C TemplateCornerCases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_ti2020/641/2021_2022_C__Templat_c97_641694.htm 编辑特别推荐: 全国计算机等级考试（等考）指定教材 全国计算机等级考试学习视频 全国计算机等级考试网上辅导招生 全国计算机等级考试时间及科目预告 百考试题教育全国计算机等级考试在线测试平台 全国计算机等级考试资料下载 全国计算机等级考试论坛 Following are some corner cases of C template features. A lot of the text is simply extracted from "C Templates: The Complete Guide", with some of my personal understanding. These features are trivial and easily neglected, but you should have some impression to them in case you run into troubles caused by the neglect. I made my notes in English, and I dont bother to translate them into Chinese, forgive my laziness. 0. use "typename" to extract type defined within a type parameter, like this: typename T::iterator itr. otherwise the itr is treated as a data member of T. 1. zero initialization When using a var x of type T, we MUST initialize it like this: T x = T(). so that when T is a primitive type, x can also be initialized with 0, or NULL (when T is a pointer type). Of course we must make sure when T is a class type, it has a default constructor. Simply using T x. cant initialize x when T is a primitive type. 2. .template The .template Construct A very similar problem was discovered after the introduction of typename. Consider the following example using the standard bitset type: templategt. void printBitset (std::bitsetgt. constlt.lt.char,char_traitsgt., allocatorgt. lt.) that follows is not really "less than" but the beginning of a template

argument list. Note that this is a problem only if the construct before the period depends on a template parameter. In our example, the parameter bs depends on the template parameter N. 3. char star string type at reference type parameter Suppose we have a string str: char str[32]. The type of str symbol is "a char string" and "of 32 chars long". similarly the literal "abc" s type is: " a char string" and "of 3 chars long", that is, the "type" information consists of both "base type", which is "char", and "length", which is 32 here. So such a string is of different type to a char* pointer, such as char*p. , a type conversion is done if passing a string literal as argument to a function with char* formal parameter. template gt. inline T constamp. a, T constlt. b ? b : a. } So when we use max("abc", "def") to call above function, it is OK. but if we use max("abc", "defg") to call it, it is wrong, because "abc" and "defg" are of different types --- the length is different. And automatical type conversion is not done for reference types here. This means that the above array str, and string literals like "abc", is not of the same type as "char *pstr.", as most people may believe. Actually it takes a conversion to convert str array or the string literals to a char* type. However, during template argument deduction array-to-pointer conversion (often called decay) occurs only if the parameter does not have a reference type. Thus we have the above issue. And if we dont use reference in above code, like this: template gt. inline T max (T a, T b) { return a &lt. b ? b : a. } Both max("abc", "def") and max("abc", "defg") can build OK, since a automatic conversion from string literal to char* is done, so finally we have the same type char* as T. 4. template template types After

reading the whole lot of text in the book, I realized that this is really a very particular feature, too complicated and restricted to use widely. Though, since it is a very recently added new C template feature, it can be used in your configure script to test whether your compiler conforms to C language standard. The piece of code in the book can directly be used in your m4 file for configure to use. 