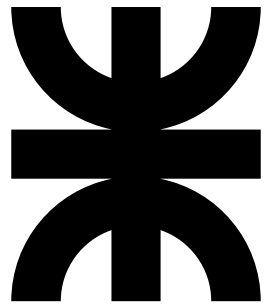
Informática II
Uniones y campos de bits

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2021 –

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.

Declaración:

```
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.

Declaración:

```
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.

Declaración:

```
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.

Declaración:

```
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 una 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 una 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 una 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 una 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 una unión.

Se pueden comparar las uniones? NO . . . y las estructuras? 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 una unión.

Se pueden comparar las uniones? NO . . . y las estructuras? NO

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

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

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

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};
```

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};

void imprimir_union_int_float(union int_float u)
{
  printf("- El miembro INT de la union es: %d\n", u.entero);
  printf("- El miembro FLOAT de la union es: %f\n", u.real);
}
```

Ejemplo de union `int` y `float`

```c
#include <stdio.h>

union int_float {
  int entero;
  float real;
};

void imprimir_union_int_float(union int_float u)
{
  printf("- El miembro INT de la union es: %d\n", u.entero);
  printf("- El miembro FLOAT de la union es: %f\n", u.real);
}

int main(void)
{
  union int_float u;

  /* Solicitar valor 'int' e imprimir union */
  /* Solicitar varlor 'float' e imprimir union */

  return 0;
}
```

Más ejemplos

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

Más ejemplos

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

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

Más ejemplos

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

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

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

Más 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);
```

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

```
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);
```

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;
  };
};
```

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

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

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

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

4. Modificar la función `imprimir_binario` para que imprima un `float`.

Actividad práctica

1. Escribir un programa que defina una union entre un `float` y un vector de cuatro `unsigned char` (4 bytes) e imprima los campos. La interacción con el usuario debe ser la siguiente:

```
Ingrese un valor real: 3.14
- El valor del FLOAT es: 3.140
- El vector de UCHAR es: 195 245 72 64
```

2. Modificar la función que imprime un `unsigned char` en binario (`imprimir_binario`) para que no modifique el valor a imprimir. Modificar el prototipo de función para que el parámetro sea `const`.

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

4. Modificar la función `imprimir_binario` para que imprima un `float`.

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

```
./float2bin 123.123
123.123001 = 11111010 00111110 11110110 01000010
```