**摘要：**

Sizeof的作用非常简单：求对象或者类型的大小。然而sizeof又非常复杂，它涉及到很多特殊情况，本篇把这些情况分门别类，总结出了sizeof的10个特性：

(0)sizeof是运算符，不是函数；

(1)sizeof不能求得void类型的长度；

(2)sizeof能求得void类型的指针的长度；

(3)sizeof能求得静态分配内存的数组的长度!

(4)sizeof不能求得动态分配的内存的大小!

(5)sizeof不能对不完整的数组求长度；

(6)当表达式作为sizeof的操作数时，它返回表达式的计算结果的类型大小，但是它不对表达式求值！

(7)sizeof可以对函数调用求大小，并且求得的大小等于返回类型的大小，但是不执行函数体！

(8)sizeof求得的结构体(及其对象)的大小并不等于各个数据成员对象的大小之和！

(9)sizeof不能用于求结构体的位域成员的大小，但是可以求得包含位域成员的结构体的大小！

**特性1：sizeof不能求得void类型的长度**

是的，你不能用sizeof(void)，这将导致编译错误：illegal sizeof operand。事实上你根本就无法声明void类型的变量，不信你就试试void a;这样的语句，编译器同样会报错：illegal use of type 'void'。或许你要问为什么，很好，学东西不能只知其然，还要知其所以然。我们知道声明变量的一个重要作用就是告诉编译器该变量需要多少存储空间。然而，void是“空类型”，什么是空类型呢，你可以理解成不知道存储空间大小的类型。既然编译器无法确定void类型的变量的存储大小，那么它自然不让你声明这样的变量。当然了，声明void类型的指针是可以的！这就是特性2的内容。

**特性2：sizeof能求得void类型的指针的长度**

在特性1中说过，可以申明void类型的指针，也就是说编译器可以确定void类型的指针所占用的存储空间。事实上确实如此，目前，几乎所有平台上的所有版本的编译器都把指针的大小看做4byte，不信你试试sizeof(int\*);sizeof(void\*);sizeof(double\*)；sizeof(Person\*)；等等，它们都等于4！为什么呢？问得好，我将尽全力对此作出解释：其实**指针也是变量**，只不过这个变量很特殊，它是**存放其他变量的地址的变量**。又由于目前32位计算机平台上的程序段的寻址范围都是4GB，寻址的最小单元是byte，4GB等于232Byte，这么多的内存其地址如果编码呢，只需要用32个bit就行了，而32bit = 32/8 = 4byte，也就是说只需要4byte就能存储这些内存的地址了。因此对任何类型的指针变量进行sizeof运算其结果就是4！

**特性3：sizeof能求得静态分配内存的数组的长度!**

Int a[10];int n = sizeof(a);假设sizeof(int)等于4，则n= 10\*4=40；特别要注意：char ch[]=”abc”;sizeof(ch);结果为4，注意字符串数组末尾有’\0’！通常我们可以利用sizeof来计算数组中包含的元素个数，其做法是：int n = sizeof(a)/sizeof(a[0]);

**非常需要注意的是**对函数的形参数组使用sizeof的情况。举例来说，假设有如下的函数：

void fun(int array[10])

{

         int n = sizeof(array);

}

你会觉得在fun内，n的值为多少呢？如果你回答40的话，那么我很遗憾的告诉你，你又错了。这里n等于4，事实上，不管形参是int的型数组，还是float型数组，或者其他任何用户自定义类型的数组，也不管数组包含多少个元素，这里的n都是4！为什么呢？**原因是在函数参数传递时，数组被转化成指针了，或许你要问为什么要转化成指针，原因可以在很多书上找到，我简单说一下：假如直接传递整个数组的话，那么必然涉及到数组元素的拷贝(实参到形参的拷贝)，当数组非常大时，这会导致函数执行效率极低！而只传递数组的地址(即指针)那么只需要拷贝4byte**。

**特性4：sizeof不能求得动态分配的内存的大小!**

假如有如下语句：int\* a = new int[10];int n = sizeof(a);那么n的值是多少呢？是40吗？答案是否定的！其实n等于4，因为a是指针，在特性2中讲过：在32位平台下，所有指针的大小都是4byte！切记，这里的a与特性3中的a并不一样！很多人(甚至一些老师)都认为数组名就是指针，其实不然，二者有很多区别的，要知详情，请看《c专家编程》。通过特性3和特性4，我们看到了数组和指针有着千丝万缕的关系，这些关系也是导致程序潜在错误的一大因素，关于指针与数组的关系问题我将在《**C/C++刁钻问题各个击破之指针与数组的秘密》**一文中进行详细介绍。

特性3指出sizeof能求静态分配的数组的大小，而特性4说明sizeof不能求的动态分配的内存的大小。于是有人认为sizeof是编译时进行求值的，并给出理由：语句int array[sizeof(int)\*10];能编译通过，而很多书上都说过数组大小是编译时就确定下来的，既然前面的语句能编译通过，所以认为sizeof是编译时进行求值的。经过进一步测试我发现这个结论有些武断！至少是有些不严谨！因为在实现了c99标准的编译器(如DEV C++)中可以定义动态数组，即：语句：int num;cin>>num; int arrary[num];是对的(注意在vc6.0中是错的)。因此我就在DEV C++中对刚才的array利用语句int n =sizeof(array);cout<<n<<endl来求大小，结果编译通过，运行时输入num的值10之后，输出n等于40！在这里很明显num的值是运行时才输入的，因此sizeof不可能在编译时就求得array的大小！这样一来sizeof又变成是运行时求值的了。

那么到底sizeof是编译时求值还是运行时求值呢？最开初c标准规定sizeof只能编译时求值，后来c99又补充规定sizeof可以运行时求值。但值得注意的是，即便是在实现了c99标准的DEV C++中仍然不能用sizeof求得动态分配的内存的大小！

**特性5：sizeof不能对不完整的数组求长度！**

在阐述该特性之前，我们假设有两个源文件：file1.cpp和file2.cpp，其中file1.cpp中有如下的定义：

int arrayA[10] = {1,2,3,4,5,6,7,8,9,10};

int arrayB[10] = {11,12,13,14,15,16,17,18,19,20};

file2.cpp包含如下几个语句：

         extern arrayA[];

         extern arrayB[10];

         cout<<sizeof(arrayA)<<endl;            //编译出错！！

         cout<<sizeof(arrayB)<<endl;

在file2.cpp中第三条语句编译出错，而第条语句正确，并且能输出40！为什么呢？原因就是sizeof(arrayA)试图求不完整数组的大小。这里的不完整的数组是指数组大小没有确定的数组！sizeof运算符的功能就是求某种对象的大小，然而声明：extern int arrayA[]只是告诉编译器arrayA是一个整型数组，但是并没告诉编译器它包含多少个元素，因此对file2.cpp中的sizeof来说它无法求出arrayA的大小，所以编译器干脆不让你通过编译。

那为什么sizeof(arrayB)又可以得到arraryB的大小呢?关键就在于在file2.cpp中其声明时使用extern int arrayB[10]明确地告诉编译器arrayB是一个包含10个元素的整型数组，因此大小是确定的。

到此本特性讲解差不多要结束了。其实本问题还能引申出**连接和编译**等知识点，但是目前我暂时还没自信对这两个知识点进行详细的，彻底的讲解，因此不便在此班门弄斧，不久的将来我会在本系列中加上相关问题的阐述

**特性7：sizeof可以对函数调用求大小，并且求得的大小等于返回类型的大小，但是不执行函数体！**

假设有如下函数（是一个写得很不好的函数，但是能很好的说明需要阐述的问题）：

int fun(int& num,const int& inc)

{

         float div = 2.0;

         double ret =0;

         num = num+inc;

         ret = num/div;

         return ret;

}那么语句：

int a = 3;

         int b = 5;

         cout<<sizeof(fun(a,b))<<endl;

         cout<<a<<endl;输出多少呢？不同的人会给出不同的答案，我将对sizeof(fun(a,b))的值和a的值分别进行讨论：

首先sizeof(fun(a,b))的值：其正确是4，因为用sizeof求函数调用的大小时，它得到的是函数**返回类型**的大小，而fun(a,b)的返回类型是int，sizeof(int)等于4。很多人把函数的**返回类型**和**返回值的类型**弄混淆了，认为sizeof(fun(a,b))的值是8,因为函数返回值是ret,而ret被定义成double，sizeof(doube)等于8。注意，虽然函数返回值类型是double，但是在函数返回时，将该值进行了类型转换(这里的转换不安全)。也有人错误的认为sizeof(fun(a,b))的值是12，它们的理由是：fun内部定义了两个局部变量，一个是float一个是double，而sizeof(float)+sizeof(doube)= 4+8=12。这样的答案看似很合理，其实他们是错误地认为这里的sizeof是在求函数内部的变量的大小了。这当然是错误的。

接下来看a的值：其正确答案是3！还记得特性6吗？这里很类似，sizeof的操作对象是函数调用时，它不执行函数体！为此，建议大家不要把函数体放在sizeof后面的括号里，这样容易让人误以为函数执行了，其实它根本没执行。

既然对函数条用使用sizeof得到的是函数返回类型的大小，那么很自然能得出这样的结论：不能对返回类型为void的函数使用sizeof求其大小！原因请参考特性1。同理，对返回类型是任何类型的指针的函数调用使用sizeof求得的大小都为4，原因请参考特性2。

最后我们来看看这样的语句：cout<<sizeof(fun)；其答案是多少呢？其实它得不到答案，原因是编译就通不过！最开始，我以为能输出答案4，因为我认为fun是函数名，而我知道函数名就是函数的地址，地址就是指针，于是我认为sizeof(fun)其实就是对一个指针求大小，根据特性2，任何指针的大小都是4。可是当我去验证时，编译器根本不让我通过！这个是为什么呢？我一时半会想不到，所以还请朋友们补充！