# 关于C++编写的程序静态存储区的数据存储的实验

陈力 2018.11.21

目录

[关于C++编写的程序静态存储区的数据存储的实验 1](#_Toc530694440)

[第零节 引例 1](#_Toc530694441)

[第一节 数组的寻值 2](#_Toc530694442)

[第二节 程序在内存中的空间分配 4](#_Toc530694443)

[第三节 Auto变量和Static变量 4](#_Toc530694444)

[第四节 更加安全的实现杨辉三角 7](#_Toc530694445)

[附录 11](#_Toc530694446)

[编译测试环境 11](#_Toc530694447)

[参考 11](#_Toc530694448)

## 第零节 引例

使用静态数组实现的杨辉三角的示例

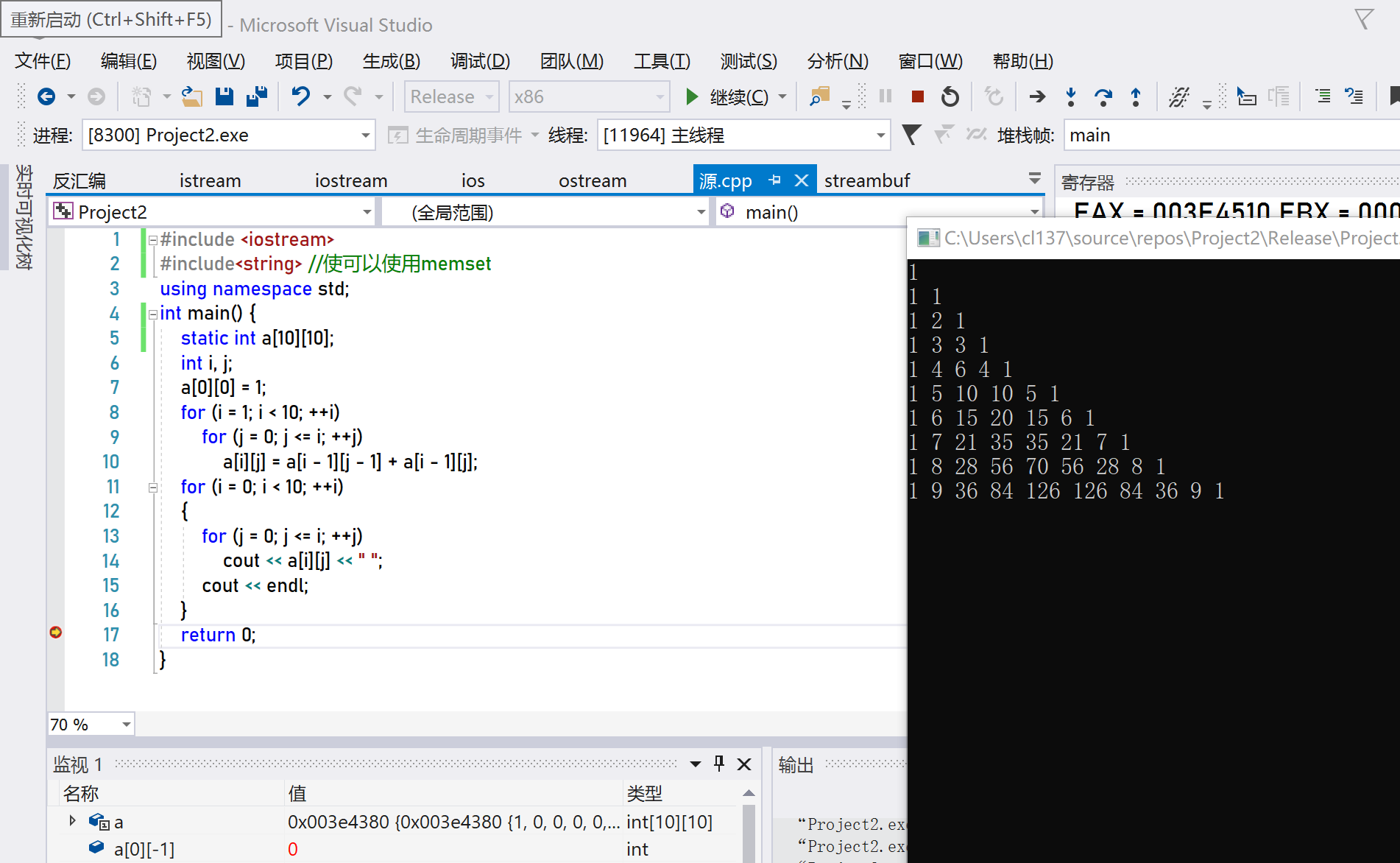


图 0‑1

而去掉static，不使用静态数组的效果：

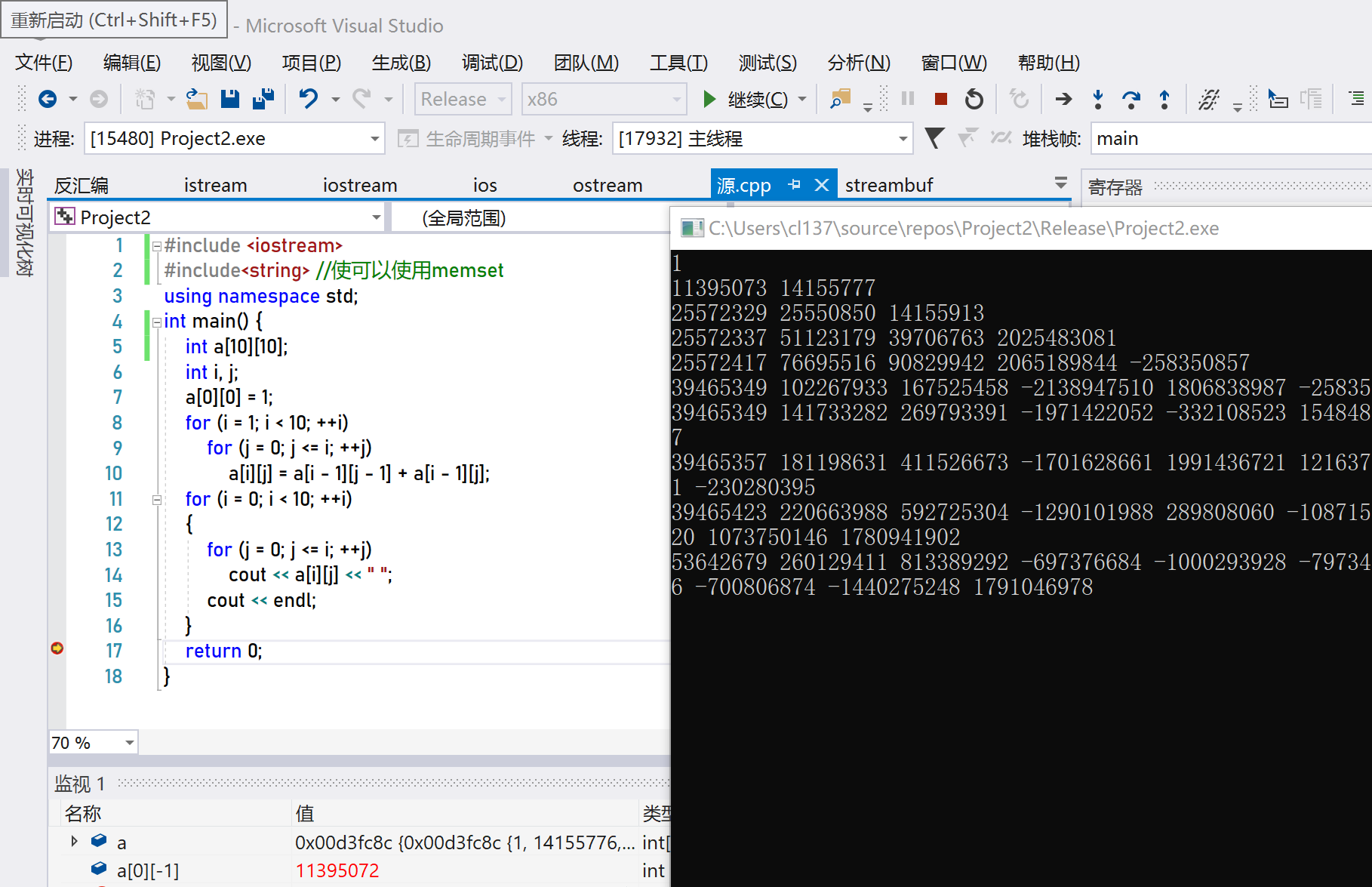


图 0‑2

显然使用静态存储的答案似乎是正确的，而动态存储的结果显然是有问题的。我们接下来逐步讨论产生这样问题的原因。

## 第一节 数组的寻值

引例中计算时我们可以发现其数组使用了a[0][-1]等数组下标时负数的位置，发生了数组越界。首先要知道越界后的数组指向的是什么地方的数据才能解释引例中的问题，我们需要找到数组是如何寻值。例如，我们声明数组，利用以下程序打印相关值前后的地址。（我们这里打印从a[-5][-5]到a[4][4]的地址，输出顺序参考源代码（图1-1））

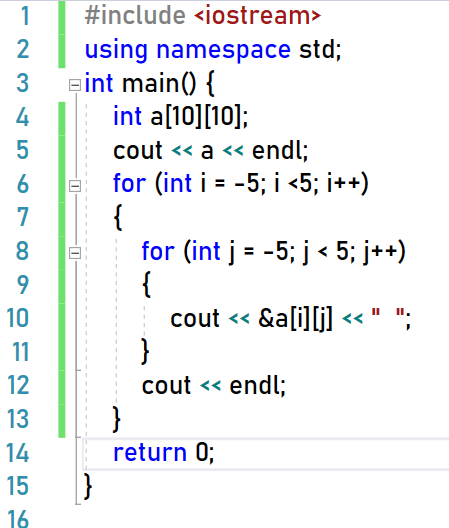


图 1‑1

我们得到一下输出结果（图1-2）：

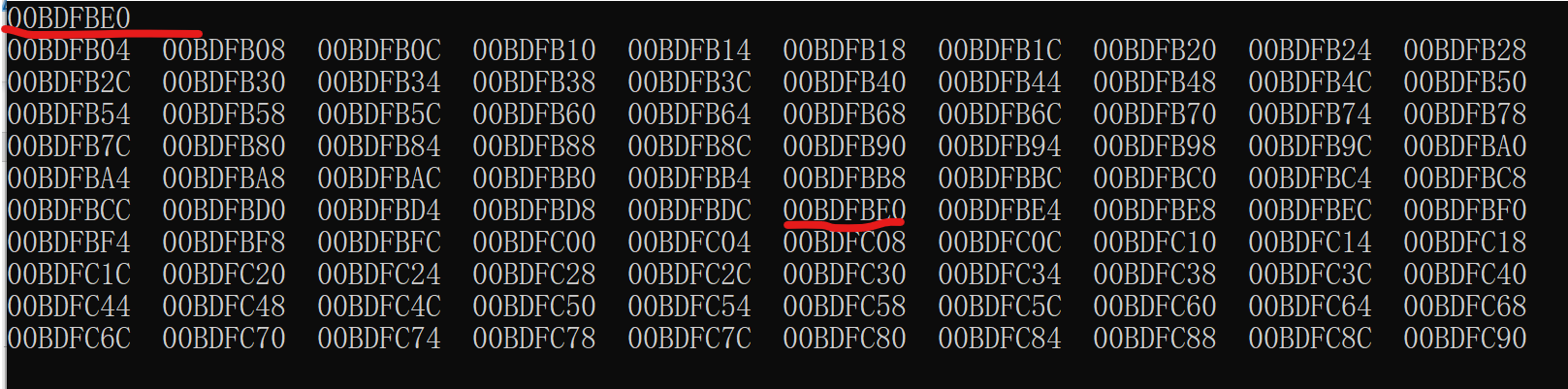


图 1‑2

如图（图1-2）我们可以知道每一个地址相隔4 bytes（此处为数组元素int型的大小）。所以对于编译系统只要知道起始的地址在这个基础上进行加减乘除就可以得到对应元素的地址。（）

对于二维数组a[I][J]元素a[i][j],编译系统是通过“S（首地址）+(i\*J+j)\* T（类型大小）”来计算得到对应元素地址的。

对于C++来说，没有对数组是否越界进行检查，根据上面的公式推论得到显然a[0][-1]就是a[0][0]地址前面的那4 bytes内存空间所表示的数，如果a[0][0]的首地址是00BDBF0那么a[0][-1]的首地址就是00BDFBDC。

## 第二节 程序在内存中的空间分配

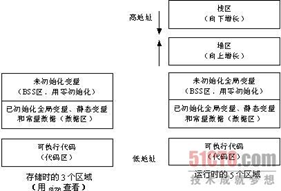


图 2‑1

注意：未初始化的static变量（.bss区）和已初始化的static变量不是连续储存在内存中。而全局变量和static变量存放在同一个内存区域。

## 第三节 Auto变量和Static变量

在引例中我们可以看到使用auto变量和static变量越界造成的结果是不同的。

引例中，我们递推出现的问题的主要原因是出现在引用的a[0][-1]的值。由第一节的内容我们可以得知a[0][-1]就是a[0][0]地址前面的那4 bytes内存空间所表示的数，那么使用auto变量和static变量是如何造成不同结果的？

首先，使用static申明的变量储存在静态存储区（未初始化的变量在*.bss*区，不占exe文件空间；已经初始化的变量，值直接定义在exe文件中，直接加载到系统分配的内存中），加载程序时进行初始化并分配内存空间，系统会自动将相关内存区域置为0。

使用auto申明的变量存于栈区，由**运行时**分配，系统不会对相关内存区域进行自动置零，使用我们程序拿到的数组所用的空间里面有很多不可预料的数据，要根据需要置0。

现在可以很好说明为什么auto申明的数组计算的杨辉三角是有问题的，因为a[0][-1]的位置在内存中很有可能本来就存在非0的数据，而且显然这个数据的类型和值是无法预料的。系统只是简单将a[0][0]前面的4 bytes当作int型的数进行操作，自然我们无法保证越界位置或者a数组里面的数是0，这样就无法保证得到正确答案。

那么至于在static中申明的，存在静态区的数据由于在系统初始化的时候置零，因为编译系统的分配策略我们确保a[0][0]到a[9][9]是0。但是对于a[0][-1]是否一定为0依然有疑问，我们将在接下来结合例子进行进一步的讨论。

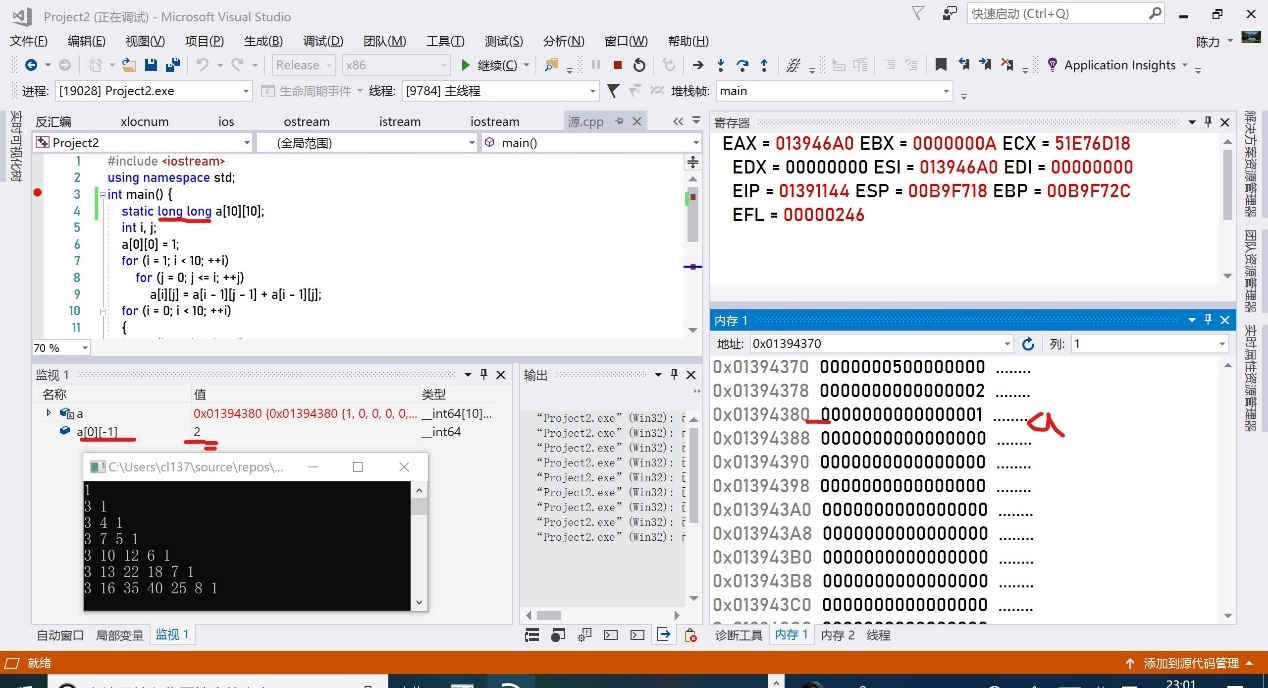


图 3‑1

由上图（图3-1）我们可以推测在定义a[10][10]这个数组之前，系统定义了很多“全局（静态）变量”（此处仅疑似静态或全局变量，也可能是其他东西，下同），而且显然这些“全局（静态）变量”不可能在**运行时**全是0。如果系统在分配内存时将一个在**运行时**不是0的“全局变量”刚好分配到我们a数组的前面，那么a[0][-1]就很有可能不是一个空置的0，而是与那个“全局（静态）变量”有关，如上图3-1中静态数组的a[0][-1]的值就是2，而不是0，导致即使使用了static数组计算杨辉三角还是得到错误的结果。

通过进一步查询，上面例子中输出的“2”是属于的，这就说明a数组之前不是绝对是空置的，存在许多其他编译系统或系统定义的数据，上例就是一个叫“\_\_favor”的“全局（静态）变量”影响了我们的结果，使得a[0][-1]=2。为了验证这个“全局（静态）变量”，我将所编译的exe文件使用VS的dumpbin工具进行dump（图3-2），在“*.reloc”*节中发现也发现了“\_\_favor”。同时它的虚拟地址正好在a的虚拟地址（00404380）的前8 bytes（此处a定义为*long long* 型，一个变量8 bytes），在所以说在静态区进行越界也不是万无一失的！

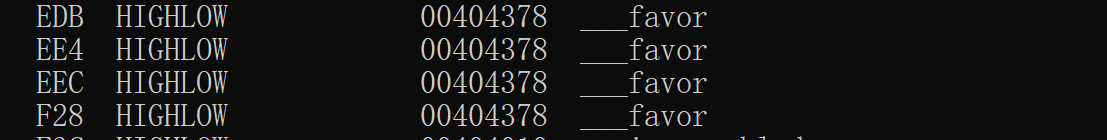


图 3‑2

## 第四节 更加安全的实现杨辉三角

为了使程序可控有条理，系统显然不会在连续定义的变量（同是static（全局）或同栈）之间插入其他变量（可能空间不一定连续），据此我们可以让数组在可控的范围内越界。下面将讨论如何可控的越界。为了方便起见，先不使用杨辉三角的例子。

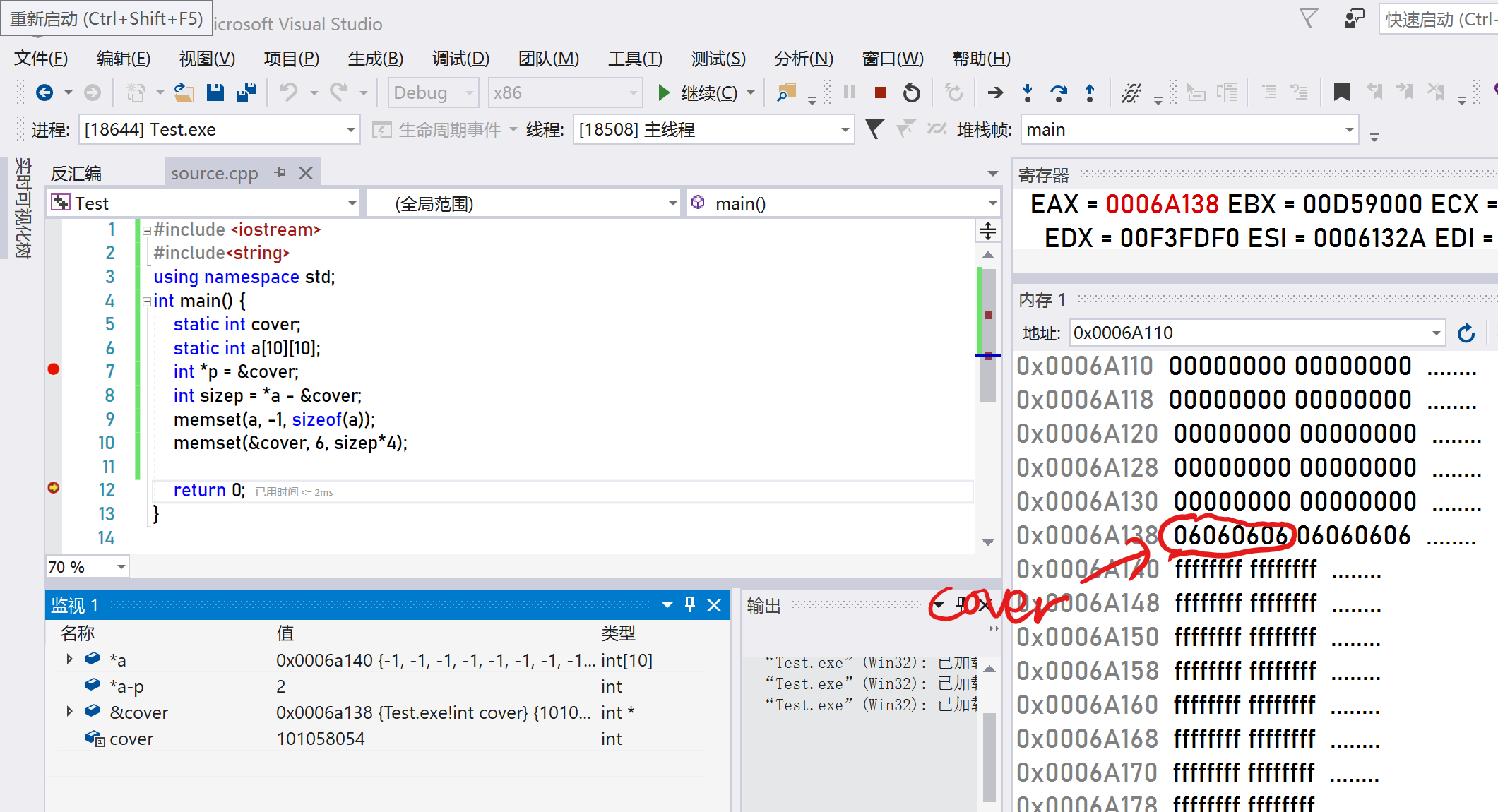


图 4‑1

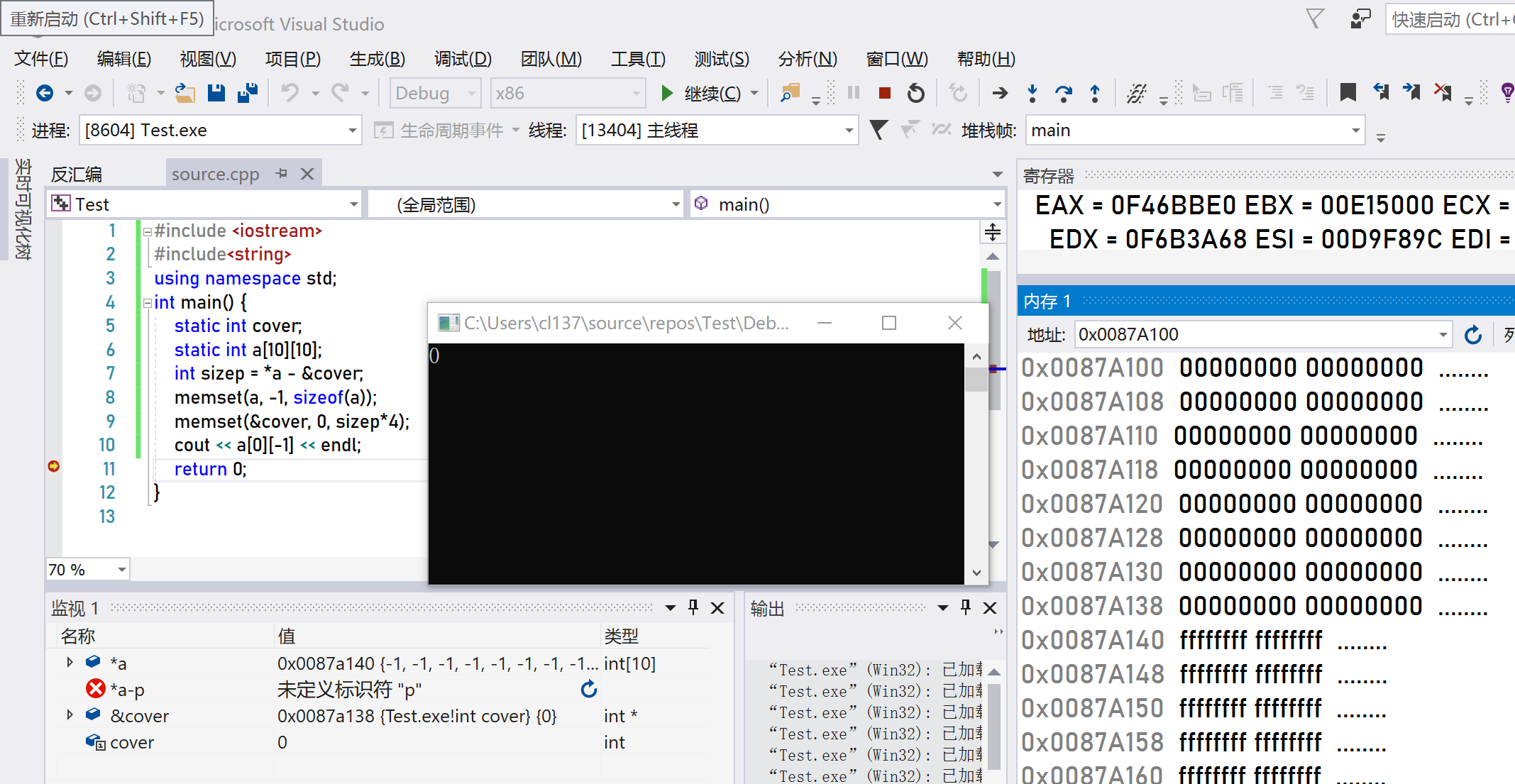
在此例（图 4‑1）中我们定义了两个静态变量，cover和a。由于内存对齐的要求，cover和a并没有在内存中连在一起，于是我们使用memset将cover和a之间的字节全部初始化为6。并且为了方便观察将a内存字节初始化为ff。在vs调试的内存图，和监视图中的地址中我们可以看见两者在内存中是比较紧密。

图 4‑2

如果使用memset将cover和a之间的字节全部初始化为0，就可以比较安全地使a[0][-1]初始化为0。（按照变量地址分配的规则不会在相邻定义的变量之间的空间分配给其他变量）注意：没有在a之前申明cover直接对a[0][-1]赋予0是很危险的行为，因为你无法确认a[0][-1]所占的数据空间是否有用，是否为其他的关键数据！

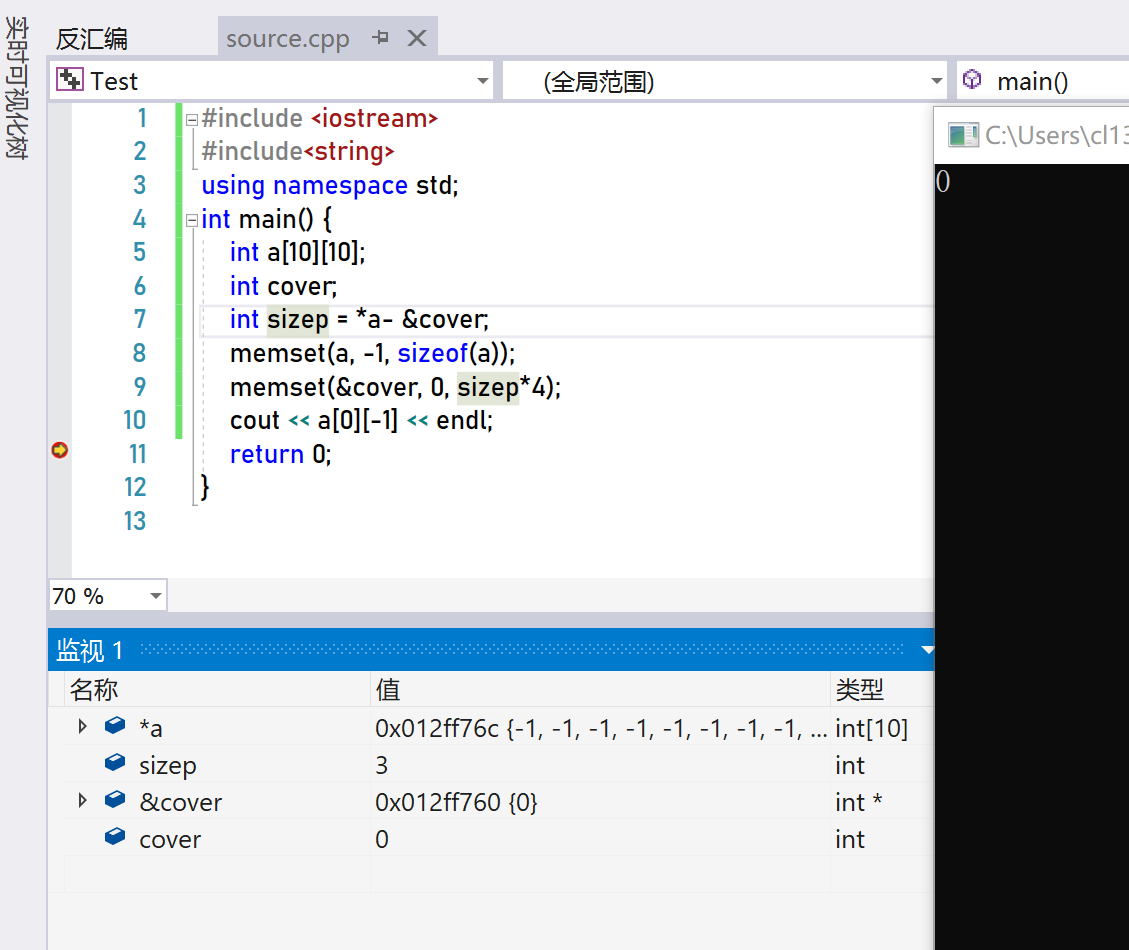


图 4‑3

同样的，对于auto定义的变量这样的操作也是可行的。注意：在main函数的栈内进行内存分配是从高地址到低地址，与全局（静态）区相反。

现在我们回到一开始的杨辉三角，我们提供连续定义变量，并将之间赋值为0使初始化将要越界访问的内存位置。利用auto变量也可以通过“越界法”实现要求（图4-4）。

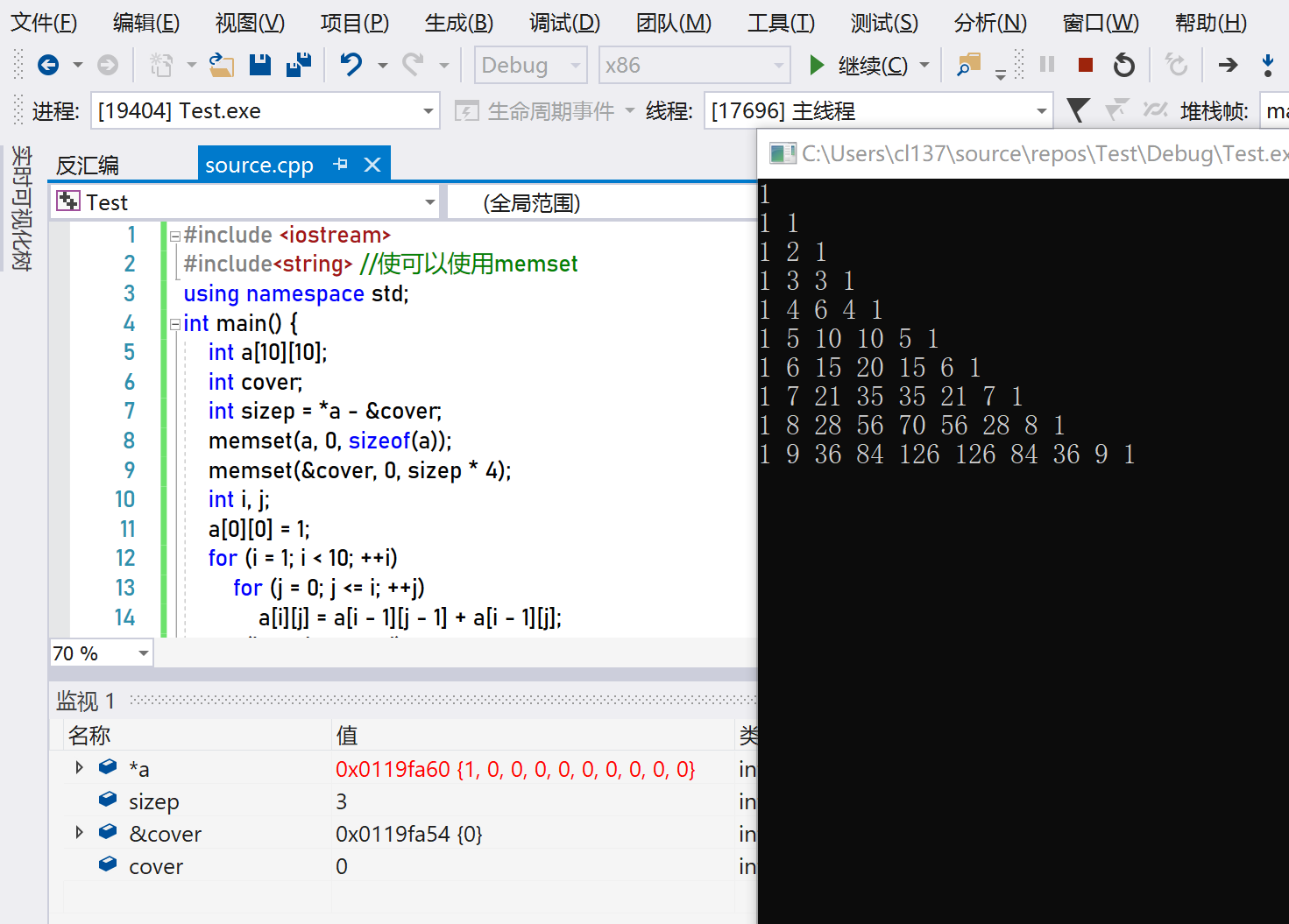


图 4‑4

或使用static数组变量也可以（图4-5）。

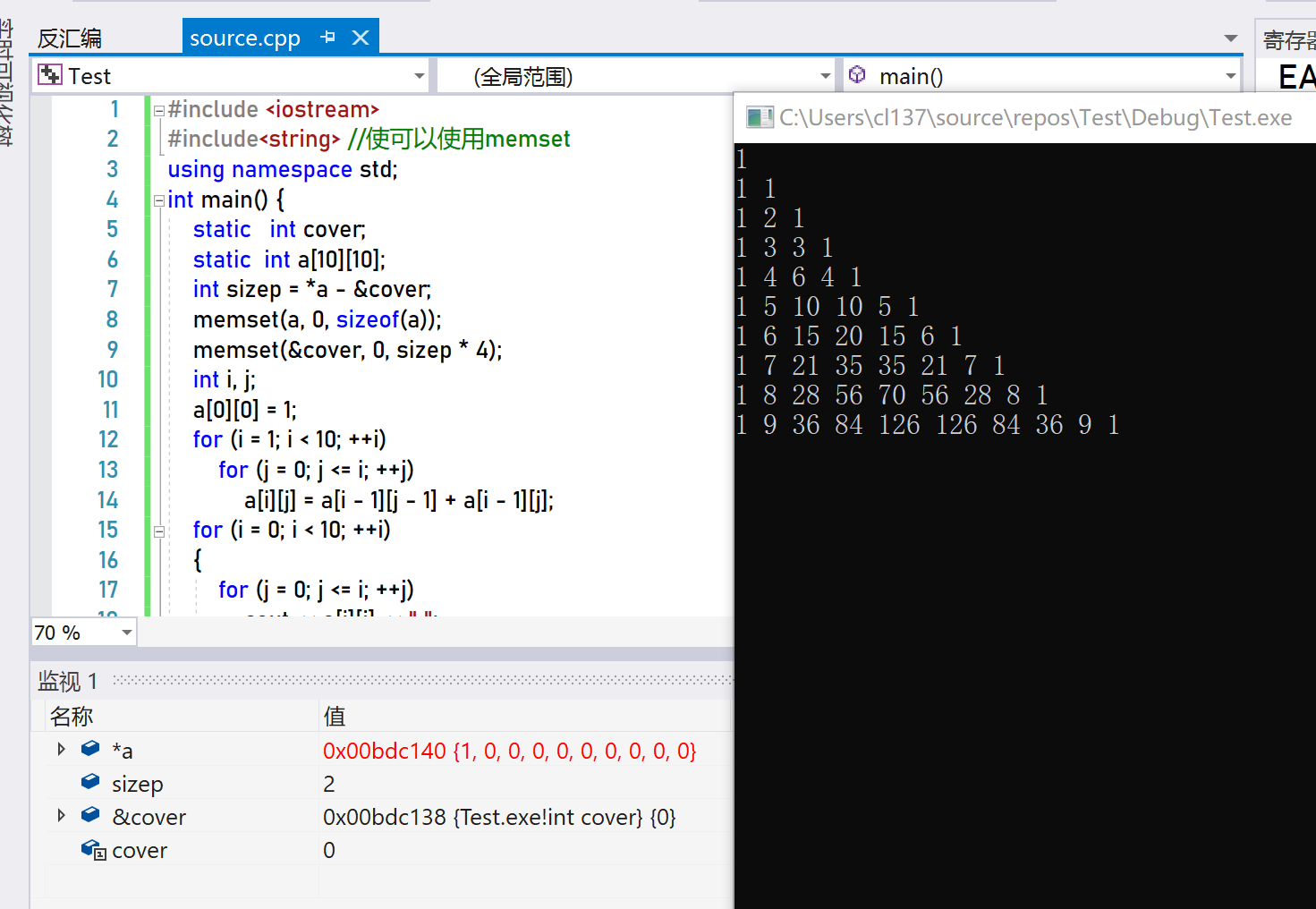


图 4‑5

这样我们就相对安全地实现了对a[0][-1]的初始化，并正确地实现了杨辉三角。

## 附录

### 编译测试环境

Windows 10 Build 18282.rs\_prerelease.181109-1613 64bit

Visual Studio 15.9.1

cl．exe编译参数：

/permissive- /GS /GL /analyze- /W3 /Gy /Zc:wchar\_t /Zi /Gm- /O2 /sdl /Fd"Release\vc141.pdb" /Zc:inline /fp:precise /D "\_MBCS" /errorReport:prompt /WX- /Zc:forScope /Gd /Oy- /Oi /MD /FC /Fa"Release\" /EHsc /nologo /Fo"Release\" /Fp"Release\Project2.pch" /diagnostics:classic

32位编译

### 参考

（日）青木峰郎《自制编译器》中国工信出版社

（美）基普•欧文《汇编语言 基于X86处理器》机械工业出版社

[流沙的刺客](https://me.csdn.net/shimachao)《C语言全局未初始化数据段分析》 [CSDN](https://blog.csdn.net/candcplusplus/article/details/12576185)

《DUMPBIN 命令行》 [MSDN](https://msdn.microsoft.com/zh-cn/library/20y05bd5.aspx)