Punteros

Concepto de puntero

El valor de cada variable está almacenado en un lugar determinado de la memoria, caracterizado por una dirección (que se suele expresar con un número hexadecimal). Se mantiene una tabla de direcciones que relaciona el nombre de una variable con la dirección de memoria. Gracias a los identificadores (nombre de las variables) no hace falta que el programador sepa siquiera la dirección de memoria donde están almacenados sus datos. Sin embargo, en ocasiones es más conveniente trabajar con las direcciones de memoria que con los nombres de las variables.
El lenguaje C dispone del operador de dirección (&) que permite determinar la dirección de una variable y de un tipo de variable destinadas a contener la dirección de otra variable. Éstas últimas se llaman punteros o apuntadores.

Un puntero es una variable que puede contener la dirección de otra variable.

Los punteros están almacenados en algún lugar de la memoria y tienen su propia dirección (incluso se puede tener un puntero a puntero).
Se dice que un puntero apunta a una variable si su contenido es la dirección de esa variable. Un puntero ocupa 4 bytes de memoria y se debe declarar o definir de acuerdo al tipo de dato al que apunta.

```
int *aptrEntero;
char *aptrChar;
float *aptrFloat;
```

A partir de ese momento, se dice que la variable aptrEntero podrá contener una dirección de memoria de una variable del tipo de datos int, lo mismo con aptrChar con char y aptrFloat con float.

Operador de dirección (&)

Mediante el operador de dirección podremos obtener la dirección de memoria de una variable (inclusive si ésta variable es un puntero). Podemos almacenar una dirección de memoria de la siguiente manera:

```
#include <iostream>
using namespace std;

int main(void){
   int *aptrEntero;
   int varEntero = 10;
   aptrEntero = &varEntero;
   cout << "Direccion de memoria de varEntero : " << &varEntero;
   cout << endl << endl;
   cout << "Contenido de aptrEntero : " << aptrEntero;
   cout << endl << endl;
   cout << "Direccion de memoria de aptrEntero : " << &aptrEntero;
   system("pause >nul");
}
```

Salida:

```
Direccion de memoria de varEntero : 0x22ff40

Contenido de aptrEntero : 0x22ff40

Direccion de memoria de aptrEntero : 0x22ff44
```

En el ejemplo visto anteriormente se puede observar como en la línea 7 se le asigna a nuestro puntero a entero *aptrEntero* la dirección de memoria de la variable *varEntero* y luego como al listar la dirección de memoria de varEntero y el contenido de aptrEntero ambos tienen la misma dirección de memoria (líneas 8 y 10).
Por último se observa como en la línea 12 se lista la dirección de memoria de nuestro puntero aptrEntero y ésta es una dirección distinta a la de varEntero, lo cual tiene sentido porque como habíamos dicho anteriormente, los punteros también son variables.
Se podría decir entonces que de un puntero se pueden llegar a obtener tres valores:
1 – La dirección de memoria de nuestra variable puntero.
2 – La dirección de memoria a donde está apuntado (el valor de nuestra variable puntero).
3 – El valor de la variable a donde está apuntado

A partir de lo visto hasta ahora, sólo podemos obtener lo explicado en 1 y 2. Para poder obtener el valor de la variable a donde el puntero está apuntando, necesitamos hacer uso del operador de indirección (*).

Operador de indirección (*)

Para acceder al valor depositado en la dirección de memoria a la que apunta un puntero se debe utilizar el operador de indirección (*).

Ejemplo:

```
#include <iostream>
using namespace std;

int main(void){
   int *aptrEntero, **punteroaPuntero, *otroPuntero;
   int varEntero = 10;
   aptrEntero = &varEntero;
   cout << "El valor de varEntero mediante el puntero 'aptrEntero': " << *aptrEntero;
   cout << endl << endl;
   *aptrEntero = 90;
   punteroaPuntero = &aptrEntero;
   otroPuntero = aptrEntero;
   cout << "El valor de varEntero : " << varEntero;
   cout << endl << endl;
   cout << "El valor de varEntero mediante un puntero del puntero 'aptrEntero' : " <<
*(*punteroaPuntero);
   *(*punteroaPuntero) = -5;
   cout << endl << endl;
   cout << "El valor de varEntero mediante el puntero 'otroPuntero' : " << *otroPuntero;
   system("pause >nul");
}
```

Salida:

```
El valor de varEntero mediante el puntero 'aptrEntero': 10

El valor de varEntero : 90

El valor de varEntero mediante un puntero del puntero 'aptrEntero' : 90

El valor de varEntero mediante el puntero 'otroPuntero' : -5
```

El ejemplo anterior es un tanto complejo, comencemos por lo fácil, en la línea 6 se declara la variable entera *varEntero* y se le asigna el valor de 10. Como puede observarse en la línea 5 se declaran tres variables: *aptrEntero* y *otroPuntero* que son punteros a entero y *punteroaPuntero* que como su nombre lo indica es un puntero a puntero de entero.
Repasemos por un momento esto, habíamos dicho que un puntero puede almacenar la dirección de memoria de una variable siempre y cuando el puntero sea declarado como un apuntador al mismo tipo de datos que la/s variable/s que va a apuntar.

Entonces:

int *aptrEntero	Podrá almacenar la dirección de memoria de una variable del tipo de dato *int*.
char *aptrChar	Podrá almacenar la dirección de memoria de una variable *char*.
float *aptrFloat	Podrá almacenar la dirección de memoria de una variable *float*.
const int *aptrConstInt	Podrá almacenar la dirección de memoria de una y sólo una variable del tipo de dato *const int*. Por lo que es un puntero constante a una constante del tipo de dato int.
int **punteroaPuntero	Podrá almacenar la dirección de memoria de una variable del tipo de dato *int**. Por lo que es un puntero que almacenará la dirección de memoria de otro puntero.

Una vez claro esto, se entiende como en la línea 7 se le asigna la dirección de memoria de la variable entera *varEntero* al puntero *aptrEntero*. Luego, en la siguiente línea se puede ver como se lista por pantalla el valor de la variable *varEntero* pero mediante el puntero *aptrEntero*, aquí se utiliza el operador de indirección. Para acceder a la zona de memoria donde la variable que contiene el puntero *aptrEntero* está apuntado se antepone el caracter asterisco (*), de esa forma podemos acceder al valor de *varEntero* (osea 10).
Lo mismo se hace en la línea 10, pero en este caso se le asigna el valor 90 al espacio de memoria donde *aptrEntero* está apuntando (*varEntero* ahora vale 90).
La línea 11 puede ser confusa pero tal y como se aclaró en la tabla, a nuestro puntero *punteroaPuntero* se le asigna la dirección de memoria del puntero *aptrEntero*.
La línea 12 es más sencilla de entender, al puntero *otroPuntero* se le está asignando el valor de *aptrEntero* (que sólo puede ser una dirección de memoria ya que es un puntero) y que es equivalente a la dirección de memoria de *varEntero* (*&varEntero*).
En la línea 13 se lista el valor de *varEntero*, sólo para demostrar que efectivamente el valor ha cambiado a 90.
En la línea 15 empieza lo confuso, como habíamos dicho *punteroaPuntero* almacenaría la dirección de memoria de un puntero a entero. Por lo tanto *punteroaPuntero* sólo mostraría la dirección de memoria de un puntero a entero (en este caso, la dirección de memoria de *aptrEntero* → *&aptrEntero*) y ***punteroaPuntero** mostraría el contenido de un puntero a entero (en este caso el

contenido de *aptrEntero* → *&varEntero*). Por lo tanto, es necesario indireccionar a *punteroaPuntero* dos veces, la primera para poder acceder a la zona de memoria de donde está apuntado y una vez más para poder acceder a la zona de memoria del puntero que está apuntado, osea **(*punteroaEntero)* → **aptrEntero* → *varEntero*.
Por eso se entiende porqué **(*punteroaPuntero)* es equivalente a *varEntero*, por lo tanto es utilizado en la línea siguiente para modificarle su valor a -5.
Luego, como se había asignado en la línea 12 a *otroPuntero* el valor de *aptrEntero* que ya era la dirección de memoria de *varEntero* mediante la igualación que se realiza en la línea 7. Se puede observar como en la línea 18 el indireccionamiento de *otroPuntero* muestra como resultado -5.

Bibliografía:

- García de Jalón. J, Rodríguez. J, Sarriegui. J, Goñi. R, Brazales. A, Funes. P, Larzabal. A, Rodríguez. R, Aprenda C++ como si estuviera en primero, San Sebastián, Abril 1998.