Operador sizeof con estructuras

Podemos usar el operador **sizeof** para calcular el espacio de memoria necesario para almacenar una estructura.

Sería lógico suponer que sumando el tamaño de cada elemento de una estructura, se podría calcular el tamaño de la estructura completa, pero no siempre es así. Por ejemplo:

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

struct A {
   int x;
   char a;
   int y;
   char b;
};

struct B {
   int x;
   int y;
   char a;
   char b;
};
```

El resultado, usando Dev-C++, es el siguiente:

```
Tamaño de int: 4
Tamaño de char: 1
Tamaño de estructura A: 16
Tamaño de estructura B: 12
```

Si hacemos las cuentas, en ambos casos el tamaño de la estructura debería ser el mismo, es decir, 4+4+1+1=10 bytes. Sin embargo en el caso de la estructura *A* el tamaño es 16 y en el de la estructura *B* es 12, ¿por qué?

La explicación es algo denominado alineación de bytes (*byte-aling*). Para mejorar el rendimiento del procesador no se accede a todas las posiciones de memoria. En el caso de microprocesadores de 32 bits (4 bytes), es mejor si sólo se accede a posiciones de memoria múltiplos de cuatro, de modo que el compilador intenta alinear los objetos con esas posiciones.

En el caso de objetos *int* es fácil, ya que ocupan cuatro bytes, pero con los objetos *char* no, ya que sólo ocupan uno.

Cuando se accede a datos de menos de cuatro bytes la alineación no es tan importante. El rendimiento se ve afectado sobre todo cuando hay que leer datos de cuatro bytes que no estén alineados.

En el caso de la estructura *A* hemos intercalado campos **int** con **char**, de modo que el campo **int** *y*, se alinea a la siguiente posición múltiplo de cuatro, dejando tres posiciones libres después del campo *a*. Lo mismo pasa con el campo *b*.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x				a	vacío			y				b	vacío		

En el caso de la estructura *B* hemos agrupado los campos de tipo **char** al final de la estructura, de modo que se aprovecha mejor el espacio, y sólo se desperdician los dos bytes sobrantes después de *b*.

0	1	2	3	4	5	6	7	8	9	10	11
x				y				a	b	vacío	