大端序和小端序

何为大端序，小端序？
简单点说，就是字节的存储顺序，如果数据都是单字节的，那怎么存储无所谓了，但是对于多字节数据，比如 int，double 等，就要考虑存储的顺序了。注意字节序是硬件层面的东西，对于软件来说通常是透明的。再说白一点，字节序通常只和你使用的处理器架构有关，而和编程语言无关，比如常见的 Intel x86 系列就是小端序。

Big-endian（大端序）
数据的高位字节存放在地址的低端 低位字节存放在地址高端

Little-endian（小端序）
数据的高位字节存放在地址的高端 低位字节存放在地址低端

字节的高位与低位
举个例子，int a = 0x12345678；那么左边 12 就是高位字节，右边的 78 就是低位字节，从左到右，由高到低，（注意，高低乃相对而言，比如 56 相对于 78 是高字节，相对于 34 是低字节）

地址的高端与低端
0x00000001
0x00000002
0x00000003
0x00000004
从上倒下，由低到高，地址值小的为低端，地址值大的为高端。

不同字节序如何存储数据？
看看两种方式如何存储数据，假设从地址 0x00000001 处开始存储十六进制数 0x12345678，那么

Bit-endian 如此存放(按原来顺序存储)
0x00000001 -- 12
0x00000002 -- 34
0x00000003 -- 56
0x00000004 -- 78

Little-endian 如此存放(颠倒顺序储存)
0x00000001 -- 78
0x00000002 -- 56
0x00000003 -- 34
0x00000004 -- 12

一个很好的记忆方法是，大端序是按照数字的书写顺序进行存储的，而小端序是颠倒书写顺序进行存储的。

编程判断大端序和小端序
方法一

```
bool IsBigEndian()
{
int a =1 ;
if(((char*)&a)[3] ==1)
return true ;
else
return false ;
}
```

打开 VS 的内存窗口，看一下 a 的存储方式，一目了然

由于 a 是 int，所以占四个字节，其值是 00000001，存储方式如下。所以 a[3]是 0，不是大端序。一个更标准的写法是将 a[3]替换为 a[sizeof(int) - 1]。

0x0012FE88　　01

0x0012FE89　　00

0x0012FE8A　　00

0x0012FE8B　　00

方法二，使用 union，原理见后面的面试题。

```
bool IsBigEndian()
{
    union
    {
        unsigned short a ;
char b ;
    } c;

    c.a =0x0102 ;

if(c.b ==1)
return true ;
else
```

```
return false ;
}
```
一道面试题
来道题巩固一下，下面代码输出什么？

```
union u
{
int i ;
char x[2] ;
} a ;

int main(void)
{
	a.x[0] ='1' ;
	a.x[1] ='2' ;

	cout << a.i << endl ;

	getchar() ;
return 0 ;
}
```
这个题，要看你使用的是什么系列的 CPU，姑且假设是 Intel 系列的。Union 是一个特殊的结构，其中所有成员共享结一个内存地址，任意时间只能存储一个成员，上面的 Union 大小为 4 个字节，所以上面的代码存储完字符 1 和 2 之后，Union 的存储貌似应该是 0x31320000，31 和 32 分别是字符'1'和'2'的十六进制 ASCII 码（注意是字符 1 和 2，而不是整数），但是 Intel 系列的 CPU 都是按照小端序存储的，所以，正确的顺序是 0x00003231，对应的十进制数是 12849，你答对了么？