Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Tabla de contenidos

5.1. Actividades

5.1.1. Arrays como casos especiales de punteros
5.1.2. Gestión de celdas en el móvil
5.1.3. Arrays de punteros
5.1.4. Uso de punteros en C

5.1. Actividades

5.1.1. Arrays como casos especiales de punteros

Plan de trabajo

Supón que tienes una declaración como ésta:

```
int table[] = {10, 20, 30, 40, 50};
```

Si imprimes `table[0]`, obtendrás el valor `10`. Ahora bien, ¿qué pasa si imprimes el valor `*table`? ¿Imprimirá algo o dará error? Si crees que dará un error porque `table` es un array y no un puntero, te equivocas; el nombre de un array es tratado en C como un puntero. Si imprimes `*table`, obtendrás también `10`.

Puedes hacer que un puntero apunte al primer elemento de un array asignando simplemente el nombre del array a la variable puntero. Si un array es referenciado por un puntero, podrás acceder a los elementos en el array con ayuda del puntero, como muestra el siguiente ejemplo:

```
#define SIZE 10
char *ptr_char;
char chars[SIZE];
ptr_char = chars;
```

El nombre del array es un atajo de acceder a la dirección del primer elemento del array. Siguiendo con el ejemplo, la última línea podría sustituirse por `ptr_char = &chars[0];`.

Así mismo, puedes incrementar o decrementar un puntero. Si lo incrementas, la dirección que tiene como valor incrementa: `ptr_char++;` Esto no significa que el puntero se incremente siempre en 1; se incrementa en tantos bytes como tamaño ocupe la variable a la que referencia. De esta manera, `chars[2]` y `*(ptr_char + 2)` dan el mismo resultado.

Teniendo esto en cuenta, calculad la media entre los elementos de dos arrays. Se pide para ello que imprimáis por pantalla los elementos de dos arrays y que calculéis e imprimáis la media entre cada par de elementos. Para ello, cread en vuestro entorno de desarrollo un fichero llamado `arrays_as_pointers.c`, e implementad en él lo siguiente (veréis que es el mismo programa que el de arrays básicos, pero ahora tendréis que enviar el array en forma de puntero):

1. Una función que imprima por pantalla los elementos de un array de enteros: `void print_array(int *array);`
2. Una segunda función que calcule la media entre cada par de elementos de un array y que la vaya

imprimiendo por pantalla: `void calculate_average(int *array1, int *array2);`. Para la primera posición de cada array, sumará los dígitos y los dividirá por dos, y ese resultado lo sacará por pantalla, y así con el resto de posiciones.

3. Una función `main` que declare e inicialice dos arrays de 10 enteros con los dígitos que queráis, por ejemplo `int array1[] = {1,5,7,3,12,...};`, y que haga uso de la primera función para imprimir los arrays y de la segunda para calcular la media.

5.1.2. Gestión de celdas en el móvil

Plan de trabajo

Una aplicación del móvil necesita almacenar las celdas que ofrecen cobertura en cada momento. El número de celdas no se sabe de antemano pues depende de la situación del dispositivo. Cada celda tiene un identificador único (entero positivo) y un nivel de la calidad de la señal (número entre 0 y 100). En todo momento, la aplicación distingue a una de las celdas como la "actual" (la que da mejor cobertura en ese instante). Cada cierto tiempo, esta información se actualiza dependiendo de la calidad de la señal.

Las siguientes dos funciones ya están implementadas en la aplicación:

- `... new_cell()`. Devuelve un puntero a una nueva estructura de celda. El tipo de dato del resultado es el utilizado para definir la estructura con la información de la celda.
- `void remove_cell(...)`. Función opuesta a la anterior, recibe un puntero a una estructura de datos de una celda y se deshace de ella. Hay que llamar a esta función para cerciorarse de que esta estructura "desaparece".

Tienes que escribir las siguientes porciones de código:

- Definición de la estructura de datos para almacenar la información de una celda y declaración de las variables necesarias para almacenar los datos que se piden.
- `void fill_fields(...)`. Función que asigna los valores de los campos pasados como parámetros a las variables de una estructura cuyo puntero se recibe como parámetro. Decide primero qué parámetros debe recibir la función y luego escribe el código.
- `void update_current(...)`. Función que asigna como celda "actual" la que tiene mayor calidad de señal.
- ¿Qué cambio harías en la definición de la estructura de datos para almacenar la información de forma "circular" (es decir, que estén ordenadas como un círculo)?

Para ello puedes seguir los pasos:

1. ¿Cómo vas a representar el conjunto de celdas? ¿Sabes de antemano cuántas celdas tienes que manipular?
2. Define las estructuras de datos necesarias para almacenar el conjunto de celdas. ¿Cómo distingues a la celda actual?
3. ¿Qué prototipo crees que tiene la función "`new_cell`"?
4. Define el prototipo de la función "`fill_fields`". Escribe su código.
5. Implementa la función "`update_current`".
6. ¿Qué quiere decir exactamente una "estructura circular" y cómo se puede implementar en C?

5.1.3. Arrays de punteros

 **Plan de trabajo**

En muchas ocasiones, es muy útil declarar un array de punteros. Es muy común cuando queremos tener un array de cadenas de caracteres. Si una cadena de caracteres puede escribirse como `char *c;` un array de cadenas de caracteres podrá escribirse como `char **c;` o `char *c[x];`.

Dado el array de punteros siguiente:

```
#define SIZE 7
char *str[SIZE] = {"Lunes",
                   "Martes",
                   "Miércoles",
                   "Jueves",
                   "Viernes",
                   "Sábado",
                   "Domingo"};
```

Crea un fichero llamándole `arrays_of_pointers.c` e implementa una función con el prototipo `void print_strings(char **str);` que imprima por pantalla las 7 cadenas.

5.1.4. Uso de punteros en C

 **Plan de trabajo**

1. Prepara el entorno de desarrollo para crear, compilar y ejecutar programas cortos.
2. Leer el capitulo sobre punteros (40 minutos). Escribir todas las dudas que aparezcan y clarificarlas mediante consulta con el profesor o posteando en el foro de la asignatura.
3. Resuelve los diez primeros ejercicios del capítulo. Si tras resolverlos tienes dudas sobre el funcionamiento de los punteros acude a consultas con el profesor. Si ves que el concepto lo tienes claro, resuelve los problemas restantes.