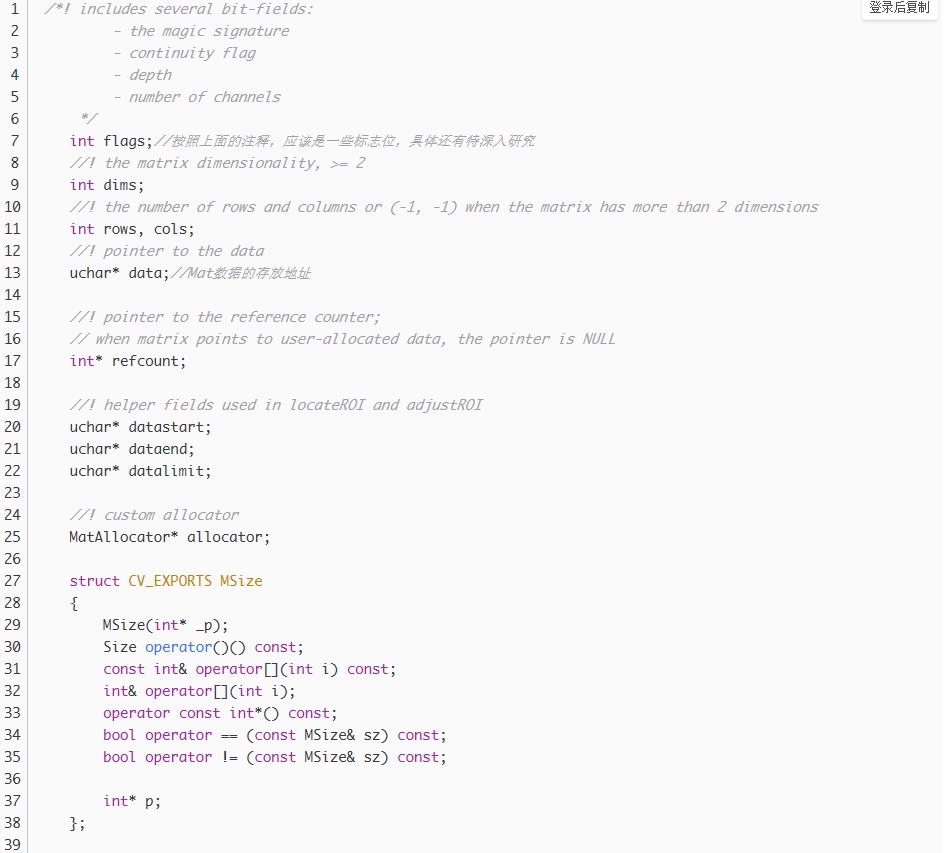
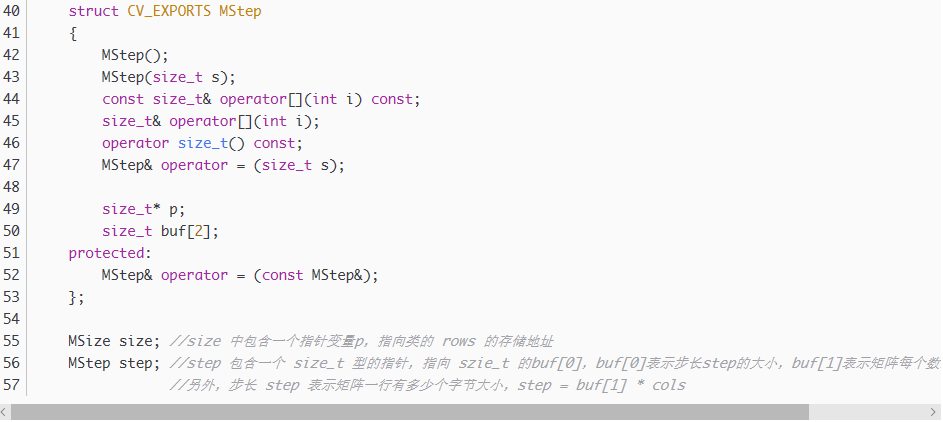
**OpenCV2 - Mat 数据类型初探**

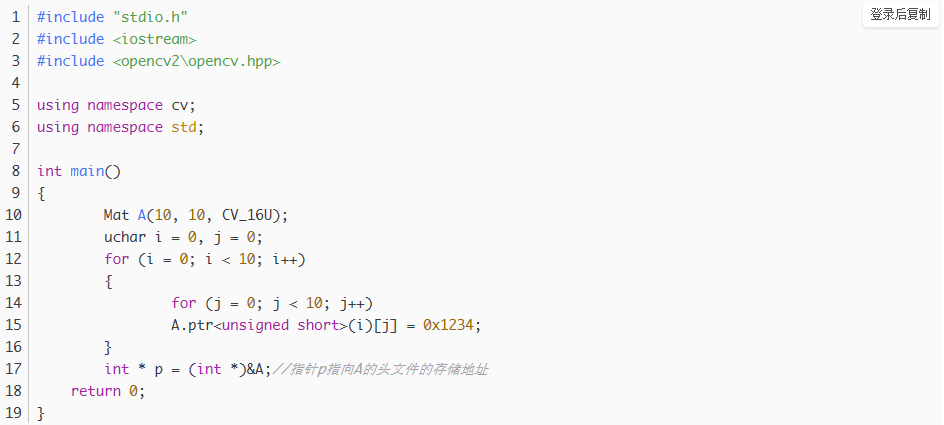
首先来看看Mat类的定义中包含的数据变量。这里有必要提一下，定义一个Mat型变量后，会有两个数据部分，一个是**数据头**部分，包含了该变量的一些必要的信息；另一个是真正的矩阵数据。**通常这两个数据块的地址并不是在一起，而是在两个内存块中，下面将通过截图详细说明。**

 Mat类中数据变量的定义：





测试代码1：



Mat A的结构，与定义是一致的。（这不废话嘛，不一致才怪了~）

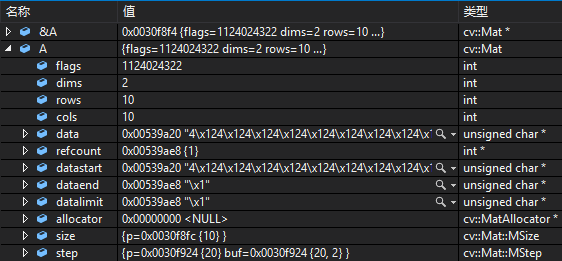


图1. A的变量结构

测试代码中还定义了一个指针变量p，用来指向矩阵A的头文件的起始地址，这里p = 0x0030F8F4，查看该处的内存：

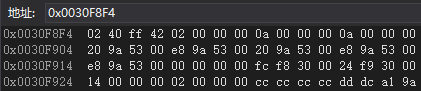


图2.矩阵A头文件在内存中的存储

通过对比观察，前四个字节分别是 02 40 ff 42，即flags = 0x42ff4002，化为十进制就是1124024322，也与图1相符。依次对比，发现确实是一一对应的。在这里，Mat A的头文件大小是56个字节，之所以强调在这里，是因为这里定义的矩阵数据类型是16U，16位无符号整型，待会儿会看看当定义其他数据类型的时候头文件的大小是不是会变化。

头文件看完了，下面我们来看看矩阵真正的数据存储情况。通过查看图1，可以看到数据的起始地址datastart = 0x00539a20，结束地址dataend = 0x00539ae8，dataend - datastart = 0xc8，十进制为200，测试代码中我们定义的10x10的16位无符号整数型的矩阵，数据大小正好为200。如下图所示：

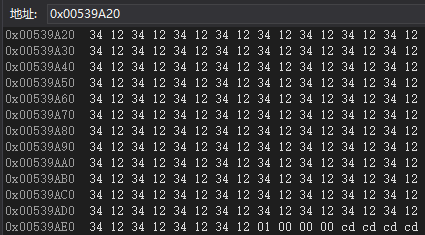


图3.矩阵数据的内存存储

通过观察发现，在矩阵数据的最后，还有四个字节为 01 00 00 00，这四个字节不是矩阵的数据，其地址已经超过了dataend。这时，观察图1矩阵头文件中的refcount，发现其大小为1，地址也恰好就是dataend的大小。说明了在矩阵数据的最后，储存了refcount的大小，是一个int型变量，占用四个四节。

**refcount**是一个很有用的东西，它记录了这个矩阵的数据被其他变量引用了多少次。在c++中矩阵的一些赋值操作往往只给新的变量赋予一个新的头文件，而数据部分只把数据指针指过去，而不重新分为内存，需要程序员来管理内存。

这样的话就会有隐患，因为如果原始的变量的数据内存释放了，新的变量却还在，指向了已经被释放的内存区域，如果对新变量进行操作，很可能出现意想不到的问题。不过不用担心，OpenCV的开发者早就想到了这个问题，并为我们想到了一个解决方法，那就是靠这个refcount来记录该数据被多少变量共用，直到最后一个变量被释放时，才释放掉这个存储数据内存块。

这也是为什么类定义中refcount为指针，让实际记录次数的变量跟在矩阵数据最后面。我们就可以放心的用啦，下面将会给出例子继续详细说明这个问题~

测试代码2：





运行结果：

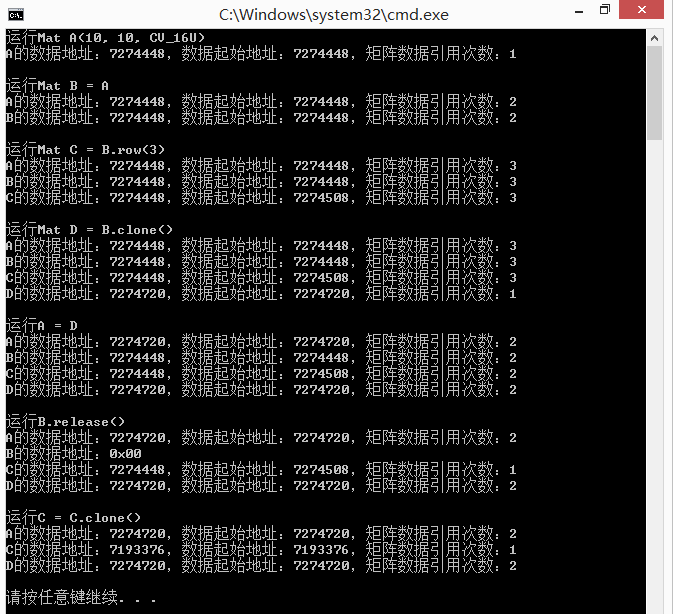


图3.测试代码2的运行结果

结果分析：前面对照着内存分析过，测试代码2就没照着内存分析了，把结果输出了，有了前面的了解这里通过输出分析一下。有兴趣的话可以自己找着前面的步骤分析，能够有一个更好的认识。

首先 Mat A(10, 10, CV\_16U) 运行后，定义了一个Mat 型的矩阵A，同时调用了构造函数生成了矩阵（包含矩阵数据区）。接着对矩阵A的数据进行了一个初始化。运行Mat B = A后，定义一个矩阵B，但是在这里并不对B进行数据内存的分配，B只有一个头文件，和A共用数据存储区。这从输出的数据地址可以看出来，A和B指向的是同一块地址，同时之前提到的refcount指向的数值变为2。运行Mat C

= B.row(3)，这里C相当于是一个10x1的行矩阵，其值为矩阵B（也是A）的第三行，同样这里也并不对C进行数据内出的分配，只是把B(A)的第三行数据的地址赋给C的数据起始地址。通过这个语句的执行，可以知道data指针和datastart指针的区别了。data就是指整个被分配的数据块的起始地址，而datastart就是当前矩阵的数据的起始地址。对于A(B)来说，两者是一样的，但是因为C只有一行，所以这两者并不一样了。同时，refcount指向的的值为3。

那么，如果我们想定义个新的矩阵，并且希望给新的矩阵重新分配数据内存，那么该怎么办呢？看下面！Mat D = B.clone()，这时发现有不一样的了，矩阵D的数据地址和A(B)就不一样了，说明给D分配了新的内存块。事实上，clone函数就是专门用来给新矩阵也分配独立内存的。注意，这时A(B)的refcount指向的数没有变为4，还是3，也印证了refcount的作用。运行A = D，改变了A，这时A的数据变为了D的数据了，A和D一样的了。B的refcount减1变为2，而D的refcount则变为2。运行B.release()，就把B给释放掉了，这时B的数据指针都是空指针了，但是最开始分配的第一块数据内存并没有被释放，因为还有C在用这块内存，虽然C只用了一部分，但是还是可以看到C的refcount指向的数变为1，说明这时候就只有C在用这块数据内存了。

最后，再做个小测试。运行C = C.clone()，可能第一眼看不明白这是在干什么。其实这是在重新给C分配内存。这时候最开始的第一条语句 Mat A(10, 10, CV\_16U)运行时分配的第一块内存终于释放了！同时注意，给C新分配内存的时候，数据内存的大小只有一行矩阵数据的大小(另外加上四个字节的refcount)，这时候C的data和datastart是一样的了！

好了！到目前为止，对于OpenCV2中的 Mat 数据类型已经有了一个比较具体的认识了。不过现在还只是有了一个基本的认识，后面将进一步通过例子，来学习 Mat 的相关操作。这才是最关键的部分！

---------------------

作者：蜗牛一步一步往上爬

来源：CSDN

原文：https://blog.csdn.net/yc461515457/article/details/48720165

版权声明：本文为博主原创文章，转载请附上博文链接！