一篇文章搞定【所有】Cpp八股文

原创 陈同学 陈同学在搬砖 2021-12-16 09:36

加我微信chen079328 拉你进技术群

往期汇总

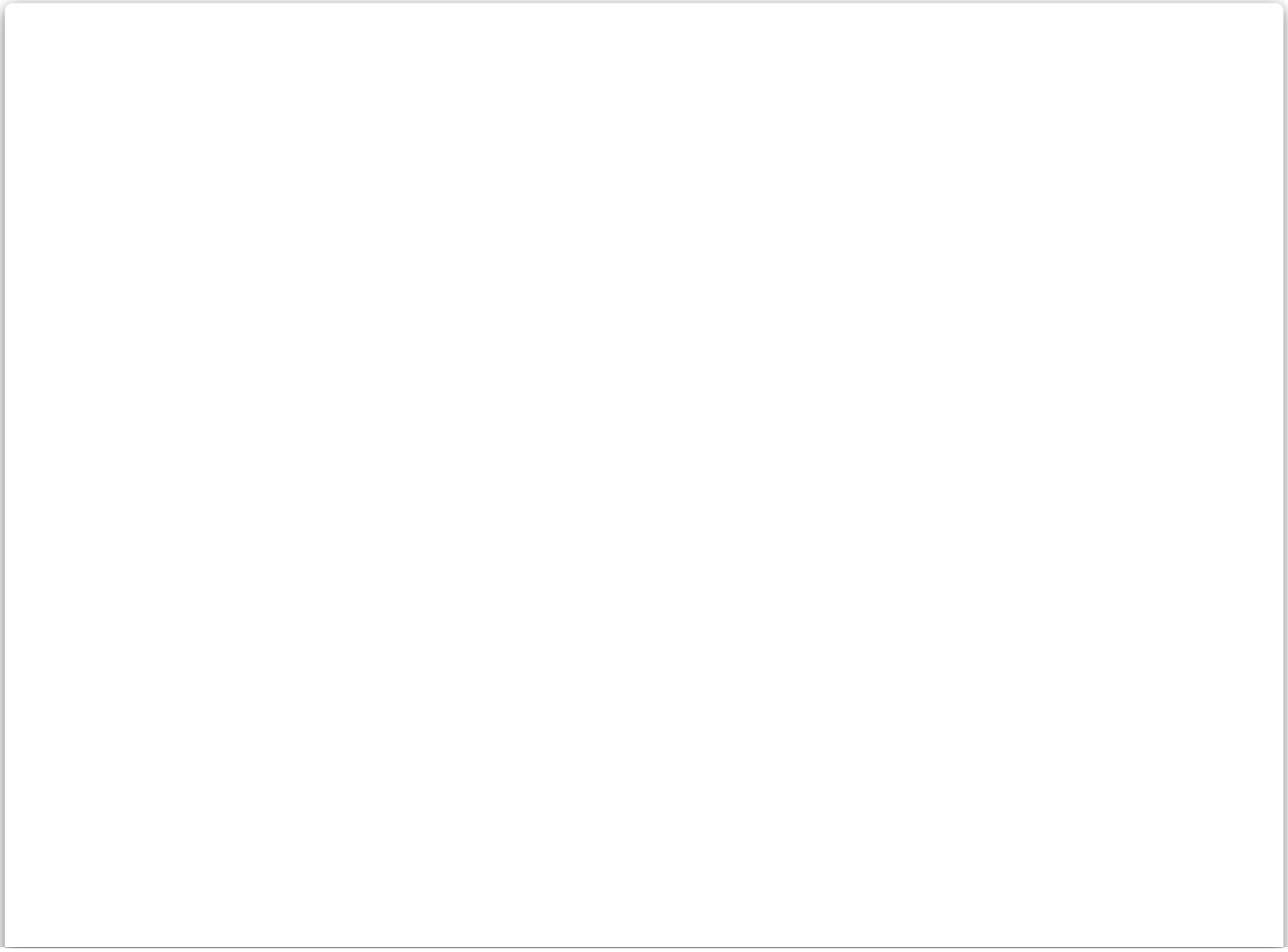
[一篇文章搞定【所有】计算机网络八股文]

[一篇文章搞定【所有】计算机网络八股文]

C++八股文

一、编译相关

Q1 讲讲动态编译和静态编译?



静态编译

编译器在编译可执行文件时，
把需要用到的
对应动态链接库中的部分
提取出来，
连接到可执行文件中去，
使可执行文件
在运行时
不需要依赖于动态链接库；

动态编译

动态编译的可执行文件
需要附带一个动态链接库，
在执行时，
需要调用其对应
动态链接库的命令。
所以其优点一方面
是缩小了执行文件本身的体积，
另一方面是加快了编译速度，
节省了系统资源。
缺点是哪怕是很简单的程序，
只用到了链接库的一两条命令，
也需要附带一个相对庞大的链接库；
二是如果其他计算机上
没有安装对应的运行库，
则用动态编译的可执行文件
就不能运行。

Q2 讲讲Debug和Release区别?

- Denug调试版本，包含调试信息，所以容量比Release大很多， 并且不进行任何优化（优化会使调试复杂化，因为源代码和生成的指令间关系会更复杂）， 便于程序员调试。Debug模式下生成两个文件， 除了.exe或.dll文件外，还有一个.pdb文件，该文件记录了代码中断点等调试信息；

- release发布版本，不对源代码进行调试，编译时对应用程序的速度进行优化， 使得程序在代码大小和运行速度上都是最优的。（调试信息可在单独的PDB文件中生成）。Release模式下生成一个文件.exe或.dll文件。实际上，Debug 和 Release 并没有本质的界限，他们只是一组编译选项的集合， 编译器只是按照预定的选项行动。事实上，我们甚至可以修改这些选项，从而得到优化过的调试版本或是带跟踪语句的发布版本。

Q3 讲讲动态联编和静态联编?

在C++中，联编是指一个计算机程序的不同部分彼此关联的过程。按照联编所进行的阶段不同，可以分为静态联编和动态联编；

静态联编是指联编工作在编译阶段完成的，这种联编过程是在程序运行之前完成的，又称为早期联编。

要实现静态联编，在编译阶段就必须确定程序中的操作调用（如函数调用）与执行该操作代码间的关系，确定这种关系称为束定，在编译时的束定称为静态束定。静态联编对函数的选择是基于指向对象的指针或者引用的类型。其优点是效率高，但灵活性差。

动态联编是指联编在程序运行时动态地进行，根据当时的情况来确定调用哪个同名函数，实际上是在运行时虚函数的

实现。这种联编又称为晚期联编，或动态束定。动态联编对成员函数的选择是基于对象的类型，针对不同的对象类型将做出不同的编译结果。C++中一般情况下的联编是静态联编，但是当涉及到多态性和虚函数时应该使用动态联编。动态联编的优点是灵活性强，但效率低。动态联编规定，只能通过指向基类的指针或基类对象的引用来调用虚函数，其格式为：指向基类的指针变量名->虚函数名（实参表）或基类对象的引用名.虚函数名（实参表）

实现动态联编三个条件：必须把动态联编的行为定义为类的虚函数；

类之间应满足子类型关系，通常表现为一个类从另一个类公有派生而来；

必须先使用基类指针指向子类型的对象，然后直接或间接使用基类指针调用虚函数；

Q4 讲讲整个编译的过程?

源代码－－>预处理－－>编译－－>优化－－>汇编－－>链接-->可执行文件

- 预处理 读取c源程序，对其中的伪指令（以#开头的指令）和特殊符号进行处理。包括宏定义替换、条件编译指令、头文件包含指令、特殊符号。 预编译程序所完成的基本上是对源程序的“替代”工作。经过此种替代，生成一个没有宏定义、没有条件编译指令、没有特殊符号的输出文件。.i预处理后的c文件，.ii预处理后的C++文件。

- 编译阶段 编译程序所要作得工作就是通过词法分析和语法分析，在确认所有的指令都符合语法规则之后，将其翻译成等价的中间代码表示或汇编代码。.s文件

- 汇编过程 汇编过程实际上指把汇编语言代码翻译成目标机器指令的过程。对于被翻译系统处理的每一个C语言源程序，都将最终经过这一处理而得到相应的目标文件。目标文件中所存放的也就是与源程序等效的目标的机器语言代码。.o目标文件

- 链接阶段 链接程序的主要工作就是将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来， 使得所有的这些目标文件成为一个能够诶操作系统装入执行的统一整体。

Q5 讲讲条件编译(ifdef/endif)

一般情况下，源程序中所有的行都参加编译。但是有时希望对其中一部分内容只在满足一定条件才进行编译，也就是对一部分内容指定编译的条件，这就是“条件编译”。有时，希望当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句。

条件编译命令最常见的形式为：

#ifdef 标识符

程序段1

#else

程序段2

#endif

它的作用是：当标识符已经被定义过(一般是用#define命令定义)，则对程序段1进行编译，否则编译程序段2。

其中#else部分也可以没有，即： #ifdef 程序段1 #denif

在一个大的软件工程里面，可能会有多个文件同时包含一个头文件， 当这些文件编译链接成一个可执行文件上时，就会出现大量"重定义"错误。在头文件中使用#define、#ifndef、#ifdef、#endif能避免头文件重定义。

Q6 讲讲一段HelloWorld程序从编译到显示在屏幕中的过程?

hello world 程序开始到打印到屏幕上的全过程?

1.用户告诉操作系统执行HelloWorld程序（通过键盘输入等）

2．操作系统：找到helloworld程序的相关信息，检查其类型是否是可执行文件；并通过程序首部信息，确定代码和数据在可执行文件中的位置并计算出对应的磁盘块地址。

3．操作系统：创建一个新进程，将HelloWorld可执行文件映射到该进程结构，表示由该进程执行helloworld程序。

4．操作系统：为helloworld程序设置cpu上下文环境，并跳到程序开始处。

5．执行helloworld程序的第一条指令，发生缺页异常

6．操作系统：分配一页物理内存，并将代码从磁盘读入内存，然后继续执行helloworld程序

7．helloword程序执行puts函数（系统调用），在显示器上写一字符串

8．操作系统：找到要将字符串送往的显示设备，通常设备是由一个进程控制的，所以，操作系统将要写的字符串送给该进程

9．操作系统：控制设备的进程告诉设备的窗口系统，它要显示该字符串，窗口系统确定这是一个合法的操作，然后将字符串转换成像素，将像素写入设备的存储映像区

10．视频硬件将像素转换成显示器可接收和一组控制数据信号

11．显示器解释信号，激发液晶屏

12．OK，我们在屏幕上看到了HelloWorld

Q7 讲讲C++11的新特性?

二、变量相关

Q8 C++中各种类型变量所占空间分别为多大?

Q9 C++不同类型变量如何进行零值比较

bool类型：if(flag)

int类型：if(flag == 0)

指针类型：if(flag == null)

float类型：if((flag >= -0.000001) && (flag <= 0. 000001))

Q10 讲讲C++中的引用类型?

- 原理
- 注意点
- 分类

Q11 讲讲C++中的指针类型?

- 基本原理与操作
- 指针与引用
- 指针与函数
- 指针与数组
- 指针与const

Q12 讲讲C++中的struct类型?

- C语言中struct和C++中struct的区别
- struct和class的区别
- struct中的内存对齐

默认的对齐方式：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。

注：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。

⑴示例代码一：

1 struct MyStruct 2 { 3 double dda1; 4 char dda; 5 int type; 6 }; 7 //错：sizeof(MyStruct)=sizeof(double)+sizeof(char)+sizeof(int)=13。8 //对：当在VC中测试上面结构的大小时，你会发现sizeof(MyStruct)为16。

注：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。

⑴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；

⑵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；

⑶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。

⑵示例代码二：交换一下上述例子中MyStruct的成员变量的位置

复制代码 1 struct MyStruct 2 { 3 char dda; 4 double dda1; 5 int type; 6 }; 7 //错：sizeof(MyStruct)=sizeof(double)+sizeof(char)+sizeof(int)=13。8 //对：当在VC中测试上面结构的大小时，你会发现sizeof(MyStruct)为24。

注：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。

⑴先为第一个成员dda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(char)的倍数）， 该成

员变量占用sizeof(char)=1个字节；

⑵接下来为第二个成员dda1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1， 不是sizeof(double)=8的倍数，需要补足7个字节才能使偏移量变为8（满足对齐方式）， 因此VC自动填充7个字节，dda1存放在偏移量为8的地址上，它占用8个字节；

⑶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16， 是sizeof(int)=4的倍数，满足int的对齐方式，所以不需要VC自动填充， type存放在偏移量为16的地址上，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：1+7+8+4=20， 不是结构的节边界数 （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， 所以需要填充4个字节，以满足结构的大小为sizeof(double)=8的倍数。所以该结构总的大小为：sizeof(MyStruct)为1+7+8+4+4=24。其中总的有7+4=11个字节是VC自动填充的，没有放任何有意义的东西。

字节的对齐方式：

在VC中提供了#pragmapack(n)来设定变量以n字节对齐方式。n字节对齐就是说变量存放的起始地址的偏移量有两种情况：第一，如果n大于等于该变量所占用的字节数，那么偏移量必须满足默认的对齐方式；第二，如果n小于该变量的类型所占用的字节数，那么偏移量为n的倍数，不用满足默认的对齐方式。结构的总大小也有个约束条件， 分下面两种情况：如果n大于所有成员变量类型所占用的字节数， 那么结构的总大小必须为占用空间最大的变量占用的空间数的倍数；否则必须为n的倍数。

注：VC对结构的存储的特殊处理确实提高了CPU存储变量的速度， 但有时也会带来一些麻烦，我们也可以屏蔽掉变量默认的对齐方式，自己来设定变量的对齐方式。

⑴示例代码：

1 #pragmapack(push)//保存对齐状态 2 3 4 #pragmapack(4)//设定为4字节对齐 5 6 struct test 7 { 8 char m1; 9 double m4; 10 int m3; 11 }; 12 13 #pragmapack(pop)//恢复对齐状态 复制代码

注：以上结构的大小为16，下面分析其存储情况。

⑴首先为m1分配空间，其偏移量为0，满足我们自己设定的对齐方式（4字节对齐），m1占用1个字节；

⑵接着开始为m4分配空间，这时其偏移量为1，需要补足3个字节，这样使偏移量满足为n=4的倍数（因为sizeof(double)大于n）,m4占用8个字节；

⑶接着为m3分配空间，这时其偏移量为12，满足为4的倍数，m3占用4个字节；这时已经为所有成员变量分配了空间，共分配了16个字节，满足为n的倍数。如果把上面的#pragmapack(4)改为#pragma pack(8)， 那么我们可以得到结构的大小为24。

- 内存对齐的作用

Q13 讲讲C++中的类型转换?

隐式类型转换

强制类型转换

- 1.static_cast
- 2.const_cast
- 3. reinterpret_cast
- 4.dynamic_cast

Q14 讲讲C++中变量是如何初始化的?

Q15 讲讲变量的声明和定义有何区别?

Q16 讲讲C++中的extern关键字?

Q17 讲讲变量的static关键字和const关键字?

Q18 讲讲变量的sizeof运算符?

sizeof与strlen的区别

1)strlen计算字符串的具体长度（只能是字符串），不包括字符串结束符。返回的是字符个数。

2)sizeof计算声明后所占的内存数（字节大小），不是实际长度。

3)sizeof是一个取字节运算符，而strlen是个函数。

4)sizeof的返回值=字符个数*字符所占的字节数，字符实际长度小于定义的长度，此时字符个数就等于定义的长度。若未给出定义的大小，分类讨论，对于字符串数组，字符大 小等于实际的字符个数+1；对于整型数组，字符个数为实际的字符个数。字符串每个字符占1个字节，整型数据每个字符占的字节数需根据系统的位数类确定，32位占4个字节。

5)sizeof可以用类型做参数，strlen只能用char*做参数，且必须以'\0'结尾，sizeof还可以用函数做参数；

6)数组做sizeof的参数不退化，传递给strlen就退化为指针；

Q19 讲讲define关键字?

- define宏定义和函数有何区别?

宏在编译时完成替换，之后被替换的文本参与编译，相当于直接插入了代码，运行时不存在函数调用，执行起来更快；函数调用在运行时需要跳转到具体调用函数。

宏函数属于在结构中插入代码，没有返回值；函数调用具有返回值。

宏函数参数没有类型，不进行类型检查；函数参数具有类型，需要检查类型。

宏函数不要在最后加分号。

- define宏定义和const区别?

宏替换发生在编译阶段之前，属于文本插入替换；const作用发生于编译过程中。

宏不检查类型；const会检查数据类型。

宏定义的数据没有分配内存空间，只是插入替换掉；const定义的变量只是值不能改变，但要分配内存空间。

- define宏定义和typedef区别?

宏主要用于定义常量及书写复杂的内容；typedef主要用于定义类型别名。

宏替换发生在编译阶段之前，属于文本插入替换；typedef是编译的一部分。

宏不检查类型；typedef会检查数据类型。

宏不是语句，不在在最后加分号；typedef是语句，要加分号标识结束。

注意对指针的操作，typedef char * p_char和#define p_char char *区别巨大。

- define宏定义和内联函数(inline)区别?

在使用时，宏只做简单字符串替换（编译前）。而内联函数可以进行参数类型检查（编译时），且具有返回值。

内联函数本身是函数，强调函数特性，具有重载等功能。

内联函数可以作为某个类的成员函数，这样可以使用类的保护成员和私有成员。而当一个表达式涉及到类保护成员或私有成员时，宏就不能实现了。

- 条件编译#ifdef, #else, #endif作用?

可以通过加#define，并通过#ifdef来判断，将某些具体模块包括进要编译的内容。

用于子程序前加#define DEBUG用于程序调试。

应对硬件的设置（机器类型等）。

条件编译功能if也可实现，但条件编译可以减少被编译语句，从而减少目标程序大小。

- 宏定义一个取两个数中较大值的功能

#define MAX（x,y）((x>y?)x:y)

Q20 讲讲inline关键字

Q21 讲讲volatile关键字?

三、 内存管理相关

Q22 讲讲C++ 的内存结构?

- 堆和栈的区别

Q23 讲讲C++的malloc/free原理?

- malloc/free的原理
- malloc/free具体过程 情况一：malloc 小于 128K 的内存，使用 brk 分配 step1

step2

step3

情况二：malloc 大于 128K 的内存，使用 mmap 分配（munmap 释放）

Q24 讲讲C++的new/delete原理?

new

- new所做的工作: 调用operator new分配足够的空间，并调用相关对象的构造函数 默认情况下编译器会将new这个关键字翻译成这个operator new和相应的构造函数。
- operator new
- placemet new
-

new[]

delete

delete[]

Q25 讲讲C++的malloc/free 和 new/delete的使用方法

Q26 讲讲C++的malloc/free 和 new/delete的异同

Q27 讲讲C++智能指针原理?

引入

发展历史

- C++98时代的智能指针
- C++11时代的智能指针 `unique_ptr` `shared_ptr`

`weak_ptr`

底层原理

四、面向对象相关

Q28 讲讲类的构造函数?

Q29 讲讲类的拷贝构造函数?

Q30 讲讲类的析构函数?

Q31 讲讲类的this指针?

Q32 讲讲类的private/public/protected关键字

Q33 构造函数和析构函数调用时机?

Q34 空类的大小是多少？为什么？

Q35 设计一个类计算子类的个数？

Q36 如何计算一个类对象的大小？

Q37 讲讲类的继承？

Q38 讲讲继承类之间的类型转换？

Q39 讲讲虚函数？

由来

声明方式

调用过程

回避机制

纯虚函数

实现机制

其他问题

Q40 讲讲类之间的多继承？

Q41 讲讲类的继承和组合之间的区别与联系？

Q42 讲讲类的多态？

Q43 讲讲静态多态和动态多态的区别?

Q44 讲讲函数重载?

Q45 讲讲函数模板?