Aritmética de punteros

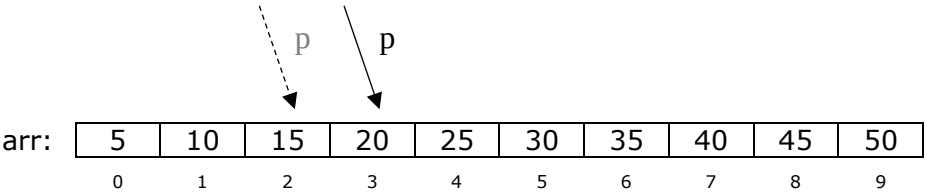
Ya sabemos que un puntero señala una dirección específica de la memoria, con lo cual podemos procesar el valor que se encuentra almacenado a partir de esa dirección. Si lo que nos interesa es procesar un conjunto de datos que se encuentran en memoria, lo más apropiado es que un puntero pueda desplazarse sobre ellos; es decir, que pueda ir recorriendo los espacios de almacenamiento contiguos.

Para realizar este desplazamiento se utiliza la aritmética de puteros. Es importante entender que la aritmética de punteros no se utiliza para sumar, multiplicar o dividir punteros. Esta sirve para desplazar los puteros a lo largo de la memoria, en un sentido o en otro: avanzando en la memoria o retrocediendo. Es decir, la aritmética de punteros se utiliza para el barrido de la memoria.

Los operadores que comúnmente se utilizan para la aritmética de punteros son los de incremento y de decremento:

+, **–**, **++**, **––**, **+=** y **–=**

Por ejemplo, si *p* apunta a *arr[2]*, para hacer que apunte a la siguiente casilla habrá que sumarle 1 y para que apunte a la casilla anterior habrá que restarle uno:



En la siguiente tabla se explica el efecto de estos operadores en un puntero:

Las operaciones	Producen este resultado
p**++** ó **++**p p**––** ó **––**p p**+=** Expresión P**–=** Expresión	Cambian la dirección de memoria que está almacenada en *p*, pues son operadores de asignación. Así que, luego de aplicarlas, *p* apuntará a otra dirección de memoria.
p **+** Expresión p **–** Expresión	Calculan/producen una nueva dirección de memoria a partir de la dirección a la que *p* apunta. Esto no altera el valor de *p* (la dirección a la que *p* apunta realmente), a menos que esa nueva dirección se le asignara a *p* misma.

(Expresión debe devolver un entero)

Para avanzar en la memoria utilizaremos p**++** , **++**p , p**+=** Expresión , p **+** Expresión.

Para retroceder en la memoria utilizaremos p**––** , **––**p , p**–=** Expresión , p **+** Expresión.

Pero, ¿cuánto avanza o cuanto retrocede el puntero? La respuesta es: el puntero avanza o retrocede dando la cantidad de saltos que le indica el entero que se le está sumando o restando. Por ejemplo, si *p* es un puntero de tipo entero y se ejecuta la operación *p + 3*, dará tres saltos hacia adelante; si se ejecuta la operación *p - 5*, dará cinco saltos hacia atrás.

El tamaño del paso, tmaño del salto, lo define el tipo del puntero. Por ejemplo, si p es un puntero de tipo entero, p + 1 indica un saltó de 4 bytes, para apuntar al

siguiente entero; si se ejecuta p + 3, se darán 3 saltos de 4 bytes cada uno, para apuntar al entero que está tres posiciones más adelante.

Es importante aclarar que, al realizar un desplazamiento del puntero, por ejemplo, sumándole 1, el puntero no avanza al siguiente byte de memoria, sino que avanza al siguiente elemento del tipo definido en la declaración del puntero. Si el puntero apunta a datos de tipo entero, el puntero avanzará 4 bytes. Si se le resta 1, entonces el puntero retrocederá cuatro bytes para apuntar al entero que está antes.

Por ejemplo:

```
double *p, *q, i;

i = 2;
q = p + i;
```

El puntero q apuntará 16 bytes más lejos en la memoria RAM porque dará dos saltos de 8 bytes.

Si anteponemos el operador de indirección, *, a una expresión de este tipo, *(p + i)*, nos referimos al contenido de la celda que encontramos a partir de la celda de memoria que está en la dirección *p + i*.

Por ejemplo:

Escriba un programa que recorra e imprima los elementos de un arreglo de enteros con un puntero y la dirección de memoria de cada uno.

```
#include "iostream"

using namespace std;

int main(void)
{
    int arr[ ] = {13, -4, 6, 21, 8}, *p, i;

    cout << "RECORRER UN ARREGLO CON UN PUNTERO" << endl << endl;

    p = arr;

    cout << "Los elementos del arreglo son:" << endl;
    for(i = 0; i < 5; i++)
        cout << "Elemento " << i << "    Dirección: " << (p + i) << endl;

    cout << endl;
    return 0;
}
```

La corrida de este programa es:

```
RECORRER UN ARREGLO CON UN PUNTERO

Los elementos del arreglo son:
Elemento 0    Dirección: 0xffffcb80
Elemento 1    Dirección: 0xffffcb84
Elemento 2    Dirección: 0xffffcb88
Elemento 3    Dirección: 0xffffcb8c
```

Elemento 4 Dirección: 0xffffcb90