Relación entre Apuntadores y Arreglos

Si se declara el siguiente arreglo

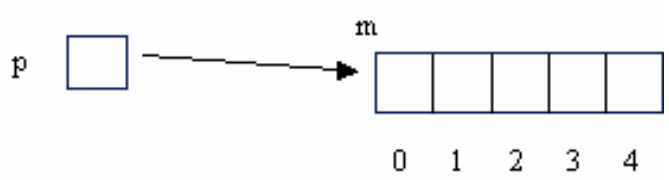
```
int m[5];
```

internamente C++ declara m como un apuntador a entero y coloca ahí la dirección del primer elemento del arreglo.

Supongamos que tenemos lo siguiente:

```
int *p;
p = m;
```

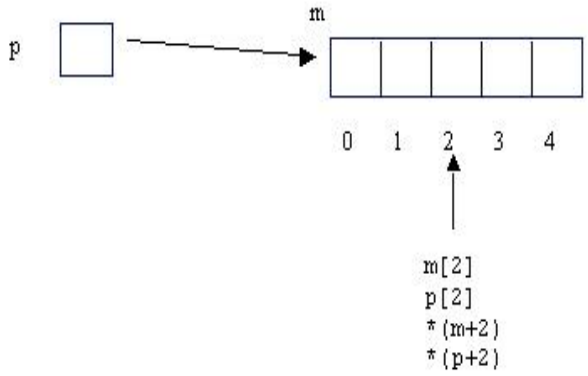
gráficamente lo podemos representar así:



y para accesar los elementos del arreglo también se puede usar el apuntador de cualquiera de las siguientes maneras:

m[2]	se refiere al elemento de la casilla 2 del arreglo
p[2]	se refiere al elemento de la casilla 2 del arreglo
*(p+2)	se refiere al elemento que está 2 posiciones enteras más allá del inicio del arreglo, lo cual es equivalente a decir que es el elemento de la casilla 2 del arreglo
*(m+2)	se refiere al elemento que está 2 posiciones enteras más allá del inicio del arreglo, lo cual es equivalente a decir que es el elemento de la casilla 2 del arreglo

es decir:



En la expresión *(p+2) al sumar 2 al apuntador p, no se está sumando el número 2 sino que se está avanzando 2 posiciones en la memoria; es decir, p es un apuntador a entero, por lo que al sumarle 2 se avanza el espacio de memoria necesario para avanzar 2 posiciones enteras.

El paréntesis es necesario porque el operador * tiene una prioridad más alta que el operador +.

Aritmética de apuntadores

Cada vez que un apuntador se incrementa, apunta a la localidad de memoria del siguiente elemento de su tipo base (o sea, del tipo del que fue definido).

Cada vez que se decrementa, apunta a la localidad de memoria del elemento anterior. Esta forma de trabajar asegura que siempre un apuntador apunta a un elemento apropiado de su tipo base.

Por ejemplo, supón que tienes la siguiente declaración y que un entero ocupa 2 bytes:

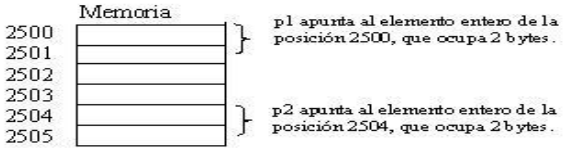
```
int *p1, *p2;
```

y supón que p1 contiene la dirección de memoria 2500, al ejecutar la operación:

```
p2 = p1 + 2;
```

la memoria queda de la siguiente manera

Memoria



Verificación de los limites del arreglo

En base a lo que se ha explicado en esta sección, se puede entender porque C++ no hace verificación de los límites de un arreglo; es decir, si se tiene el siguiente caso:

```
int m[5];
m[15] = 10;
```

se está asignando el valor 10 a una posición de memoria que no pertenece al arreglo (porque el arreglo m tiene casillas de la 0 a la 4); entonces, es responsabilidad del programador verificar que esto no ocurra.

Ejercicio

Supón que tienes declarado el siguiente arreglo:

```
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
```

dí que muestra en la pantalla el siguiente segmento de código:

```
int *p;

p = arr;
*p = 20;
*(p + 2) = 40;
*(arr + 5) = 60;

for (int i =0; i<10; i++)
   cout<<arr[i]<<"\t";
```

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo