#pragmapack与sizeof计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022__23pragmapac_c97_645569.htm #pragma 是一个预处理指令，其中#pragma pack指令直接影响到一个结构体、联合体、类的内存布局。那么它是如何影响的，首先我们来看两个类 #pargam pack(8) struct TEST1 { char v[9]. int x. }. struct TEST2 { char v[9]. long long x. }. 请问sizeof(TEST1)和sizeof(TEST2)的值是多少？如果当第一行的指令改成#pargam pack(1)，哪么sizeof又应该是多少呢，呵呵。如果你能马上正确的答出这几个问题，我想你一定对下面的计算方式了如指掌。 对齐参数 指某个变量在内存中的起始位置是对齐参数的整数倍。对于给定的#pargma pack(n)，n可以是集合中{1，2，4，8，16}中的任一个值。 第一步：第一个成员总是从零开始对齐 第二步：计算每个成员的对齐参数。 对于一个简单类型成员，每种类型有一个基本对齐参数（ba)，也就是sizeof的值，如 char 是 1，int 是4。对于简单类型变量数组，其对应对齐参数与其简单类型相同。然后比n比较，取较小值为该成员的实际对齐参数b 对于复合类型成员，如结构体对象成员，取该结构体所有成员变量中最大的对齐参数作该成员的实际对齐参数b。 第三步：结构体的总尺寸应为 所有成员中最大的对齐参数的整数倍。 编辑特别推荐: 送给正在学习C 朋友的50条建议 C 笔试考前练习 了解了计算方法，我们来看一下具体的例子。 #pargam pack(8) // 也是默认的对齐方式 struct TEST1 { char v[9]. //实际对齐参数是 1 int x. //实际对齐参数为4 }. 第一个成员有9个char值，因为x要从4的倍数开始，所以填充了三个byte 所以计算总尺寸为9 3 4

= 16 ,而16正好是4的倍数，所以sizeof(TEST1)=16 #pargam pack(8) // 也是默认的对齐方式 struct TEST2 { char v[9]. //实际对齐参数是 1 long long x. //实际对齐参数为8 }. 而对于TEST2而言，X的对齐参数为8 所以要填充7个byte作为 所以总尺寸为 9 7 8 =24 正好是8的倍数，所以sizeof（TEST2）=24 当TEST2的对齐条件改成了 #pargam pack(4) 那么 成员 x 的实际对齐参数就是4，基本对齐参数与 n的最小值 所以只要填充3个字节，所以总尺寸变成了 9 3 8 =20 正好是4的倍数 所以现在的sizeof(TEST2)=20 我们看一个更复杂的例子，如下 #pragma pack(4) struct TEST1 { char a[9]. long long s. }. struct TEST2 { char a. TEST1 test. }. 由上个例子我们知道sizeof(TEST1)=20,哪么现在sizeof(TEST2)是多少？根据前面所说的计算方式，对于复杂类型的成员按其复杂类型最大的对齐参数计算。在这里是S,它的对齐尺寸是4，因此 sizeof(TEST2) = 24. 更极端的例子，如果TEST1和TEST2它们的对齐条件不一致的情况下，会是如何呢？ #pragma pack(2) struct TEST1 { char a[9]. long long s. }. #pragma pack() #pragma pack(4) struct TEST2 { char a. TEST1 test. }. #pragma pack() 我不给出答案，你可以找个工具编译运行一下，想想这是为什么？ 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com