详细解析C语言中的sizeof PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/166/2021_2022__E8_AF_A6_E7_BB_86_E8_A7_A3_E6_c97_166358.htm 计算机等级考试训练软件《百宝箱》sizeof是c语言的一种单目操作符，如c语言的其他操作符 、--等。它并不是函数。sizeof操作符以字节形式给出了其操作数的存储大小。操作数可以是一个表达式或括在括号内的类型名。操作数的存储大小由操作数的类型决定。 二、sizeof的使用方法 1、用于数据类型 sizeof使用形式：sizeof（type） 数据类型必须用括号括住。如sizeof（int）。 2、用于变量 sizeof使用形式：sizeof（var_name）或sizeof var_name 变量名可以不用括号括住。如sizeof (var_name)，sizeof var_name等都是正确形式。带括号的用法更普遍，大多数程序员采用这种形式。 注意：sizeof操作符不能用于函数类型，不完全类型或位字段。不完全类型指具有未知存储大小的数据类型，如未知存储大小的数组类型、未知内容的结构或联合类型、void类型等。 如sizeof(max)若此时变量max定义为int max(),sizeof(char_v) 若此时char_v定义为char char_v[max]且max未知，sizeof(void)都不是正确形式。 三、sizeof的结果 sizeof操作符的结果类型是size_t，它在头文件中typedef为unsigned int类型。该类型保证能容纳实现所建立的最大对象的字节大小。 1、若操作数具有类型char、unsigned char或signed char，其结果等于1。 ansi c正式规定字符类型为1字节。 2、int、unsigned int 、short int、unsigned short 、long int 、unsigned long 、float、double、long double类型的sizeof在ansi c中没有具体规定，大小依赖于实现，一般可能分别为2

、2、2、2、4、4、4、8、10。 3、当操作数是指针时，sizeof依赖于编译器。例如microsoft c/c 7.0中，near类指针字节数为2，far、huge类指针字节数为4。一般unix的指针字节数为4。 4、当操作数具有数组类型时，其结果是数组的总字节数。 5、联合类型操作数的sizeof是其最大字节成员的字节数。结构类型操作数的sizeof是这种类型对象的总字节数，包括任何垫补在内。 让我们看如下结构： struct {char b. double x.} a. 在某些机器上sizeof（a）=12，而一般sizeof（char） sizeof（double）=9。 这是因为编译器在考虑对齐问题时，在结构中插入空位以控制各成员对象的地址对齐。如double类型的结构成员x要放在被4整除的地址。 6、如果操作数是函数中的数组形参或函数类型的形参，sizeof给出其指针的大小。 四、sizeof与其他操作符的关系 sizeof的优先级为2级，比/、%等3级运算符优先级高。它可以与其他操作符一起组成表达式。如i*sizeof（int）；其中i为int类型变量。 五、sizeof的主要用途 1、sizeof操作符的一个主要用途是与存储分配和i/o系统那样的例程进行通信。例如： void *malloc（size_t size）, size_t fread(void * ptr,size_t size,size_t nmemb,file * stream)。 2、sizeof的另一个的主要用途是计算数组中元素的个数。例如： void * memset（void * s,int c,sizeof(s)）。 六、建议 由于操作数的字节数在实现时可能出现变化，建议在涉及到操作数字节大小时用sizeof来代替常量计算。 本文主要包括二个部分，第一部分重点介绍在vc中，怎么样采用sizeof来求结构的大小，以及容易出现的问题，并给出解决问题的方法，第二部分总结出vc中sizeof的主要用法。 1、 sizeof应用在结构上的情况 请看下面的结构：struct mystruct { double dda1. char dda. int type }.

对结构mystruct采用sizeof会出现什么结果呢？sizeof(mystruct)为多少呢？也许你会这样求： sizeof(mystruct)=sizeof(double) sizeof(char) sizeof(int)=13 但是当在vc中测试上面结构的大小时，你会发现sizeof(mystruct)为16。你知道为什么在vc中会得出这样一个结果吗？ 其实，这是vc对变量存储的一个特殊处理。为了提高cpu的存储速度，vc对一些变量的起始地址做了“对齐”处理。在默认情况下，vc规定各成员变量存放的起始地址相对于结构的起始地址的偏移量必须为该变量的类型所占用的字节数的倍数。下面列出常用类型的对齐方式(vc6.0,32位系统)。 类型 对齐方式（变量存放的起始地址相对于结构的起始地址的偏移量） char 偏移量必须为sizeof(char)即1的倍数 int 偏移量必须为sizeof(int)即4的倍数 float 偏移量必须为sizeof(float)即4的倍数 double 偏移量必须为sizeof(double)即8的倍数 short 偏移量必须为sizeof(short)即2的倍数 各成员变量在存放的时候根据在结构中出现的顺序依次申请空间，同时按照上面的对齐方式调整位置，空缺的字节vc会自动填充。同时vc为了确保结构的大小为结构的字节边界数（即该结构中占用最大空间的类型所占用的字节数）的倍数，所以在为最后一个成员变量申请空间后，还会根据需要自动填充空缺的字节。 下面用前面的例子来说明vc到底怎么样来存放结构的。 struct mystruct{ double dda1. char dda. int type }；为上面的结构分配空间的时候，vc根据成员变量出现的顺序和对齐方式，先为第一个成员dda1分配空间，其起始地址跟结构的起始地址相同（刚好偏移量0刚好为sizeof(double)的倍数），该成员变量占用sizeof(double)=8个字节；接下来为第二个成员dda分配空间，这时下一个可以分配的地址对于结构的起始

地址的偏移量为8，是sizeof(char)的倍数，所以把dda存放在偏移量为8的地方满足对齐方式，该成员变量占用sizeof(char)=1个字节；接下来为第三个成员type分配空间，这时下一个可以分配的地址对于结构的起始地址的偏移量为9，不是sizeof(int)=4的倍数，为了满足对齐方式对偏移量的约束问题，vc自动填充3个字节（这三个字节没有放什么东西），这时下一个可以分配的地址对于结构的起始地址的偏移量为12，刚好是sizeof(int)=4的倍数，所以把type存放在偏移量为12的地方，该成员变量占用sizeof(int)=4个字节；这时整个结构的成员变量已经都分配了空间，总的占用的空间大小为：8 1 3 4=16，刚好为结构的字节边界数（即结构中占用最大空间的类型所占用的字节数sizeof(double)=8）的倍数，所以没有空缺的字节需要填充。所以整个结构的大小为：sizeof(mystruct)=8 1 3 4=16，其中有3个字节是vc自动填充的，没有放任何有意义的东西。 