¿Qué es un puntero?

Por: Sebastián Sánchez.

A pesar de tener conocimientos de las estructuras de datos lineales más sencillas, parece que el concepto de puntero es aún nuevo. La primera vez que nos resulta extraño es cuando vemos el * en las declaraciones de C++ y para eso debemos darle una definición en este lenguaje.

Parece ser que el poder que emana C++ se debe al manejo y la implementación de los punteros, es por eso que muchos dicen que es el lenguaje perfecto para aprender estructuras de datos.

Un puntero es una variable que almacena la direccion de memoria de otro objeto. Por ejemplo, si una variable llamada **p** contiene la dirección de otra variable llamada **q**, entonces decimos que **p** apunta a **q**. Por tanto, si **q** está en la posición de memoria 100, entonces **p** tendría el valor de 100 (la dirección de **q**).

Por lo general, utilizamos esta notación en C++:

```
tipo *nombre-variable;
```

Podemos definir varios tipos de punteros y además estos pueden apuntar a un valor nulo (NULL ó nullptr). Sin los punteros, la asignación de memoria dinámica en C++ no sería posible, esta es la razón por la que su importancia es gigantezca. Pero, ¿a qué nos referimos cuando hablamos de memoria dinámica?

La memoria del programa se divide en dos grandes entidades:

Stack:
Todas tus variables locales toman la memoria del stack.

Heap:
Es la memoria del programa no utilizada que puede ser utilizada cuando el programa es ejecutado para asignar *dinámicamente* la memoria.

Es posible asignar memoria dentro del heap con la palabra reservada **new**, es así como llenamos los punteros con información nueva.

Por ejemplo, si escribimos en C++:

```
Objeto miObjeto;
miObjeto.metodo();
```

No recibiremos ninguna excepción, el compilador reserva un espacio en el stack para la variable **miObjeto** y le asigna un valor inicial.

Pero, si escribimos:

```
Objeto *miObjeto;
miObjeto->metodo();  //Los miembros del puntero se escriben "->"
```

Recibiremos una excepción de “puntero nulo”, por lo tanto, debemos asignarle un valor en el heap de esta manera:

```
Objeto *miObjeto = new Objeto();
```

Por lo tanto, la dirección del miembro **miObjeto** es estática pero la dirección del puntero **miObjeto** podrá cambiar dinámicamente. Esto es muy importante para la creación de estructuras, los miembros de una clase Nodo para una lista doblemente enlazada serían, por ejemplo:

```
class Nodo
{
     private:
          T elemento;
          Nodo *anterior;
          Nodo *siguiente;
}
```

Sí usas un elemento en el heap y no aplicas las operaciones necesarias para “eliminarlo”, seguirá allí, ocupando espacio aunque no lo utilices. Esto no sucede en lenguajes como Java o C#, donde el “recolector de basura” (garbage collector) se encarga de liberar la memoria que no sé utiliza durante la ejecución, un proceso que no siempre es confiable pero nos ahorra mucho trabajo.

Punteros en Java

En Java la historia es distinta, no hay alguna notación u operador que nos indique la presencia de un puntero pero están presentes. Toda expresión compleja, como objetos o arreglos son guardados en el heap y se asignan dinámicamente.

Es una desventaja y una ventaja en ciertos aspectos, pero si vemos cada elemento, variable, simbolo de Java como un puntero o referencia, la creación de estructuras de datos es incluso mucho más sencilla que en C++. En conclusión, todo en Java, exceptuando los tipos de datos primitivos, es un puntero.

Punteros en C#

Los puntero en C# son una cosa muy complicada a mi parecer. C# trabaja con referencias al igual que Java, prácticamente con el mismo patrón, cada cosa se asigna de manera dinámica, es como si todo fuera un puntero, pero al mismo tiempo como que no.

C# menciona que existe una diferencia entre puntero y referencia. Con los punteros podemos manipular la dirección como tal pero con las referencias no. Es decir, **miObjeto** sigue siendo una referencia de tipo **Objeto**, con una direccion 0x004FFFA8, pero no podemos manipular esa dirección. Si **miObjeto** fuera un puntero, podriamos sumarle cuatro y la nueva dirección sería 0x004FFFAB (la dirección de la memoria se representa con un número en formato hexadecimal).

La notación de los punteros es la misma que C++ pero la porción de codigo donde se utilice un puntero debe englobarse e identificarse con la etiqueta **unsafe**, indicandole al compilador que es código inseguro.

Lo más importante de todo esto es que, para la creación de estructuras de datos en C# no es necesario utilizar punteros, lo complicarían todo; el proceso de creación e implementación es practicamente igual que en Java.

Es importante conocer todo el tipo de operaciones y como podemos manejar los punteros en C++, será una herramienta indispensable para la creación de cualquier estructura de datos en este lenguaje. El entender el concepto de puntero nos deja claro como funcionan las estructuras de datos en los lenguajes que trabajan implícitamente con referencias como C# o Java.

Una comparación de un Nodo de una lista simplemente enlazada en los tres lenguajes mencionados:

C++

```
class Nodo
{
     private:
          T elemento;
          Nodo *siguiente;

     public:
          Nodo(T elemento);
}

Nodo::Nodo(T elemento)
{
     this->elemento = elemento;
     siguiente = nullptr;
}
```

C# y Java

```
public class Nodo
{
     T elemento;
     Nodo siguiente;

     public Nodo(T elemento)
     {
          this.elemento = elemento;
          siguiente = null;
     }
}
```