

5

Capítulo

Apuntadores, estructuras y uniones

```
struct robot{  
    float posiciones[6];  
    float velocidades[6];  
    float torques[6];  
    int puerto0 : 1;  
    int puerto1 : 1;  
} robotA, robotB, robotC;
```





Capítulo Web

- 5.1 Introducción
- 5.2 Apuntadores
- 5.3 Estructuras
- 5.4 Uniones
- 5.5 Resumen
- 5.6 Referencias selectas
- 5.7 Problemas propuestos

Competencias


Presentar los elementos de programación de apuntadores, estructura de datos y uniones a través de ejemplos didácticos y pedagógicos que se ejecutan en las plataformas electrónicas de tarjetas Arduino con el propósito de ilustrar los conceptos y aspectos técnicos de su empleo e implementación.

Desarrollar habilidades en:




-  Que el lector desarrolle habilidades para programar aplicaciones de robótica y mecatrónica mediante:
-  Variables puntero, inicialización y asignación dinámica de memoria.
-  Estructura de datos, acceso y manipulación de sus campos.
-  Unión de datos, acceso y manipulación de sus campos.

5.1 Introducción

EL lenguaje C proporciona elementos de programación que mejoran las aplicaciones en todas las áreas de la ingeniería, adicional al conjunto de instrucciones, tiene un conjunto de herramientas con propiedades importantes de programación que facilitan la implementación de tareas específicas, estas herramientas de programación se basan en los apuntadores, estructuras y unión de datos.

En este capítulo se revisan aspectos y detalles de programación con variables puntero o apuntadores, estructuras y uniones de datos; la presentación de los conceptos se apoyan por medio de ejemplos didácticos de programación con código fuente en lenguaje C y documentación detallada de los programas o sketches que se ejecutan directamente en las plataformas electrónicas de los diversos modelos de tarjetas Arduino. Todos los ejemplos desarrollados se encuentran documentados en detalle y disponibles en el sitio  de esta obra.

El análisis, discusión y presentación de apuntadores, estructuras y uniones se realiza con particular énfasis sobre los siguientes tópicos:

-  Variables puntero o apuntadores de los diversos tipos de datos del lenguaje C, operadores, operaciones aritméticas, inicialización, asignación dinámica de memoria usando las funciones `malloc(...)` y `free(...)`, similitud con arreglos.
-  Estructura (**structure**), corresponde a una agrupación de variables de diferentes tipos de datos; tienen asociados ciertas clase de operadores y forma de acceso a sus campos; las estructura de datos permiten definir campos de bits para manipular datos y manejar de manera adecuada interfaces electrónicas como banderas, puertos, comandos, etc.
-  Unión de datos (**union**) es una agrupación de diversos tipos de datos parecida a la estructura, pero en este caso permite compartir el mismo espacio de memoria para todos sus elementos o campos que se encuentran agrupados. Por su naturaleza requiere de una clase particular de operadores, forma de acceso y manipulación computacional de sus elementos.

5.2 Apuntadores

LOS apuntadores también conocidos como variables tipo puntero son una de las herramientas más importantes del lenguaje C y al mismo tiempo el recurso más peligroso sino se tiene completo dominio de la forma en que funcionan, aspectos de aritméticos y tipo de operadores que se emplean. Por ejemplo, un apuntador mal utilizado puede ocasionar serios problemas de cómputo, esta parte es muy sutil, ya que la fase de compilación no necesariamente detecta aspectos de inicialización, es decir a pesar que la sintaxis del sketch esté bien escrita, al momento de ejecutarse en las tarjetas Arduino se colapsa el programa por un inadecuado manejo de los apuntadores; este tipo de fallas es difícil de encontrar, teniendo como consecuencia modificación del valor de variables o borrado de datos importantes del sketch.




El uso adecuado de los apuntadores resulta clave para mejorar la eficiencia de un sketch; técnicamente los apuntadores intervienen en el pase de parámetros de las funciones, modificando el valor de los argumentos, también se utilizan en asignación dinámica de memoria, pueden acceder a la memoria de todos los tipos de datos del lenguaje C, estructuras y uniones.



Apuntadores

Los apuntadores, también conocidos como variables tipo puntero contienen direcciones de memoria de otra variable, para acceder y manipular dichas localidades de memoria; la variable puntero a través de la dirección de memoria puede leer y escribir información. Es decir, está apuntando a esa variable.

La declaración de variables tipo puntero corresponde a los mismos tipos de datos del lenguaje C, la sintaxis de un apuntador consiste de lo siguiente:

-  Utilizar un tipo de dato válido del lenguaje C.
-  Seguido de un espacio en blanco.
-  Continúa con el operador *, en seguida sin dejar espacio en blanco con el nombre o identificador, si hay una lista de apuntadores, entonces estarán separados por comas y se finaliza con punto y coma.

Por ejemplo, en el código ejemplo 5.1 se presenta la forma de declarar apuntadores de diversos tipos de datos:

∞ Código ejemplo 5.1

Declaración de apuntadores o variables tipo puntero

```
char *c, *b, *r; //apuntadores tipo char.
```

```
unsigned char *cadena, *letra, *palabra; //apuntadores tipo char sin signo.
```

```
int *ap_int, *u, *k; //variables puntero tipo de datos entero.
```

```
unsigned int *uns_i, *puerto, *p; //apuntadores del tipo de datos entero sin signo.
```

```
long *torque, *velocidad, *pos; //apuntadores del tipo entero largo.
```

```
float *x, *y, *z; //variables puntero del tipo flotante.
```

```
double *xo, *xyu, *salida; //apuntadores flotantes con doble precisión.
```

El tipo de dato define el tipo de apuntador, es decir, a lo que puede apuntar la variable puntero, esto significa que un apuntador tipo char solo puede apuntar a este tipo de datos y no a los del tipo flotante. Sin embargo, técnicamente cualquier tipo de apuntador puede apuntar a cualquier lugar de memoria, no obstante, la aritmética de los apuntadores se realiza en base al tipo de dato declarado. Existen dos operadores monarios para apuntadores: `&` y `*`, el término monario se refiere a que sólo necesitan un operando. La tabla 5.1 contiene su descripción.

Tabla 5.1 Operadores para variables tipo puntero.

Operador	Descripción
<code>&</code>	<p>Devuelve la dirección de memoria de su operando. Por ejemplo:</p> <pre>float *z, x=3.14159; //variable puntero z, variable del tipo flotante x. z=&x; //z recibe la dirección de memoria de la variable x.</pre> <p>La dirección de memoria de la variable x contiene los registros o valores de esta variable, por lo tanto, el apuntador z puede acceder a esa dirección para leer o cambiar la información.</p>
<code>*</code>	<p>Devuelve el valor de la variable localizada en la dirección a la que apunta. Por ejemplo:</p> <pre>*z=86.789; //el valor de x se modifica de 3.14159 a 86.789.</pre> <p>Debido a que el apuntador z contiene la dirección de memoria de la variable x, puede acceder a su contenido para leer o modificar.</p>

Supóngase que la dirección de memoria de la variable **x** es 0x1000 y esta localidad de memoria almacena el valor de 3.14159, la variable puntero **z** adquiere la dirección de **x** a través de la sentencia **z=&z**; para modificar el valor de **x** se ejecuta la sentencia ***z=86.789**; entonces, la variable **x** ahora tiene el nuevo valor, debido a que el apuntador **z** accedió a su dirección de memoria, modificando su contenido, la figura 5.1 ilustra este proceso.

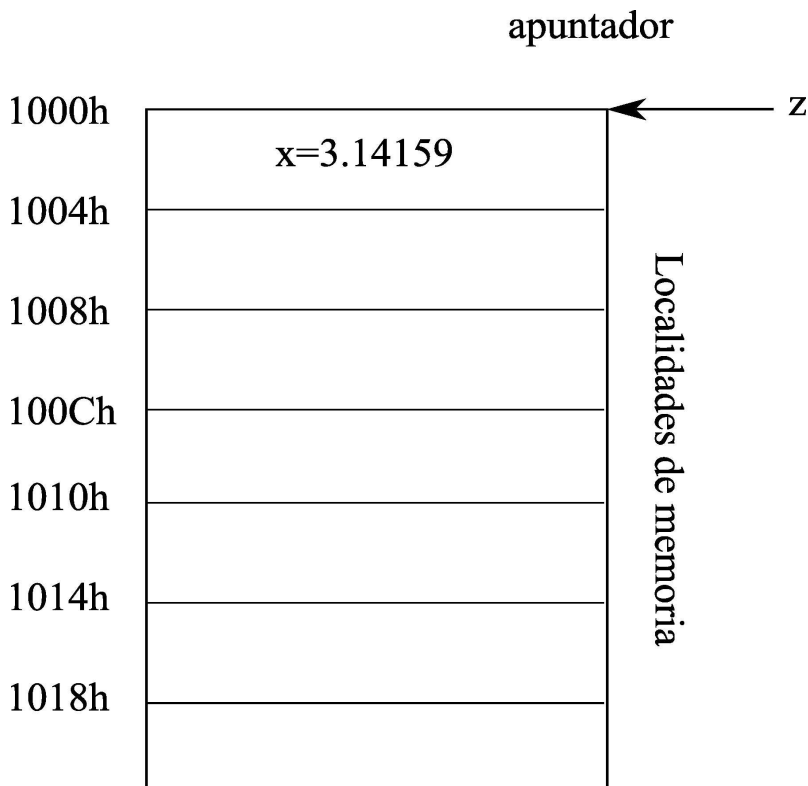


Figura 5.1 Variable puntero **z** direcciona la localidad de memoria de **x**.

Los operadores para apuntadores **&** y ***** no deben ser interpretados, ni confundidos como los operadores **and** a nivel de bits y multiplicación aritmética, respectivamente. Ambos operadores de apuntadores tienen mayor prioridad que los otros operadores, la diferencia se encuentra en la forma de utilizar la sintaxis o gramática para variables de diversos tipos de datos y apuntadores.

♣ Ejemplo 5.1

Escribir un sketch que modifique el valor de una variable tipo flotante **x**, la cual tiene inicialmente el valor de 3.14159 y adquiera el valor de 86.789 por medio de un apuntador. Desplegar la información en el monitor serial del ambiente de programación Arduino.

Solución

El sketch **cap5_apuntador** contiene la solución al problema planteado, cuya documentación se encuentra detallada en el cuadro de código Arduino 5.1; en la línea 1 se declara las variables apuntador ***z** y **x**, ambas del tipo flotante, en esta fase el apuntador ***z** no se encuentra inicializado.









La velocidad de transmisión serial entre la tarjeta Arduino y la computadora ha sido establecida en 9600 Baudios en la rutina de configuración **setup()** ubicada en la línea 2 y el lazo principal **loop()** del sketch inicia en la línea 5.

La forma de demostrar la eficiencia de los apuntadores es asignar en la variable de tipo flotante **x** un valor de referencia 3.14159 (línea 6) y desplegarlo en el monitor serial. La línea 7 es clave, debido a que el apuntador ***z** queda anclado o apuntando en la dirección de memoria de la variable **x** por medio del operador **&**, note que en la línea 11 se asigna el valor 86.789 al apuntador **z** usando el operador ***** y como dicha variable puntero ya contiene la dirección de memoria de la variable **x**, entonces puede acceder a modificar el contenido de esa localidad. La línea 12 envía al monitor serial el valor de la variable **x** para verificar que efectivamente ha cambiado su valor que almacena en memoria debido a la manipulación del apuntador **z**. El sketch se ejecuta de nuevo después de 3 segundos (línea 14).

Los pasos requeridos para ejecutar el sketch **cap5_apuntador** en cualquier modelo de las tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**; sin embargo, por comodidad al lector a continuación se presentan en forma sencilla:



Descargar del sitio Web de este libro  el código fuente en lenguaje C del sketch **cap5_apuntador**.

-  En el menú **Herramientas** del ambiente de programación Arduino seleccionar modelo de tarjeta y velocidad de comunicación serial en 9600 Baudios.
-  Compilar el sketch mediante el icono .
-  Descargar el código de máquina a la tarjeta Arduino usando .
-  Desplegar resultados con el monitor serial (activar con .
-  **Importante:** para que los resultados del sketch **cap5_apuntador** se exhiban adecuadamente en el monitor serial del ambiente de programación, maximice la ventana y active la opción **desplazamiento automático**.

Código Arduino 5.1: sketch cap5_apuntador

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones**.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_apuntador.ino

```

1 float *z, x; //variable apuntador z.
2 void setup() { //subrutina de configuración.
3     Serial.begin(9600); //configuración de comunicación serial.
4 }
5 void loop() { //inicia lazo principal del sketch.
6     x=3.14159;
7     z=&x;
8     Serial.print("Valor de x=" );
9     Serial.println(x,5);
10    Serial.print("Valor nuevo de x=" );
11    *z=86.789;
12    Serial.println(x,5);
13    Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n\n" );
14    delay(3000); //retardo entre transmisión serial.
15 }
```

El sketch **cap5_apuntador** produce el siguiente resultado en el monitor serial del ambiente de programación Arduino:

Valor de x= 3.14159

Valor nuevo de x=86.78900

En 3 segundos se ejecuta una vez más el sketch.



5.2.1 Inicialización de apuntadores

Cuando se declara un apuntador de cualquier tipo de dato válido del lenguaje C, éste contiene un valor desconocido, es decir, no apuntada a una dirección específica. Por ejemplo, **int *ap**; el apuntador **ap** de tipo entero no tiene una dirección de memoria asignada, por lo que si llegara a utilizar el sketch o programa fallará, a pesar que en la fase de compilación no fue detectado como un error de sintaxis.



Es **responsabilidad** del usuario inicializar correctamente un apuntador antes de utilizarlo.

Una forma de inicializar un apuntador es por medio de la asignación de la dirección de una variable, como en el siguiente caso:

```
float *z, x;
```

```
z=&x;//obtiene la dirección de memoria de la variable x.
```

en este caso se dice que la variable puntero **z** apunta a la variable **x**.

Asignación dinámica

Otra forma de inicializar a una variable puntero es por medio de asignación dinámica de memoria utilizando la función **malloc(num_bytes)**, retorna un apuntador con la dirección de memoria asignada, donde **num_bytes** representa la cantidad de memoria en bytes por asignar, la cual depende de las características técnicas y

disponibilidad de recursos de los modelos de tarjetas Arduino. Para liberar la memoria asignada al apuntador se emplea la función **free(apuntador)**. Ambas funciones requieren utilizar en la cabecera (*header*) del sketch el archivo de librerías estándar **stdlib.h** (ver detalles de sintaxis en el capítulo 6 **Librerías Arduino**).

♣ ♣ Ejemplo 5.2

Desarrollar un sketch que permita asignar memoria dinámica a una variable puntero del tipo flotante.

Solución

La asignación dinámica de memoria se realiza por medio de la función **malloc(num_bytes)**, donde **num_bytes** establece la cantidad de memoria requerida expresada en bytes. Por otro lado, la función que libera la memoria asignada se denomina **free(...)**. El algoritmo de asignación dinámica se encuentra en el sketch **cap5_apdinamico**, el cual se encuentra documentado en el cuadro de código Arduino 5.2.

La línea 1 describe el uso de la librería **stdlib.h**, la cual se coloca en la cabecera o *header* del sketch y permite el uso de las funciones **malloc(...)** y **free(...)**. La declaración de la variable puntero tipo flotante **z** se lleva a cabo en la línea 2, en este momento apuntador **z** no se encuentra inicializado, es decir no apunta a ninguna dirección de memoria específica.

La configuración de velocidad de transmisión serial queda establecida en 9600 Baudios en la subrutina **setup()** como se ilustra en la línea 3 y el lazo principal **loop()** del sketch inicia en la línea 6.

La asignación dinámica de memoria de 4 bytes para un número flotante se lleva a cabo en la línea 9 a través de la función **malloc(4 bytes)**; observe que se emplea *casting* (**float ***)**malloc(4)** para adecuar el tipo de apuntador que retorna esta función y que sea compatible con el tipo de dato flotante del apuntador **z**. En la línea 10 se le asigna el valor 3.14159 al apuntador **z** por medio del operador *****, esta información se envía por transmisión serial y se despliega en el monitor del ambiente de programación Arduino con 5 fracciones utilizando la función **Serial.print(*z,5)**, tal y como se indica en la línea 12.

Para liberar la memoria asignada al apuntador **z** se emplea la función **free(z)** (ver línea 13), después de una pausa de 3 segundos, el sketch se ejecuta una vez más.

Para de descargar y ejecutar el sketch **cap5_apdinamico** en las tarjetas Arduino referirse al procedimiento descrito en la página 7.

🔗 Código Arduino 5.2: sketch cap5_apdinamico

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_apdinamico.ino

```

1 #include <stdlib.h>//librería para las funciones malloc(...) y free(...).
2 float *z;//declaración del apuntador z tipo de dato flotante.
3 void setup() { //subrutina de configuración.
4     Serial.begin(9600); //configuración de comunicación serial.
5 }
6 void loop(){ //inicia lazo principal del sketch.
7     Serial.print("Asignación dinámica de apuntadores " );
8     //Asignación dinámica de 4 bytes de memoria en punto flotante usando malloc(...).
9     z=(float *)malloc(4);//convierte el resultado en apuntador punto flotante.
10    *z=3.14159;//valor asignado en la localidad de memoria que apunta z.
11    Serial.print("Valor asignado al apuntador=: " );
12    Serial.print(*z,5);//desplegar información con 5 fracciones.
13    free(z);//libera espacio de memoria (4 bytes) asignado al apuntador z.
14    Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n");
15    delay(3000);//retardo entre transmisión serial.
16 }
```



5.2.2 Operaciones aritméticas con apuntadores

Cuando se asigna memoria dinámica a una variable puntero, se puede recorrer ese espacio de memoria para leer o escribir información; lo anterior se realiza a través de

operadores aritméticos para apuntadores: +, -, ++, --. La tabla 5.2 contiene la descripción de dichos operadores; considere a **p** una variable apuntador de un tipo válido de datos del lenguaje C, entonces:

Tabla 5.2 Operadores aritméticos para apuntadores.

Operador	Descripción
p+n	Incrementa en n localidades de memoria la posición donde apunta la variable apuntador p .
p-n	Decrementa en localidades de memoria la posición donde apunta la variable apuntador p .
p++	Incrementa una localidad de memoria la posición donde apunta la variable apuntador p .
p--	Decrementa una localidad de memoria la posición donde apunta la variable apuntador p .

El incremento o decremento de localidades depende de la longitud del tipo de dato de la variable apuntador, por ejemplo:

`char *c;`//entero de un byte.

`int *i;`//entero de 2 bytes.

`float *x;`//número real de 4 bytes.

En la variable apuntador **c**, los incrementos/decrementos de las localidades de memoria corresponden a un byte, para **i** de 2 bytes y en el caso de **x** son de 4 bytes.

Otro ejemplo, corresponde a: **c+4** significa que la posición de memoria se incrementa en 4 bytes, **i+4** la posición del apuntador se incrementa en 8 bytes, mientras que **x+4** lo realiza en 16 bytes.

Cada vez que se incremente una variable puntero, apunta al elemento de la siguiente posición de memoria, dependiendo de la naturaleza del tipo de dato del apuntador, el incremento/decremento se realiza en función de la longitud del tipo de dato correspondiente.



La aritmética de apuntadores se relaciona con el tipo de dato, es decir los incrementos/decrementos están directamente relacionados con el número de bytes que definen la longitud del tipo de dato, para **boolean**, **char** y **byte** 1 byte, **word**, **int** 2 bytes, **long** 4 bytes, **float** y **double** 4 bytes.

Como una aplicación de los operadores aritméticos de apuntadores considere el siguiente ejemplo donde se considera la suma de vectores.

♣ ♣ ♣ Ejemplo 5.3

Desarrolle un sketch que realice la suma de dos vectores $x, y \in \mathbb{R}^{1 \times 8}$; utilizar un algoritmo en base a la asignación dinámica de apuntadores y despliegue el resultado en el monitor serial del ambiente de programación Arduino.

Solución

Un ejemplo práctico sobre el uso de asignación dinámica de apuntadores, es el algoritmo de suma de vectores que se describe en el sketch **cap5_apdinamico1**, cuya documentación se encuentra en el cuadro de código Arduino 5.3. En la cabecera o *header* del sketch, línea 2 se utiliza la librería **stdlib.h** la cual contiene la sintaxis correcta de las funciones **malloc(...)** y **free(...)**. En la línea 3 se realiza la declaración de los apuntadores de tipo flotante **x**, **y** y **z**. En el proceso iterativo de las suma de dos vectores se requiere un índice entero, como el que se define en la línea 4.

La velocidad de transmisión serial se establece en 9600 Baudios en la rutina de configuración **setup()** ubicada en la línea 5 y el lazo principal del sketch **loop()** inicia en la línea 8.

Cada vector tiene una dimensión de un renglón por ocho columnas, lo números que se manejan son del tipo flotante, es decir se requiere 4 bytes por 8 números; en otras palabras se requieren 32 bytes de asignación dinámica para cada apuntador. Las líneas 12 a la 16 llevan a cabo este proceso de inicialización de los apuntadores por medio de la función **malloc(...)**, el apuntador **z** se utiliza para registrar la suma o resultado de los vectores: **z=x+y**. Observe que se utiliza en cada apuntador un *casting* (float *)**malloc(8*4)** para convertir adecuadamente el tipo de apuntador que retorna **malloc(...)** al tipo de dato flotante.

Para propósitos ilustrativos considere los siguientes vectores:

$$\mathbf{x} = [0 \quad 3.14159 \quad 6.28318 \quad 9.42477 \quad 12.56636 \quad 15.70795 \quad 18.84954 \quad 21.99113]$$

$$\mathbf{y} = [0 \quad 2.345 \quad 4.69 \quad 7.035 \quad 9.38 \quad 11.725 \quad 14.07 \quad 16.415]$$

∞ Código Arduino 5.3: sketch cap5_apdinamico1

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_apdinamico1.ino

```

1 //Algoritmo para sumar vectores.
2 #include <stdlib.h>//librería para las funciones malloc(...) y free(...).
3 float *x, *y, *z;//declaración de apuntadores de tipo flotante.
4 int j;//índice o pivote para los apuntadores.
5 void setup(){//subrutina de configuración.
6     Serial.begin(9600); //configuración de comunicación serial.
7 }
8 void loop(){//inicia lazo principal del sketch.
9     Serial.println("Suma de vectores transpuestos: un renglón x 8 columnas " );
10    //Asignación dinámica de memoria, 32 bytes a cada apuntador, representan
11    //8 elementos de 4 bytes.
12    x=(float *)malloc(8*4);//asignación dinámica de 8 elementos de 4 bytes,
13    //es decir, 32 bytes de memoria para el apuntador x.
14    y=(float *)malloc(8*4);//asignación dinámica de 8 elementos de 4 bytes
15    //en otras palabras, se requieren 32 bytes de memoria para el apuntador y.
16    z=(float *)malloc(8*4);//asignación dinámica de 8 elementos de 4 bytes
17    // 32 bytes de memoria para el apuntador z.
18    for (j=0; j<8; j++){//inicialización de valores de los apuntadores.
19        *(x+j)=3.14159*j;//valores de los elementos del apuntador x.
20        *(y+j)=2.345*j;//valores de los elementos del apuntador y.
21        *(z+j)=0; //limpiar los elementos de z.
22    }
23    //Algoritmo de suma de vectores tipo columna o transpuesto.
24    for (j=0; j<8; j++){//sumar elemento a elemento.
25        *(z+j)=*(x+j)+*(y+j);//suma de vectores.
26    }
27    Serial.println("Valores asignados a los vectores presentados en formato de una columna y 8 renglones" );
28    Serial.println("  x          y          z=x+y" );

```

☉ Continúa código Arduino 5.3a: sketch cap5_apdinamico1

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Continuación del sketch cap5_apdinamico1.ino

```

29   for (j=0; j<8; j++){//sumar elemento a elemento.
30       Serial.print(*(x+j),5);//envía elemento del vector x con 5 fracciones.
31       delay(10);//retardo para mantener la secuencia de transmisión serial.
32       Serial.print("    ");//espacio tabular.
33       delay(10);//retardo para mantener la secuencia de transmisión serial.
34       Serial.print(*(y+j),5);//envía elemento del vector y con 5 fracciones.
35       delay(10);//retardo para mantener la secuencia de transmisión serial.
36       Serial.print("    ");//espacio tabular.
37       delay(10);//retardo para mantener la secuencia de transmisión serial.
38       Serial.println(*(z+j),5);//envía elemento del vector z con 5 fracciones.
39       delay(10); //retardo para mantener la secuencia de transmisión serial.
40   }
41   free(x);//libera espacio de memoria asignado al apuntador x.
42   free(y);//libera espacio de memoria asignado al apuntador y.
43   free(z);//libera espacio de memoria asignado al apuntador z.
44   Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n\n" );
45   delay(3000);//retardo entre transmisión serial.
46 }

```

los vectores \mathbf{x} , \mathbf{y} tienen dimensión de un sólo renglón por 8 columnas, así que la suma se realiza componente a componente, en este caso, columna a columna. La línea 18 contiene el código para inicializar los valores de cada apuntador, en el caso del vector \mathbf{z} se limpia con valores cero. Note que, para todos los casos la asignación de valores a los apuntadores se realiza con el operador $*$ y de manera indexada con el pivote \mathbf{j} : $*(\mathbf{x}+\mathbf{j})$, $*(\mathbf{y}+\mathbf{j})$ y $*(\mathbf{z}+\mathbf{j})$.

En la línea 24 se realiza la suma de vectores elemento a elemento entre los apuntadores \mathbf{x} y \mathbf{y} , el resultado se registra en el apuntador \mathbf{z} . A partir de la línea 29

el valor de cada localidad de memoria de los tres apuntadores se envía al monitor serial del ambiente de programación Arduino para su despliegado en forma de texto con cinco fracciones. Para obtener un flujo de comunicación serial estable entre la tarjeta Arduino y la computadora, se utilizan retardos de 10 milisegundos.

Son de particular interés las líneas 19 y 25 donde se ejecutan las siguientes sentencias respectivamente:

$$*(x+j)=3.14159*j; // \text{línea 19.}$$

$$*(z+j)=*(x+j)+*(y+j); // \text{línea 25}$$


En la sentencia $*(x+j)=3.14159*j$; $*$ se emplea como operador de apuntador y como operador aritmético entre una constante de tipo flotante y un número entero. El operador de apuntador tiene mayor prioridad que el operador aritmético de multiplicación, por lo cual no existe ambigüedad en la sintaxis.



La expresión $*(x+j)$ se interpreta como el apuntador x está indexado por el pivote j para recorrer la memoria asignada por **malloc(...)**, los paréntesis () delimitan a la expresión $x+j$ indicando que se escribirá en la j -ésima localidad de memoria a través del operador de apuntadores $*$.



En la sentencia $*(z+j)=*(x+j)+*(y+j)$; $*$ se utiliza como operador a variable puntero, los operandos que se encuentra en el lado derecho del signo igual se lee el valor que contiene la j -ésima localidad de memoria de los apuntadores x y y , en el lazo izquierdo del signo igual se escribe el valor de la suma en la j -ésima localidad de memoria del apuntador z .

Para liberar espacio asignado a cada apuntador se utiliza la función **free(...)**, como se ejemplifica en la línea 41, posteriormente, después de 3 segundos el sketch se repite de manera indefinida (línea 45).

Para descargar el código de máquina del sketch **cap5_apdinamico1** y ejecutarlo en las tarjetas Arduino, el procedimiento es similar al recomendado en la página 7.





5.2.3 Apuntadores y arreglos

Existe una fuerte relación entre las variables puntero y los arreglos, el lenguaje C proporciona dos métodos para acceder a los elementos de un arreglo: la aritmética de los apuntadores y la forma indexada o por pivote de los arreglos, es importante remarcar que la aritmética de las variables puntero es mucho más rápida que la indexada; sin embargo, hay que saber utilizar los operadores aritméticos para evitar colapsos en la ejecución del sketch.

La forma de manipular a un apuntador se realiza de la misma forma que un arreglo y viceversa, en el siguiente cuadro se detallan estas características.

Considere las siguientes sentencias:

```
float *x;  
float vector[5];
```



La sentencia **float *x**; se declara el apuntador **x** de tipo flotante, el cual no está inicializado, es decir no apunta a ninguna localidad de memoria.



La sentencia **float vector[5]**; declara un arreglo de tipo flotante con 5 elementos. Dichos elementos se ubican en las localidades `vector[0]`, `vector[1]`, `vector[2]`, `vector[3]` y `vector[4]`. El elemento `vector[5]` se emplea para indicar el término del arreglo y generalmente tiene el elemento nulo (**null**).



El arreglo **vector** es por naturaleza un apuntador con memoria asignada en su propia declaración, por lo que se pueden referenciar sus elementos de la siguiente forma: ***vector** para el primer elemento, ***(vector+1)**, ***(vector+2)**, ***(vector+3)** y ***(vector+4)** para el segundo al quinto elemento, respectivamente.



La sentencia **x=vector**; obtiene la dirección de memoria del apuntador (arreglo) **vector** y se la asigna al apuntador **x**. En el caso de apuntadores y arreglos no se requiere el operador **&** para obtener la dirección de memoria.



Una vez que la variable puntero x está apuntando al arreglo **vector**, dicho apuntador puede ser utilizado como un arreglo, es decir: para acceder a cada elemento del arreglo se puede utilizar: $x[0]$, $x[1]$, $x[2]$, $x[3]$ y $x[4]$.

El siguiente ejemplo muestra la similitud que existe entre los arreglos y apuntadores.

♣ ♣ ♣ Ejemplo 5.4

Desarrolle un sketch que apuntadores y despliegue el resultado en el monitor serial del ambiente de programación Arduino.

Solución

El cuadro de código Arduino 5.4 describe al sketch **cap5_apuntadorArr** con la programación en lenguaje C que permite ilustrar la similitud que existe entre arreglos y apuntadores. La línea 1 declara un apuntador de tipo flotante, en este momento dicho apuntador no se encuentra inicializado, ya que no tiene una dirección de memoria específica donde apuntar, mientras que, en la línea 2 se declara un arreglo de datos tipo flotante denominado **vector** con 5 elementos, automáticamente se le asigna la cantidad de memoria necesaria (en este caso 5 elementos \times 4 bytes =20 bytes de memoria), el elemento **vector[5]** indica la terminación del arreglo y tiene al elemento nulo (**null**). Es importante reiterar que el arreglo **vector** al momento de la declaración se le asigna una dirección de memoria, por lo tanto, ya está inicializado y puede ser visto como un apuntador ***vector** de tipo flotante con dirección y cantidad de memoria necesaria para almacenar a sus elementos. Para propósitos de indexado sobre el arreglo **vector** y el apuntador **x**, se utiliza la variable de tipo entera **j** en la línea 3.

La velocidad de transmisión serial ha quedado especificada en 9600 Baudios en la subrutina de configuración **setup()** ubicada en la línea 5 y el lazo principal **loop()** del sketch empieza en la línea 7.

En la línea 11 se inicializa al apuntador **x** con la dirección de memoria que ocupa el arreglo **vector**, es decir, **x** apunta a la misma dirección de memoria de **vector**. Observe que, no se utiliza el operador **&** para obtener la dirección de memoria, esto

☞ Código Arduino 5.4: sketch cap5_apuntadorArr

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 Apuntadores, estructuras y uniones.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “Te acerca al conocimiento”.

Sketch cap5_apuntadorArr.ino

```

1 float *x;//declaración de apuntadores de tipo flotante.
2 float vector[5];
3 int j; //índice o pivote para los apuntadores.
4 void setup(){//subrutina de configuración.
5     Serial.begin(9600); //configuración de comunicación serial.
6 }
7 void loop(){//inicia lazo principal del sketch.
8     Serial.println("Relación entre apuntadores y arreglos " );
9     //Inicialización del apuntador x, obtiene la dirección del arreglo vector, en este
10    //caso no se requiere el operador &, por la similitud entre apuntadores y arreglos.
11    x=vector;//obtener la dirección de memoria del arreglo vector.
12    for (j=0; j<5; j++){//inicialización de los elementos del apuntador x y vector.
13        *(x+j)=1.12345*j;//valores de los elementos de x y de vector.
14    }
15    Serial.println("Valores asignados al apuntador x arreglo vector." );
16    Serial.println(" x          vector " );
17    delay(10);//retardo para mantener la secuencia de transmisión serial.
18    for (j=0; j<5; j++){//sumar elemento a elemento.
19        Serial.print(x[j],5);//lee el apuntador x como si fuera arreglo.
20        delay(10);//retardo para mantener la secuencia de transmisión serial.
21        Serial.print("      ");//espacio tabular.
22        delay(10);//retardo para mantener la secuencia de transmisión serial.
23        Serial.println(*(vector+j),5);//lee el arreglo vector como si fuera apuntador.
24        delay(10);//retardo para mantener la secuencia de transmisión serial.
25    }
26    Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n\n" );
    delay(3000);//retardo entre transmisión serial.
27 }
```

se debe a que un apuntador y un arreglo del mismo tipo de datos mantienen la misma naturaleza y por lo tanto, comparten las mismas propiedades.



La asignación de memoria (inicialización) entre apuntadores y arreglos se realiza con el signo igual =, el arreglo debe estar del lado derecho del signo igual, por ejemplo:

```
float *x;//declaración del apuntador  $x$  de tipo flotante.
```

```
float vector[5];//declaración de un arreglo de tipo flotante con 5 elementos.
```

```
x=vector;//asignar dirección de memoria que tiene vector al apuntador  $x$ .
```



La asignación de memoria entre un apuntador y una variable se lleva a cabo con el operador &. Por ejemplo,

```
float *x;//declaración del apuntador de tipo flotante  $x$ .
```

```
float y;//declaración de una variable de tipo flotante.
```

```
x=&y;//asignar la dirección de memoria de  $y$  usando el operador &,
//la variable puntero  $x$  apunta a la dirección de memoria de  $y$ .
```

En la línea 11 se asigna la dirección de memoria que ocupa el arreglo **vector** al apuntador x , de esta forma, apunta a la misma dirección, por lo tanto, puede manipular (leer/escribir) la información registrada en esa localidad de memoria. Observe que no se utiliza el operador & para asignar la dirección de memoria, sólo el signo igual. Observe que en la línea 13 se inicializan los elementos del arreglo **vector** a través del apuntador x , se utiliza la forma indexada $*(x+j)=1.12345*j$.

A partir de la línea 18 se envía la información para ser desplegada con 5 fracciones en el monitor serial del ambiente de programación Arduino, particularmente note que, en la línea 19 el apuntador x se indexa como si fuera arreglo, es decir $x[j]$, mientras que en la línea 23, el arreglo **vector** se procesa como si fuera apuntador $*(vector+j)$. En otras palabras, una vez declarado un apuntador e inicializado, éste puede ser manipulado como si fuera un arreglo de datos. Asimismo, un arreglo puede ser procesado como si fuera un apuntador. Recuerde que la declaración de un arreglo ya incluye la asignación de memoria. Entre cada transmisión de la información por protocolo serial, se utiliza un retardo de 10 milisegundos usando la función **delay(10)**

para mantener la estabilidad de comunicación serial entre la tarjeta Arduino y la computadora donde radica el ambiente de programación Arduino.

Cuando termine el proceso de transmisión de la información, se emplea un retardo de 3 segundos (línea 26), antes que se ejecute una vez más el sketch. La respuesta que se despliega en el monitor serial es el que a continuación se presenta:

Relación entre apuntadores y arreglos.

Valores asignados al apuntador **x**, y al arreglo vector.

x	vector
0.00000	0.00000
1.12345	1.12345
2.24690	2.24690
3.37035	3.37035
4.49380	4.49380

En 3 segundos se ejecuta una vez más el sketch.

Para descargar el código de máquina del sketch **cap5_apuntadorArr** y ejecutarlo en las tarjetas Arduino, consulte la página 7.



5.2.4 Apuntadores y funciones



Apuntadores y funciones

Los capítulos 6 al 14 incluyen ejemplos en lenguaje C de funciones y apuntadores que el lector puede descargar con la programación específica de aplicaciones en ingeniería y ciencias exactas; se resalta la sintaxis de pase de argumentos y tipos de datos que retornan las funciones.

Los apuntadores se encuentran fuertemente relacionados con las funciones mediante pase de argumentos y retorno de resultados. Una función puede manejar de manera normal argumentos de diversos tipos de datos del lenguaje y el pase de argumentos puede ser por medio de apuntadores, en este caso indicando la dirección de la variable por medio del apuntador **&**. En el siguiente ejemplo, se ilustran estos conceptos a detalle.

♣ ♣ Ejemplo 5.5

Desarrolle un sketch que procese como argumentos de entrada apuntadores en una función que retorne la suma de sus argumentos; desplegar el resultado en el monitor serial del ambiente de programación Arduino.

Solución

El sketch **cap5_funcionesap** que se describe en el cuadro de código Arduino 5.5, contiene la programación en lenguaje C que ejemplifica la forma de usar funciones con pase de argumentos a través de apuntadores. Como ha sido costumbre, la velocidad de transmisión serial se establece en 9600 Baudios en la subrutina de configuración **setup()** ubicada en la línea 3 y el lazo principal del sketch, función **loop()** empieza en la línea 11.

El algoritmo consiste en declarar apuntadores de tipo flotante en la línea 1, en este punto, los apuntadores no se encuentran inicializados, es decir, no están apuntando a ninguna localidad de memoria. En la línea 2 se declaran las variables auxiliares que las variables puntero direccionarán.

En la línea 6 se declara la función de prueba de tipo flotante denominada **suma_numeros(float x, float y)** con dos argumentos de tipo flotante, esta función realiza la suma de sus argumentos y retornar este resultado. Note que, en las líneas 13 y 14 los apuntadores **x** y **y** obtienen a través del operador **&** la dirección de memoria que ocupan las variables **w1** y **w2**, respectivamente.

La línea 15 es particularmente clave, debido a que los apuntadores **x** y **y** son utilizados como argumentos en la función **suma_numeros(*x, *y)**, es decir, los apuntadores acceden al contenido de la localidad de memoria de las variables **w1** y **w2** para utilizar el valor registrado en esa localidad y pasarlo como argumento de entrada. Los argumentos de entrada de la función **suma_numeros** son variables de tipo flotante, de ahí que, requieren el valor numérico de los apuntadores, es decir ***x, *y**. El resultado que devuelve la rutina **suma_numeros(...)** es registrado en la variable de tipo flotante **z**, la cual se despliega con cinco fracciones en el monitor serial del ambiente de programación Arduino (línea 16). El sketch se repite después de una pausa de 3 segundos generada por la función **delay(3000)**, línea 18.

☞ Código Arduino 5.5: sketch cap5_funcionesap

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_funcionesap.ino

```

1 float *x, *y;//declaración de apuntadores (globales) de tipo flotante.
2 float w, w1=3.3, w2=4.5;
3 void setup(){//subrutina de configuración.
4     Serial.begin(9600); //configuración de comunicación serial.
5 }
6 float suma_numeros(float x, float y){
7     float z;
8     z=x+y;
9     return z;
10 }
11 void loop(){//inicia lazo principal del sketch.
12     Serial.println("Funciones y apuntadores" );
13     x=&w1;
14     y=&w2;
15     w=suma_numeros(*x, *y);
16     Serial.println(w,5);
17     Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n\n" );
18     delay(3000);//retardo entre transmisión serial.
19 }

```

Otra forma opcional de realizar la programación del mismo algoritmo es a través del sketch **cap5_funcionesap1**, el cual se presenta en el cuadro de código Arduino 5.6. Observe en este caso, que no se requieren declarar apuntadores como en el sketch **cap5_funcionesap**, de hecho en la línea 1, sólo se declaran las variables de trabajo de tipo flotante (globales), que se emplean más adelante en la programación del propio sketch. La función **suma_numeros(float *x, float *y)** ubicada en la línea 5 tiene como argumentos de entrada apuntadores de tipo flotante, en lugar de

variables como en la línea 6 del cuadro de código 5.5. Técnicamente son estilos de programación muy diferentes, ya que observe que en la línea 7 del cuadro de código Arduino 5.6 se realiza la suma aritmética de los argumentos con el contenido de la dirección de memoria a donde apuntan las variables punteros x y y , es decir: $z=*x+*y$; dicha dirección se obtiene con el operador $\&$ con el pase de parámetros a la función o subrutina `suma_numeros` en la línea 12.

∞ Código Arduino 5.6: sketch cap5_funcionesap1

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponibile en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_funcionesap1.ino

```

1 float w, w1=3.3, w2=4.5;
2 void setup(){//subrutina de configuración.
3     Serial.begin(9600); //configuración de comunicación serial.
4 }
5 float suma_numeros(float *x, float *y){
6     float z;
7     z=*x+*y;
8     return z;
9 }
10 void loop(){//inicia lazo principal del sketch.
11     Serial.println("Funciones y apuntadores " );
12     w=suma_numeros(&w1, &w2);
13     Serial.println(w,5);
14     Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n\n" );
        delay(3000);//retardo entre transmisión serial.
15 }
```

Para descargar los códigos de máquina generados en la fase de compilación de los sketches `cap5_funcionesap` y `cap5_funcionesap1` y ejecutarlos en las tarjetas Arduino, consultar el procedimiento recomendado en la página 7.



5.3 Estructuras

UNA estructura es una agrupación de variables que se encuentran declaradas con el mismo nombre (identificador), es decir, se crean variables de diferentes tipos del lenguaje C contenidas en una estructura de datos, cada elemento, forma un campo de la estructura.

La palabra clave **struct** se utiliza para definir una estructura, seguida de al menos un espacio en blanco, nombre o identificador, ya que la estructura de datos es una sentencia. Posteriormente, continúa con llave de apertura {, en otra línea inicia el conjunto de campos formados por las variables que la integran, si son variables del mismo tipo de datos, se utilizan comas para separarlas. Al final del listado de cada campo se inserta punto y coma. En el último campo de la estructura, después del punto y coma se inserta una línea para incluir llave de cierre }, finalizando con el operador punto y coma. Por ejemplo:

```
struct robot{//agrupación de variables de diversos tipos de datos.
    byte bandera_activa;//campo de un byte en memoria.
    int posiciones[6];//arreglo: dos bytes × 6 elementos enteros = 12 bytes.
    int puerto;//campo de dos bytes en memoria
    float vel[6];//arreglo de 4 bytes × 6 elementos flotantes, requiere 24 bytes.
    float torque[6];//4 bytes × 6 elementos=24 bytes
};//la estructura robot necesita un total de 63 bytes en memoria.
```

Una vez que se ha definido la estructura **robot**, se puede utilizar para declarar variables del tipo **robot** de la siguiente manera:

```
struct robot robotA, robotB, robotC;//declaración de variables tipo robot.
```

Con la anterior declaración, las variables **robotA**, **robotB**, **robotC** son del tipo **struct robot**, cada una de estas variables está compuesta por los campos que forman a la estructura robot.

Otra forma de realizar la declaración de variables del tipo robot es la siguiente:

```

struct robot{//agrupación de variables de diversos tipos de datos.
    byte bandera_activa;//un byte en memoria.
    int posiciones[6];//dos bytes × 6 elementos enteros = 12 bytes en memoria.
    int puerto;//dos bytes en memoria
    float vel[6];//4 bytes × 6 elementos flotantes, requiere 24 bytes.
    float torque[6];//4 bytes × 6 elementos, requiere 24 bytes
} robotA, robotB, robotC;

```

Observe que después de la llave de cierre, se declaran las variables **robotA**, **robotB** y **robotC** que son del tipo **robot**, la declaración de las variables finaliza con el operador punto y coma.

Cuando se necesita declarar una sola variable de estructura, entonces en este caso, no se requiere que el nombre o identificador de la estructura esté después de la palabra clave **struct**, después de dicha palabra clave se continúa con llave {, en otra línea de texto inicia el listado de campos con su respectivas declaraciones de variables separadas por comas, al final de cada campo se utiliza el operador punto y coma, cuando se termina el listado de campos se utiliza la llave de cierre }, a continuación el nombre de la estructura de datos cerrando la sintaxis con el operador punto y coma, el ejemplo siguiente ilustra este concepto:

```

struct { //agrupación de variables de diversos tipos de datos.
    byte bandera_activa;//un byte en memoria.
    int posiciones[6];//dos bytes × 6 elementos enteros = 12 bytes en memoria.
    int puerto;//dos bytes en memoria
    float velocidades[6];//4 bytes × 6 elementos flotantes, requiere 24 bytes.
    float torque[6];//4 bytes × 6 elementos, requiere 24 bytes.
} robot;//la variable robot contiene los campos de la estructura que le precede.

```



5.3.1 Referencia a los elementos de una estructura

Para referenciar a los elementos individuales de un campo de la estructura se utiliza el operador punto. Primero se escribe el nombre de la variable, seguido del operador

punto y a continuación (sin dejar espacio en blanco) el nombre del elemento. Por ejemplo, para activar la bandera del robot y asignar la posición en 90 grados al elemento 3 del arreglo de posiciones de la variable **robotA** de estructura **robot** se realiza de la siguiente forma:

```
robotA.bandera_activa=1;//activa bandera del robot.  
robotA.posiciones[3]=90;//posicionar a la articulación tres en 90 grados.
```

El ejemplo 5.6 ilustra la forma de utilizar los elementos o campos de una estructura.

♣ Ejemplo 5.6

Escribir un sketch que asigne valores a los diferentes campos de la estructura tipo **robot**; desplegar la información en el monitor serial del ambiente de programación Arduino.

Solución

El cuadro de código Arduino 5.7 contiene el sketch **cap5_estructuraRobot** que ilustra la forma de asignar valores a los campos que forman la estructura de datos **robot**, la cual inicia en la línea 1. Se realiza la declaración global de tres variables **robotA**, **robotB** y **robotC** del tipo **robot** en la línea 7.

La subrutina de configuración **setup()** se encuentra declarada en la línea 8, la velocidad de comunicación serial se ha programado en 9600 Baudios (línea 9) y la rutina principal **loop()** inicia en la línea 11.

Observe que en las líneas 13 a la 17 se asignan valores a los campos **bandera_activa**, posición de 90 grados (segunda articulación del robot) y número de puerto de la estructura **robotA**, mientras que los campos correspondientes para los puertos de **robotB** y **robotC** se les asigna los números 12 y 13, respectivamente.

A partir de la línea 18 se envía información al monitor serial del ambiente de programación Arduino para desplegar el valor de los campos asignados en las estructuras **robotA**, **robotB** y **robotC**. En la línea 27 se realiza una pausa de un segundo, antes de la próxima ejecución del sketch.

☞ Código Arduino 5.7: sketch cap5_estructuraRobot

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.











Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_estructuraRobot.ino

```

1 struct robot { //variables claves de programación del robot.
2     byte bandera_activa; //bandera que habilita el movimiento del robot.
3     int posiciones[6]; //registro de posiciones de los 6 ejes del robot.
4     int puerto; //puerto de interface.
5     float velocidades[6]; //registro de velocidades de los ejes del robot.
6     float torque[6]; //registro de los pares aplicados a los servomotores del robot.
7 } robotA, robotB, robotC;
8 void setup() {
9     Serial.begin(9600); //configuración de comunicación serial.
10 }
11 void loop() { //inicia lazo principal.
12     Serial.println("Estructuras de datos" );
13     robotA.bandera_activa=1; //activa robot A.
14     robotA.posiciones[1]=90; //Posición de la segunda articulación del robot A.
15     robotA.puerto=11; //número de puerto del robot A.
16     robotB.puerto=12; //número de puerto del robot B.
17     robotC.puerto=13; //número de puerto del robot C.
18     Serial.print("Bandera activa " ); //envía información al monitor serial.
19     Serial.println(robotA.bandera_activa);
20     Serial.print("Posicion de la segunda articulacion robot A " );
21     Serial.println(robotA.posiciones[1]);
22     Serial.print("Puertos de los robots " );
23     Serial.print(robotA.puerto); Serial.print("\t" );
24     Serial.print(robotB.puerto); Serial.print("\t" );
25     Serial.println(robotC.puerto);
26     Serial.println("Espere un segundo: se ejecuta una vez mas el sketch." );
27     delay(1000); //retardo de un segundo en la transmisión serial.
28 }
```

Los pasos requeridos para ejecutar el sketch **cap5_estructuraRobot** en cualquier modelo de tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**; sin embargo, por comodidad al lector a continuación se presentan en forma resumida:

-  Descargar del sitio Web de este libro  el código fuente en lenguaje C del sketch **cap5_estructuraRobot**.
-  En el menú **Herramientas** del ambiente de programación Arduino seleccionar el modelo de tarjeta, puerto USB y configurar velocidad de comunicación serial USB en 9600 Baudios.
-  Compilar el sketch mediante el icono .
-  Descargar el código de máquina a la tarjeta Arduino usando .
-  Desplegar resultados con el monitor serial (activar con .
-  **Importante:** para que los resultados del sketch **cap5_estructuraRobot** se exhiban adecuadamente en el monitor serial del ambiente de programación, maximice la ventana y active la opción **desplazamiento automático**.



5.3.2 Arreglos de estructuras

Para trabajar con variables de arreglos de estructuras se requiere declarar la estructura de datos y posteriormente definir una variable arreglo del tipo de esa estructura. Por ejemplo, para declarar un arreglo de estructuras de 20 elementos de tipo **robot** (definida anteriormente) se realiza de la siguiente forma:

```
struct robot robots_en_proceso[20]; //arreglo de estructura para 20 robots.
```

La anterior declaración crea 20 variables **robots_en_proceso**, cada una de las variables de ese arreglo tiene elementos y campos de la estructura **robot**. En este

punto es importante remarcar que los campos de cada estructura del arreglo tienen diferentes localidades de memoria a pesar que compartan el mismo nombre del campo, los valores que se les asignan son específicos para cada elemento del arreglo y por lo tanto no son iguales entre sí. Por ejemplo:

```
float x,y;  
x=robots_en_proceso[3].posiciones[1];  
y=robots_en_proceso[49].posiciones[1];
```

los valores que adquieren las variables **x,y** no son los mismos, ya que el valor numérico que pueda tener el campo **robots_en_proceso[3].posiciones[1]** es muy diferente al campo **robots_en_proceso[49].posiciones[1]**. Los campos del elemento del arreglo **robots_en_proceso[3]** tienen localidades de memoria diferentes al elemento **robots_en_proceso[49]**, es decir son variables completamente distintas.

♣ ♣ Ejemplo 5.7

Desarrollar un sketch que configure la estructura de datos de un conjunto de 20 robots manipuladores, con 6 ejes cada uno; considere limpiar los registros de velocidad y pares aplicados de cada servomotor e inicialice las posiciones articulares de los 6 ejes en las siguientes posiciones: $[100, 120, 140, 160, 180, 200]^T$ grados.

Solución

El cuadro de código Arduino 5.8 contiene al sketch **cap5_estructuraA** con la declaración de 20 elementos de arreglos del tipo estructura **robots_en_proceso**. Se contempla la estructura descriptiva **robot** para un proceso de automatización con 20 robots manipuladores, definido este indicador como constante tal y como se ilustra en la línea 1, así como el número de ejes de cada robot (ver línea 2). Las variables contador que permiten indizar a los registros de la estructura y también a los arreglos correspondientes a los campos de dicha estructura se declaran en la línea 3. La estructura **robot** se declara de manera global en la línea 4 y el arreglo de 20 estructuras **robots_en_proceso** en la línea 10; la velocidad de transmisión serial entre la computadora y la tarjeta Arduino se establece en 9600 Baudios (línea 12) y la subrutina o función principal **loop()** inicia en la línea 14.

☞ Código Arduino 5.8: sketch cap5_estructuraA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_estructuraA.ino

```

1 #define      num_robots    20 //número de robots.
2 #define      num_ejes      6 //número de ejes de cada robot.
3 int i, j; //variables contadores para las instrucciones for( ; ; ){...}
4 struct robot { //estructura de un robot manipulador.
5     byte bandera_activa; //robot activo.
6     int posiciones[num_ejes]; //registra posiciones de cada servomotor.
7     int puerto; //puerto de interfaz.
8     float velocidades[num_ejes]; //velocidades de movimiento.
9     float torque[num_ejes]; //pares aplicados a los servomotores.
10 } robots_en_proceso[num_robots]; //número de robots en proceso.
11 void setup() { //subrutina de configuración.
12     Serial.begin(9600); //configuración de comunicación serial.
13 }
14 void loop() { //subrutina principal.
15     for(i=0; i<num_robots; i++){ //asigna valores a los campos de la estructura.
16         robots_en_proceso[i].bandera_activa=1; //activa robots.
17         robots_en_proceso[i].puerto=i; //asigna número de puerto para interfaz.
18         Serial.print("Num de robots en proceso ");
19         Serial.print(i); //envía información al monitor serial.
20         Serial.print("\t");
21         Serial.print("Bandera activa ");
22         Serial.print(robots_en_proceso[i].bandera_activa); //indica bandera activa.
23         Serial.print("\t");
24         Serial.print("Num de puerto ");
25         Serial.println(robots_en_proceso[i].puerto);
26         Serial.println("Posiciones velocidades torques");
27         for(j=0; j<num_ejes; j++){ //inicializa posiciones, velocidades y torque.
28             robots_en_proceso[i].posiciones[j]=100+20*j; //posiciones de referencia.

```

∞ Continúa código Arduino 5.8a: sketch cap5_estructuraA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 Apuntadores, estructuras y uniones.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “Te acerca al conocimiento”.

Continuación del sketch cap5_estructuraA.ino

```

29     robots_en_proceso[i].velocidades[j]=0;//limpia velocidades de cada eje.
30     robots_en_proceso[i].torque[j]=0;//limpia torque de cada servomotor.
31     Serial.print(robots_en_proceso[i].posiciones[j]);//datos al monitor serial.
32     Serial.print("      " );
33     Serial.print(robots_en_proceso[i].velocidades[j]);
34     Serial.print("      " );
35     Serial.println(robots_en_proceso[i].torque[j]);
36     }// fin de for(j=0; j<num_ejes; j++){...}
37 }// fin de for(i=0; i<num_robots; i++){...}
38 Serial.println("Espere 5 segundos para ejecutar una vez mas el sketch." );
39 delay(5000); //retardo entre transmisión serial.
40 }//termina función loop(){...}.
41 //Para descargar y ejecutar este sketch en las tarjetas Arduino, consulte el
42 //procedimiento descrito en la página 29.

```

La manera de procesar los campos de cada estructura **robots_en_proceso** es por medio del pivote **i** el cual sirve como índice **i=0, 1**, hasta llegar al elemento 19. Por ejemplo, en la línea 27 se utiliza una instrucción **for(; ;){...}** para acceder a los campos de banderas y asignar número de puerto de cada uno de los 20 robots manipuladores: **robots_en_proceso[i].bandera_activa=1**; y **robots_en_proceso[i].puerto=i**; además, como parte de los campos de la estructura **robot**, existen registros de posiciones, velocidades y pares aplicados de los 6 ejes que forman a un robot manipulador, por lo que, el procesamiento de estos campos se realiza por medio de la instrucción anidada **for(; ;){...}** ubicada en la línea 27 de la siguiente forma: utilizando el pivote **j** **robots_en_proceso[i].velocidades[j]=0**; se limpian los registros de velocidad para cada uno de los seis ejes del robot manipulador.

Finalmente, se emplea la función **delay**(5000) para generar una pausa de 5 segundos (antes de ejecutar una vez más el sketch) con la finalidad de darle tiempo al usuario en revisar los letreros que se despliegan sobre el monitor serial del ambiente de programación Arduino.

Para descargar el código de máquina generado en el proceso de compilación y ejecutar el sketch **cap5_estructuraA** sobre los diversos modelos de tarjetas Arduino, consultar el procedimiento descrito en la página 29.



5.3.3 Campos de bits

Dentro de las ventajas que tiene el lenguaje C, se encuentran la manipulación de bits individuales para variables del tipo **int**, **unsigned** o **signed**; esto es particularmente importante en dispositivos que transfieren información codificada y en interfaces electrónicas. La manipulación de bits es una característica clave en aplicaciones de robótica y mecatrónica, por ejemplo la interface electrónica para habilitar a robots, generalmente se realiza con un bit de encendido/apagado a través de un puerto digital, banderas de alerta del robot como límite de velocidad o torque de saturación se indican por medio de bits.

Los campos de bits son propiedades de la estructura de datos para acceder a los bits individuales de un tipo de dato, mejora la eficiencia y funcionalidad del programa.



Un campo de bits es un tipo especial de elemento dentro de una estructura que define su tamaño en bits.










Un campo de bits tiene que declararse como **int**, **unsigned** o **signed**.



Los campos de bits de longitud uno deben declararse como **unisigned**, debido a que un bit individual no puede tener signo.

La forma general para declarar de un campo de bits en una estructura es la siguiente:

-  Primero la palabra clave **struct**
-  Continúa, al menos con un espacio en blanco y después el identificador o nombre de la estructura
-  Llave que abre { (puede haber un espacio en blanco entre esta llave y el nombre de la estructura)
-  Listado de variables tipo enteros (**int**), definiendo los campos de bits utilizando los modificadores para tipos de datos **unsigned** o **signed**
-  Un espacio en blanco entre el nombre de los campos o elementos, seguido del operando dos puntos : continúa un número natural que indica la longitud o número de bits del cual consta el campo correspondiente.
-  Finaliza este campo con el operador punto y coma.
-  La estructura se cierra con la llave } y punto y coma.

El ejemplo 5.2 ilustra la forma de declarar una estructura con campos de bits.

Código ejemplo 5.2

Campos de bits de una estructura

```
struct puertos {
    unsigned puerto0 : 1;
    unsigned puerto1 : 1;
    unsigned puerto2 : 1;
    unsigned puerto3 : 1;
    unsigned puerto4 : 1;
    unsigned puerto5 : 1;
    unsigned puerto6 : 1;
    unsigned puerto7 : 1;
} puertos_robot;
```

La variable **puertos_robot** es del tipo estructura **puertos**, definida en 8 campos, cada una de un bit, por lo que, los posibles valores que pueden tomar son 0 y 1.

No obstante, la declaración de campos de una estructura no está restringida a ser de un bit, es decir, pueden ser de uno, dos o más bits, tal es el caso del siguiente ejemplo 5.3:

Código ejemplo 5.3

Campos de una estructura con número variable de bits

```
struct puertos {
    unsigned int puerto0 : 1;//campo de un bit.
    unsigned int puertoA : 2;//campo de dos bits.
    unsigned int puertob : 3;//campo de tres bits.
    unsigned int puertoc : 4;//campo de cuatro bits.
} puertos_robotA;
```

Ejemplo 5.8

Escribir un sketch que realice la manipulación de campos de bits para 1, 2, 3 y 4 bits. Desplegar la información en el monitor serial del ambiente de programación Arduino.

Solución

El sketch **cap5_estructuraBits** describe la forma de realizar la manipulación computacional a nivel campos de bits usando la estructura **puertos**; la documentación del sketch se presenta el cuadro de código Arduino 5.9.

En la línea 1 se declara la estructura **puertos** con los campos de bits requeridos para asignar valores y envío de información del monitor serial. En la línea 6 se define la variable **puertos_robotA** del tipo de estructura **puertos**. La subrutina de configuración **setup()** se define en la línea 7 estableciendo la velocidad de comunicación serial en 9600 Baudios y en la línea 10 empieza el lazo principal **loop()**.

De las línea 11 a la 14 se asigna valores a los campos de bits de 1, 2, 3 y 4 bits, respectivamente. De las líneas 16 a la 24 se envía la información al monitor serial para su presentación tabular usando las funciones **Serial.print()** y **Serial.println()**.

∞ Código Arduino 5.9: sketch cap5_estructuraBits

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_estructuraBits.ino

```

1 struct puertos {
2     unsigned int puerto0 : 1;//campo de un bit.
3     unsigned int puertoa : 2;//campo de dos bits.
4     unsigned int puertob : 3;//campo de tres bits.
5     unsigned int puertoc : 4;//campo de cuatro bits.
6 } puertos_robotA;
7 void setup() { //subrutina de configuración.
8     Serial.begin(9600); //configuración de comunicación serial.
9 }
10 void loop() { //inicia lazo principal.
11     puertos_robotA.puerto0=true;
12     puertos_robotA.puertoa=3;
13     puertos_robotA.puertob=7;
14     puertos_robotA.puertoc=15;
15     Serial.print("Campos de bits en una estructura.\n" );
16     Serial.print("Puerto de un bit " );
17     Serial.println(puertos_robotA.puerto0);
18     Serial.print("Puerto de dos bits " );
19     Serial.println(puertos_robotA.puertoa);
20     Serial.print("Puerto de tres bits " );
21     Serial.println(puertos_robotA.puertob);
22     Serial.print("Puerto de cuatro bits " );
23     Serial.println(puertos_robotA.puertoc);
24     Serial.println("Espere 3 segundos para ejecutar una vez más el sketch." );
25     delay(3000); //retardo entre transmisión serial.
26 }
27 //Para descargar y ejecutar este sketch cap5_estructuraBits en las tarjetas
28 //Arduino, consulte el procedimiento descrito en la página 29.
```

El sketch `cap5_estructuraBits` se ejecuta de manera iterativa por intervalos de 3 segundos usando la función `delay(3000)`, como se ilustra en la línea 25.

La descarga de código y ejecución del sketch `cap5_estructuraBits` en las tarjetas Arduino es similar al procedimiento que se describe en la página 29.



♣ ♣ ♣ Ejemplo 5.9

Desarrollar la programación en lenguaje C para que un sketch realice la manipulación de campos de una estructura por medio de un apuntador. Desplegar la información en el monitor serial del ambiente de programación Arduino.

Solución

El sketch `cap5_estructuraAp` (ver documentación en el cuadro de código Arduino 5.10) contiene la programación en lenguaje C para manipular computacionalmente por medio de un apuntador a la estructura `datos` (línea 2), la cual contiene varios campos, desde enteros de 1, 2, 3 y 4 bits, 2 bytes y número flotante de 4 bytes. En la línea 10 se realiza la declaración del apuntador tipo estructura `datos`; este apuntador no contiene una localidad o dirección de memoria, por lo que, requiere se inicializado posteriormente antes de utilizarse.

En la línea 11 se ubica la subrutina de configuración donde se define la velocidad de comunicación serial en 9600 Baudios entre la tarjeta Arduino y el ambiente de programación que radica en la computadora. El lazo principal del sketch inicia a partir de la línea 14.

La estructura `datos` ocupa 8 bytes de memoria, debido a la composición de sus campos (ver línea 2), es decir: 4 bytes del campo flotante `x`, 2 bytes del campo entero `i` y 9 bits de los campos enteros `puerto0`, `puertoa`, `puertob` y `puertoc`, ya que no hay fracciones de bytes, en este caso corresponde a 2 bytes. La forma de calcular la cantidad de memoria requerida por la estructura `datos` es a través de la función `sizeof(datos)` como se indica en la línea 16.

La asignación dinámica de memoria para el `apuntador` se realiza por medio de la

∞ Código Arduino 5.10: sketch cap5_estructuraAp

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_estructuraAp.ino

```

1 #include <stdlib.h> //librería para las funciones malloc(...) y free(...).
2 struct datos { //estructura de datos con sus campos asociados.
3     unsigned int puerto0 : 1; //campo de un bit.
4     unsigned int puerto1 : 2; //campo de dos bits.
5     unsigned int puerto2 : 3; //campo de tres bits.
6     unsigned int puerto3 : 4; //campo de cuatro bits.
7     int i; //campo de 2 bytes.
8     float x; //campo de 4 bytes.
9 };
10 struct datos *apuntador; // apuntador tipo estructura datos.
11 void setup() { //subrutina de configuración.
12     Serial.begin(9600); //configuración de comunicación serial.
13 }
14 void loop() { //inicia lazo principal del sketch.
15     Serial.print("Número de bytes de la estructura datos: ");
16     Serial.println(sizeof(datos)); //8 bytes de memoria para estructura datos.
17     //Asignación dinámica de memoria del apuntador por medio de casting.
18     apuntador=(struct datos *)malloc(sizeof(datos)); //casting para el apuntador.
19     Serial.print("Apuntador a estructura.\n" );
20     //operador flecha para acceder a los campos de la estructura.
21     Serial.print("Campo de un bit " );
22     apuntador->puerto0=true;
23     Serial.println(apuntador->puerto0);
24     Serial.print("Campo del numero entero de 2 bytes " );
25     apuntador->i=12345;
26     Serial.println(apuntador->i);
27     Serial.print("Campo del numero flotante de 4 bytes " );
28     apuntador->x=3.14159;

```

☞ Continúa código Arduino 5.10a: sketch cap5_estructuraAp

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Continuación del sketch cap5_estructuraAp.ino

```

29   Serial.println(apuntador->x,5);//desplegar con 5 fracciones.
30   free(apuntador);//libera espacio de memoria asignado al apuntador.
31   Serial.println("En 3 segundos se ejecuta una vez mas el sketch.\n\n");
32   delay(3000); //retardo entre transmisión serial.
33 }
```

función **malloc(...)**, cuyo argumento es **sizeof(datos)** asignando 8 bytes de memoria al **apuntador**. Observe que se utiliza un *casting* (**struct datos ***) para acoplar correctamente el resultado que retorna la función **malloc(...)** a la naturaleza específica del **apuntador** tipo estructura **datos** (ver línea 18); de esta forma, se eliminan problemas técnicos que suceden generalmente cuando se trabaja con apuntadores y éstos no están adecuadamente inicializados.

A partir de la línea 22 se asignan valores a los campos de la estructura **apuntador**, por medio del operador flecha **->**, así como la lectura de esos campos se envía por transmisión serial al ambiente de programación Arduino por su despliegue en forma de texto.







La función **free(apuntador)** en la línea 30 libera la cantidad de memoria asignada al **apuntador** por la función **malloc(...)**. Posteriormente, después de 3 segundos (línea 32), el sketch **cap5_estructuraAp** se ejecuta por ciclos indefinidos.

Los pasos requeridos para descargar el código de máquina y ejecutar el sketch **cap5_estructuraAp** en cualquiera de las tarjetas Arduino se encuentran concentrados en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**. No obstante, por sencillez al lector, en la página 29 se encuentra dicho procedimiento en forma compacta.



5.4 Uniones

EN el lenguaje C, una unión es un tipo de dato especial que está formada por varios campos de variables de diferentes tipos de datos, similar al de la estructura, con la diferencia que en la unión todos sus campos comparten la misma localidad o espacio de memoria, de tal forma que si una variable modifica su valor, de la misma manera lo harán los demás campos. La declaración de la unión es por medio de la palabra clave **union** y tiene una sintaxis similar al de la estructura. A continuación se describe el procedimiento para declarar una variable de tipo unión:

-  Se emplea la palabra clave **union**, en seguida insertan al menos un espacio en blanco, continúa el nombre o identificador, ya que la unión de datos es una sentencia.
-  Posteriormente, continúa llave de apertura {.
-  En otra línea inicia el conjunto de campos formados por las variables que la integran, si son variables del mismo tipo de datos, se utilizan comas para separarlas.
-  Al final del listado de cada campo se inserta punto y coma.
-  En el último campo de la unión, después del punto y coma se inserta una línea para incluir llave de cierre }.
-  Finalizando la unión de datos con el operador punto y coma.

Una variable de tipo unión puede tener campos de diferentes tipos de datos tales como: **boolean**, **char**, **byte**, **int**, **long**, **float**, estructuras, apuntadores, etc. El área de memoria se ajusta automáticamente al tipo de dato más grande. Todos los campos comparten el mismo espacio de memoria, aun modificando por ejemplo campos de bits de una estructura, esto alterará el contenido de los demás campos de la unión.

Considere la unión **tipo_datos** compuesta por diversos números de tipo **byte**, **int**, **long int** y **float**:

```
union tipo_datos{ //números enteros y flotante.
    byte b; //número entero de 8 bits.
    int i; //número entero de 16 bits.
    long li; //número entero de 32 bits.
    float x; //número flotante de 32 bits.
};
```

Una vez que se ha definido la unión **tipo_datos**, se puede utilizar para declarar variables de **tipo_datos** de la siguiente manera:

```
union tipo_datos datoA, datoB, datoC; //declaración de variables tipo_datos.
```

Cuando se declara una unión, el compilador del lenguaje C automáticamente asigna área de memoria de acuerdo a la variable más grande definida en los elementos de la unión, para el caso de **tipo_datos** reserva espacio de memoria de 32 bits. La figura 5.2 muestra la forma en que la unión **tipo_datos** comparte el espacio de memoria para todos sus elementos.

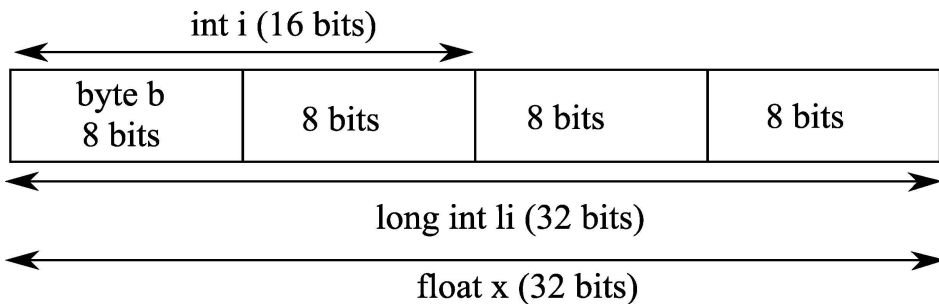


Figura 5.2 Unión **tipo_datos**.

El área de memoria que ocupa la unión **tipo_datos** es de 32 bits debido a la naturaleza de las variables **long li** y **float x**; los primeros 8 bits del espacio de memoria corresponden al dato **byte b**, el elemento de tipo entero **i** se ubica en los primeros 16 bits, mientras que el espacio completo de 32 bits está ocupado por los

elementos **li** y **x**. A pesar que todos los elementos son de diferente tipo, comparten el mismo espacio de memoria. Esto significa que cualquier asignación de valores de cualquier elemento de la unión, repercute inmediatamente en todos los elementos.



A diferencia de la estructura de datos, cuando se declara una **unión**, sus elementos no pueden ser inicializados, ya que esto generaría error de compilación.



Después de declarar la unión de datos, se puede asignar valores a cualquier elemento ya sea en el programa principal **loop() {...}** o en alguna subrutina, en este caso, debe tomarse en cuenta que el valor numérico de cada uno de sus elementos se modifica, debido a que comparten el mismo espacio de memoria.

Otra forma para realizar la declaración de una unión es de la siguiente manera:

```
union tipo_datos{//números enteros y flotante.
    byte b;//número entero de 8 bits.
    int i;//número entero de 16 bits.
    long int li;//número entero de 32 bits.
    float x;//número flotante de 32 bits.
} datosA, datosB, datosC;
```

Observe que después de la llave de cierre, se declaran las variables **datosA**, **datosB** y **datosC** (separadas por comas) que pertenecen a la unión **tipo_datos**, la declaración de las variables finaliza con el operador punto y coma. Para acceder a los elementos de la unión se procede de la misma forma como en el caso de las estructuras.



Si se accede directamente sobre la unión, se utiliza el operador punto.



Cuando se accede a través de un apuntador, se emplea el operador flecha **->**.

♣ Ejemplo 5.10

Escribir un sketch que procese la variable **datosA** que pertenece a unión **tipo_datos**, el campo del número real asigne el valor de **datosA.x=32.34567** y analice como se modifican los demás elementos de la unión; despliegue los resultados en el monitor serial del ambiente de programación Arduino.

Solución

El cuadro de código Arduino 5.11 contiene el sketch **cap5_union** con el algoritmo en lenguaje C para procesar la variable **datosA** de la unión **tipo_datos**.

En la línea 1 se realiza la declaración de la unión de datos **tipo_datos**, observe que ninguno de sus elementos se encuentra inicializado. En la línea 7 se define la variable **datosA** de la forma **tipo_datos**. La configuración de velocidad de 9600 Baudios para la transmisión serial se lleva a cabo en la subrutina **setup()** en la línea 8 y en la línea 11 empieza la rutina principal **loop(){...}**.

En la línea 12 se lleva a cabo la asignación del elemento flotante **datosA.x=32.34567**, el cual automáticamente asigna el valor de 247 al elemento de 8 bits **datosA.b**, el número entero **datosA.i** adquiere el valor de 25079 y el número entero largo de 32 bits **li** toma el valor de 1107386871. Todos los elementos de la unión **datosA** comparten la misma localidad de memoria, y la variación de cualquier bit de alguno de los elementos modifica el valor de todos los elementos de la unión.

Es importante recalcar que en el monitor serial del ambiente de programación Arduino se despliegue correctamente el valor del elemento **datosA** con las cinco fracciones que le fueron asignado, en la línea 12 se utiliza la función **Serial.println** con cinco fracciones a exhibir.

El sketch **cap5_union** se ejecuta en intervalos de tiempo de 3 segundos, por lo que, en la línea 23 se emplea la función **delay(3000)** para generar dicha secuencia.

Para compilar, descargar y ejecutar el sketch **cap5_union** en cualquier modelo de tarjetas Arduino es necesario realizar el procedimiento que se describe en la página 29.

∞ Código Arduino 5.11: sketch cap5_union

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_union.ino

```

1 union tipo_datos {
2     byte b;//número entero de 8 bits.
3     int i;//número entero de 16 bits.
4     long int li;//número entero de 32 bits.
5     float x;//número flotante de 32 bits.
6 };
7 union tipo_datos datosA;//declaración de la variable datosA.
8 void setup() { //subrutina de configuración.
9     Serial.begin(9600); //configuración de comunicación serial.
10 }
11 void loop() { //inicia lazo principal.
12     datosA.x=32.34567;//asignación del elemento flotante.
13     Serial.print("Unión de datos.\n" );
14     Serial.print("Número entero de 8 bits " );
15     Serial.println(datosA.b);
16     Serial.print("Número entero de 16 bits " );
17     Serial.println(datosA.i);
18     Serial.print("Número entero de 32 bits " );
19     Serial.println(datosA.li);
20     Serial.print("Número flotante de 32 bits " );
21     Serial.println(datosA.x,5);//desplegar información con cinco fracciones.
22     Serial.println("Espere 3 segundos para ejecutar una vez más el sketch." );
23     delay(3000); //retardo entre transmisión serial.
24 }
25 //Para descargar y ejecutar el sketch cap5_union en las tarjetas
26 //Arduino, consulte el procedimiento descrito en la página 29.
```



♣ ♣ ♣ Ejemplo 5.11

Modificar la programación del sketch 5.10 para procesar a la unión **tipo_datos** por medio de un apuntador con asignación dinámica de memoria.

Solución

El cuadro de código Arduino 5.12 describe la programación en lenguaje C con el uso de un apuntador **tipo_datos** del sketch **cap5_unionA**, el cual modifica la forma de procesar computacionalmente a la **unión** de datos del sketch **cap5_union** (ver cuadro de código 5.11) del ejemplo 5.10.

En la cabecera del sketch **cap5_unionA**, en línea de código 1 se utiliza la librería **stdlib.h** requerida para la función de asignación dinámica **malloc(...)** y para liberar el espacio de memoria del apuntador, el cual se realiza por medio de la función **free(...)**. La línea 2 contiene la declaración de la unión **tipo_datos** con sus respectivos campos, siendo los de mayor longitud de 4 bytes, debido a la naturaleza de los tipos de datos **long int** y punto flotante **float**.

La línea 8 tiene la declaración del apuntador a la unión **tipo_datos**. Es importante recordar en esta fase del sketch, el **apuntador_union** no tiene asignado memoria y que tampoco está apuntado hacia alguna localidad de memoria específica; únicamente tiene la estructura de la unión **tipo_datos** y antes de utilizarse requiere asignarle memoria SRAM.

La velocidad de comunicación serial entre la tarjeta Arduino y la computadora se ha establecido en 9600 Baudios, programada en la subrutina **setup()** de la línea de código 9; la subrutina principal **loop()** inicia en la línea 12.

La asignación dinámica de memoria SRAM para **apuntador_union** se realiza en la línea 14 por medio de la función **malloc(...)**; observe que el cálculo de bytes que determina la cantidad de memoria requerida se obtiene a través de la función **sizeof(tipo_datos)**; en este caso, corresponden 4 bytes debido a los tipos de datos **long int** y **float**. Note que, por medio de un *casting* (**union tipo_datos ***) se realiza la correcta conversión o acoplamiento de la asignación del tipo de apuntador que retorna la función **malloc(...)** con el **apuntador_union**. Para acceder a los campos










del apuntador de la unión de datos se emplean los operadores signo y flecha \rightarrow , tal y como se ilustra en la línea 16 en el caso del campo flotante **x** del **apuntador_union**.

La línea 18 verifica que efectivamente son 4 bytes de memoria los que se requieren asignar para **apuntador_union**, esta información se despliega en el monitor serial del ambiente de programación Arduino. En la línea 21 se accede al valor que contiene la sección de memoria compartida por el campo del número entero tipo byte **b** desplegando su contenido en el monitor serial; de igual forma para los demás campos de la unión: entero **i** de 16 bits, entero largo **li** de 32 bits y número flotante **x** de 32 bits que se encuentran descritos de la línea 23 a la 27.

La liberación de memoria asignada al **apuntador_union** se lleva a cabo utilizando la función **free(...)** como se indica en la línea 28. A partir de este momento, el **apuntador_union** ya no tiene memoria asignada, encontrándose sin dirección específica de memoria, se han destruido los valores de los campos que contenía.

Usando la función **delay(3000)** en la línea 30 para generar 3 segundos, el sketch **cap5_unionA** se ejecuta una vez más debido al ciclo indefinido que introduce el lazo principal **loop()**, empezando en la línea 12.

Los pasos requeridos para ejecutar el sketch **cap5_unionA** en cualquier modelo de tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**. No obstante, a continuación se presenta en forma sucinta el procedimiento recomendado:

-  Descargar del sitio Web de este libro  el código fuente en lenguaje C del sketch **cap5_unionA**.
-  En el menú **Herramientas** del ambiente de programación Arduino seleccionar el modelo específico de tarjeta, puerto USB y configurar velocidad de comunicación serial USB en 9600 Baudios.
-  Compilar el sketch mediante el icono .
-  Descargar el código de máquina a la tarjeta Arduino usando .
-  Desplegar resultados con el monitor serial (activar con .

☞ Código Arduino 5.12: sketch cap5_unionA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Sketch cap5_unionA.ino

```

1 #include <stdlib.h> //librería para las funciones malloc(...) y free(...).
2 union tipo_datos { //unión de datos y sus correspondientes campos.
3     byte b; //número entero de 8 bits.
4     int i; //número entero de 16 bits.
5     long int li; //número entero de 32 bits.
6     float x; //número flotante de 32 bits.
7 };
8 union tipo_datos *apuntador_union; //apuntador del tipo unión de datos.
9 void setup() { //subrutina de configuración.
10     Serial.begin(9600); //configuración de comunicación serial.
11 }
12 void loop() { //inicia lazo principal del sketch.
13     //Asigna espacio de memoria del apuntador de la unión tipo_datos.
14     apuntador_union=(union tipo_datos *)malloc(sizeof(tipo_datos)); //casting
15     //Operador flecha -> para acceder a los campos de la unión.
16     apuntador_union->x=32.34567; //asignación del valor deseado.
17     Serial.print( "Número de bytes de la unión tipo_datos: " );
18     Serial.println(sizeof(tipo_datos)); //el tamaño de la unión es de 4 bytes.
19     Serial.print( "Apuntador a unión de datos.\n" );
20     Serial.print( "Campo del número entero de 8 bits. " );
21     Serial.println(apuntador_union->b); //acceso al número entero tipo byte.
22     Serial.print( "Campo del número entero de 16 bits: " );
23     Serial.println(apuntador_union->i); //acceso al número entero.
24     Serial.print( "Campo del número entero de 32 bits: " );
25     Serial.println(apuntador_union->li); //acceso al número entero tipo long.
26     Serial.print( "Campo del número flotante de 32 bits " );
27     Serial.println(apuntador_union->x,5); //acceso al número flotante.
28     free(apuntador_union); //libera espacio de memoria asignado al apuntador.

```

∞+ Continúa código Arduino 5.12a: sketch cap5_unionA

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “**Te acerca al conocimiento**”.

Continuación del sketch cap5_unionA.ino

```

29   Serial.println("Espere 3 segundos para ejecutar una vez mas el sketch." );
30   delay(3000); //retardo entre transmisión serial.
31   //Para descargar y ejecutar el sketch cap5_unionA en las tarjetas Arduino,
32   //consulte el procedimiento recomendado en la página 46.
33 }
```



Importante: para que los resultados del sketch **cap5_unionA** se exhiban adecuadamente en el monitor serial del ambiente de programación Arduino, maximice la ventana y active la opción **desplazamiento automático**.



♣ ♣ Ejemplo 5.12

Modificar la programación del sketch 5.12 para procesar a la unión **tipo_datos** por medio de un apuntador con asignación directa de memoria.

Solución

La versión modificada del sketch **cap5_unionA** (ver cuadro 5.12) se encuentra ubicada en el cuadro de código Arduino 5.13 denominado **cap5_unionAp**. En este caso, no se utiliza la función **malloc(...)** para la asignación dinámica de memoria del apuntador, ni la función **free(...)** para liberar la memoria asignada, en tal circunstancia, no es necesario emplear en la cabecera del sketch la librería **stdlib.h**. Observe que en la línea 7 se declaran **datosA** y **apuntador_union**; ambos pertenecientes a la unión **tipo_datos**. Por medio del operador ampersand & el apuntador obtiene la dirección de memoria de la variable **datosA** (línea 13) y por lo tanto acceder al contenido de los campos dentro del espacio de memoria, la escritura y lectura de datos se realiza mediante los operadores signo y flecha **->**.

☞ Código Arduino 5.13: sketch cap5_unionAp

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en 

Capítulo 5 **Apuntadores, estructuras y uniones.**

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: “Te acerca al conocimiento”.

Sketch cap5_unionAp.ino

```

1 union tipo_datos { //unión de datos y la integración de sus campos.
2     byte b; //número entero de 8 bits.
3     int i; //número entero de 16 bits.
4     long int li; //número entero de 32 bits.
5     float x; //número flotante de 32 bits.
6 };
7 union tipo_datos datosA, *apuntador_union; //apuntador a tipo_datos.
8 void setup() { //subrutina de configuración.
9     Serial.begin(9600); //configuración de comunicación serial.
10 }
11 void loop() { //inicia lazo principal del sketch.
12     //Por medio del operador & se asigna la dirección de memoria al apuntador.
13     apuntador_union=&datosA; //la dirección de memoria de datosA se asigna al apuntador.
14     //Operadores signo y flecha -> para acceder a los campos de la unión de datos.
15     apuntador_union->x=32.34567; //operador flecha.
16     //Envío de información (campos de la unión) al monitor serial.
17     Serial.print( "Apuntador a unión de datos.\n" );
18     Serial.print( "Campo del número entero de 8 bits " );
19     Serial.println(apuntador_union->b);
20     Serial.print( "Campo del número entero de 16 bits " );
21     Serial.println(apuntador_union->i);
22     Serial.print( "Campo del número entero de 32 bits " );
23     Serial.println(apuntador_union->li);
24     Serial.print( "Campo del número flotante de 32 bits " );
25     Serial.println(apuntador_union->x,5);
26     Serial.println( "Espere 3 segundos para ejecutar una vez mas el sketch. " );
27     delay(3000); //retardo entre transmisión serial.
28 }

```

En forma similar al procedimiento descrito en la página 46, la descarga y ejecución del sketch **cap5_unionAp** en las tarjetas Arduino se puede realizar desde el ambiente de programación Arduino.



Estructuras

En el capítulo 14 Web Manejo de interrupciones, se presentan ejemplos con apuntadores y arreglos de estructuras y sus aplicaciones potenciales.



Uniones

Ejemplos didácticos con uniones de datos se ilustran dentro del capítulo 14 Web Manejo de interrupciones, el lector puede descargarlos y analizar el código para mejorar su dominio con este tipo de datos.

5.5 Resumen

EL lenguaje C por sus características es una de las mejoras herramientas de programación para realizar aplicaciones de automatización, robótica, mecatrónica y en general para todas las de la ingeniería y ciencias exactas. Las herramientas de programación se realizan cuando se utilizan una clase particular de variables con determinadas propiedades como lo son las variables puntero o apuntadores, estructuras y uniones de datos.

La declaración de apuntadores se caracteriza por utilizar el operador * y puede ser de cualquier tipo de dato válido del lenguaje C, incluyendo estructuras, uniones y funciones. Esta herramienta de programación es muy poderosa, pero al mismo tiempo la más peligrosa cuando no se sabe emplearla correctamente, puede ocasionar colapsos en la ejecución del sketch, cuyos errores no son fáciles de detectar, ya que la etapa de compilación no los detecta. En este capítulo se han presentado los fundamentos de los apuntadores, forma de declaración, inicialización, asignación dinámica y liberación de memoria, similitud con los arreglos, forma de indexar o

utilizarlos con pivote, tipos de operadores, así como la precaución para utilizarlos correctamente.

Otra forma de variables importantes en el lenguaje C son las estructuras de datos, las cuales agrupan a diferentes elementos o campos de diversos tipos de datos válidos del lenguaje C, pueden aceptar también la definición de enteros de uno, dos o más bits, procesarlos computacionalmente mediante asignación de valores o manipulación de operadores, ampliando el tipo de aplicaciones de la ingeniería. Se han presentado diversos ejemplos que ayudan a entender su forma de trabajo y programación, tipos de operadores y la manera de acceder a sus elementos.

Semejante a la estructura de datos, se encuentra la unión de datos la cual comparte el mismo espacio de memoria para todos sus elementos o campos. De tal forma que cuando se modifica un elemento, esto causa el mismo efecto en todos los elementos agrupados en la propia unión. Conserva los mismos tipos de operadores que la estructura.

Una cantidad importante de ejemplos didácticos y correctamente documentados han sido presentados para mejorar la comprensión, programación y aplicación de este tipo de variables del lenguaje C. El código fuente de todos los ejemplos se encuentran disponibles en el sitio Web de esta obra y corren en las plataformas electrónicas de tarjetas Arduino.

5.6 Referencias selectas

LAS variables puntero o apuntadores, estructura y uniones de datos permiten expandir las prestaciones y aplicaciones del lenguaje C. Existe una diversidad de literatura sobre estas herramientas de programación; no obstante, a continuación se recomienda al lector la siguiente literatura que le ayudara a mejorar sus conocimientos sobre los temas presentados en este capítulo.



Brian Kernighan and Dennis Ritchie. “*The C Programming Language*”.
Prentice-Hall, 1978.



Herber Schildt. “*C: The Complete Reference*”. 4th Edition, McGraw-Hill Osborne Media, 2000.



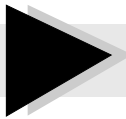
Herber Schildt. “*C: Manual de referencia*”. Cuarta edición, McGraw-Hill, 2001.



Stephen G. Kochan. “*Programming in C: A complete introduction to the C language*”. Third Edition, Sams Publishing, 2004.



<http://arduino.cc/en/Reference/HomePage>



5.7 Problemas propuestos

LOS apuntadores, estructura de datos y uniones son importantes herramientas de programación del lenguaje C. A continuación se presentan un conjunto de problemas con el propósito de evaluar los conocimientos adquiridos por el lector en este capítulo.

5.7.1 Considere las siguientes declaraciones de variables:

```
char c[14]="Hola mundo..." ;  
int i=8;  
long li=23456;  
double x=9.345;
```

Realice la correspondiente declaración de apuntadores en función de las variables previamente declaradas, desarrolle la programación necesaria en lenguaje C para que un sketch despliegue en el monitor serial del ambiente de programación Arduino los valores de la variables a través de apuntadores.

5.7.2 Sea la siguiente declaración de variables:

```
char texto[10]="Lenguaje C" ;
```

```
char buffer[10], *ap;
```

Desarrolle un programa o sketch que permita transferir carácter a carácter de la variable **texto** a **buffer** por medio del apuntador **ap**. Despliegue el resultado en el monitor serial. Precaución: considere la inicialización del apuntador **ap** antes de utilizarlo.

5.7.3 Desarrollar un sketch que permita almacenar los primeros 50 números positivos de tipo entero largo (**long**) en memoria SRAM. Para este propósito, utilice un apuntador con asignación de memoria dinámica. Desplegar el resultado en el monitor serial.

5.7.4 Considere las siguientes matrices:

$$A = \begin{bmatrix} 3.4 & 6.7 & 1.2 \\ 2.4 & 4.6 & 1.34 \\ 6.1 & 4.1 & 7.8 \end{bmatrix}$$

$$B = \begin{bmatrix} 5.1 & 4.2 & 7.4 \\ 3.3 & 9.1 & 5.4 \\ 8.3 & 8.6 & 2.1 \end{bmatrix}$$

desarrollar los siguientes algoritmos:

- Suma de matrices.
- Multipliación de matrices.
- Obtener la matriz inversa de A y B .
- Obtener la matriz transpuesta de A y B .

En la implementación de cada uno de los algoritmos utilice apuntadores de tipo flotante.

5.7.5 Una estructura de datos puede tener como elemento o campo a otra estructura de datos como la que se indica en la siguiente declaración:

```
struct robot{
    int puertoA;
    int puertoB;
```

```

float posiciones[6];
float velocidades[6];
float torques[6];
struct comandos{
    int robot1 : 1;
    int robot2 : 1;
    int robot3 : 5;
};
long bandera_activa;
};

```

Escribir un sketch que permita inicializar los campos de la estructura **robot** de la siguiente forma: los campos de tipo entero en cero, los campos flotantes con el valor de 3.45 y el de tipo **long** con el valor de 15001. Presente los resultados en el monitor serial.

5.7.6 Sea la siguiente unión de datos:

```

union datos{
    int i;
    char c;
    float x;
    long j;
    struct comandos{
        int bit0 : 1;
        int bit1 : 1;
        int datox : 6;
    };
};

```

- ¿Cómo están distribuidos los elementos de la unión **datos** en el espacio de memoria que comparten?
- ¿Cuál es el tamaño del espacio de memoria que ocupa la unión **datos**?
- Si el campo **datosx** de la estructura de datos **comandos** tiene el valor de 9 ¿Cuál será el valor de los campos restantes de la unión de datos?
- Desarrolle un sketch que permita desplegar el resultado en el monitor serial.