)
{
  union int_float u;

  printf("Ingrese un entero: ");
  scanf("%d", &(u.entero));
  imprimir_union_int_float(u);

  printf("Ingrese un real: ");
  scanf("%f", &(u.real));
  imprimir_union_int_float(u);

  return 0;
}
```

Uniones. Ejercicio 2

- Union formada por un `float` y un vector de `unsigned char` de dimensión 4
- Ingrese valor `float` por teclado e imprime el vector en pantalla

Uniones. Ejercicio 2

- Union formada por un `float` y un vector de `unsigned char` de dimensión 4
- Ingrese valor `float` por teclado e imprime el vector en pantalla

```c
/* Union con float y vector de uchar */
#include <stdio.h>

typedef union {
  float real;
  unsigned char ucvec[4];
} float_uchar_t;

int main(void)
{
  int i;
  float_uchar_t u;

  printf("Ingrese un valor real: ");
  scanf("%f", &(u.real));

  /* Imprimer float y vector uchar */
  . . .
  . . .
  . . .
  . . .

  return 0;
}
```

Uniones

Algunos ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;
```

Uniones

Algunos ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

Uniones

Algunos ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;


for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;
```

Uniones

Algunos ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;

printf("%d␣", reg2.bytes.byte0);
printf("%d␣", reg2.bytes.byte1);
printf("%d␣", reg2.bytes.byte2);
printf("%d␣", reg2.bytes.byte3);
```

Uniones

Algunos ejemplos

```
union uint_ucvec {
  unsigned int uint;
  unsigned char ucvec[4];
} reg1;

for(i = 0; i < 4; i++)
  printf("%d␣", reg1.ucvec[i]);
```

```
union uint_bytes {
  unsigned int uint;
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
} reg2;

printf("%d␣", reg2.bytes.byte0);
printf("%d␣", reg2.bytes.byte1);
printf("%d␣", reg2.bytes.byte2);
printf("%d␣", reg2.bytes.byte3);
```

```
union uint {
  unsigned int uint;
  unsigned char ucvec[4];
  struct {
    unsigned char byte0;
    unsigned char byte1;
    unsigned char byte2;
    unsigned char byte3;
  } bytes;
};
```

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Ver código de ejemplo.

Operadores a nivel de bits

Para manipular los bits individuales de los tipos de datos básicos (generalmente `unsigned`)

Operadores a nivel de bits

- `AND` a nivel de bit, `&`
- `OR` inclusivo a nivel de bit, `|`
- `OR` exclusivo a nivel de bit, `^`
- Desplazamiento a la izquierda, `<<`
- Desplazamiento a la derecha, `>>`
- Complemento, `~`

Operadores de asignación

- `AND`, `&=`
- `OR` inclusivo, `|=`
- `OR` exclusivo, `^=`
- Despl. izq., `<<=`
- Despl. der., `>>=`
- Complemento, `~=`

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Cómo forzar a cero/uno, uno o varios bits?. . . y cómo hacer toggle?

Operadores a nivel de bits

Operadores de desplazamiento

Desplazamiento a la izquierda

- Los valores desplazados se pierden
- Los valores a la derecha se rellenan con ceros

Operadores a nivel de bits

Operadores de desplazamiento

Desplazamiento a la izquierda

- Los valores desplazados se pierden
- Los valores a la derecha se rellenan con ceros

Desplazamiento a la derecha

- Los valores desplazados se pierdes
- Los valores a la izquierda dependen del tipo de dato (`signed`/`unsigned`)

Operadores a nivel de bits

Imprimir variables en binario

```
1  void print_binary(unsigned char val)
2  {
3    unsigned char b, mask = 1<<(8*sizeof(unsigned char)-1);
4
5    for(b = 1; b <= 8*sizeof(unsigned char); b++)
6    {
7      putchar(val & mask ? '1' : '0');
8      val <<= 1;
9
10     if(b % 8 == 0)
11       putchar('␣');
12   }
13
14   putchar('\n');
15 }
```

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Los miembros de los campos de bits deben ser declarados como `unsigned` o `int`

Campos de bits

Permite definir el número de bits en el cual se almacenan los miembros `unsigned` o `int` de una estructura o de una union.

Los miembros de los campos de bits deben ser declarados como `unsigned` o `int`

```
struct bitCard {
  unsigned face : 4;
  unsigned suit : 2;
  unsigned color : 1;
};
```

- Nombre del campo seguido de dos puntos `:` y una constante entera del *ancho del campo*
- La cantidad de bits se fija según el rango de valores de cada miembro
- El acceso a los miembros se realiza como en cualquier estructura

Campos de bits
Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Campos de bits
Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Y si se quisiera modificar los bits RGB todos juntos?

Campos de bits
Ejemplos

Byte/registro para manejo de colores

```
struct RGB_color {
  unsigned char r : 2; /* 2-bits */
  unsigned char g : 2;
  unsigned char b : 2;
  unsigned char : 2; /* padding */
};
```

Y si se quisiera modificar los bits RGB todos juntos?

```
union RGB_color {
  struct {
    unsigned char r:2, g:2, b:2;
    unsigned char : 2;
  };
  struct {
    unsigned char rgb : 6;
    unsigned char : 2;
  };
};
```

Uniones y manipulación de bits

Ejercicios

1. Modificar la función que imprime un `unsigned char` en binario (`print_binary`) para que no modifique el valor a imprimir.

2. Modificar la función `print_binary` para que imprima cualquier variable de tipo entero utilizando un `typedef`.

3. Modificar la función `print_binary` para que imprima un `float`.

4. Modificar el programa anterior para que reciba el valor `float` a convertir por la línea de comandos, para que pueda ser ejecutado y muestre la salida como:

```
./float2bin 123.123
123.123001 = 11111010 00111110 111101
```