**Programación en C/C++: Trabajando con punteros**

En el contexto de la programación en C/C++ trabajar con punteros es algo que asusta a la mayoría porque es un tema complicado de entender, pero una vez que se conoce la lógica te das cuenta que no es tan terrible como se pensaba. En éste artículo desarrollaré la lógica de los punteros y sus utilidades para que lo comprendamos mejor.

**Nota**

Si nunca has programado en C o C++, te recomiendo primero aprender a programar en el lenguaje, ya que es necesario conocerlo un poco para entender el tema de los punteros.

Como se que a la mayoría de nosotros, los programadores (o los que intentamos serlo), nos gusta aprender algo viendo código directamente, mil veces me han explicado algo, pero no lo entiendo hasta ver el código. Asi que trataré de explicar de esa manera para que quede clarísimo el tema. Vamos al grano.

**Entendiendo los punteros**

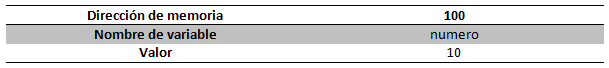
Un puntero es básicamente una **dirección de memoria**. Muchas veces hemos hecho esto:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-trabajando-con-punteros/)

|  |  |
| --- | --- |
| 1  2  3 | int numero;  numero = 10;  printf(“Numero: %i”,numero); |

Obteniendo la salida: “Numero: 10”.

Pero, ¿Qué hace el computador en su interior? El computador trabaja con direcciones de memoria, imaginemos los siguientes valores:

[](http://www.maestrosdelweb.com/images/2012/08/img11.png)

Tenemos nuestra variable *numero*, cuyo valor es *10*, y su dirección de memoria es *100*. Cuando escribimos la línea “numero = 10; “, le estamos diciendo al computador que en la dirección 100 escriba el valor 10.

Ahora es cuando veremos un puntero, he aquí un código equivalente al anterior:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-trabajando-con-punteros/)

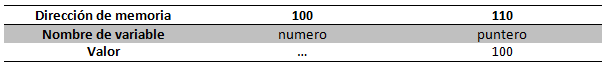
|  |  |
| --- | --- |
| 1  2  3  4 | int numero;  int \* puntero = &Numero;  \*puntero = 10;  printf(“Numero: %i”,numero); |

Nuevamente la salida es: “Numero: 10”.

¿Qué hicimos aquí? Vayamos línea por línea para que quede claro:

* **int numero**: Declaramos nuevamente la variable numero.
* **int \* puntero = &numero**: Declaramos un puntero llamado puntero al cual le asignamos la dirección de memoria de numero (el operador & nos sirve para obtener la dirección de memoria de una variable, en este caso, 100).
* **\*puntero = 10**: Con esto estamos diciéndole al computador: “A lo que esta señalando el puntero (la dirección 100), asignale el valor 10”.
* **printf(“Numero: %i”,numero)**: Mostramos el valor de numero.

Para que quede más claro, veamos los valores que corresponden a nuestras variables hasta la segunda línea:

[](http://www.maestrosdelweb.com/images/2012/08/img21.png)

**Nota**

1. Para ver el contenido de un puntero, podemos escribir “printf(“%p”,puntero);”
2. Los valores de las direcciones de memoria han sido cambiados para mayor comprensión, por ejemplo, la dirección de “numero” en mi computador dio “0022FF18”.

Como vemos, la variable puntero es una variable más, tiene su dirección de memoria (110) y un valor. Lo que hace especial a los punteros, es que alojan una dirección de memoria, en este caso 100, que corresponde a la dirección de memoria de la variable numero (que contiene basura debido a que no la hemos inicializado). Entonces cuando escribimos “\*puntero = 10”, decimos: En la dirección 100, escribe el valor 10. Quedando de esta manera:

[http://www.maestrosdelweb.com/images/2012/08/img3.png](http://www.maestrosdelweb.com/images/2012/08/img3.png)

**Operadores &, “\*” y la función Scanf**

Para que quede más claro, les dejo los casos de uso de los operadores “&” y “\*”:

**Operador &**

Se utiliza para obtener la dirección de memoria de una variable. Se puede implementar con cualquier tipo de variable.

**Operador “\*”**

Se utiliza sólo con punteros. Su función es la de acceder al contenido de la dirección almacenada.

Quiero recalcar que siempre que trabajen con punteros, deben apuntarlo a alguna variable. De lo contrario apuntará a una dirección cualquiera y puede (y es lo mas probable) que te cause errores, además de que tu programa no funcionará como debería.

**Función Scanf()**

Ahora que saben el uso del operador “&”, quizás entiendan el uso de la típica función scanf, veamos un ejemplo de como se utiliza para leer una variable de tipo entero y guardar el valor en la variable:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-trabajando-con-punteros/)

|  |  |
| --- | --- |
| 1 | scanf("%i",&variable); |

Pueden notar el uso del operador “&” antes de “variable”. Así es, scanf recibe un puntero del tipo que se le indica en el primer argumento (“%i indica un entero, por lo tanto, el segundo argumento debe ser de tipo puntero a entero).

En éste primer artículo aprendimos sobre los punteros y sus operadores. En una próxima entrega veremos casos de uso: Punteros para pasar argumentos por referencia a funciones, Arreglos y Matrices con punteros.

### Punteros para pasar argumentos por referencia a funciones

Supongamos que tenemos una función que hace unos cálculos con 2 números que le pasemos, esta es su declaración:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | void calculos(int primero, int segundo)   {   primero += 10;   segundo += 20;   } |

Básicamente esta función suma 10 al primer argumento y 20 al segundo.

Ejecutamos las siguientes líneas de código:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3 | int primero = 0, segundo=0;   calculos(primero,segundo);   printf("Primero: %i, Segundo: %i",primero,segundo); |

Cuya salida es: “Primero: 0, Segundo: 0”.

**¿Porque no cambió los valores de nuestras variables como le indicamos?** Esto es porque en C los valores que les asignamos a nuestras funciones se pasan por valor, es decir, se le pasa una copia de lo que ingresemos como argumento. Es por esto que no se modifica el valor de nuestras variables, ya que lo que modificamos en la función, se mantendrá modificado sólo en esa función.

Para modificar el valor de nuestras variables, debemos pasar estas variables por referencia, es decir, pasarle a la función una referencia de nuestra variable, en nuestro caso le pasaremos la dirección de memoria de ellas, es decir, un puntero.

Recordemos que para acceder a la dirección de memoria de una variable escribiamos “&variable”, por lo tanto, haremos esto para llamar a la función cálculos:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | calculos(&primero,&segundo); |

Ahora, lo que le estamos pasando a la función no es de tipo entero (int), si no, de tipo puntero a entero (int \*), por lo tanto debemos cambiar su declaración para que quede de esta manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | void calculos(int \* primero, int \* segundo)   {   primero += 10;   segundo += 20;   } |

Ahora analicemos esto:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | primero += 10; |

Sabemos que primero es un puntero, por lo tanto, escribir lo anterior no tiene mucha utilidad para lo que queremos hacer, ya que estamos sumándole 10 a la dirección de memoria y no al valor que contiene esa dirección, que es lo que queremos. Por lo tanto, la función final debe quedar así:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | void calculos(int \* primero, int \* segundo)   {   \*primero += 10;   \*segundo += 20;   } |

Ahora si ejecutamos nuestro código, la salida que produce es: “Primero: 10, Segundo: 20”.

### Arreglos con punteros

Acabamos de ver una utilidad de los punteros, ahora veremos otra: Utilizar punteros para crear arreglos y trabajar con ellos. Como sabemos, si queremos declarar un arreglo de 5 enteros, llenarlo con valores y mostrar en pantalla, podemos hacerlo de la siguiente manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4 | int numeros[5] = {0,1,2,3,4};   int i;   for(i=0;i<5;i++)   printf("%i ",numeros[i]); |

Este programa produce la siguiente salida: “0 1 2 3 4”.

Ahora, vean el siguiente código:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int numeros[5] = {0,1,2,3,4};   int i;   int \* puntero = numeros;   for(i=0;i<5;i++){   printf("%i ",\*puntero);   puntero++;   } |

Si lo ejecutas, verás que produce la misma salida. La explicación de esto radica en que un arreglo son sólo variables que estan dispuestas en la memoria una al lado de otra, como se ve en la siguiente tabla (la cual corresponde a los valores de nuestras variables antes de entrar al bucle for):

[http://www.maestrosdelweb.com/images/2012/09/tabla.png](http://www.maestrosdelweb.com/images/2012/09/tabla.png)

Tenemos 5 variables llamadas números (les pongo el mismo nombre para que se entienda la idea). Estas 5 variables dispuestas una al lado de otra corresponden al arreglo números. Si nos damos cuenta, puntero tiene el valor 100, el cual corresponde al primer elemento del arreglo, por lo tanto, números es un puntero, el cual apunta al primer elemento del arreglo que creamos.

Entendido esto, debería tomar sentido que en el primer ciclo del bucle for, el programa nos muestre como salida un 0, debido a que es lo que contiene la dirección 100. Luego de esto, tenemos una línea que dice “puntero++”. Lo que hace esto, es cambiar lo que apunta puntero, por la dirección que se encuentra a su lado, es decir, 110.

Ahora utilizaremos punteros para reservar memoria, que es lo equivalente a crear un arreglo.

La línea de código:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | int numeros[5]; |

Reserva espacio para 5 enteros, puede ser escrita mediante punteros de la siguiente manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | int \* numeros = (int\*)malloc(sizeof(int)\*5); |

### Nota

Para utilizar la función **malloc** es necesario incluir la librería “stdlib.h” en C o “cstdlib” en C++.

**Analicemos este código**

Tenemos un puntero que apunta a un entero, nada que no hayamos visto previamente. Luego de eso, tenemos algo completamente desconocido (o que quizás hayas usado antes sin tener idea de para que servía), la sentencia **“(int\*)malloc(sizeof(int)\*5);”**.

Malloc es una función que reserva la cantidad de memoria que le indiquemos, en este caso, estamos reservando 5 veces el peso de un entero, cuyo peso es 4 bytes, te preguntarás ¿porque no escribimos directamente “malloc(20)”?, la respuesta es que un entero no tiene el mismo peso para todos los computadores en los que se utiliza. Tenemos una función que reserva espacio y lo que retorna es un puntero al primer elemento reservador, pero el problema es que es de tipo “void \*” (recordemos que void significa que no tiene tipo), por lo cual, le hacemos un cast a “int\*” con la línea “(int\*)”.

Con nuestro arreglo listo, podemos trabajar de la misma manera que con un arreglo común y corriente, por ejemplo, podemos modificar el primer elemento de la siguiente manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | numeros[0] = 0;   numeros[1] = 1;   numeros[2] = 2;   numeros[3] = 3;   numeros[4] = 4; |

Ahora, recordando lo que vimos anteriormente, veamos la siguiente sentencia:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | \*numeros = 0; |

Con esto, estamos diciendo que el primer elemento del arreglo vale 0. Por lo tanto, podríamos hacer esto:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | (\*numeros) = 0;   numeros++;   (\*numeros) = 1;   numeros++;   (\*numeros) = 2;   numeros++;   (\*numeros) = 3;   numeros++;   (\*numeros) = 4; |

Pero el problema ahora radica en que no tenemos un puntero que apunte al primer elemento, ya que el valor de numeros es 110 (recordemos que la dirección del primer elemento es 100). Bueno, ya se, podríamos volver fácilmente escribiendo “numeros–” un par de veces, pero esto podría confundirnos si tenemos un código relativamente largo.

Por lo tanto, la forma anterior no es muy recomendada (a menos que creemos un puntero auxiliar y sea ese puntero el que aumentemos). Una forma equivalente de hacer lo anterior sin perder el inicio del arreglo, es esta:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | \*numeros = 0;   \*(numeros+1)= 1;   \*(numeros+2)= 2;   \*(numeros+3)= 3;   \*(numeros+4)= 4; |

Para entenderlo, veamos la segunda línea:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | \*(numeros+1)= 1; |

Le estamos diciendo al computador, que lo que apunta numeros más 1 (es decir, una dirección hacia la derecha), valga 1. Hubiera sido incorrecto escribir:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | (\*numeros+1)= 1; |

Ya que estaríamos diciendo que lo que apunta numeros + 1 = 1. Es decir, sería como escribir (suponiendo que la primera posición tiene el valor 5): 5 + 1 = 1. ¿No tiene sentido verdad?

Un punto importante, es que después de reservar memoria con malloc y de utilizarla (y estar seguro que no volveremos a utilizar), debemos liberarla con la función free. Su uso en este caso sería de la siguiente manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | free(numeros); |

Esto se debe hacer debido a que si no lo hacemos, la memoria seguira reservada y estaremos gastando memoria innecesaria.

Como último punto respecto a la utilidad de los punteros para ser utilizados como arreglos, quiero recalcar que podemos usar un puntero como una cadena de caracteres (String), para guardar una cadena simplemente podemos escribir:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3 | char \* cadena = "Maestros";   cadena = "del web";   printf("%s",cadena); |

Lo cual no da como salida: “del web”. Sin punteros, esto no hubiera sido posible, debido a que si hubiéramos usado arreglos, no se hubiera podido reasignar la cadena de caracteres cadena, debido a que estaríamos intentar cambiar el tamaño del arreglo de 9 a 8 (recuerda que el caracter de fin de cadena “” también cuenta como un caracter más).

### Matrices con punteros

Debemos pensar en una matriz como un puntero a un puntero. Anteriormente teníamos un arreglo hecho con punteros, pues bien, ahora tenemos un arreglo de arreglos (puntero a puntero). Un puntero a puntero (de tipo entero), se declara de la siguiente manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | int \*\* matriz; |

Pero recordemos que debemos reservar el espacio necesario para nuestra matriz, para esto utilizamos la función malloc:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1 | int \*\* matriz = (int\*\*)malloc(sizeif(int\*)\*2); |

Con esto estamos creando un puntero a puntero de tipo int, es decir, un puntero a un arreglo de tipo int. Pero este arreglo al que estamos apuntando no tiene espacio, no hemos reservador memoria, por lo cual, también debemos inicializarlo:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2  3  4 | int \*\* matriz = (int\*\*)malloc(sizeof(int\*)\*2);   int i=0;   for(i=0;i<2;i++)   matriz[i] = (int\*)malloc(sizeof(int)\*5); |

Quiero recalcar el hecho de tener que hacer esto, ya que es un error común el no hacerlo. La lógica nos dice que tenemos un arreglo de arreglos. Pero esos arreglos a los que apuntamos, no tienen espacio y ese espacio no se reserva automáticamente, debemos inicializarlo nosotros. Así que tenlo en cuenta y que no se te olvide.

Hecho esto, tenemos una matriz de 2×5 de tipo entero. Para liberar la memoria en este caso, debemos liberar cada arreglo de la matriz, de esta manera:

[?](http://www.maestrosdelweb.com/editorial/programacion-en-c-uso-de-punteros/)

|  |  |
| --- | --- |
| 1  2 | for(i=0;i<2;i++)   free(matriz[i]); |

Esto sería todo, espero que les haya quedado claro este tema que es muy complicado de entender la primera vez que se oye de él. Como consejo, practiquen utilizando punteros y así será todo mucho más claro.