在 C/C++中，64 为整型一直是一种没有确定规范的数据类型。现今主流的编译器中，对 64 为整型的支持也是标准不一，形态各异。一般来说，64 位整型的定义方式有 long long 和__int64 两种(VC 还支持_int64)，而输出到标准输出方式有 printf(”%lld”,a)，printf(”%I64d”,a)，和 cout << a 三种方式。

本文讨论的是五种常用的 C/C++编译器对 64 位整型的支持，这五种编译器分别是 gcc(mingw32)，g++(mingw32)，gcc(linux i386)，g++(linux i386)，Microsoft Visual C++ 6.0。可惜的是，**没有一种定义和输出方式组合，同时兼容这五种编译器**。为彻底弄清不同编译器对 64 位整型，我写了程序对它们进行了评测，结果如下表。

变量定义	输出方式	gcc(mingw32)	g++(mingw32)	gcc(linux i386)	g++(linux i386)	MicrosoftVisual C++ 6.0
long long	“%lld”	错误	错误	正确	正确	无法编译
long long	“%I64d”	正确	正确	错误	错误	无法编译
__int64	“lld”	错误	错误	无法编译	无法编译	错误
__int64	“%I64d”	正确	正确	无法编译	无法编译	正确
long long	cout	非 C++	正确	非 C++	正确	无法编译
__int64	cout	非 C++	正确	非 C++	无法编译	无法编译
long long	printint64()	正确	正确	正确	正确	无法编译

上表中，正确指编译通过，运行完全正确；错误指编译虽然通过，但运行结果有误；无法编译指编译器根本不能编译完成。观察上表，我们可以发现以下几点：

1. long long 定义方式可以用于 gcc/g++，不受平台限制，但不能用于 VC6.0。
2. __int64 是 Win32 平台编译器 64 位长整型的定义方式，不能用于 Linux。
3. “%lld”用于 Linux i386 平台编译器，”%I64d”用于 Win32 平台编译器。
4. cout 只能用于 C++编译，在 VC6.0 中，cout 不支持 64 位长整型。

表中最后一行输出方式中的 printint64()是我自己写的一个函数，可以看出，它的兼容性要好于其他所有的输出方式，它是一段这样的代码：

[Copy to clipboard]View Code CPP

```
void printint64(long long a)
{
 if (a<=100000000)
  printf("%d\n",a);
 else
```

```
  {
   printf("%d",a/100000000);
   printf("%08d\n",a%100000000);
  }
}
```

这种写法的本质是把较大的 64 位整型拆分为两个 32 位整型，然后依次输出，低位的部分要补 0。看似很笨的写法，效果如何？我把它和 cout 输出方式做了比较，因为它和 cout 都是 C++支持跨平台的。首先 printint64()和 cout(不清空缓冲区)的运行结果是完全相同的，不会出现错误。我的试验是分别用两者输出 1000000 个随机数，实际结果是，printint64()在 1.5s 内跑完了程序，而 cout 需要 2s。cout 要稍慢一些，所以在输出大量数据时，要尽量避免使用。