Clase 6: Arreglos y memoria

Mauricio Gómez García

Índice

Introducción **2**

Definición **3**

Memoria **5**

Direcciónes de memoria 5
Tamaños en memoria 6
Arreglos y memoria 7
Tamaño 7
Dirección 7

Práctica **9**

Conclusión **11**

Introducción

Al momento de trabajar con lenguajes de programación es común tener datos del mismo tipo y que estén relacionados, por ejemplo un índice de direcciones de memoria o una lista de caracteres en un string. Es por esto que es importante conocer la funcionalidad e implementación de los arreglos en C.

De igual modo, se verán conceptos relativos a memoria y cómo funcionan los accesos a memoria en C. No se abarcará manejo de memoria, ya que es un tema mucho más complejo de lo que se puede abarcar en la sesión.

En esta sesión se explorará qué es un arreglo, cómo definirlo y cómo trabajar con ello, así como las partes más técnicas de cómo funcionan. Esto se verá en conjunto com los temas de memoria para tener ejemplos prácticos de cómo funciona la memoria en los programas que se desarrollan.

Definición

Un arreglo es una lista de valores que se pueden guardar y acceder desde un mismo nombre de variable. Es decir, una vez que se tenga declarada la variable y los valores del arreglo, se pueden acceder a cada uno de sus elementos.

En C los arreglos tienen muchas propiedades que son importantes conocer ántes de empezar a implementarlos. En primer lugar, es importante conocer que cada arreglo realmente apunta al primer elemento del arreglo, o más precísamente al sector de memoria en donde existe dicho elemento. Seguido del primer elemento se guardan los valores de todos los demás elementos sequencialmente. Debido a esto, es obligatorio asignar el espacio en memoria del arreglo prévio a establecer valores.

Dado que los elementos se encuentran en orden secuencial en memoria, y al no saber qué habrá después de dichos sectores (ya que la memoria es volátil) no es posible cambiar el tamaño de un arreglo.

Por último, es importante conocer que el tamaño de un arreglo dependerá de la cantidad de elementos que hayan en el, y el tipo de dato que contiene.

Los arreglos en C se declaran de dos modos, ya sea con la cantidad de elementos que tendrá sin definir valores o con los valores definidos de entrada:

```
int arr1[100];
int arr2[] = {1, 2, 3};
```

Nótese que en ambos casos se usan los corchetes `[]` al momento de definir el nombre de nuestros arreglos. Esto indica que la variable será un arreglo, en contraste con variables regulares.

En la primera línea se define un arreglo que contiene valores de tipo `int` y con un tamaño de `100` elementos.

En la segunda línea definimos un arreglo sin establecerle explícitamente la cantidad de elementos, pero dado que le damos los valores del arreglo el compilador puede saber que hay tres (`1, 2, 3`). Al momento de establecer los valores de un arreglo, se deben de poner separados por comas y entre llaves `{}`.

Se pueden acceder a los elementos de un arreglo llamando el nombre de la variable en donde se guarda con el índice del elemento que queremos buscar. Es importante conocer que el índice **siempre** inicia en `0`. Utilizando el ejemplo prévio de `arr2`:

```
printf("El primer elemento es %d\n", arr2[0]);
```

Utilizando este código la salida a la consola debe ser la siguiente:

```
El primer elemento es 1
```

Memoria

En C es importante conocer los conceptos básicos de memoria, ya que dadas muchas implementaciónes y métodos que se utilizan, se considera un lenguaje que no es seguro en memoria.

Ya hemos mencionado la función de los arreglo, y ahora profundizaremos en el concepto de memoria para conocer por qué es importante entender el rol de manejo de memoria en C.

Toda información que fluye por nuestro programa viaja a traves de la memoria RAM, la cual es volátil. Gracias a que es posible asignar datos a un espacio o dirección de memoria, también es posible conocer en dónde se asignó el dato.

Direcciónes de memoria

Al asignar memoria para nuestras variables, también debemos asignar el tamaño o espacio que ocupará el dato. Por ejemplo, un valor `int` ocupará `4` bytes de memoria, mientras que un `char` tendrá un tamaño de `1`. Conociendo esto, si no conocemos en dónde se guardó el dato, o el tipo de dato que se guardó (lo cual podemos saber del tipo de dato) nos será imposible leer y escribir datos a la memoria. De hecho, si se intenta acceder a espacios que no han sido asignados de la manera correcta se obtendrá un error de sistema por violación a espacio de memoria no autorizado.

Al trabajar con variables de cualquier tipo, es importante reconocer que al llamar a una variable lo que realmente se hace es llamar a una dirección de memoria en específico que contiene el valor que se guardó en nuestra variable, y esa dirección se puede conseguir al preceder el nombre de una variable existente con el operador `&`:

```
int x = 10;
printf("La direccion de la variable x es %x", &x);
```

Utilizando este ejemplo, podríamos esperar una salida parecida a la siguiente (***OJO***: El valor de la dirección siempre será distinto en cada ejecución del programa):

```
La direccion de la variable x es a82a844
```

Tamaños en memoria

Como se menciona préviamente, las variables ocupan cierto espacio en bytes de memoria. En muchos casos conocer el tamaño en memoria es sencillo, únicamente se requiere conocer el tipo de dato que es para obtener su valor. La instrucción `sizeof()` devuelve el valor del tamaño en bytes de la variable o valor que recibe como parámetro, lo cual nos ayuda a trabajar con tamaños en memoria.

Utilizando el ejemplo prévio, el valor de `x` sería de `4` bytes de memoria, ya que es un valor de tipo `int`.

Arreglos y memoria

Como se ha mencionado préviamente, entender el rol que tiene la memoria relativo a los arreglos es súmamente importante. Tomando en cuenta lo visto préviamente es más fácil conocer por qué se debe conocer el tamaño del arreglo prévio a utilizarlo. Pero entonces, ¿Cómo conocemos el tamaño de un arreglo?

Tamaño

Como se menciona préviamente, un arreglo es símplemente una lista contígua en memoria de elementos del mismo tipo. Conociendo esto, debemos conocer dos cosas para obtener el tamaño del arreglo:

1) La cantidad de elementos que conformarán el arreglo
2) El tipo de dato que almacenará

Observemos el siguiente ejemplo:

```
int arr[] = {1, 2, 3, 4, 5}
```

Conociendo cómo se definió el arreglo podemos calcular su tamaño. En primera instancia conocemos que tenemos cinco valores, y también conocemos que son de tipo `int`. Entonces conocer el tamaño total del arreglo es símplemente multiplicar el valor del tamaño del tipo de dato por la cantidad de elementos:

```
sizeof(int)*5
```

En este caso, un valor `int` tiene un tamaño de `4` bytes, y multiplicado por `5` nos da un tamaño total de `20` bytes en memoria.

Dirección

Como hemos visto, las variables en donde se guardan los arreglos realmente nos apuntan al primer elemento de dicho arreglo. Esto es importante conocer, ya que la variable en si no guarda los datos del arreglo, pero gracias a que los valores se encuentran de manera contigua (es decir, uno tras otro) podemos conseguir los valores de todos los elementos.

Conociendo esto, si se requiere acceder a todos los elementos de un arreglo es necesario aceder a cada uno de ellos individualmente,

ya que acceder a únicamente la variable nos devolverá la dirección en donde se encuentra el primer elemento. Observemos el siguiente ejemplo:

```
int arr[] = {1, 2, 3};
printf("Los valores del arreglo son %x\n", arr);
```

En otros lenguajes, se esperaría que se impriman todos los elementos de nuestro arreglo, pero conociendo mejor su verdadero funcionamiento se puede entender por qué obtendríamos una salida como la siguiente:

```
Los valores del arreglo son 6a891e1c16
```

Entonces, si se quisieran acceder a todos los elementos de nuestro arreglo será necesario iterar sobre cada uno de los elementos. Observemos el siguiente ejemplo:

```
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3};
    printf("Los valores del arreglo son:\n");
    for (int i=0; i<3; i++) {
        printf("    (%x: %d)\n", &arr[i], arr[i]);
    }
    printf("\n");
    return 0;
}
```

La salida de este programa será parecida a lo siguiente:

```
Los valores del arreglo son:
    (91b835dc: 1)
    (91b835e0: 2)
    (91b835e4: 3)
```

Nótese como hemos impreso tanto la dirección de memoria como el valor de cada elemento. En este caso, como se tienen elementos de tipo `int` podemos observar como se van aumentando `4` bytes en la dirección de cada elemento.

Práctica

En esta sección se realizará un ejercicio para cimentar mejor los conceptos prévios, incluyendo conceptos de sesiones pasadas. Se realizará un programa que inicialize un arreglo con caracteres `char`, tambien conocido como un string, y se imprimirá la cadena de texto al revés.

```c
#include <stdio.h>

int main() {
  // Creacion de arreglo
  char str[] = "Este string es un arreglo!";

  // Imprimir arreglo regularmente
  printf("Imprimamos el arreglo\n");
  printf("%s\n", &str);

  // Calcular cantidad de elementos
  int len = sizeof(str)/sizeof(str[0]);

  // Imprimir cadena al reves
  printf("Imprimamos el arreglo al reves\n");
  for (int i=len-1; i>=0; i--) {
    printf("%c", str[i]);
  }
  printf("\n");

  return 0;
}
```

Nótese que en este caso no conocemos la cantidad de elementos, por lo que no podemos diréctamente decir cuántas iteraciones hacer en nuestro bucle `for`. Es por esto que es necesario calcular la cantidad de elementos. Si sabemos que todos los elementos tienen el mismo tamaño (ya que son del mismo tipo de dato) y podemos obtener el tamaño de la variable (`sizeof()` nos da el tamaño del espacio de memoria reservado para variables), podemos calcular el tamaño dividiendo el tamaño total sobre el tamaño de un elemento del arreglo.

Seguido de esto, podemos iniciar en el último elemento. Sabemos el

tamaño del arreglo, por lo que el último valor será el que tenga el índice del tamaño **menos uno**, debido a que los arreglos empiezan en `0`.

Por último, iteramos en sentido contrario (es decir, reduciendo el valor de nuestro iterador) para imprimir todos los caracteres en orden opuesto.

La salida de este programa es la siguiente:

```
Imprimamos el arreglo
Este string es un arreglo!
Imprimamos el arreglo al reves
!olgerra nu se gnirts etsE
```

Conclusión

Conocer cómo funcionan los arreglos en conjunto con la memoria en C forman parte escencial para poder realizar implementaciones de la manera correcta. Entender cómo funciona a nivel estructural un arreglo y saber cómo acceder a sus elementos nos permite crear herramientas que pueden tener muchos usos, y nos permite manejar datos de una manera mucho más óptima a crear una infinidad de variables.

De igual modo, conocer más a profundidad cómo funciona la memoria en C nos permite saber cómo funcionan las computadoras y los programas que ejecutamos en ellas, y así realizar mejores implementaciones de código.

Tarea opcional

Para el mejor entendimiento de los conceptos tocados en la sesión se recomienda implementarlos en un ejercicio que los utilice. Se recomienda realizar un programa que tome el valor de diez valores y los guarde programáticamente en un arreglo (con un bucle). Seguido de esto, realizar la suma de los valores e imprimir la dirección de cada elemento junto con el valor de la suma de los elementos. Esta actividad ayudará a cimentar mejor las ideas vistas en cuestión de arreglos y memoria.

De igual modo se recomienda investigar los conceptos básicos de los apuntadores, concepto que será muy importante conocer al trabajar con C, y que se podrá revisar a profundidad en sesiones futuras.