支持作者劳动成果，欢迎购买纸质图书，京东、当当、亚马逊、淘宝均有售

程序员面试笔试真题与解析

猿媛之家　编著

程序员最可信赖的求职帮手

机械工业出版社

前　言

程序员求职始终是当前社会的一个热点，而市面上有很多关于程序员求职的书籍，例如《程序员代码面试指南》（左程云著）、《剑指 offer》（何海涛著）、《程序员面试笔试宝典》（何昊编著）、《Java 程序员面试笔试宝典》（何昊编著）、《编程之美》（《编程之美》小组著）和《编程珠玑》（Jon Bentley 著）等。它们都是针对基础知识的讲解，各有侧重点，而且在市场上反映良好。但是，我们发现，当前市面上没有一本专门针对 C/C++程序员、Java 程序员的面试笔试真题的分析与讲解，很多读者朋友们向我们反映，他们经过了精心地准备以后，感觉自己什么知识都会了，但又感觉自己什么都不会，不知道自己是否真的能够在程序员面试笔试中得心应手，心里一点底都没有，偶尔会搜索一下网上一些 IT 企业的面试笔试真题，但这些题大都七拼八凑，毫无系统性可言，而且绝大多数都是一些博主自己做的，简单答案，准确性不高，即使偶尔答案正确了，也没有详细的讲解，这就导致读者做完了这些真题之后，根本就不知道自己做得是否正确。如果下一次这个题目再次被考查，自己还是不会。更有甚者，网上的答案还有可能是错误的，误导读者。

针对这种情况，我们创作团队经过精心准备，从互联网上的海量面试笔试真题中，选取了当前顶级企业（包括微软、谷歌、百度、腾讯、阿里巴巴、360 和小米等）的面试笔试真题，挑选出其中最典型、考察频率最高、最具代表性的真题，做到难度适宜，兼顾各层次读者的需求，同时对真题进行知识点的分门别类，做到层次清晰、条理分明、答案简单明了。最终形成了这样一本《程序员面试笔试真题与解析》。本书特点鲜明，所选真题以及写作手法具有以下特点：

第一，考查率高；本书中所选真题绝非泛泛之辈，其内容全是程序员面试笔试常考点，例如语言基础、操作系统、计算机网络、数据结构与算法和海量数据处理等。

第二，行业代表性强；本书中所选真题全部来自于顶级知名企业，它们是行业的风向标，代表了行业的高水准，其中绝大多数真题因为题目难易适中，而且具有非常好的区分度，通常会被众多小企业全盘照搬，具有代表性。

第三，答案详尽；本书对每一道题目都有非常详细的解答，庖丁解牛，不只是告诉读者答案，还提供了参考答案。授之以鱼的同时还授之以渔，不仅告诉答案，还告诉读者同类型题目以后再遇到了该如何解答。

第四，分类清晰、调理分明；本书对各个知识点都进行了分门别类，这种写法有利于读者针对个人实际情况做到有的放矢，重点把握。

由于图书的篇幅所限，我们没法将所有的程序员面试笔试真题内容都写在书稿中，鉴于此，我们猿媛之家在官方网站（www.yuanyuanba.com）上提供了一个读者交流平台，读者朋友们可以在该网站上上传各类面试笔试真题，也可以查找到自己所需要的知识，同时，读者朋友们也可以向本平台提供当前最新、最热门的程序员面试笔试题、面试技巧、程序员生活等相关材料。除此以外，我们还建立了公众号：**猿媛之家**，作为对外消息发布平台，以期最大限度地满足读者需要。欢迎读者关注探讨新技术。

本书主要针对 C/C++用户，我们还有专门针对 Java 用户的图书，同期出版发行。有需要的读者可以在各大电商网站或是实体书店进行购买。

感谢在我们成长道路上帮助我们的父母、亲人、同事、朋友和同学等，无论我们遇到了多大的挫折与困难，他们对我们都能不离不弃，一如既往地支持与帮助我们，使我们能够开开心心地度过每一天。在此对以上所有人一并致以最衷心的感谢。

所有的成长和伟大，如同中药，都是一个时辰一个时辰熬出来的，所有的好书，都是逐字逐

句琢磨出来的。在技术的海洋里，我们不是创造者，但我们更愿意去当好一名传播者的角色，让更多的求职者能够通过本书的系统学习，找到一份自己满意的工作，实现自己的人生理想与抱负。

我们每个人的人生都是一场戏剧，我们每个人都要成为戏剧的主角，而不应该沦为别人的配角，所以，我建议所有的求职者在求职的道路上，无论遇到了多大的困难，遭遇了多大的挫折，都不要轻言放弃，你们的母校可能不是“985”“211”，你们的学历可能不是本科生、研究生，你们的专业可能也不是计算机相关，但这些都不要紧，只要你认真努力，立志成为一名程序员，以我们辅导过的成千上万的求职者经验而言，百分之九十以上的企业是完全可以进去的。请记住：**在这个世界上，没有人可以让你仰视，除非你自己跪着。**

由于编者水平有限，书中不足之处在所难免，还望读者见谅。读者如果发现问题或是有此方面的困惑，都可以通过邮箱 yuancoder@foxmail.com 联系我们。

猿媛之家

于镐京

程序员最可信赖的求职帮手

目　　录

前言

面试笔试经验技巧篇

面试笔试经验技巧 1　如何巧妙地回答面试官的问题……2
面试笔试经验技巧 2　如何回答技术性问题……3
面试笔试经验技巧 3　如何回答非技术性问题……4
面试笔试经验技巧 4　如何回答快速估算类问题……5
面试笔试经验技巧 5　如何回答算法设计问题……5
面试笔试经验技巧 6　如何回答系统设计问题……7
面试笔试经验技巧 7　如何解决求职中的时间冲突问题……9
面试笔试经验技巧 8　如果面试问题曾经见过，是否要告知面试官……10
面试笔试经验技巧 9　在被企业拒绝后是否可以再申请……11
面试笔试经验技巧 10　如何应对自己不会回答的问题……11
面试笔试经验技巧 11　如何应对面试官的“激将法”语言……12
面试笔试经验技巧 12　如何处理“与面试官持不同观点”这个问题……13
面试笔试经验技巧 13　什么是“职场暗语”……13

面试笔试真题解析篇

第 1 章　C/C++语言基础知识……18

1.1　变量……18
1.2　表达式……19
1.3　数组……20
　1.3.1　一维数组……20
　1.3.2　二维数组……21
1.4　字符串……22
1.5　结构体……31
1.6　指针与引用……32
　1.6.1　指针……32
　1.6.2　引用……39
1.7　预处理……40
　1.7.1　基本概念……40
　1.7.2　#define……40
　1.7.3　#include……44
1.8　循环……46
　1.8.1　while……46

1.8.2 do/while ····· 47
1.8.3 for ····· 47
1.8.4 switch ····· 48
1.9 数据类型 ····· 49
1.9.1 概念 ····· 49
1.9.2 const ····· 50
1.9.3 static ····· 52
1.10 运算 ····· 54
1.10.1 常规运算 ····· 54
1.10.2 ++与-- ····· 60
1.10.3 位运算 ····· 63
1.11 sizeof ····· 67
1.11.1 数组求 sizeof ····· 67
1.11.2 struct 求 sizeof ····· 70
1.12 函数 ····· 71
1.12.1 函数调用 ····· 71
1.12.2 函数参数 ····· 80
1.12.3 重载与覆盖 ····· 81
1.12.4 其他 ····· 83
1.13 Hash（哈希） ····· 83
1.14 内存分配 ····· 86
1.14.1 内存的分配形式 ····· 86
1.14.2 malloc/free 与 new/delete ····· 89
1.14.3 内存泄漏 ····· 91
1.15 编译原理 ····· 92
1.16 面向对象技术 ····· 99
1.16.1 概念 ····· 99
1.16.2 构造函数与析构函数 ····· 105
1.16.3 继承 ····· 111
1.16.4 虚函数 ····· 115
1.16.5 模板 ····· 118
1.17 编程技巧 ····· 119
1.18 其他 ····· 120
第 2 章 数据库 ····· 122
2.1 基本概念 ····· 122
2.2 数据库设计 ····· 134
第 3 章 网络与通信 ····· 137
3.1 网络模型 ····· 137
3.2 网络设备 ····· 138
3.3 网络协议 ····· 139
3.4 网络编程 ····· 150
3.5 网络安全 ····· 153
3.6 其他 ····· 155

第 4 章 操作系统 161
4.1 基本概念 161
4.2 进程与线程 173
4.3 内存管理 184
第 5 章 计算机组成与原理 192
第 6 章 软件工程与设计模式 199
6.1 软件工程与 UML 199
6.2 设计模式 201
6.3 软件测试 203
第 7 章 数据结构与算法 206
7.1 数组与线性表 206
7.2 链表 221
7.3 字符串 231
7.4 栈、队列 246
7.5 STL 容器 252
7.6 排序 253
7.7 查找 260
7.8 二叉树 261
7.9 图 275
7.9.1 有向图 275
7.9.2 无向图 277
7.9.3 遍历 278
7.10 其他 279
第 8 章 前端技术 294
第 9 章 数学知识 297
9.1 逻辑推理 297
9.2 概率与组合 301
9.3 数学计算 309
第 10 章 系统设计题 315
第 11 章 海量数据处理 322
11.1 问题分析 322
11.2 基本方法 322
11.2.1 Hash 法 322
11.2.2 Bit-map 法 324
11.2.3 Bloom filter 法 325
11.2.4 数据库优化法 326
11.2.5 倒排索引法 327
11.2.6 外排序法 328
11.2.7 Trie 树 329
11.2.8 堆 332
11.2.9 双层桶法 332

11.2.10 MapReduce 法……334
11.3 经典实例分析……334
11.3.1 top K 问题……334
11.3.2 重复问题……336
11.3.3 排序问题……338

面试笔试经验技巧篇

想找到一份程序员的工作，没有技术显然是不行的，但只有技术也是不够的。面试笔试经验技巧篇主要针对程序员面试笔试中遇到的 13 个常见问题进行深度解析，并且结合实际情景，给出了一个较为合理的参考答案供读者学习与应用。掌握这 13 个问题的应答技巧，对于求职者大有裨益。

面试笔试经验技巧 1　如何巧妙地回答面试官的问题

在程序员面试中，求职者不可避免地需要回答由面试官提出的各种刁钻、犀利的问题，这时不能简单地回答“是”或者“不是”，而应该具体分析“是”或者“不是”的理由。

那么，面对面试官提出的各类问题，如何才能条理清晰地回答呢？如何才能让自己的回答令面试官满意呢？

谈话是一种艺术，回答问题也是一种艺术。同样的问题，不同的回答方式，往往会产生不同的效果，甚至是截然不同的效果。在此，编者提出以下几点建议，供读者参考。

首先，回答问题务必谦虚谨慎。既不能让面试官觉得自己很自卑，唯唯诺诺，也不能让面试官觉得自己清高自负，而应该通过回答问题表现出自己自信从容、不卑不亢的一面。例如，当面试官提出“你在项目中起到了什么作用”的问题时，如果回答“我完成了团队中最难的工作”，可能会给面试官一种居功自傲的感觉；而如果回答“我完成了文件系统的构建工作，这个工作被认为是整个项目中最具挑战性的一部分内容，因为它几乎无法重用以前的框架，需要重新设计”，这种回答有理有据，更能打动面试官。

其次，回答面试官的问题时，要适当地留有悬念。人一般都有猎奇的心理，面试官自然也不例外，而且人们往往对好奇的事情更有兴趣，也更加记忆深刻。所以，在回答面试官问题时，应说关键点而非细节，说重点而非和盘托出，通过关键点，吸引面试官的注意力，等待他们继续“刨根问底”。例如，当面试官希望对简历中的一个算法问题有进一步了解时，求职者可以这样回答“我设计的这种查找算法，对于 80%以上的情况，都可以将时间复杂度从 O(n)降低到 O(logn)，如果您有兴趣，我可以详细给您分析具体的细节。”

最后，回答问题要条理清晰、简单明了，最好使用“三段式”方式。所谓“三段式”，有点类似于中学作文中的写作风格，包括“场景/任务”“行动”“结果”三部分内容。以面试官提的问题“你在团队建设中，遇到的最大挑战是什么”为例，第一步，分析“场景/任务”：在我参与的一个企业资源计划（Enterprise Resource Planning，ERP）项目中，我们团队一共四个人，除了我以外的其他三个人中，有两个人能力较好，人也比较好相处，但有一个人却不太好相处，每次小组讨论问题时，他都不太爱说话，也很少发言，分配给他的任务也很难完成。第二步，分析“行动”：为了提高团队的综合实力，我决定找个时间和他单独谈谈。于是我利用周末时间，约他一起吃饭，顺便讨论项目问题，并询问了一些项目中他遇到的问题，通过他的回答，我发现他并不懒，只是对项目不太了解，缺乏经验，缺乏自信而已，所以越来越孤立，越来越不愿意讨论问题。为了解决这个问题，我尝试着把问题细化到他可以完成的程度，从而帮助他建立自信心。第三步，分析“结果”：慢慢地，他的技术水平有了大幅提升，不仅能够按时完成安排给他的工作，人也越来越自信了，也越来越喜欢参与小组的讨论，并能良好地表达自己的想法了。由此，团队整体协作能力也得到了提升。“三段式”回答一个最明显的好处就是条理清晰，既有描述，也有结果，有理有据，让面试官一目了然。

回答问题是一门大学问。求职者可以在平时的生活中加以练习，提高自己与人沟通的技

能，等到面试时，自然也得心应手了。

如何回答技术性问题

在面试中，面试官会经常询问一些技术性的问题，有的问题可能是历年的笔试面试真题，求职者在平时的复习中会经常遇到，应对自然不在话下，但有的题目可能比较难，来源于 Google、Microsoft 等大企业的题库或是企业为了招聘需要自己设计的题库，求职者可能从来没见过或从来都不能完整地、独立地想到解决方案，而这些题目往往又是企业比较关注的。

如何能够回答好这些技术性的问题呢？编者建议，会做的一定要拿满分，不会做的一定要拿部分分。也就是说，对于简单的题目，求职者要努力做到完全正确；对于难度比较大的题目，不要有畏难心理，即使无法完全做出来，也要努力思考问题，至少要把自己的思路表达清楚，而不是完全回答“不会”或放弃，因为面试官除了关注你独立思考问题的能力以外，还会关注你技术能力的可塑性，观察求职者是否能够在别人的引导下正确地解决问题，所以，对于你不会的问题，他们很有可能会循序渐进地启发你去思考，以期通过这个过程进一步了解你。

一般而言，在回答技术性问题时，求职者大可不必胆战心惊，除非是没学过的新知识，否则，一般都可以采用以下六个步骤来分析解决。

（1）勇于提问　面试官提出的问题有时可能过于抽象，让求职者不知所措或无从下手，所以，对于面试中的疑惑，求职者要勇敢地提出来，多向面试官提问，把不明确或是有二义性的情况都问清楚。这样做不仅不会让面试官烦恼，影响你的面试成绩，相反还会对面试结果产生积极影响：一方面，提问可以让面试官知道你在思考，也可以给面试官留下一个心思缜密的好印象；另一方面，提问有利于后续自己对问题的解答。

例如，面试官提出一个问题——设计一个高效的排序算法。求职者可能摸不到头脑，排序对象是链表还是数组？数据类型是整型、浮点型、字符型，还是结构体类型？数据基本有序还是杂乱无序？数据量有多大？1000 以内还是百万以上个数？此时，求职者大可以将自己的疑问提出来，问题清楚了，就可以有针对性地设计解决方案了。

（2）高效设计　对于技术性问题，如何才能打动面试官？完成基本功能肯定是必需的，仅此而已吗？显然不是，完成基本功能只能算是及格，要想达到优秀，至少还应该考虑更多的内容。以排序算法为例，时间是否高效？空间是否高效？数据量不大时也许没有问题，如果是海量数据呢？是否考虑了相关环节，例如数据的“增删改查”？是否考虑了代码的可扩展性、安全性、完整性以及鲁棒性？如果是网站设计，是否考虑了大规模数据访问的情况？是否需要考虑分布式系统架构？是否考虑了开源框架的使用？

（3）伪代码先行　有时实际代码会比较复杂，上手就写很有可能漏洞百出、条理混乱，所以，求职者可以先征求面试官的同意，在编写实际代码前，写一个伪代码或画好流程图，这样做往往会让思路更加清晰明了。

切记在写伪代码前要告诉面试官，否则他们很有可能对你产生误解，认为你只会纸上谈兵，缺乏实际编码能力。

（4）控制节奏　如果是算法设计题，面试官都会给求职者一个用以完成设计的时间限制，

一般为 20min 左右。完成得太慢，会给面试官留下能力不够的印象；但完成得太快，如果不能保证 100%正确，也会给面试官留下毛手毛脚的印象。速度快当然是好事情，但只有速度没有质量，就不会起到加分的作用。所以，编者建议控制好答题节奏，如果完成得比较快，也不要急于提交给面试官，最好能够利用剩余的时间，认真仔细地检查一些边界情况、异常情况、极性情况等是否也能满足要求。

（5）规范编码　回答技术性问题时，多数都是在纸上写代码，离开了编译器的帮助，求职者要想让面试官对自己的代码一看即懂，除了字迹工整以外，最好能够严格遵循编码规范，同时，代码设计应该具有完整性，保证代码能够完成基本功能、输入边界值能够得到正确的输出、对各种不合规范的非法输入能够做出合理的错误处理，否则，写出的代码即使很高效，面试官也不一定看得懂或看起来非常费劲，这对面试成功都是非常不利的。

（6）精心测试　众所周知，任何软件都有缺陷（Bug）。但不能因为如此，就让自己的代码漏洞百出。尤其是在面试过程中，实现功能也许并不那么困难，难的是在有限的时间内设计出的算法是否使各种异常都得到了有效的处理，是否使各种边界值都在算法设计的范围内，等等。

测试代码是让代码变得更加完美的高效方式之一，也是一名优秀程序员必备的素质之一。所以，在编写代码前，求职者最好能够了解一些基本的测试知识，做一些基本的单元测试、功能测试、边界测试以及异常测试。

在回答技术性问题时，即便是在思考问题，也不要“惜字如金”。面试官面试的时间是有限的，他们希望在有限的时间内尽可能地去了解求职者，求职者的“惜字如金”很有可能会让面试官觉得思考问题能力以及沟通能力存在问题。

其实，在面试时，求职者往往会存在一种思想误区——把技术性面试的结果看得过于重要。对于面试过程中的技术性问题，结果固然重要，但也并非最重要的内容，因为面试官看重的不仅仅是最终的结果，还包括求职者在解决问题的过程中体现出来的逻辑思维能力以及分析问题能力。所以，求职者在面试过程中，要适当地提问，通过提问获取面试官的反馈信息，并抓住这些有用信息进行辅助思考，从而提高面试的成功率。

如何回答非技术性问题

评价一个人的能力，除了专业能力，还有一些非专业能力，如智力、沟通能力、反应能力等，所以在 IT 企业招聘过程的笔试面试环节中，并非所有的笔试内容都是 C/C++、数据结构与算法、操作系统等专业知识，也包括其他一些非技术类的知识，如智力题、推理题、作文题等。技术水平类测试可以考查一个求职者的专业素养，而非技术类测试则更侧重于考查求职者的综合素质，包括数学分析能力、反应能力、临场应变能力、思维灵活性、文字表达能力、性格特征等。考查的形式多种多样，但与公务员考查相似，主要包括行测（占大多数）、性格测试（大部分都有）、应用文、开放问题等内容。

每个人都有自己的答题技巧，答题方式也各不相同，以下是一些相对比较好的答题技巧（以行测为例）。

1）合理有效的时间管理。由于题目的难易不同，因此不要对所有题目都“绝对公平”，要有轻重缓急，最好的做法是不按顺序回答。行测有各种题型，如数量关系、图形推理、应

用题、资料分析、文字逻辑等，而不同的人擅长的题型是不一样的，因此应该先回答自己最擅长的问题。例如，如果对数字比较敏感，那么就先答数量关系。

2）注意时间的把握。由于题量一般都比较大，因此可以先按照总时间/题数来计算每道题的平均答题时间，如 10s，如果看到某一道题 5s 后还没思路，则应马上放弃。在做行测题目时，以在最短的时间内拿到最多分为目标。

3）平时多关注图表类题目，培养迅速抓住图表中的各个数字要素间相互逻辑关系的能力。

4）做题要集中精力，只有集中精力、全神贯注，才能将自己的水平最大限度地发挥出来。

5）学会关键字查找，能够提高做题效率。

6）提高估算能力，很多时候，估算能够极大地提高做题速度，同时保证正确率。

除了行测以外，一些企业非常相信个人性格对职位的影响，所以都会引入相关的性格测试题用于测试求职者的性格，看其是否适合所申请的职位。大多数情况下，只要按照自己的真实想法选择就行了，不要弄巧成拙，因为测试是为了得出正确的结果，所以大多数测试题前后都有相互验证的题目。如果求职者自作聪明，选择该职位可能要求的性格选项，则很可能导致测试前后不符，这样很容易让企业认为你是个不诚实的人，从而不予考虑。

如何回答快速估算类问题

有些企业的面试官喜欢出一些快速估算类问题。对他们而言，这些问题只是手段，不是目的，能够得到一个满意的结果固然是他们所需要的，但他们更希望通过这些题目去考查求职者的快速反应能力以及逻辑思维能力。由于求职者平时准备的时候可能对此类问题有所遗漏，因此一时很难想到解决的方案。这些题目看似毫无头绪，但实际上比较灵活，属于开放性试题，一般没有标准答案，只要找准回答要点，分析合理到位，使答案具有说服力，就可以了。

例如，面试官可能会问这样一个问题：“请你估算一家商场在促销时一天的营业额”，求职者如何能够得出一个准确的数据呢？

其实本题只要能够分析出一个概数就行了，不一定要得出精确数据，而分析概数的前提就是做出各种假设。以该问题为例，可以尝试从以下思路入手：从商场规模及商铺规模入手，通过每平方米的租金估算出商场的日租金，再根据商铺的成本构成得到全商场日均交易额，然后考虑促销时的销售额与平时销售额的倍数关系，乘以倍数，即可得到促销时一天的营业额。具体而言，包括以下估计数值：

1）以一家较大规模商场为例，商场一般按 6 层计算，每层大约长 100m，宽 100m，合计 60000m^2。

2）商铺规模约占商场规模的一半，合计 30000m^2。

3）商铺租金约为 40 元/m^2，估算出年租金为 40 元/m^2×30000m^2×365 天=4.38 亿元。

4）对商户而言，租金一般占销售额的 20%左右，则年销售额为 4.38 亿元÷20%=21.9 亿元，则计算平均日销售额为 21.9 亿元/365 天=600 万。

5）促销时的日销售额一般是平时的 10 倍，所以大约为 600 万×10=6000 万。

此类题目涉及面比较广，例如，“估算北京小吃店的数量”“估算中国在过去一年方便面的市场销售额”“估算长江的水的质量”“估算一个行进在小雨中的人 5min 内身上淋到的雨的质量”等等。但一般都是即兴发挥，遇到此类问题，读者应一步步地抽丝剥茧，找准要点，合理分析，给出具有说服力的答案。

如何回答算法设计问题

在面试中，很多算法设计问题都是各家企业历年来的现成题目，不管求职者以前对算法知识学习得是否扎实，理解得是否深入，学上一段时间，牢记于心，应付此类题目应该没有问题，但遗憾的是，很多世界级知名企业也深知这一点，如果纯粹是出一些毫无技术含量的题目，会让考前“突击手”占尽便宜，这样不利于人才的选拔，所以，为了把优秀的求职者与一般的求职者能够更好地区分开来，他们往往会推陈出新，越来越倾向于出一些有技术含量的新题。

在面试中，算法的地位如同托福考试在出国考试中的地位——是必需的但不是最重要的，它只是众多考核方面中的一个，不直接决定求职者的“生死”。虽然如此，但并不是说不用去准备算法知识，因为算法知识回答得好，必然会成为面试的加分项，对于求职成功，百利而无一害。那么，如何应对此类题目呢？很显然，编者不可能将此类题目都在《程序员面试笔试宝典》中一一解答，一来由于内容众多，篇幅有限；二来也没必要，今年考过了，以后一般就不会再考了，不然还是没有区分度。编者以为，靠死记硬背肯定是行不通的，解答此类算法设计问题，需要求职者具有扎实的基本功以及良好的运用能力，这里仅提供一些比较好的答题方法和解题思路，以供求职者在面试时应对此类算法设计问题。

（1）归纳法　此方法通过写出问题的一些特例来分析总结其中的一般规律。具体而言，就是通过列举少量的特殊情况，经过分析，最后找出其中的一般关系。例如，某人有一对兔子饲养在围墙中，如果它们每个月生一对兔子，且新生的兔子在第二个月后也是每个月生一对兔子，问：一年后围墙中共有多少对兔子？

使用归纳法解答此题，首先想到的就是第一个月有多少对兔子。第一个月，最初的一对兔子生下一对兔子，此时围墙内共有两对兔子。第二个月仍是最初的一对兔子生下一对兔子，共有 3 对兔子。到第三个月，除最初的兔子新生一对兔子外，第一个月生的兔子也开始生兔子，因此共有 5 对兔子。通过举例，可以看出，从第二个月开始，每一个月兔子总数都是前两个月兔子总数之和，$U_{n+1}=U_n+U_{n-1}$，一年后，围墙中的兔子总数为 377 对。

此种方法比较抽象，也不可能对所有的情况进行列举，所以，得出的结论只是一种猜测，还需要进一步证明。

（2）相似法　如果面试官提出的问题与求职者以前用某个算法解决过的问题相似，此时就可以触类旁通，尝试改进原有算法来解决这个新问题。而通常情况下，此种方法都会比较有效。

例如，实现字符串的逆序打印，也许求职者从来就没遇到过此问题，但将字符串逆序肯定是在求职准备的过程中见过的。将字符串逆序的算法稍加处理，即可实现字符串的逆序打印。

（3）简化法　使用此方法，应先将问题简单化，例如改变一下数据类型、空间大小等，然后尝试着将简化后的问题解决，一旦有了一个算法或思路可以解决这个被简化过的问题，再将问题还原，尝试用此类方法解决原有问题。

例如，在海量日志数据中提取出某日访问×××网站次数最多的 IP。很显然，由于数据量巨大，直接进行排序不可行，但如果数据规模不大，采用直接排序不失为一种好的解决方法。那么，如何将问题规模缩小呢？于是想到了散列法，散列往往可以缩小问题规模，然后在简化过的数据里面使用常规排序算法即可找出此问题的答案。

（4）递归法　为了降低问题的复杂度，很多时候都会将问题逐层分解，最后分解为一些最简单的问题，这就是递归。使用此种方法，应先解决最基本的情况，然后以此为基础，解决其他问题。

例如，在寻求全排列时，可能会感觉无从下手，但仔细推敲，会发现后一种排列组合往往是在前一种排列组合的基础上进行重新排列，一旦知道了前一种排列组合的各类组合情况，只需要把最后一个元素插入到前面各种组合的排列里面，就解决了这一问题，即先截去字符串 s[1，…，n]中的最后一个字母，生成所有 s[1，…，n−1]的全排列，然后再将最后一个字母插入到每一个可插入的位置。

（5）分治法　任何一个可以用计算机求解问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。而分治法也是如此，即将一个难以直接解决的大问题，分割成一些规模较小的相同问题，各个击破，分而治之。分治法一般包含三个步骤：①将问题的实例划分为几个较小的实例，最好具有相等的规模；②对这些较小的实例求解，而最常见的方法一般是递归；③如果有必要，则合并这些较小问题的解，以得到原始问题的解。

分治法是程序员面试常考的算法之一，一般适用于二分查找、大整数相乘、求最大子数组和、找出伪币、金块问题、矩阵乘法、残缺棋盘、归并排序、快速排序、距离最近的点对、导线与开关等。

（6）散列法　很多面试笔试题目都要求求职者给出的算法尽可能高效。那么，究竟什么样的算法是高效的？一般而言，时间复杂度越低，算法越高效。而要想达到时间复杂度的高效，很多时候就必须在空间上有所牺牲，即“用空间来换时间”。而用空间换时间最有效的方式就是散列法、大数组、位图法。当然，此类方法并非“包治百病”，有时面试官也会对空间大小进行限制，此时，求职者只能再去思考其他的方法了。

其实，凡是涉及大规模数据处理的算法设计，散列法就是最好的方法之一。

（7）轮询法　每道面试笔试题，在设计时，往往会有一个载体，这个载体便是数据结构，例如数组、链表、二叉树、图等。当载体确定后，可用的算法自然而然地就会显露出来。可问题是，很多时候并不确定这个载体是什么。当无法确定这个载体时，求职者一般也就很难想到合适的方法了。

编者建议，此时求职者可以采用最原始的方法——轮询，在脑海中轮询各种可能的数据结构与算法，从中找出一种合适的解法。

常考的数据结构与算法知识点见下表。

数据结构	算法	概念
链表	广度（深度）优先搜索	位操作
数组	递归	设计模式

二叉树	二分查找	内存管理（堆、栈等）
树	排序（归并排序、快速排序等）	
堆（大顶堆、小顶堆）	树的插入、删除、查找、遍历等	
栈	图论	
队列	散列法	
向量	分治法	
散列表	动态规划	

此种方法看似笨拙，其实很实用，只要求职者对常见的数据结构与算法烂熟于心。

为了更好地理解这些方法，求职者可以在平时的准备过程中，应用此类方法去答题，练习得多了，自然就熟能生巧了，面试时遇到此类问题，也就能够应对自如了。

如何回答系统设计问题

在面试时，求职者偶尔也会遇到一些系统设计问题，而这些题目往往只是测试求职者的知识面，或者测试求职者对系统架构知识的了解，一般不会涉及具体的编码工作。虽然如此，对于此类问题，很多人还是感觉难以应对。

如何应对此类题目呢？在正式介绍基础知识之前，首先罗列几个常见的与系统设计相关的面试笔试题。

1）设计一个域名系统（Domain Name System，DNS）的 Cache 结构，要求能够满足每秒 5000 次以上的查询，满足 IP 数据的快速插入，查询的速度要快（题目还给出了一系列的数据，例如，站点数总共为 5000 万，IP 地址有 1000 万，等等）。

2）有 N 台机器，M 个文件，文件可以以任意方式存放到任意机器上，文件可任意分割成若干块。假设这 N 台机器的宕机率小于 1/3，要求在宕机时可以从其他未宕机的机器中完整地导出这 M 个文件，求最好的存放与分割策略。

3）假设有 30 台服务器，每台服务器中都存有上百亿条数据（有可能重复），如何根据某关键字，从中找出重复出现次数最多的前 100 条数据？要求使用 Hadoop 来实现。

4）设计一个系统，要求写速度尽可能快，并说明设计原理。

5）设计一个高并发系统，说明架构和关键技术要点。

6）假设有一个 25T 的 log(query->queryinfo)，log 的大小还在不段地增长，设计一个方案，给出一个 query 能快速返回 queryinfo。

以上所有问题中，凡是不涉及高并发的，基本可以采用 Google 的三个技术解决，即 GFS、MapReduce 和 Bigtable，这三个技术被称为“Google 的三驾马车”，Google 只公开了论文而未公开源代码，开源界对此非常感兴趣，于是仿效这三篇论文实现了一系列软件，如 Hadoop、HBase、HDFS、Cassandra 等。

在 Google 这些技术还未出现之前，企业界在设计大规模分布式系统时，采用的架构往往是 Database+Sharding+Cache，现在很多公司仍采用这种架构。在这种架构中，仍有很多问题值得去探讨。如采用什么数据库，是 SQL 界的 MySQL 还是 NoSQL 界的 Redis/TFS，两者有何优劣？采用什么方式进行数据分片（Sharding）？是水平分片，还是垂直分片？

据网上资料显示，weibo.com 和 taobao 图片存储中曾采用的架构是 Redis/MySQL/TFS+Sharding+Cache。该架构解释如下：前端 Cache 是为了提高响应速度；后端数据库则用于数据永久存储，防止数据丢失；而 Sharding 是为了在多台机器间分摊负载。最前端由大块的 Cache 组成，要保证至少 99%（该数据在 weibo.com 架构中的占比是编者推测的，而在 taobao 图片存储模块是真实的）的访问数据落在 Cache 中，这样可以保证用户访问速度，减少后端数据库的压力。此外，为了保证前端 Cache 中的数据与后端数据库中的数据一致，需要有一个中间件异步更新（这是因为同步代价太高。请思考，异步有缺点，如何弥补？）数据。另外，为了分摊负载压力和海量数据，会将用户的微博信息经过分片（即 Sharding）后存放到不同的结点上。

这种架构的优点非常明显——简单，在数据量和用户量较小时完全可以胜任。但其扩展性和容错性差、维护成本高，尤其是数据量和用户量暴增之后，系统不能通过简单地增加机器解决该问题。

于是，新的架构便出现了，即前面讲到的“Google 的三驾马车”。

1）**GFS** 是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，提供容错功能。现在开源界有 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS），该文件系统虽然弥补了数据库+Sharding 的很多缺点，但自身仍存在一些问题，例如，由于采用 master/slave 架构，因而存在单点故障问题；元数据信息全部存放在 master 端的内存中，因而不适合存储小文件，或者说如果存储的大量小文件，那么存储的总数据量不会太大。

2）**MapReduce** 是针对分布式并行计算的一套编程模型。其最大的优点是编程接口简单、自动备份（数据默认情况下会自动备三份）、自动容错和隐藏跨机器间的通信。在 Hadoop 中，MapReduce 作为分布计算框架，而 HDFS 作为底层的分布式存储系统，但 MapReduce 不是与 HDFS 耦合在一起的，用户完全可以使用自己的分布式文件系统替换 HDFS。当前 MapReduce 有很多开源实现，如 Java 实现 Hadoop MapReduce、C++实现 Sector/sphere 等，甚至有些数据库厂商将 MapReduce 集成到数据库中。

3）**BigTable** 俗称“大表”，用来存储结构化数据。编者个人认为，BigTable 在开源界最火爆，其开源实现最多，包括 HBase、Cassandra、levelDB 等，使用也非常广泛。

除了 Google 的这“三驾马车”以外，还有其他一些技术可供学习与使用。

1）**Dynamo**：亚马逊的 key-value 模式的存储平台，可用性和扩展性都很好，采用分布式散列表（Distributed Hash Table，DHT）对数据分片，解决单点故障问题。在 Cassandra 中，也借鉴了该技术，在 BT 和电驴中，也采用了类似算法。

2）**虚拟结点技术**：该技术常用于分布式数据分片中。具体应用场景是：有一大堆数据（maybe TB 级或者 PB 级），需按照某个字段（key）分片存储到几十（或者更多）台机器上，同时想尽量负载均衡且容易扩展。传统的做法是“Hash(key) mod N”，这种方法最大的缺点是不容易扩展，即增加或者减少机器均会导致数据全部重新分布，代价太大。这时便可以使用上面提到的 DHT（现在已经被很多大型系统采用）。还有一种是对“Hash(key) mod N”的改进：假设要将数据分布到 20 台机器上，传统做法是“Hash(key) mod 20”，而改进后，N 的取值要远大于 20，如 20000000，然后额外采用一张表记录每个结点存储的 key 的模值，例如，

node1：0～1000000

node2：1000001～2000000
……

这样当添加一个新的结点时，只需将每个结点上的部分数据移动给新结点，同时修改一下这个表即可。

3）**Thrift**：它是一个跨语言的远程过程调用协议（Remote Procedure Call Protocol，RPC）框架。RPC 的使用方式与调用一个普通函数一样，但执行体发生在远程机器上。跨语言是指不同语言之间进行通信，例如 C/S 架构中，Server 端采用 C++编写，Client 端采用 PHP 编写，若要让两者之间通信，则 thrift 是一种很好的方式。

对于这一节开始提出的几道笔试题，都可以通过上面介绍的知识点来加简答，例如：

1）关于高并发系统设计，主要有以下几个关键技术点：缓存、索引、数据分片、锁粒度尽可能小。

2）问题 2 涉及现在通用的分布式文件系统的副本存放策略。一般是将大文件切分成小的块（如 64MB）后，以块为单位存放三份到不同的结点上，这三份数据的位置需根据网络拓扑结构配置。一般而言，如果不考虑跨数据中心，可以这样存放：两个副本存放在同一个机架的不同结点上，而另外一个副本存放在另一个机架上，这样从效率和可靠性上都是最优的（Google 公布的文档对此有专门的证明）。如果考虑跨数据中心，可将两份存在一个数据中心的不同机架上，另一份存到另一个数据中心。

3）问题 4 涉及 BigTable 的模型。主要思想是将随机写转化为顺序写，进而大大提高写速度。具体方法为：由于磁盘物理结构的独特设计，其并发的随机写（主要是因为磁盘寻道时间长）非常慢，考虑到这一点，在 BigTable 模型中，首先会将并发写的大批数据放到一个内存表（称为“memtable”）中，当该表大到一定程度后，会顺序写到一个磁盘表（称为“SSTable”）中，这种写是顺序写，效率极高。说到这里，可能有读者问，随机读可不可以这样优化？答案是：看情况而定。通常而言，如果读并发度不高，则不可以这么做，因为如果将多个读重新排列组合后再执行，则系统的响应时间太慢，用户可能无法接受，而如果读并发度极高，也许可以采用类似机制。

如何解决求职中的时间冲突问题

对于求职者而言，求职季堪比个“赶场季”，一天少则几家、十几家企业入校招聘，多则几十家、上百家企业招兵买马，企业多，选择自然也多，这固然是一件好事情，但由于招聘企业实在是太多了，自然会导致另外一个问题的发生——同一天企业扎堆，且都是自己心仪或欣赏的知名企业。如果不能够提前掌握企业的宣讲时间、地点，则很容易迟到或错过。但有时候即使掌握了宣讲时间、笔试时间、面试时间，还是有可能错过，为什么呢？时间冲突，人不是神仙，不可能具有分身术，也不可能同一时间做两件不同的事情，所以，就必须有所取舍。

到底该如何取舍呢？到底该如何应对这种时间冲突的问题呢？在此，编者将自己的一些想法经验与分享出来，以供读者参考。

1）如果多家心仪企业的校园宣讲时间发生冲突（前提是只宣讲、不笔试），此时最好的解决方法是和同学或朋友商量好，各去一家，然后大家进行信息共享。

2）如果多家心仪企业的笔试时间发生冲突，此时只能选择其一，毕竟企业的笔试时间都是考虑到了成百上千人的安排，需要提前安排考场、考务人员、阅卷人员等，不可能为了一个人而轻易改变。所以，最好选择自己更有兴趣的企业参加笔试。

3）如果多家自己心仪企业的面试时间发生冲突，求职者不要轻易放弃。对于面试官而言，面试任何人都是一样的，因为面试官谁都不认识。而面试时间也是比较灵活的，一般可以通过电话协商。求职者可以与相关工作人员（一般是企业的人力资源师）进行沟通，给出不能参加面试的原因，让其调整时间。但一般要接到面试通知后的第一时间联系相关工作人员变更时间。

那如果多家企业的校园宣讲时间、笔试时间、面试时间发生冲突，到底是去宣讲会，还是去笔试，或是去面试呢？大原则是选择自己最有兴趣的企业。但也要灵活处理，如果仅仅只是宣讲而不涉及笔试，则完全可以不去，可寻求同学或是朋友的帮忙，回来后信息共享即可。如果可以协调面试时间，那就打电话给相关工作人员，换个不冲突的时间去面试，笔试一般时间比较固定，需要协调的事情很多，所以调整时间的可能性也不大，此时如果协调得当，基本能够保证不放弃任何一个机会。

正如世界上没有能够包治百病的药物一样，以上这些建议在应用时，很多情况下也做不到全盘兼顾，当必须进行多选一时，求职者就要对此进行评估了，评估的项目可以包括对企业的中意程度、获得录用通知的概率、去工作的可能性等。评估的结果往往具有很强的参考性，求职者依据评估结果做出的选择一般也会比较合理。

面试笔试经验技巧 8　如果面试问题曾经见过，是否要告知面试官

其实在面试中，大多数题目都是有章可循的，只要求职者肯花时间，复习得当，基本上在面试前都会见过相同的或类似的问题（当然，很多知名企业每年都会推陈出新，这些题目是很难完全复习到位的）。所以，在面试中，求职者遇到自己见过的问题也就不足为奇了。那么，一旦出现这种情况，求职者是否要如实告诉面试官呢？

选择不告诉面试官的理由比较充分：首先，面试的题目60%～70%都是重复以前的，见过或见过类似的不足为奇，难道要一一告知面试官吗？如果那样，估计就没有几个题不用告知面试官了。其次，即使曾经见过该问题，也是自己辛勤耕耘、努力奋斗的结果，很多人复习不用功或方法不到位，也许从来就没见过，而这些题也许正好是拉开求职者差距的分水岭，就是面试官用来区分求职者实力的内容。

同样，选择告诉面试官的理由也比较充分：第一，如实告诉面试官，不仅可以彰显出求职者个人的诚实品德，还可以给面试官留下良好的印象。第二，有些问题，即使求职者曾经复习过，但也无法保证完全回答正确，如果向面试官如实相告，或许还可以规避这一问题，避免错误的发生。第三，求职者如果见过该问题，也能轻松应答，题目简单倒也无所谓，一旦题目难度比较大，求职者却对面试官有所隐瞒，就极有可能给面试官造成一种求职者水平很强的假象，导致面试官的判断出现偏差，后续的面试有可能向着不利于求职者的方向发展。

其实，仁者见仁，智者见智，这个问题并没有固定的答案，需要根据实际情况来决定。

面试笔试经验技巧 9　在被企业拒绝后是否可以再申请

很多企业为了能够在一年一度的招聘季节中，提前将优秀的程序员招至“麾下”，往往会“先下手为强”。他们通常采取的措施有以下两种：第一种，招聘实习生；第二种，多轮招聘。

对求职者来讲，招聘开始后，往往是几家欢喜几家愁，提前拿到企业“绿卡”的，可谓是对酒当歌、欢天喜地；而没有被选上的，担心从此与这家企业无缘了，忧心忡忡。难道一次失望的表现就会被企业拉入“黑名单”了吗？难道一次失败的经历就会永远被记录在个人历史的“耻辱柱”上了吗？

答案当然是否定的，对心仪的女孩表白，即使第一次被拒绝了，还可以一而再再而三地表白，多次表白后成功的案例比比皆是，更何况是求职找工作。一般而言，企业是不会“记仇”的，尤其是知名的大企业，对此都会有明确的表示。如果在企业的实习生招聘或在企业以前的招聘中不幸被淘汰了，一般是不会被拉入企业的“黑名单”的。在下一次招聘中，和其他求职者，具有相同的竞争机会（有些企业可能会要求求职者等待半年到一年时间才能再次应聘该企业，但上一次求职的糟糕表现不会被计入此次招聘中）。

对心仪的对象表白被拒绝了，不是一样还可以继续表白吗？也许是在考验，也许是在等待，也许真的是拒绝，但无论出于什么原因，求职者都不要对自己丧失信心。工作也是如此，以编者身边的很多同学和朋友为例，很多人最开始被一家企业给拒绝了，过了一段时间，他们也成了该企业的员工。所以，即使被企业拒绝了，也不是什么大不了的事情，以后还有机会。有志者自有千计万计，无志者只感千难万难，关键是看你愿意成为什么样的人了。

面试笔试经验技巧 10　如何应对自己不会回答的问题

在面试的过程中，求职者对面试官提出的问题并不是每个问题都能回答上来。计算机技术博大精深，很少有人能对计算机技术的各个分支学科都了如指掌，而且抛开技术层面的问题，在面试那种紧张的环境中，回答不上来的情况也容易出现。在面试中遇到自己不会回答的问题时，错误的做法是保持沉默或者支支吾吾、不懂装懂，硬着头皮胡乱说一通，这样会使面试气氛很尴尬，很难顺利进行。

其实面试遇到不会的问题是一件很正常的事情，没有人是万事通，即使对自己的专业有相当的研究与认识，也可能会在面试中遇到感觉没有任何印象、不知道如何回答的问题。在面试中遇到实在不懂或不会回答的问题，正确的办法是本着实事求是的原则，态度诚恳地告诉面试官自己不知道答案。例如，“对不起，不好意思，这个问题我回答不出来，我能向您请教吗?”

征求面试官的意见时可以说说自己的个人想法，如果面试官同意听，就将自己的想法说出来，回答时要谦逊有礼，切不可滔滔不绝。然后应该虚心地向面试官请教，表现出强烈的学习欲望。

所以，当遇到自己不会的问题时，正确的做法是：“知之为知之，不知为不知”，一定要实事求是，坦然面对，起码应该给面试官留下诚实、坦率的好印象。

面试笔试经验技巧 11　如何应对面试官的“激将法”语言

“激将法”是面试官用以淘汰求职者的一种惯用方法，它是指面试官采用怀疑、尖锐、咄咄逼人的交流方式来对求职者进行提问的方法。例如，“我觉得你比较缺乏工作经验”“我们需要活泼开朗的人，你恐怕不合适”“你的教育背景与我们的需求不太适合”“你的成绩太差”“你的英语没过六级”“你的专业和我们不对口”“为什么你还没找到工作”“你竟然有好多门课不及格”等，很多求职者遇到这样的问题，会很快产生“我是来面试而不是来受侮辱”的想法，往往会被“激怒”。千万要记住，面试的目的是要获得工作，而不是与面试官争个高低。所以对于此类问题，求职者应该巧妙地回答，一方面化解不友好的气氛，另一方面得到面试官的认可。

具体而言，受到这种“激将”时，求职者首先应该保持清醒的头脑。企业让你来参加面试，说明你已经通过了他们第一轮的筛选，至少从简历上看，已经表明你符合求职岗位的需要。其次，做到不卑不亢，不要被面试官的思路带着走，要时刻保持自己的思路和步调。此时可以换一种方式，如介绍自己的经历、工作、优势，来表现自己的抗压能力。针对面试官提出的“非名校毕业”的问题，求职者可以给出巧妙的回答，例如，“比尔盖茨也并非毕业于哈佛大学，但他一样成为了世界首富，成为举世瞩目的人物”；针对缺乏工作经验的问题，不妨回答：“每个人都是从没经验变为有经验的，如果有幸最终能够成为贵公司的一员，我将很快成为一个经验丰富的人”；针对专业不对口的问题，可以回答：“专业人才难得，复合型人才更难得，在某些方面，外行的灵感往往超过内行，他们一般没有思维定势，没有条条框框”。面试官还可能抛出“你的学历对我们来讲太高了”这

样的问题，此时也可以很巧妙地回答：“今天我带来的三张学历证书，您可以从中挑选一张您认为合适的”。针对“性格内向”的问题，可以回答：“内向的人往往具有专心致志、锲而不舍的品质，而且我善于倾听，我觉得应该把发言机会更多地留给他人”。

面对面试官的“挑衅”行为，如果求职者回答得结结巴巴，或者无言以对，抑或怒形于色、据理力争，那就掉进了对方所设的“陷阱”，所以当求职者碰到此种情况时，最重要的一点就是保持头脑冷静，以平常心对待。

面试笔试经验技巧 12　如何处理“与面试官持不同观点”这个问题

在面试的过程中，求职者不可能所有的观点都与面试官保持一致，甚至很有可能在对某个问题的看法上相差甚远。

出现这种情况时，应该委婉地表达自己的真实想法，不妨对其观点的某些方面予以肯定，然后在此基础上说明自己的观点。

什么是“职场暗语”

以往常规的面试套路，已经被众多“面试达人”们挖掘出了各种“破解秘诀”，形成了类似“求职宝典”的各类“面经”（面试经验）。所谓“道高一尺，魔高一丈”，面试官们也纷纷升级面试模式，制作了更为隐蔽、间接、含混的面试题目，让那些早已流传开来的

“面试攻略”毫无用武之地，一些蕴涵丰富信息但以更新面目出现的问话屡屡“秒杀”求职者，让求职者深感困惑，例如“面试官从头到尾都表现出对我很感兴趣的样子，营造出马上就要录用我的氛围，为什么我最后还是没有被录用？”“为什么人力资源师会问我一些与专业、能力根本无关的怪问题，我感觉回答得也还行，为什么最后还是被拒了？”。其实，这都是没有听懂面试“暗语”，没有听出面试官“弦外之音”的表现。“暗语”已经成为一种测试求职者心理素质、挖掘求职者内心真实想法的有效手段。理解这些面试中的暗语对于求职者而言是不可或缺的。

以下是一些常见的面试暗语，求职者一定要弄清楚其中蕴含的深意，不然很可能铩羽而归。

1）“请把简历先放在这，有消息我们会通知你的”。面试官说出这句话，表明他对你已经“兴趣不大”了。

2）“我不是人力资源的，你别拘束，咱们就当是聊天，随便聊聊”。

一般来说，能当面试官的人都是久经沙场的“老将”，所以，求职者应该时刻保持高度警觉，认真对待面试官不经意间问出来的问题，因为这些问题看似随意，很可能是他最想知道的。所以千万不要把面试过程当作聊天，当作朋友之间的闲谈，而应该仔细思考每一个问题，认真回答，切忌随意地接话和回答。

3）“是否可以谈谈你的要求和打算”。面试官在翻阅了求职者的简历后，说出这句话，很有可能是对求职者有兴趣，此时求职者应该尽量全方位地表现个人水平与才能，但也不能“王婆卖瓜”，以免引起对方的反感。

4）“面试时只是‘例行公事’式的问答”。如果面试时只是“例行公事”式的问答，没有什么激情或主观性的赞许，此时希望就很渺茫了。但如果面试官对你的专长问得很细，而且表现出一种极大的关注与热情，那么此时希望会很大，求职者一定要抓住机会，将自己最好的一面展示给面试官。

5）“你好，请坐”。一般而言，面试官说出此话，求职者回答“你好”或“您好”不重要，重要的是求职者是否“礼貌回应”和“坐不坐”。有的求职者的回应是“你好”或“您好”后直接落座，也有求职者回答“你好，谢谢”或“您好，谢谢”后落座，还有求职者一声不吭地坐下去，极个别求职者回答“谢谢”但不坐下来。前两种方法都可接受，通过问候语，可以体现一个人的基本修养，直接影响在面试官心目中的第一印象。

6）面试官向求职者探过身去。在面试的过程中，面试官会有一些肢体语言，了解这些肢体语言对于了解面试官的心理情况以及面试的进展情况非常重要。例如，当面试官向求职者探过身去时，一般表明面试官对求职者很感兴趣；当面试官打呵欠或者目光游移不定，甚至打开手机看时间或打电话、接电话时，一般表明面试官此时有了厌烦的情绪；而当面试官收拾文件或从椅子上站起来，一般表明此时面试官打算结束面试。针对面试官的肢体语言，求职者也应该给出恰当的回应：当面试官很感兴趣时，应该继续陈述自己的观点；当面试官厌烦时，此时最好停下来，询问面试官是否愿意再继续听下去；当面试官打算结束面试，领会其用意，并准备好结束语，尽快结束面试。

7）“你从哪里知道我们的招聘信息的？”面试官提出这种问题，一方面是在评估招聘渠道的有效性，另一方面是想知道求职者是否有熟人介绍。一般而言，熟人介绍总体上会有加分，但是也不全是如此。如果是一个在单位里表现不佳或者其推荐的历史记录不良的熟人介绍，则会起到相反的效果。而大多数面试官询问这一问题，主要是为了评估自己企业发布招聘广告的有效性。

8）“你念书的时间还是比较充裕的”。表面上看，这是对他人的高学历表示赞赏，但同时也是一语双关，如果“高学历”的同时还搭配上一个“高年龄”，就一定要提防面试官的质疑：如有些人因为上学晚或是工作了以后再回来读的研究生，毕业年龄明显高出平均年龄。此时一定要向面试官解释清楚，否则，面试官如果自己揣摩，往往会向不利于求职者的方向思考，如求职者年龄大的原因是高考复读过、考研用了两年甚至更长时间或者是先工作后读研等，如果面试官有了这种想法，很可能会影响最终的求职结果。

9）“你有男/女朋友吗？对异地恋爱怎么看待？”一般而言，面试官都会询问求职者的婚恋状况，一方面是对求职者个人问题的关心，另一方面，对于女性而言，绝大多数面试官不是看中求职者的美貌、温柔贤惠，也并非特意来窥探你的隐私，很有可能是在试探你是否近期要结婚生子，将会给企业带来什么程度的负担。“能不能接受异地恋”，很有可能是考察你是否能够安心在一个地方工作，或者是暗示该岗位可能需要长期出差，试探求职者如何在感情和工作上做出抉择。与此类似的问题还有，如果求职者已婚，面试官会问是否生育；如果已育，可能还会问小孩谁带？所以，如果面试官有这一层面的意思，尽量要明确表态，避免将来的麻烦。

10）“你还应聘过其他什么企业？”面试官提出这种问题是在考核你的职业生涯规划，同时顺便评估下你被其他企业录用或淘汰的可能性。当面试官对求职者提出此种问题，表明面试官对求职者是基本肯定的，只是还不能下决定是否最终录用。如果你还应聘过其他企业，则最好选择相关联的岗位或行业回答。一般而言，如果应聘过其他企业，则一定要说自己拿到了其他企业的录用通知，如果其他的行业影响力高于现在面试的企业，则无疑可以加大你自身的筹码，有时甚至可以因此拿到该企业的顶级录用通知，如果行业影响力低于现在面试的企业，若回答没有拿到录用通知，则会给面试官一种误导——连这家企业都没有给你录用通知，我们如果给你录用通知了，岂不是说明我们不如这家企业。

11）“这是我的名片，你随时可以联系我。”在面试结束，面试官起身将求职者送到门口，并主动与求职者握手，提供给求职者名片或是自己的个人电话，希望日后多加联系，此时，求职者一定要明白，面试官已经对自己非常肯定了，这是被录用的表示。

12）“你担任职务很多，时间安排得过来吗？”对于有些职位，如销售等，学校的积极分子往往更具优势，但在应聘研发类岗位时，却并不一定是这样。面试官提出此类问题，有可能认为求职者因大量的社交活动占据学业时间，从而导致专业基础不牢固。所以，针对上述问题，求职者在回答时，一定要告诉面试官，自己参与组织的“课外活动”并没有影响到自己的专业技能。

13）“如果有消息，我们会通知你的。”一般而言，面试官让求职者等通知，有多种可能性：没戏了；给你面试的人不是负责人，无最终决定权，还需要请示上级；公司对你不是特别满意，希望再多面试一些人；公司需要对一次面试通过的人进行重新选择，可能会安排二次面试。所以，当面试官说这句话时，表明此时成功的可能性不大，至少这一次不能给出肯定的回复。

14）“我们会在几天后联系你。”一般而言，面试官说出这句话，表明了面试官对求职者还是很感兴趣的，尤其是当面试官仔细询问你所能接受的薪资情况等相关情况，否则他们会尽快结束面谈。

15）面试官认为该结束面试时的暗语。一般而言，求职者自我介绍之后，面试官会相应地提出各类问题，然后转向谈工作内容。面试官先会把工作、内容、职责介绍一番，接着让求职者谈谈今后工作的打算和设想，然后，双方会谈及福利待遇问题，谈完之后，

求职者就应该主动做出告辞的姿态，不要盲目拖延时间。

面试官认为该结束面试时，往往会说以下暗示的话语来提醒求职者：

①“我很感激你对我们公司这项工作的关注”。

②“真难为你了，跑了这么多路，多谢了”。

③“谢谢你对我们招聘工作的关心，我们一旦做出决定就会立即通知你”。

④“你的情况我们已经了解。你知道，在做出最后决定之前我们还要面试几位申请人”。

此时，求职者应该主动站起身来，露出微笑，和面试官握手告辞，并且表达谢意，然后有礼貌地退出面试室。适时离场还包括不要在面试官结束谈话之前表现出浮躁不安、急欲离去或另去赴约的样子，过早地想离场会使面试官认为你应聘没有诚意或做事情没有耐心。

16）“如果让你调到其他的岗位，你愿意吗？”有些企业招收岗位和人员较多，在面试中，当听到面试官说出这句话时，言外之意是该岗位也许已经“人满为患”或“名花有主”了，但企业对你兴趣不减，还是很希望你能成为企业的一员。面对这种提问，求职者应该迅速做出回应，如果认为对方是个不错的企业，你对新的岗位又有一定的把握，也可以先进单位再选岗位；如果对方情况一般，且新岗位又不太适合自己，最好明确回绝。

17）“你能来实习吗？”对于实习这种敏感的问题，面试官一般是不会轻易提及的，除非是确实对求职者很感兴趣。当求职者遇到这种情况时，一定要清楚面试官的意图。如果确实可以去实习，求职者一定要及时地在面试官面前表达出来，为自己争取更多的机会。

18）“你什么时候能到岗？”面试官问及到岗的时间，一般是为了确定求职者是否能够及时到岗并开始工作。如果确有难题，求职者千万不要遮遮掩掩、含糊其辞，应说清楚情况、诚实守信。

针对面试中存在的这种暗语，求职者在面试过程中，应多推敲推敲面试官的深意，仔细想想其中的“潜台词”，从而为求职成功增大概率。

面试笔试真题解析篇

面试笔试技术攻克篇主要针对近 3 年以来近百家顶级 IT 企业的面试笔试真题而设计，这些企业涉及业务包括系统软件、搜索引擎、电子商务、手机 APP、安全关键软件等，面试笔试真题难易适中，覆盖面广，非常具有代表性与参考性。本篇对这些真题进行了合理地划分与归类（包括 C/C++语言基础知识，操作系统、计算机网络与通信、数学知识、数据库、系统设计题、海量数据处理等内容），并且对其进行了庖丁解牛式地分析与讲解，针对真题中涉及的部分重难点问题，本篇都进行了适当地扩展与延伸，力求对知识点的讲解清晰而不紊乱，全面而不啰嗦，使得读者能够通过本书不仅获取到求职的知识，同时更有针对性地进行求职准备，最终能够收获一份满意的工作。

第1章　C/C++语言基础知识

1.1　变量

【真题 1】 使用全局数据的有益做法很多，以下说法正确的是（　　）。

A．使用访问器子程序来存取全局数据

B．为全部的全局变量创建一份注释良好的清单

C．用全局变量来存放中间结果

D．将数据放在一个大对象中并到处传递，以满足不使用全局变量的要求

答案：A、B。

全局变量是可以被本程序所有对象或函数引用的对象。

《代码大全》一书指出，降低使用全局数据的风险通常有如下 4 种方法：①创建一种命名规则来突出全局变量；②为全部的全局变量创建一份注释良好的清单；③不要用全局变量来存放中间结果；④不要把所有的数据都放在一个大对象中并到处传递，以说明你没有使用全局变量。所以，选项 B 正确，选项 C 与选项 D 错误。

同时，根据《代码大全》的建议：避免用全局变量，不只是因为它们很危险，而是因为你可以用其他更好的方法来取代它们，如果不得不使用全局变量，那么就通过访问器来使用它们，访问器子程序能带来全局变量所能带来的一切优点，还有一些额外的好处。所以，选项 A 正确。

【真题 2】 float x 与“零值”比较的 if 语句是什么？

答案：const float EPSINON = 0.000001;

if(x<=EPSINON &&x>=－ EPSINON)

为什么浮点数与零值比较不能使用判等符“==”呢？这与浮点数在内存中的存储形式有关系。具体而言，在计算机中，浮点数不像整型数那样表示地十分准确，它通常取的是近似的值，所以，判断两个浮点数是否相等，需要判断其是否落在同一个区间，这个区间就是[－EPSINON,EPSINON]，EPSINON 一般很小，通常取 10^{-6} 以下。

【真题 3】 全局变量和局部变量有什么区别？二者是否可以重名？

答案：全局变量和局部变量是变量的两种不同类型，从分配内存空间看，全局变量、静态局部变量、静态全局变量都在静态存储区分配空间，而局部变量则在栈里分配空间。

局部变量可以与全局变量重名，但此时局部变量会屏蔽全局变量。当要使用全局变量时，需要使用作用域解析运算符“::”。需要注意的是，对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。

【真题 4】 如何引用一个已经定义过的全局变量？

答案：使用关键字 extern 修饰该全局变量或者引用头文件的方式。

如果使用 extern 方式引用，假定将那个变量写错了，那么在编译期间不会报错，而在链接期间报错。如果使用引用头文件的方式来引用某个在头文件中声明的全局变量，如果将那个变量写错了，那么在编译期间会报错。

【真题 5】 全局变量是否可以定义在被多个.c 文件包含的头文件中？

答案：可以。

在不同的.c 文件中以 static 形式来声明同名全局变量。可以在不同的.c 文件中声明同名的全局变量，前提是其中只能有一个.c 文件中对此变量赋初值，此时链接不会出错。

1.2 表达式

【真题 6】 定义有 a=1, b=2, c=3, d=4，那么表达式 a<b ? a:c < d ? a:d 的值是（ ）。

答案：1。

条件表达式的一般形式为：

x=<表达式 1>?<表达式 2>:<表达式 3>

其意义是：先求解表达式 1，若返回值为真，则求解表达式 2，将表达式 2 的值赋给 x。若返回值为假，则求解表达式 3，将表达式 3 的值赋给 x。

对于本题而言：表达式 a<b ? a:c < d ? a:d 等价于 a<b ? a:(c < d ? a:d)。如果满足条件 a<b，那么运算结果为 a，否则，运算结果为 c < d ? a:d。在本题中，由于 a<b，所以，这个表达式的运算结果为 a=1。如果将这道题目中 a 的值改为 5，由于 a<b 不成立，因此，表达式的运算结果实际上为 c < d ? a:d，由于 c<d，所以，运算结果为 a=5。

【真题 7】 已知 double d = 3.2，int n = 3，那么下列表达式中，不合法的是（ ）。

A．d<<2　　B．d/n　　C．!d && (n−3)　　D．(d−0.2)|n

答案：A、D。

按位运算只适用于字符型和整数型变量以及它们的变体，对其他数据类型不适用。本题中，由于 d 是 double 类型的变量，所以，它不适用于移位操作。所以，选项 A 与选项 D 错误。

对于选项 B，在算术运算式中，低类型能够转换为高类型，因此，在计算 d/n 的时候，会首先把 int 类型的 n 隐式转换为 double 类型，然后参与计算。所以，选项 B 正确。

对于选项 C，&&是对 bool 表达式求与运算。在选项 C 中，0 代表 false，非 0 代表 true，!d && (n−3)=false&&fase=false。所以，选项 C 正确。

【真题 8】 有如下代码：

```
#include<stdio.h>
int main()
{
    unsigned int a = 6;
    int b = −20;
    (a + b > 6) ? printf(">6") : printf("<=6");
    return 0;
}
```

程序的输出结果是（ ）。

答案：>6。

隐式类型转换分为三种，即算术转换、赋值转换和输出转换。对于其中的算术转换，指的是如果参与运算的数字有不同的类型，那么在运算的过程中，总会把低精度的类型自动转换为高精度的类型进行运算。对于本题而言，当执行 unsigned int a=6; int b=−20 语句后，在计算表达式 a+b 结果的时候，首先会把 int 型变量 b 转换为 unsigned int（无符号 int）类型（无符号 int 的值域是[0,4294967295]）的值，转换后的值为 4294967276，因此，a+b 的结果为 4294967282，该值显然大于 6，因此，输出结果为：>6。

【真题 9】 计算表达式 x^6+4x^4+2x^3+x+1 最少需要做（　　）次乘法。

A．3　　B．4　　C．5　　D．6

答案：A。

本题中，要想知道表达式需要做多少次乘法运算，就需要知道如何将这个表达式进行拆分，而表达式的拆分涉及因式分解，这又回归到了数学问题。本题中，x^2 表示的是 x*x，x^3 是由 3 个 x 相乘得到，可以用 x^2*x 表示，x^4 是由 4 个 x 相乘得到，可以用 x^2*x^2 表示，x^6 是由 6 个 x 相乘得到，可以用 x^4*x^2 来表示，所以，原式=x^3 * (x^3 + 4 * x + 2) + x + 1= x^3 * (x^3 + x+x+x+x+ 2) + x + 1，x^3 需要两次乘法，再与括号内的值相乘又需要一次乘法，所以，需要 3 次乘法。所以，选项 A　　正确。

【真题 10】 有如下代码：

```
int a=5, b=4, c=3, d=2;
if(a>b>c)
    printf("%d\n",d);
else if((c−1>=d)==1)
    printf("%d\n", d+1);
else
    printf("%d\n", d+1);
```

上述代码的输出结果为（　　）。

A．2　　B．3　　C．4　　D．编译错误

答案：B。

本题中，在第一个 if 语句中，内容为连续的比较判定，根据比较运算符>的运算优先级顺序可知，首先执行变量 a 与变量 b 的比较判断，由于变量 a 的值为 5，变量 b 的值为 4，此时满足 a>b 的判断条件，所以，返回值为 true，即为 1，而变量 c 的值为 3，显然，1<3，所以，a>b>c 这一判断条件不成立，此时 if 中语句不执行，继续后续的判断。

当执行到第二个 if 判断时，首先执行 c−1 与 d 值的判断，由于 c−1 的值为 2，d 的值为 2，二者相等，正好满足条件 c−1>=d，所以，此时，表达式 c−1>=d 的返回值为 true，即为 1，正好满足整个条件判断，所以，if 语句的条件为 true，此时，执行 printf("%d/n", d+1);语句，d 的值为 2，d+1 的值为 3，所以，程序的输出结果为 3，选项 B 正确。

1.3 数组

【真题 11】 以下叙述中正确的是（　　）。

A．对于 double 类型数组，不可以直接用数组名对数组进行整体输入或输出

B．当程序执行时，数组元素的下标超出所定义的下标范围时，系统将给出“下标越界”的出错信息

C．可以通过赋初值的方式确定数组元素的个数

D．数组名代表的是数组所占存储区的首地址，其值不可改变

答案：A、C、D。

对于选项 A，只有是字符数组可以整体输出，而对于 double 数组，不可以直接用数组名对

数组进行整体输入或输出。所以，选项 A 正确。
对于选项 B，由于 C/C++语言没有严格审查数组下标越界问题，所以，当程序执行时，如果数组的下标超出所定义的下标范围，系统也没法给出“下标越界”的出错信息。所以，选项 B 错误。
对于选项 C，可以通过赋初值的方式确定数组元素的个数。例如：int a[]={1,2,3}，对于该定义语句，可以自动确定数组 a 包含 3 个元素。所以，选项 C 正确。
对于选项 D，数组名为地址常量，其值不可改变。所以，选项 D 正确。
【真题 12】 已知一个数组 table，用一个宏定义，求出数组的元素个数。
答案：可采用宏定义的方式进行求解。定义如下：#define LENGTH (sizeof(table)/sizeof(table[0]))。
【真题 13】 下面关于数组的描述中，错误的是（ ）。
A．在 C++语言中，数组的名字就是指向该数组第一个元素的指针
B．长度为 n 的数组，下标的范围是 0～n−1
C．数组的大小必须在编译时确定
D．数组只能通过值参数和引用参数两种方式传递给函数
答案：C、D。
对于选项 A，在 C++语言中，数组的名字就是指向该数组第一个元素的指针。所以，选项 A 正确。
对于选项 B，由于数组下标通常是从 0 开始计数，而不是从 1 开始计数，所以，对于长度为 n 的数组，数组下标的范围是 0～n−1。所以，选项 B 正确。
对于选项 C，数组的大小不仅可以静态分配，即在编译的时候申请数组空间，也可以动态分配，即在编译时不能确定数组长度，而是在运行时动态分配内存空间，例如，使用 new 动态分配长度为 len*sizeof(int)的内存空间，写法如下：int *p=new int[len];。所以，选项 C 错误。
对于选项 D，数组除了可以通过值参数和引用参数两种方式传递给函数，还可以通过指针的方式传递给函数。所以，选项 D 错误。

【真题 14】 有如下定义语句：int a[][3]={{1},{3,2},{6,7,8},{9}};，那么 a[2][1]的值是（ ）。
A．3 B．6 C．2 D．7
答案：D。
对于二维数组的初始化，必须指定其列的个数。
本题中，申请并初始化了一个四行三列的二维数组，第一行中除了第一列的值为 1 以外，其余列的值都为 0，第二行前两列的值分别为 3 和 2，第三列的值为 0，第三行初始化的值分别为 6、7、8，第四行第一列的值为 9，其余两列的值为 0，如下所示：

1	0	0
3	2	0
6	7	8
9	0	0

因此，a[2][1]=7。所以，选项 D 正确。
【真题 15】 在程序设计中，当需要对两个 16K×16K 的多精度浮点数二维数组进行矩阵求和时，行优先读取和列优先读取的区别是（ ）。
A．没区别 B．行优先快
C．列优先快 D．2 种读取方式速度为随机值，无法判断

答案：B。

在 C/C++等语言中，数组的存储都是按行优先顺序存放的。行优先顺序也称为低下标优先或左边下标优先于右边下标。具体实现时，按行号从小到大的顺序，先将第一行中元素全部存放好，再存放第二行元素，然后存放第三行元素，依此类推。而与之相对的就是列优先顺序，列优先顺序也称为高下标优先或右边下标优先于左边下标。具体实现时，按列号从小到大的顺序，先将第一列中元素全部存放好，再存放第二列元素，然后存放第三列元素，依此类推。如果按列访问就需要跳过很大一串内存地址，这样可能需求的内存地址不在当前页中，需要页置换，则需要硬盘 IO，通常会较慢。所以，通常情况下，行优先顺序会比列优先顺序速度更快。所以，选项 B 正确。

【真题 16】 定义 int **a[3][4];，则变量占用的内存空间为（　　）。

答案：48。

此处定义的是指向指针的指针数组，对于 32 位系统而言，指针占用内存空间 4 个字节，因此，变量占用的总空间为 3*4*4＝48。

【真题 17】 有数组定义 int a[2][2]={{1},{2,3}};，则 a[0][1]的值为 0。

答案：正确。

【真题 18】 定义有二维数组 int A[2][3] = {1,2,3,4,5,6};，则 A[1][0]和*(*(A+1)+1)的值分别是（　　）。

A．4，5　　B．4，3　　C．3，5　　D．3，4

答案：A。

本题中，数组 A 是一个有 2 行 3 列的二维数组，数组下标从 0 开始计数。A[1][0]表示的是第 2 行第 1 列，该值为 4，*(*(A+1)+1)表示的是第 2 行第 2 列，该值为 5。所以，选项 A 正确。所以，本题的答案为 A。

【真题 19】 假定一个二维数组的定义语句为“int a[3][4]={{3,4},{2,8,6}};”，则元素 a[1][2]的值为（　　）。

A．6　　B．4　　C．2　　D．8

答案：A。

【真题 20】 以下不能正确定义二维数组的选项是（　　）。

A．int a[2][2]={{1},{2}};　　B．int a[][2]={1,2,3,4};

C．int a[2][2]={{1},2,3};　　D．int a[2][]={{1,2},{3,4}};

答案：D。

对于二维数组而言，行数可以省略，但列数一定要指定，因为编译器根据列数来进行寻址。所以，选项 D 正确。

1.4 字符串

【真题 21】给定一个指向字符串的指针 char *p_str，要把字符串中第 4 个字符的值改为‘a’，正确的做法是（　　）。

A．p_str[3]='a';　　B．*(ptr+3)= 'a';　　C．p_str[4]= 'a';　　D．*(ptr+4)= 'a';

答案：A。

通常，字符串可以被看成字符数组，因此，本题中第四个字符也等价于数组下标为 3 的值（数组下标从 0 开始）。因此，选项 C 与选项 D 错误。

对于选项 A，数组下标通常从 0 开始，p_str[3]可以用来访问字符串中第 4 个字符。因此，选项 A 正确。

对于选项 B，正确的写法为*(p_str +3)= 'a';。因此，选项 B 错误。

【真题 22】 有如下代码：

```
#include<iostream>
using namespace std;
void swap_int(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
void swap_str(char* a, char* b)
{
    char* temp = a;
    a = b;
    b = temp;
}
int main(void)
{
    int a = 10;
    int b = 5;
    char* str_a = "hello world";
    char* str_b = "world hello";
    swap_int(a, b);
    swap_str(str_a, str_b);
    printf("%d %d %s %s\n", a, b, str_a, str_b);
    return 0;
}
```

以上程序的打印结果是（　　）。

A．10 5 hello world world hello　　B．10 5 world hello hello world

C．5 10 hello world world hello　　D．5 10 hello world world hello

答案：A。

值传递是指在函数调用时，编译器会将实际参数复制一份传递到函数中，这样在函数中如果对参数进行修改，将不会影响到实际参数。本题中，由于两个交换函数都是按值传递进行，当函数执行完后，并没有改变原有变量的值。最终程序的输出结果还是原变量的值。所以，选项 A 正确。

需要注意另外一种传递参数的方式，即引用传递。所谓引用传递指的是在调用函数时将实际参数的别名传递到函数中，当在函数中对参数进行修改时，将影响到实际参数的值。

【真题 23】 有如下代码：

```
#include<stdio.h>
char *myString()
{
    char buffer[6] = { 0 };
    char *s = "Hello World!";
    for (int i = 0; i <sizeof(buffer)−1; i++)
    {
        buffer[i] = *(s + i);
    }
    return buffer;
}
int main(int argc, char **argv)
```

```
{
    printf("%s\n", myString());
    return 0;
}
```

程序的输出结果是（　　）。

A．Hello　　B．Hello World!　　C．Well　　D．以上全部不正确

答案：D。

本题中，函数 char *myString()中没有使用 new 或者 malloc 分配内存，所以 buffer 数组的内存区域在栈区，随着 char *myString()的结束，栈区内存被自动释放，字符数组也就不存在了，所以，此时会产生野指针，输出结果未知，buffer 的内存位于 myString 函数的调用栈中，函数调用结束后栈空间已经释放，打印输出已释放的栈空间，结果未知。所以，选项 D 正确。

【真题 24】 不能把字符串"HELLO!"赋给数组 b 的语句是（　　）。

A．char b[10]={'H'，'E'，'L'，'L'，'O'，'!'，'\0'};　　B．char b[10];b="HELLO!";

C．char b[10];strcpy(b，"HELLO!");　　D．char b[10]="HELLO!";

答案：B。

字符数组初始化通常有两种方法：第一种是逐个字符赋给数组中每个元素，例如选项 A 中所描述的情况，第二种是用字符串常量对整个数组进行赋值，例如选项 D 中所描述的情况。显然，第一种初始化方式比第二种方式繁琐复杂。

选项 C 采用的是字符串复制函数的方法，strcpy 函数的格式为 char *strcpy (char *s1, const char *s2)，它的作用是将字符串 s2 所指的字符串复制到 s1 所指的字符串中。需要注意的是，参数 s1 与参数 s2 都是指向字符串的指针，s1 可以是字符数组名或字符指针，但不能是字符型常量，s2 可以是字符串常量、字符数组或字符指针，而且要保证 s1 的长度足够大，以便能容纳下 s2 所指的字符串，否则会引起错误。很多读者可能有疑问了，当需要将 s2 所指的字符串复制到 s1 所指的字符串中时，为什么不能直接使用赋值语句 s1=s2 呢？那样不是更为直接吗？其实，s1 与 s2 都是指向字符数组的指针，s1=s2 会把 s2 的地址赋值给 s1，这样 s1 与 s2 就指向同一块内存空间，对字符串 s2 的任何修改都会影响字符串 s1 的值。

对于选项 B，试图将一个字符串常量赋值给数组 b，而这是不被允许的，因为 b 是一个地址常量，只能指向本身的内存空间，不可以指向字符串常量“Hello!”的首地址。以下这种写法也是不被允许的：char b[10]; b[10]="hello"，为什么呢？首先，b[10]表示的是一个字符变量，而“hello”却是一个字符串常量，一个字符怎么能容纳一个字符串？而且，数组下标是从 0 开始计数，长度为 10 的字符串数组其下标索引的范围为[0,9]，所以，b[10]是一个非法变量，本身也是不存在的。

所以，本题的答案为 B。

引申：不能使用关系运算符“==”来比较两个字符串的内容是否相等，只能用 strcmp()函数来处理。

【真题 25】字符串 www.qq.com 的所有非空子串（如果两个子串内容相同，那么只算一个）个数是（　　）。

A．1024　　B．1018　　C．55　　D．50

答案：D。

字符串中任意个连续的字符组成的子序列称为该串的子串，若字符串的长度为 n，则子串的个数就是[n+(n−1)+.....+2+1]个，但是，本题中，字符串 www.qq.com 一共有 10 个字符，由于存在着连续重复字符，所以，不能完全套用上面的公式，需要针对重复字符串逐一进行确认。

通过分析可以知道，本题中的字符串存在着如下一些可能性：
1）长度 1 的非空子串有 10−2−1−1=6 个，其中，10 表示有 10 个字符串，2 表示字符 w 有两次重复计数，两个 1 分别表示字符.和字符 q 各有一次重复计数，非空子串分别为 w、.、q、c、o、m。
2）长度 2 的非空子串有 9−1=8 个，其中，9 表示有 9 个子串，1 表示字符串 ww 有 1 次重复计数，非空子串分别为 ww、w.、.q、qq、q.、.c、co、om。
3）长度 3 的非空子串有 8 个，分别为 www、ww.、w.q、.qq、qq.、q.c、.co、com。
4）长度 4 的非空子串有 7 个，分别为 www.、ww.q、w.q、.qq、qq.、q.c、.co、com。
5）长度 5 的非空子串有 6 个，分别为 www.q、ww.qq、w.qq.、.qq.c、qq.co、q.com。
6）长度 6 的非空子串有 5 个，分别为 www.qq、ww.qq.、w.qq.c、.qq.co、qq.com。
7）长度 7 的非空子串有 4 个，分别为 www.qq.、ww.qq.c、w.qq.co、.qq.com。
8）长度 8 的非空子串有 3 个，分别为 www.qq.c、ww.qq.co、w.qq.com。
9）长度 9 的非空子串有 2 个，分别为 www.qq.co、ww.qq.com。
10）长度 10 的非空子串有 1 个，为 www.qq.com。
所有以上这些量加起来 6+8+8+7+6+5+4+3+2+1，其和为 50。所以，选项 D 是正确的。

【真题 26】 有如下程序片段，以下描述正确的是（　　）。

```
char *p = "hello";
*(p + 1) = 'w';
```

A．语法错误，编译通不过　　B．链接通不过　　C．执行错误　　D．都不对

答案：C。
本题中，p 是一个指向字符串常量的指针，本质上来说，它是该字符串的首地址，也就是说，p 指向 h，p+1 就指向下一个字符 e 了，而字符串常量是不可以更改的，因此，会出现运行时错误。所以，选项 C 正确。

【真题 27】 下面程序的输出结果是（　　）。

```
char* getCompany()
{
    char* pName = "ZTE";
    return pName;
}
char* getDepartment()
{
    char pName[10] = "3G  平台";
    return pName;
}
void getSystems(int i, char* pName)
{
    char* paucSystems[4] = { "SCS", "OSS", "BSP", "OAM" };
    pName = paucSystems[i];
    return;
}
int main(int argc, char* argv[])
{
    char* pName = "DB";
    printf("%s, ", getCompany());
    printf("%s, ", getDepartment());
    getSystems(2, pName);
    printf("%s     \n", pName);
    return 0;
```

```
}
```

答案：ZTE, 瘙 3 蹯 H? DB。

在字符串“ZTE”与字符串“DB”之间为乱码。字符串常量存储于字符串常量区，在整个程序运行完毕后才释放，因此对于 getCompany 函数而言，pName 在 main 函数退出前一直有效。而在函数 getDepartment 和 getSystems 内部定义的数组存储在这个函数对应的栈中，当函数调用结束，函数的栈就会被回收，此时在函数中申请的数组的内容是不确定的。因此这些在函数中定义的数组，在函数结束后生命周期就结束了，它们的值是不确定的。

【真题 28】 如下代码是否正确？

```
void test1()
{
    char string[10];
    char* str1 = "0123456789";
    strcpy(string, str1);
}
```

答案：程序编译正常，运行错误。

string 数组的长度为 10，但是 str1 字符串的长度却为 11（字符串以‘\0’为结束符），所以，在执行函数 strcpy 的时候会存在问题。

【真题 29】 如下代码是否正确？

```
void tet2()
{
    char string[10], str1[10];
    for (inti = 0; i<10; i++)
    {
        str1[i] = 'a';
    }
    strcpy(string, str1);
}
```

答案：strcpy 使用错误。

strcpy 只有遇到字符串末尾的‘\0’才会结束，而 str1 并没有结尾标志，导致函数 strcpy 越界访问，解决方法可以是人为设置 str1[9]='\0'。

【真题 30】 如下代码是否正确？

```
void test3(char* str1)
{
    char string[10];
    if (strlen(str1) <= 10)
    {
        strcpy(string, str1);
    }
}
```

答案：代码错误。strlen(str1)的值是不包含结尾符‘\0’的，如果 str1 刚好为 10 个字符+1 个结尾符，string 就得不到结尾符了。解决的办法是将 strlen(str1)<=10 改为 strlen(str1)<10。

【真题 31】 如下代码是否正确？

```
char* s="AAA";
printf("%s",s);
s[0]='B';
printf("%s",s);
```

答案：代码错误。“AAA”是字符串常量，s 是指针，指向这个字符串常量，因此，不能修改这个字符串常量。所以，对字符 s[0]的赋值操作是不合法的。

【真题 32】 如下代码是否正确？

```
int a[60][250][1000], i, j, k;
for (k = 0; k <= 1000; k++)
    for (j = 0; j <250; j++)
        for (i = 0; i <60; i++)
            a[i][j][k] = 0;
```

答案：循环语句内外颠倒。

解决方法是将内外语句调换顺序。因为在 C/C++语言中，数据是按行存储的，所以，顺序颠倒后就是行优先的遍历方法，会有更高的遍历效率。

【真题 33】 如下代码是否正确？

```
int   main()
{
    char a;
    char *str = &a;
    strcpy(str, "hello");
    printf(str);
    return 0;
}
```

答案：代码错误。没有为 str 分配内存空间，将会产生异常。问题出在将一个字符串复制进一个字符变量指针所指地址。虽然可以正确输出结果，但因为对内存进行越界读写而导致程序崩溃。

【真题 34】 如下代码是否正确？

```
char *my_cpy(char* src, int len)
{
    char dest[1024];
    memcpy(dest, src, len);
    return dest;
}
```

答案：代码错误。

1）数组应该初始化。

2）memcpy 没有判断是否越界，所以，调用函数前应该判断数组是否越界。

3）不应该返回 dest，因为这个数组是在函数内部申请的，属于局部变量，所以，函数结束之后就会消失，指针也会变成“野指针”，所以，会指向非法地址。

【真题 35】 有如下代码：

```
char str1[] = "abc";
char str2[] = "abc";
const char str3[] = "abc";
const char str4[] = "abc";
const char *str5 = "abc";
```

```
const char *str6 = "abc";
char *str7 = "abc";
char *str8 = "abc";
cout << (str1 == str2) << endl;
cout << (str3 == str4) << endl;
cout << (str5 == str6) << endl;
cout << (str7 == str8) << endl;
```

程序的输出结果为（　　）。

答案：0，0，1，1。

str1、str2、str3、str4 是数组变量，它们有各自的内存空间；而 str5、str6、str7、str8 是指针，它们指向相同的常量区域。

【真题 36】 有如下代码：

```
#include <stdio.h>
typedef struct object object;
struct object
{
    char data[3];
};
int main(void)
{
    object obj_array[3] = { { 'a', 'b', 'c' }, { 'd', 'e', 'f' }, { 'g', 'h', 'i' } };
    object* cur = obj_array;
    printf("%c %c\n", *(char*)((char *)(cur)+2), *(char*)(cur + 2));
    return 0;
}
```

以上程序打印的两个字符分别是（　　）。

A．c g　　B．b d　　C．g g　　D．g c

答案：A。

本题中，cur 是一个指向数组的指针，其中存储的是字符‘a’的地址，当 cur 是 object 指针时，cur+1 表示的是数组下一个元素的首地址，即字符‘d’的地址。当 cur 是 char 指针时，cur+1 是字符‘a’的下一个字符的地址，即‘b’的地址。所以，*(char*)((char *)(cur)+2)表示的是字符‘c’，*(char*)(cur+2)表示的是字符‘g’。所以，选项 A 正确。

【真题 37】 不能把字符串“HELLO!”赋给数组 b 的语句是（　　）。

A．char b[10]={'H'，'E'，'L'，'L'，'O'，'!'，'\0'};　　B．char b[10];b="HELLO!";

C．char b[10];strcpy(b，"HELLO!");　　D．char b[10]="HELLO!";

答案：B。

b 是数组的首地址，对于选项 B，b="HELLO!";改变了指针的指向，是错误的。所以，选项 B 错误。

【真题 38】 有如下代码：

```
char *ptr;
char myString[] = "abcdefg";
ptr = myString;
ptr += 5;
```

代码执行之后 ptr 指向的内容是（　　）。

A．Compiler error　　B．fg　　C．efg　　D．defg

答案：B。

本题中，当执行完语句 ptr = myString;后，ptr 指向字符串“abcdefg”的起始位置，当执行完语句 ptr += 5;后，ptr 向前移动 5 个位置，指向字符 f，所以，选项 B 正确。

【真题 39】 在审计某一开源项目的代码时，假设有下面一个 foo()子函数的实现。从安全的角度看，会存在安全漏洞吗？有的话，请①描述漏洞细节；②说明可以利用的方法；③还有该怎么修补漏洞。没有的话，也请说明为什么。

```
int foo( (void*funcp)( ) ){
    char * ptr = pointer_to_an_array;
    char  buf[128];
    gets(buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
```

提示：函数指针的缺陷、gets 函数的缺陷、foo 函数传参的问题等。

答案：1）没有判断*funcp()指针是否为空。

2）调用 get(buf)的时候，没有判断 buf 数组是否越界。因为 ptr 定义在 buf 之前，在栈上开辟内存空间，后定义的地址在低位，所以，ptr 紧跟在 buf 之后，如果 buf 越界会覆盖 ptr 指向的内容。

例如：如果 gets(buf)读取的字符串的长度超过了 128 字节，那么第 129～132 字节的内容会覆盖 ptr 的值，那么在 strncpy(ptr,buf,8)的时候，就会改变第 129～132 字节指定地址开始的 8 字节的值（假设这 8 字节是一段可执行的代码），然后第 133～136 字节的值又会覆盖 funcp 的值（假设覆盖后的值正好是之前第 129～132 字节指定地址），那么当执行(*funcp)()的时候，程序就会跳过去执行 129～132 字节中注入的代码了，会有意想不到的结果，也有非常大的安全隐患。

【真题 40】 有如下代码：

```
char s[]="\\123456\123456\t";
printf("%d\n",strlen(s));
```

程序的输出结果是（　　）。

A．12　　B．13　　C．16　　D．以上都不对

答案：A。

strlen 是一个计数器，它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描，直到碰到第一个字符串结束符‘\0’为止，然后返回计数器值(长度不包含‘\0’)。本题中，由于字符\为转义字符，\\表示字符\，\123 表示字符 S，\t 表示制表符。所以，选项 A 正确。

【真题 41】 从程序健壮性进行分析，下面的 FillUserInfo 函数和 main 函数分别存在什么问题?

```
#include<iostream>
#include<string>
#define MAX_NAME_LEN 20
struct USERINFO
{
    int nAge;
    char szName[MAX_NAME_LEN];
};
void FillUserInfo(USERINFO* parUserInfo)
{
```

```
        stu::cout <<"请输入用户的个数:";
        int nCount = 0;
        std::cin >> nCount;
        for (int i = 0; i<nCount; i++)
        {
            std::cout <<"请输入年龄:";
            std::cin >> parUserInfo[i]->nAge;
            std::string strName;
            std::cout <<"请输入姓名:";
            std::cin >> strName;
            strcpy(parUserInfo[i].szName, strName.c_str());
        }
    }
    int main(int argc, char* argv[])
    {
        USERINFO arUserInfos[100] = { 0 };
        FillUserInfo(arUserInfos);
        printf("The first name is : ");
        printf(arUserInfos[0].szName);
        printf("\n");
        return 0;
    }
```

答案：

1）在函数 FillUserInfo 中没有验证 parUserInfo 是否为空就直接使用了。

2）结构体中 szName 的长度为 20，如果名字的长度超过 20，数组就会越界会引起段错误。

3）arUserInfos 数组的大小为 100，如果在输入的用户个数超过 100，数组会越界，也会引发段错误。如果输入用户的个数小于 100 就会造成空间浪费。

4)如果输入用户的个数为 0，那么在 main 函数中输出的第一个用户名将会得到不确定的值。

5）没有判断输入的年龄是否是一个合法的值（比如负数）。

6）使用 strcpy 可能会导致 szName 数组越界，建议使用 strncpy。

7）初始化结构体数组使用的代码为：USERINFO arUserInfos[100]={0}; 一般不建议采用这种方式来初始化结构体，比较安全的方法为：ZeroMemory(arUserInfos,sizeof(arUserInfos));或者采用结构体的构造函数来初始化。

【真题 42】 有如下代码：

```
    int fun (char *s)
    {
        char *p=s;
        while(*p++)
        ;
        return p-s-1;
    }
```

该函数的功能是（　　）。

A．计算字符串的位（bit）数　　B．复制一个字符串

C．求字符串的长度　　D．求字符串存放的位置

答案：C。

本题中，首先定义了一个字符指针 p，它指向字符串 s 的首地址，然后执行一个 while 循环，该循环的条件是*p++，循环体内容为空，当 p 指向的字符为'\0'的时候，循环结束，由于执

行的是后置++操作，循环结束后 p 又执行了递增操作，也就是说 p 指向'\0'后面的一个字符，因此，p-s-1 就是字符串的长度。所以，选项 C 正确。

【真题 43】 下面程序段的运行结果是（　　）。

```
int main()
{
    char *s = "abcdefg";
    s += 2;
    fprintf(stderr, "%d\n", s);
    return 0;
}
```

A．cde　　B．字符'c'　　C．字符'c'的地址　　D．不确定

答案：C。

1.5 结构体

【真题 44】 在一个 64 位的操作系统中，定义如下结构体：

```
struct st_task
{
    uint16_t id;
    uint32_t value;
    uint64_t timestamp;
}
```

同时定义 fool 函数如下：

```
void fool()
{
    st_task task = {};
    uint64_t a = 0x00010001;
    memcpy(&task, &a, sizeof(uint64_t));
    printf("% 11u, % 11u, % 11u", task.id, task.value, task.timestamp);
}
```

上述 fool()函数的执行结果为（　　）。

A．1，0，0　　B．1，1，0　　C．0，1，1　　D．0，0，1

答案：A。

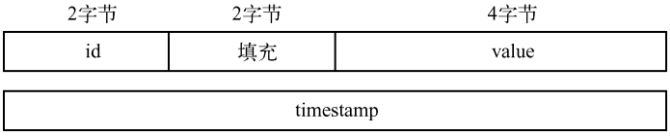
内存对齐的细节和编译器实现相关，但一般而言，满足以下三个准则：

1）结构体变量的首地址能够被其最宽基本类型成员的大小所整除。

2）结构体中每个成员相对于结构体首地址的偏移量（offset）都是成员大小的整数倍，如果有需要，编译器会在成员之间加上填充字节。

3）结构体的总大小为结构体中最宽基本类型成员大小的整数倍，如果有需要，编译器会在最末一个成员之后加上填充字节。

对于本题而言，这个结构体在内存中所占的空间如下图所示：



其中，id 占用两个字节，为了字节对齐，接下来的两个字节为填充的空白，value 占用四个字节，timestamp 占用八个字节，调用 memcpy 函数后，id 被初始化为 0001，两字节的填充地址也被初始化为 0001。而 value 与 timestamp 所占的内存被初始化为 0。因此，只有 id 的值为 1，value 与 timestamp 的值都为 0。

为了验证以上分析，在 Visual Studio 2010 下运行上述代码，其结果为 1，0，0，与选项 A 符合。所以，选项 A 正确。

【真题 45】 下述代码定义了一个结构体：

```
struct Date
{
    char a;
    int b;
    int64_t c;
    char d;
};
Date data[2][10];
```

如果 Date 的地址是 x，那么 data[1][5].c 的地址是（　　）。

A．x+195　　B．x+365　　C．x+368　　D．x+215

答案：C。

本题中，结构体成员地址对齐

a　　b　c　d

1+（3）+4+8+1+（7）= 24，（　　）内的数表示为了满足对齐填充的大小。

&data[1][5].c = x+10*24+5*24+1+(3)+4=368。 所以，选项 C 正确。

【真题 46】 C 语言和 C++语言中的 struct 有什么不同？

答案：C 语言与 C++语言中都存在 struct，C 语言和 C++语言中 struct 的主要区别是 C 语言中的 struct 不可以含有成员函数，而 C++语言中的 struct 可以含有成员函数。C++语言中 struct 和 class 的主要区别在于默认的存取权限不同，struct 默认为 public，而 class 默认为 private。

1.6 指针与引用

【真题 47】 int *p = &n;，那么*p 的值是（　　）。

A．p 的值　　B．p 的地址　　C．n 的值　　D．n 的地址

答案：C。

【真题 48】 下列 C 语言代码中，属于未定义行为的有（　　）。

A．int i=0;i=(i++);　　B．char *p="hello";p[1]= ‘E’;

C．char *p="hello";char ch=*p++;　　D．int i=0;printf("%d%d\n",i++,i--);

E．都是未定义行为　　F．都不是未定义行为

答案：B。

本题中，选项 A、选项 C 和选项 D 都能编译并且正确运行，但是选项 B 却属于未定义行为，在选项 B 中，p 是一个指向 const char 的指针，它指向的东西不能被改变，由于修改了它指向的字符，所以，选项 B 不正确。相对而言，选项 C 中没有改变 p 所指向的东西，只是修改了 p 指针，所以，选项 C 正确。

【真题 49】 有如下代码：

```
int main(int argc, char **argv)
{
    int a[4] = { 1, 2, 3, 4 };
    int *ptr = (int *)(&a + 1);
    printf("%d", *(ptr − 1));
}
```

程序的输出结果是（　　）。

A．1　　B．2　　C．3　　D．4

答案：D。

&a+1 不是首地址+1，系统会认为加了一整个 a 数组，偏移了整个数组 a 的大小（也就是 4 个 int 的大小）。所以，语句 int *ptr=(int *)(&a+1);执行完后，ptr 实际是&(a[4])，也就是 a+4。&a 是数组指针，其类型为 int(*)[4]，而指针加 1 要根据指针类型加上一定的值，不同类型的指针+1 之后增加的大小不同，a 是长度为 4 的 int 数组指针，要加 4*sizeof(int)，所以，ptr 实际是 a[4]，但是 ptr 与(&a+1)的类型是不一样的，这点非常重要，所以，ptr−1 只会减去 sizeof(int)。a 与&a 的地址是一样的，但意思就不一样了，a 是数组首地址，也就是 a[0]的地址，&a 是对象（数组）首地址，a+1 是数组下一元素的地址，即 a[1]，&a+1 是下一个对象的地址，即 a[4]。接着把&a+1 转换为指向 int 的指针，因此，ptr−1 指向 a[3]，所以，程序的输出结果为 4。

【真题 50】 在 C++语言中，有如下代码：

```
const int i = 0;
int *j = (int *)&i;
*j = 1;
printf("%d, %d", i, *j);
```

程序的输出结果是（　　）。

A．0,1　　B．1,1　　C．1,0　　D．0,0

答案：A。

本题中，对于赋值语句 const int i = 0，i 表示的是一个常量，因此，对常量 i 的值进行修改是不被允许的（例如 i=1 是不允许的），但是可以通过获取 i 的指针来修改 i 的值，即用这个指针来修改 i 的值。对于赋值语句 int *j = (int *)&i，它表示的是获取到 i 的地址给变量 j，然后执行*j=1 操作，通过该语句，实际上间接地修改了常量 i 的值。由于 i 是个常量，通常编译器在优化阶段都会把所有出现 i 的地方替换为 0，因此，语句 printf("%d, %d", i, *j)实际上等价于 printf("%d, %d", 0, *j)，输出结果为 0,1。所以，选项 A 正确。

【真题 51】 定义一个 int 类型的指针数组，数组元素个数为 10 个。以下定义方式中，正确的是（　　）。

A．int a[10];　　B．int (*a)[10];　　C．int *a[10];　　D．int (*a[10])(int);

答案：C。

在 C/C++语言中，数组指针是指向数组地址的指针，其本质为指针。指针数组指的是数组元素全为指针的数组，其本质为数组，例如 int *p[3];，定义了 p[0]、p[1]、p[2]三个指针。一维指针数组的定义形式为：“类型名 *数组标识符[数组长度]”。例如，定义一个一维指针数组：int *array[10];。

本题中，对于选项 A，表示的是 int 类型的数组，选项 B 表示的是 int 类型的数组的指针，选

项 C 表示的是 int 类型的指针的数组，选项 D 表示的是函数指针的数组，[]优先级高于*，说明 a 是一个数组，数组元素的类型为函数指针。所以，选项 A 错误。

对于选项 B，运算符“()”的优先级比运算符“[]”高，为了便于理解，在这里引入一个临时变量 tmp，那么语句 int (*a)[10];可以等价为：int tmp[10]; tmp=*a;，由此可以得出 a=&tmp。显然，a 是 int 类型的数组的指针，它指向一个包含 10 个 int 类型数据的数组，即 a 是一个数组指针。所以，选项 B 错误。

对于选项 C，a 先与“[]”结合，构成一个数组的定义，数组名为 a，int *修饰的是数组的内容，即数组的每个元素。因此，a 是一个指针数组，包含 10 个指向 int 类型数据的指针。所以，选项 C 正确。

对于选项 D，同样可以采用引入临时变量的方法来理解。对于一个函数 int f(int)，定义这个函数的一个函数指针 p 的写法为：int (*p)(int);，p 就是一个函数的指针，把 p 改成 a[10]，则说明定义了一个大小为 10 的数组，数组中存放的是返回值 int 且参数为 int 的函数指针。所以，选项 D 错误。

【真题 52】 若已定义：int a[9], *p = a;，并在以后的语句中未改变 p 的值，不能表示 a[1]地址的表达式是（ ）。

A．++p　　B．a+1　　C．p+1　　D．a++

答案： D。

a 和 p 都指向数组的首元素 a[0]，显然，a+1 与 p+1 都指向 a[1]。因此，选项 B 和选项 C 正确。

对于选项 A，++p 是首先执行++操作，它的结果也就等价于 p+1，也指向 a[1]。因此，选项 A 正确。

对于选项 D，为了便于理解，可以引入一个临时变量 int* tmp=a++;，这行代码执行的结果为：tmp 仍然指向 a[0]，当这行代码结束运行后才执行 a 的自增操作，此时 a 才指向 a[1]。由此可见，a++的值仍然是指向 a[0]的指针。因此，选项 D 错误。

【真题 53】 下面程序的运行结果是（ ）。

```
#include<stdio.h>
int main()
{
    int a[5] = { 1, 2, 3, 4, 5 };
    int *p, **k;
    p = a;
    k = &p;
    printf("%d ", *(p++));
    printf("%d", **k);
    return 0;
}
```

A．11　　B．21　　C．22　　D．12

答案：D。

本题中，p 是指向数组 a 的指针，k 是指向指针 p 的指针。第一次 printf 的时候由于 p++是后置++操作，因此，首先输出*p 的值 1，接着完成 p 的自增操作，因此，p 指向数组的第 2 个元素，此时：*k 的值即为 p 的值，而**k 的值即为*p 的值，也就是数组元素 a[1]的值了。所以，选项 D 正确。

引申：有如下代码：

```
#include<stdio.h>
#include<stdlib.h>
```

```
int main()
{
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, }, k, *p = a + 5, *q = &k;
    *q = *(p + 5);
    printf("%d %d\n", *p, *q);
    return 0;
}
```

程序的输出是（　　）。

答案：6，11。

*p=a+5 表示的是指针 p 指向数组 a 的第六个元素，即*p 值为 6。*q=&k，则 q 是 k 的别名，和 k 是同一个值。*q = *(p+5)，则*q 的值是从 p 开始再往后 5 个，即*q 值为 11，此时 k 的值也会被修改为 11。

【真题 54】 有以下程序：

```
void main()
{
    char *pn[] = { "abcd", "efghe", "ijklij", "mnopmno"};
    char **p;
    p = pn;
    printf("%c\n", *(*(p + 2) + 1));
}
```

程序运行后的输出结果是（　　）。

A．f　　B．g　　C．j　　D．k

答案：C。

本题定义了一个字符串数组 pn，同时定义了一个二级指针 p，通过指针 p 来访问字符串数组 pn 的值。p+2 表示访问的是 pn 中第三个字符串的首地址，然后再加 1，表示访问的是第三个字符串的第二个字符，该值为字符 j。（注意，数组下标都是从 0 开始）

【真题 55】 设计如下定义：int (*ptr)()，则以下叙述中不正确的是（　　）。

A．ptr 是指向一维数组的指针变量

B．ptr 是指向 int 型数据的指针变量

C．ptr 是指向函数的指针，该函数返回一个 int 型数据

D．ptr 是一个函数名，该函数的返回值是指向 int 型数据的指针

答案：A、B、D。

本题中，“int(*ptr)()”表明 ptr 是一个指向函数的指针，该函数的返回值是整型变量。*ptr 两边的括号非常关键，表示*和 ptr 组合成一个指针变量，然后再和后边的圆括号结合，表示该指针变量*ptr 是一个指向函数的指针变量。如果没有*ptr 两边的括号，“int* ptr()”则表示 ptr 是个函数，该函数返回指向整型数据的指针。所以，选项 A、选项 B、选项 D 正确。

【真题 56】 有如下代码：

```
short a[5];
short *p;
p =a;
```

如果要访问 a[1]的地址，下面写法正确的是（　　）。

A．++a　　B．a+1　　C．p+1　　D．++p

答案：B、C、D。

本题中，a 是数组名，p 是一个指针，a+1、p+1、++p 都表示的是 a[1]，只有++a 不合法，所

以，选项 B、选项 C、选项 D 都是正确的，只有选项 A 是错误的。

【真题 57】 对 C 语言类型声明语句“double *const cptr; ”的正确解释是 cptr 指向的地址不能改变。（ ）。

答案：正确。const 在 cptr 的前面用来修饰 cptr，是不可变量，cptr 是一个指向 double 类型的指针，因此，cptr 指向的地址不能改变。

【真题 58】 有如下代码：

```
int a[3];
a[0] = 0; a[1] = 1; a[2] = 2;
int *p, *q;
p = a;
q = &a[2];
```

则 a[q－p] =（ ）。

答案：2。

a[q−p]=a[2]=2。

【真题 59】 有一个 32 位的机器，则该机器的指针是多少位？

答案：要想知道指针是多少位，只要看地址总线的位数就行了。以 80386 以后的芯片为例，它们都是 32 位的数据总线，所以，指针的位数就是 4 个字节了。

【真题 60】 int (*s[10])(int) 表示的是什么？

答案：函数指针数组，每个指针指向一个 int func(int param)的函数。

【真题 61】 (void *)ptr 和 (*(void**))ptr 的结果是否相同？其中 ptr 为同一个指针。

答案：(void *)ptr 和 (*(void**))ptr 值是相同的。

【真题 62】 要对绝对地址 0x100000 赋值，可以用 (unsigned int*)0x100000 = 1234，那么要是想让程序跳转到绝对地址 0x100000 去执行，应该怎么做？

答案：*((void (*)( ))0x100000) ( );。首先要将 0x100000 强制转换成函数指针，即：(void (*)())0x100000，然后再调用它：*((void (*)())0x100000)();，用 typedef 可以看得更直观些：typedef void(*)() voidFuncPtr; *((voidFuncPtr)0x100000)();。

【真题 63】 unsigned char *p1; unsigned long *p2; p1=(unsigned char *)0x801000; p2=(unsigned long *)0x810000; 请问 p1+5=（ ），p2+5=（ ）。

答案：801005； 810014。

这个是十六进制的数字，p2 要加 20 变为十六进制就是 14。

【真题 64】 下列用法错误的是（ ）。

A．typedef void (*FUN)(); B．const int &ra;

C．extern const int array[256]; D．int* a;

答案：B。

对于选项 A，typedef void(*fun)(); 表示定义了一个函数指针的类型。如下例所示：

```
#include<iostream>
using namespace std;
void testF()
{
    cout<<"hello world"<<endl;
}
typedef void(*fun)();          //定义了一个函数指针类型
int main()
{
    fun f=testF;               //定义一个 fun 类型的变量 f，并初始化为 testF
    f();                       //调用函数 testF
    return 0;
}
```

程序的运行结果为：

hello World

因此，选项 A 正确。

对于选项 B，引用变量在定义的时候必须初始化，不能单独存在。而这个选项中定义了一个引用变量 ra，却没有初始化，因此，选项 B 错误。

对于选项 C，extern 表示声明的函数或变量可以在本模块或者其他模块中使用（只是声明，不是定义）。如果另外一个模块要引用这个变量，只需包含头文件即可。const 表示常量。extern const int array[256]声明了一个大小为 256 的常量数组。因此，选项 C 正确。

对于选项 D，定义一个整型指针变量 a。所以，选项 D 正确。

【真题 65】 假设指针变量 p 定义为：“int *p = new int[100]”，要释放指针 p 所指向的动态内存，应使用语句（　　）。

A．delete p　　B．delete *p　　C．delete &p　　D．delete[] p

答案：D。

operator new 和 operator delete 函数有两个重载版本，每个版本支持相关的 new 表达式和 delete 表达式：

1）void* operator new (size_t);　　// 分配一个对象

2）void* operator new [] (size_t);　　// 分配一个数组

3）void operator delete (void*);　　// 释放一个对象

4）void operator delete [] (void*);　　// 释放一个数组

对于 new 分配的单个对象的内存空间，回收的时候使用 delete，对于 new[]分配的一组对象的内存空间，回收的时候使用 delete[]。本题中，通过 new 分配了一个长度为 100 的 int 型数组，所以，需要使用 delete []p 释放指针 p 所指向的动态内存。因此，选项 D 正确。

在此，展开讲一个问题，关于 malloc/free 与 new/delete 的区别。malloc/free 通常应用于 C 语言中的内存分配与回收，而 new/delete 通常应用于 C++语言的内存分配与回收。相比较 malloc，new 内置了 sizeof、类型转换和类型安全检查功能，对于非内部数据类型的对象而言，new 在创建动态对象的同时完成了初始化工作，如果对象有多个构造函数，那么 new 的语句也可以有多种形式。所以，选项 D 正确。

【真题 66】 int (*ptr) ()，则 ptr 是一维数组的名字。

答案：错误。语句 int (*ptr)()定义了一个指向函数的指针变量。

【真题 67】 指针在任何情况下都可进行>，<，>=，<=，==运算。

答案：错误。不同类型的指针是不能比较的。

【真题 68】 定义有数组 int a[10]，下面不可以表示 a[1] 的地址的是（　　）。

A．a+sizeof(int)　　B．&a[0]+1　　C．(int*)&a+1　　D．(int*)((char*)&a+sizeof(int))

答案：A。

对于选项 A，在 32 位机器上，sizeof(int)的值为 4，而 a+sizeof(int) 相当于指针运算 a+4，其中，a 表示的是数组首地址，a+4 表示的是 a[4]的地址，而不是 a[1] 的地址。所以，选项 A 错误。

对于选项 B，&a[0]表示的是取数组首元素地址，而&a[0]+1 表示的是数组首元素地址加 1，即 a+1，根据指针运算可知，即表示的是 a[1]的地址。所以，选项 B 正确。

对于选项 C，(int*)&a 表示数组地址 a 被强制类型转换为 int*，然后再加 1，与&a[0]+1 等价。所以，选项 C 正确。

对于选项 D，sizeof(int)=4，数据地址 a 首先被转换为 char*类型，然后再加 4，根据指针运算公式可知，向前移动 4*sizeof(char)，最后整个结果被转换为 int*类型，此时表示的正好是 a[1]的地址。所以，选项 D 正确。

【真题 69】 int (*s[10])(int) 表示的是（　　）。

A．指针数组，每个指针指向长度为 1 的 int 数组

B．指针数组，每个指针指向长度为 10 的 int 数组

C．函数指针数组，每个指针指向一个 int func(int* param)的函数

D．函数指针数组，每个指针指向一个 int func(int param)的函数

答案：D。

【真题 70】 有如下代码：

```
char fun(char *);
main()
{
    char *s = "one", a[5] = {0}, (*f1)(char *) = fun, ch;
}
```

则对函数 fun 的调用语句正确的是（　　）。

A．*f1(&a);　　B．f1(*s);　　C．f1(&ch)　　D．ch = *f1(s);要改成(*f1)(s)才正确

答案：C。

在计算机中，每一个函数都占用一段内存单元，它们有一个起始地址，指向函数入口地址的指针称为函数指针。

指向函数的指针变量的一般定义形式为：数据类型 (*指针变量名)(参数表);

在使用函数指针的时候，通常需要注意以下内容：

1）函数指针的定义形式中的数据类型是指函数的返回值的类型。

2）区分下面两个语句：

```
int (*p)(int a, int b);    //p 是一个指向函数的指针变量，所指函数的返回值类型为整型
int *p(int a, int b);      //p 是一个函数名，此函数的返回值类型为整型指针
```

3）指向函数的指针变量不是固定指向哪一个函数的，而只是表示定义了一个这种类型的变量，它是专门用来存放函数的入口地址的；在程序中把哪一个函数的地址赋给它，它就指向哪一个函数。

4）在给函数指针变量赋值时，只需要给出函数名，而不必给出参数。

例如函数 max 的原型为：int max(int x, int y);，指针 p 的定义为：int (*p)(int a, int b); ，则语句 p = max;的作用是将函数 max 的入口地址赋给指针变量 p。此时，p 就是指向函数 max 的指针变量，也就是说，p 和 max 都指向函数的开头。

5）在一个程序中，指针变量 p 可以先后指向不同的函数，但一个函数不能赋给一个不一致的函数指针（即不能让一个函数指针指向与其类型不一致的函数）。

有如下函数：int fn1(int x, int y); int fn2(int x);，定义如下的函数指针：int (*p1)(int a, int b); int (*p2)(int a); 则：

```
p1 = fn1;    //正确，p2 = fn2;
p1 = fn2;    //产生编译错误
```

6）当定义了一个函数指针并让它指向了一个函数后，对函数的调用就可以通过函数名实现，也可以通过函数指针实现（即用指向函数的指针变量调用）。

例如语句：c = (*p)(a, b);，表示调用由 p 指向的函数（max），实参为 a 与 b，当函数调用结束后，再把得到的函数值赋给 c。

7）函数指针只能指向函数的入口处，而不可能指向函数中间的某一条指令。不能用*(p+1)来表示函数的下一条指令。

8）函数指针变量常用的用途之一是把指针作为参数传递到其他函数。

调用函数指针通常的写法为：(*指针变量)(参数表)。C/C++也支持缩写，即省去*：指针变量 (参数表)

就本题而言，对于选项 A，&a 的类型为指向有 5 个元素的 char 类型的数组的指针，与 char* 类型不匹配；另外，由于()的优先级比*高，这种写法的意思如下：对函数调用的结果做*运算。因此，选项 A 错误。

对于选项 B，*s 的类型为 char，而函数参数为 char*，所以，类型不匹配。因此，选项 B 错误。

对于选项 C，ch 的类型为 char，所以，&char 的类型为 char*，类型匹配。因此，选项 C 正确。

对于选项 D，如果改成 f1 (s)也是正确的，所以，这个描述不是非常精确。因此，选项 D 错误。

【真题 71】 引用与指针有什么区别？

答案：程序设计中的引用其实就是别名的意思，它用于定义一个变量来共享另一个变量的内存空间，变量是一个内存空间的名字，如果给内存空间起另外一个名字，那么就能够共享这个内存了，进而提高程序的开发效率。指针是指向另一个内存空间的变量，可以通过它来索引另一个内存空间的内容，需要注意的是，指针本身也有自己的内存空间。

通过上面的定义不难发现，引用与指针有着相同的地方，即指针指向一块内存，它的内容是所指内存的地址，引用是某块内存的别名。但是，二者并非完全相同，它们之间也存在着差别，具体表现在如下几个方面的内容：

1）从本质上讲，指针是存放变量地址的一个变量，在逻辑上是独立的，它可以被改变，即其所指向的地址可以被改变，其指向的地址中所存放的数据也可以被改变。而引用则只是一个别名而已，它在逻辑上不是独立的，它的存在具有依附性，所以，引用必须在一开始就被初始化，而且其引用的对象在其整个生命周期中是不能被改变的，即自始至终只能依附于同一个变量，具有“从一而终”的特性。

2）作为参数传递时，二者意义不同。在 C++语言中，指针与引用都可以用于函数的参数传递，但是指针传递参数和引用传递参数有着本质的不同。指针传递参数本质上是值传递的方式，它所传递的是一个地址值。值传递过程中，被调函数的形式参数作为被调函数的局部变

量处理，即在栈中开辟了内存空间以存放由主调函数放进来的实参的值，从而成为了实参的一个副本。值传递的特点是被调函数对形式参数的任何操作都是作为局部变量进行，不会影响主调函数的实参变量的值。而在引用传递过程中，被调函数的形式参数虽然也作为局部变量在栈中开辟了内存空间，但是这时存放的是由主调函数放进来的实参变量的地址。被调函数对形参的任何操作都被处理成间接寻址，即通过栈中存放的地址访问主调函数中的实参变量。正因为如此，被调函数对形参做的任何操作都影响了主调函数中的实参变量。虽然它们都是在被调函数栈空间上的一个局部变量，但是任何对于引用参数的处理都会通过一个间接寻址的方式操作到主调函数中的相关变量。而对于指针传递的参数，如果改变被调函数中的指针地址，它将影响不到主调函数的相关变量。如果想通过指针参数传递来改变主调函数中的相关变量，那就得使用指向指针的指针，或者指针引用。

3）使用引用时，不需要解引用（*），而指针需要解引用。

4）引用只能在定义时被初始化一次，之后不能被改变，即引用具有“从一而终”的特性。而指针却是可变的，指针的初始化不是指指针的定义，定义只是分配一个地址用来存储这个指针的值，而初始化是用来初始化这个指针的值。例如定义 float *a，会为 a 分配一个地址，初始值是随机的值，初始化可以将指针 a 赋值为 NULL，即 a = NULL，这样在以后的程序中可以增加 if(a == NULL)来判断指针是否有效，否则不行。因此，一般建议在定义指针的时候给指针初始化一个默认的值，例如 float*a = new float;，或者 float b; float *a = &b;，都可以使指针指向一块内存以实现初始化。

5）引用不可以为空，而指针可以为空。引用必须与存储单元相对应，一个引用对应一个存储单元。

6）对引用进行 sizeof 操作得到的是所指向的变量（对象）的大小，而对指针进行 sizeof 操作得到的是指针本身（所指向的变量或对象的地址）的大小，typeid(T) == typeid(T&) 一直为真，sizeof(T) == sizeof(T&)一直为真，但是当引用作为成员时，其占用空间与指针相同。

7）指针和引用的自增（++）运算意义不一样。

8）如果返回动态分配的对象或内存，必须使用指针，引用可能会引起内存泄露。

由于引用与指针的区别，所以，并非所有使用指针的地方都可以使用引用，也并非所有使用引用的地方都可以使用指针，二者的使用也有其特定的环境。以如下代码为例分析。

1）int *a; int * & p=a; int b=8; p=&b;　　//正确，指针变量的引用

```
void & a=3;                 //不正确，没有变量或对象的类型是 void
int & ri=NULL;              //不正确，有空指针，无空引用
```

2）int & ra=int;　　//不正确，不能用类型来初始化

```
int *p=new int; int & r=*p;  //正确
```

3）引用不同于一般变量，下面类型声明是非法的：

```
int &b[3];      //不能建立引用数组
int & *p;       //不能建立指向引用的指针
int &&r;        //不能建立引用的引用
```

4）当使用取址运算符&取一个引用的地址时，其值为所引用变量的地址。

通过上面的实例可以发现，引用与指针都有其特定的使用场景，所以，该使用指针的时候就使用指针，该使用引用的时候就使用引用，切不可混淆。

【真题 72】 引用可以是 void 类型吗？

答案：不可以。

1.7 预处理

【真题 73】 什么是预编译？

答案：预编译又称为预处理，主要工作是执行代码/文本的替换工作，即处理以#开头的指令，例如拷贝#include 包含的文件代码，#define 宏定义的替换，条件编译等。

以下两种情况下需要使用预编译：

1）总是使用不经常改动的大型代码体。

2）程序由多个模块组成，所有模块都使用一组标准的包含文件和相同的编译选项。在这种情况下，可以将所有包含文件预编译为一个预编译头。

【真题 74】 头文件中的 ifndef/define/endif 的作用是什么？

答案：ifndef/define/endif 为预处理命令，作用是防止头文件被重复引用。

【真题 75】 有如下代码：

```
#define add(a,b) a+b
int main()
{
    printf("%d\n", 5 * add(3, 4));
    return 0;
}
```

程序的输出结果是（　　）。

A．23　　B．35　　C．16　　D．19

答案：D。

宏是一种语法替换，用于说明某一特定输入（通常是字符串）如何根据预定义的规则转换成对应的输出（通常也是字符串），使用带参数的宏只是进行简单的字符替换。通常，使用带参数的宏有以下几个特点：

1）程序运行可能会稍微快些。一个函数调用在执行时通常会有些额外开销——存储上下文信息、复制参数的值等，而一个宏的调用则没有这些运行开销。

2）通用。与函数的参数不同，宏的参数没有类型。因此，只要预处理后的程序依然合法，宏就可以接受任何类型的参数。例如，可以使用 MAX 宏从两个数中选出较大的一个，数的类型可以是：int、long int、float、double 等。

3）编译后的代码通常会变大。每一处宏调用都会导致插入宏的替换列表，由此导致程序源代码增加（因此，编译后的代码数量变大）。宏被使用得越频繁，这种效果就越明显。当宏调用嵌套时，这个问题会相互叠加从而使程序变得更加复杂。

本题中，当调用函数 add(3,4)时，会把其替换成 3+4，因此，5*add(3,4)=5*3+4=19。正因为如此，一般在宏定义的时候最好通过增加括号的方式来避免产生不期望的结果。如果本题中把宏定义改为：#define add(a,b) (a+b)，那么上述程序的运行结果为：5*(3+4)=35。

引申：含参数的宏与函数有什么区别？

含参数的宏有时完成的是函数实现的功能，但是并非所有的函数都可以被含参数的宏所替

代，具体而言，含参数的宏与函数的优缺点如下：
1）函数调用时，首先求出实参表达式的值，然后带入形参。而使用带参的宏只是进行简单的字符替换。
2）函数调用是在程序运行时处理的，它需要分配临时的内存单元；而宏展开则是在编译时进行的，在展开时并不分配内存单元，也不进行值的传递处理，也没有“返回值”的概念。
3）对函数中的实参和形参都要定义类型，二者的类型要求一致，如果不一致，应进行类型转换；而宏不存在类型问题，宏名无类型，它的参数也无类型，只是一个符号代表，展开时带入指定的字符即可。宏定义时，字符串可以是任何类型的数据。
4）调用函数只可得到一个返回值，而用宏可以设法得到几个结果。
5）使用宏次数多时，宏展开后源程序会变得很长，因为每展开一次都使程序内容增长，而函数调用不使源程序变长。
6）宏替换不占用运行时间，而函数调用则占运行时间（分配单元、保留现场、值传递、返回）。
7）参数每次用于宏定义时，它们都将重新求值，由于多次求值，具有副作用的参数可能会产生不可预料的结果。而参数在函数被调用前只求值一次，在函数中多次使用参数并不会导致多种求值过程，参数的副作用并不会造成任何特殊的问题。
一般来说，用宏来代表简短的表达式比较合适。

【真题 76】 有如下代码：

```
#define INT_PTR int *
INT_PTR p1,p2;
int a,b;
```

那么，下面代码中，在编译时可能产生 error 或 warning 的是（　　）。

A．p1 = &a;　　B．p2 = &b;　　C．A 和 B 均可　　D．A 和 B 均不可

答案：B。

由于宏定义进行的是文本的替换，本题的原意是希望定义两个指针变量 p1 与 p2，但是由于采用的是宏定义的方式，使得在替换后，在进行类型匹配的时候，原语句 INT_PTR p1,p2 被替换为了 int *p1,p2，显然，只有 p1 为 int*类型，而 p2 则是 int 类型。所以，当执行 p2 = &b 语句时，会报 warning（警告）。所以，选项 B 正确。

【真题 77】 求数 x 的平方，宏定义可以写为：#define SQR(x) (x)*(x)。（　　）。

答案：正确。

【真题 78】 写一个“标准”宏，这个宏输入两个参数并返回较小的一个。

答案：#define Min(X, Y) ((X)>(Y)?(Y):(X))。

【真题 79】 对于一个频繁使用的短小函数，在 C 语言中应使用什么实现？在 C++语言中应使用什么实现？

答案：在 C 语言中使用宏定义，在 C++语言中使用 inline。

【真题 80】 typedef 和 define 有什么区别？

答：typedef 与 define 都是替一个对象取一个别名，以此来增强程序的可读性，但是它们在使用和作用上也存在着以下几个方面的不同。

（1）原理不同

#define 是 C 语言中定义的语法，它是预处理指令，在预处理时进行简单而机械地字符串替换工作，不作正确性检查，不管含义是否正确，照样带入，只有在编译已被展开的源程序时，才会发现可能的错误并报错。

例如#define PI 3.1415926

当程序中执行 area=PI*r*r 语句时，PI 会被替换为 3.1415926，于是该语句被替换为 area=3.1415926*r*r，如果把#define 语句中的数字 9 写成了 g，预处理也照样带入，而不去检查其是否合理、合法。

typedef 是关键字，它在编译时处理，所以，typedef 有类型检查的功能。它在自己的作用域内为一个已经存在的类型定义别名，但是不能在一个函数定义里面使用标识符 typedef。例如：typedef int INTEGER，这以后就可以用 INTEGER 来代替 int 作整型变量的类型说明了，例如：

INTEGER a, b;

用 typedef 定义数组、指针、结构等类型将带来很大的方便，不仅使程序书写简单，而且使意义更为明确，因而增强了可读性。例如：

typedef int a[10];

表示 a 是整型数组类型，数组长度为 10。然后就可用 a 说明变量，例如：a s1,s2;完全等效于 int s1[10],s2[10]。同理，typedef void (*p)(void)表示 p 是一种指向 void 型的指针类型。

（2）功能不同

typedef 用来定义类型的别名，这些类型不只包含内部类型（例如 int、char 等），还包括自定义类型（例如 struct），可以起到使类型易于记忆的功能。

例如：typedef int (*PF) (const char *, const char *);

定义一个指向函数的指针的数据类型 PF，其中，函数返回值为 int，参数为 const char *。

typedef 还有另外一个重要的用途，那就是定义机器无关的类型，例如，可以定义一个叫 REAL 的浮点类型，在目标机器上它可以获得最高的精度：typedef long double REAL，在不支持 long double 的机器上，该 typedef 看起来会是下面这样：typedef double REAL，在连 double 都不支持的机器上，该 typedef 看起来会是这样：typedef float REAL。

#define 不只是可以为类型取别名，还可以定义常量、变量和编译开关等。

（3）作用域不同

#define 没有作用域的限制，只要是之前预定义过的宏，在以后的程序中都可以使用，而 typedef 有自己的作用域。

程序示例如下：

```
void    fun()
{
    #define   A    int
}
void    gun()
{
    //在这里也可以使用 A，因为宏替换没有作用域，但如果上面用的是 typedef，那这里就不能用 A，不过一般不在函数内使用 typedef
}
```

（4）对指针的操作不同

二者修饰指针类型时，作用不同。

```
#define INTPTR1 int*
typedef int* INTPTR2;
INTPTR1 p1,p2;
INTPTR2 p3,p4;
```

语句 INTPTR1 p1,p2;和语句 INTPTR2 p3,p4;的效果是截然不同的。语句 INTPTR1 p1,p2;进

行字符串替换后变成 int* p1,p2;，要表达的意义是声明一个指针变量 p1 和一个整型变量 p2，而对于语句 INTPTR2 p3,p4;，由于 INTPTR2 是具有含义的，告诉我们是一个指向整型数据的指针，那么 p3 和 p4 都为指针变量，这句相当于 int* p1,*p2;，从这里可以看出，进行宏替换是不含任何意义的替换，仅仅为字符串替换，而用 typedef 为一种数据类型起的别名是带有一定含义的。

以如下代码为例：

```
#define INTPTR1 int*
typedef int* INTPTR2;
int a=1;
int b=2;
int c=3;
const INTPTR1 p1=&a;
const INTPTR2 P2=&b;
INTPTR2 const p3=&c;
```

上述代码中，const INTPTR1 p1 表示 p1 是一个常量指针，即不可以通过 p1 去修改 p1 指向的内容，但是 p1 可以指向其他内容；而对于 const INTPTR2 p2，由于 INTPTR2 表示是一个指针类型，因此，使用 const 去限定，表示封锁了这个指针类型，因此，p2 是一个指针常量，不可使 p2 再指向其他的内容，但可以通过 p2 修改其当前指向的内容，语句 INTPTR2 const p3 同样声明的是一个指针常量。

【真题 81】 有如下代码：

```
#define ADD(x,y) x+y
int m=3;
m+=m*ADD(m,m);
```

则 m 的值为（　　）。

A．15　　B．12　　C．18　　D．58

答案：A。

宏定义过程执行的是文本替换的工作，所以，对语句 m+=m*ADD(m,m)进行文本替换后，变为 m+=m*m+m（注意没有括号），即 m=m+(m*m+m)，由于 m 的初始值为 3，所以，m=3+(3*3+3)=15。所以，选项 A 正确。

【真题 82】 有如下代码：

```
#define    M(x,y,z)    x*y+z
main()
{
    int a=1, b=2, c=3;
    printf("%d/n",M(a+b,b+c,c+a));
}
```

以上程序的输出结果是（　　）。

A．19　　B．17　　C．15　　D．12

答案：D。

在 C/C++语言中，宏定义的规则是直接进行文本的替换。本题中，由于采用宏定义的方式定义了 M(x,y,z)，而 M(x,y,z)展开后的表达式为 x*y+z，所以，M(a+b,b+c,c+a)经过宏替换之后的表达式是 a+b*b+c+c+a（需要注意的是，本题的宏定义中没有括号，如果宏定义中有括号，此时需要将括号带入），因为变量 a 的值为 1，变量 b 的值为 2，变量 c 的值为 3，所以，将各变量带入到表达式以后，表达式 a+b*b+c+c+a=1+2*2+3+3+1=12。所以，选项 D 正确。

【真题 83】 下列对于宏的描述中，不正确的是（　　）。

A．宏会带来性能的缺失　　B．宏不进行类型检查
C．宏可以做到函数无法做到的功能　　D．编译时宏的处理早于函数
答案：A。
宏的作用是在编译的时候将对应函数用宏命令替换，在编译阶段不进行类型安全性检查，而且，这一过程对程序性能无影响。所以，选项 A 错误。

【真题 84】 以下关于头文件的描述中，正确的是（　　）。
A．#include<filename.h>，编译器寻找头文件时，会从当前编译的源文件所在的目录去找
B．#include“filename.h”，编译器寻找头文件时，会通过编译选项指定的目录去找
C．多个源文件同时用到的全局整数变量，它的声明和定义都放在头文件中，是好的编程习惯
D．在大型项目开发中，把所有自定义的数据类型、全局变量、函数声明都放在一个头文件中，各个源文件都只需要包含这个头文件即可，省去了要写很多#include 语句的麻烦，是好的编程习惯
答案：B。
对于选项 A，对于#include <filename.h>引用形式，编译器先从标准库路径开始搜索，然后再从本地目录搜索。所以，选项 A 错误。
对于选项 B，对于#include “filename.h”引用形式，编译器先从用户的工作目录开始搜索（用户的工作目录是通过编译器指定的），然后再去系统路径寻找。所以，选项 B 正确。
对于选项 C，在 C/C++语言中，头文件只能存放全局变量的声明，其定义只能放在.c、.cpp 文件中。所以，选项 C 错误。
对于选项 D，一般而言，在开发大型项目的时候，会把不同的声明放在不同的头文件中。所以，选项 D 错误。
【真题 85】 以下说法正确的是（　　）。
A．头文件中的 ifndef/define/endif 是为了防止该头文件被重复引用
B．对于#include<filename.h>，编译器从标准库路径开始搜索 filename.h，对于#include “filename.h”，编译器从用户的工作路径开始搜索 filename.h
C．C++语言支持函数重载，C 语言不支持函数重载
D．fopen 函数只是把文件目录信息调入内存
答案：A、B、C。
对于选项 A，头文件中的 ifndef/define/endif 的作用是为了防止该头文件被重复引用。所以，选项 A 正确。
对于选项 B，当使用#include<filename.h>包含方式时，编译器从标准库路径开始搜索 filename.h，当使用#include“filename.h”包含方式时，编译器从用户的工作路径开始搜索 filename.h。所以，选项 B 正确。
对于选项 C，C++语言支持函数重载，而 C 语言不支持函数重载。函数被 C++编译后，在库中的名字与 C 语言是不同的。假设某个函数的原型为：void foo(int x, int y)，那么该函数被 C 编译器编译后，在库中的名字为_foo，而 C++编译器则会产生像_foo_int_int 之类的名字，通常，C++语言提供了 C 连接交换指定符号 extern “C”来解决名字匹配问题。所以，选项 C 正确。
对于选项 D，fopen 函数用来打开一个文件，它只是建立了文件对象和进程的关联关系，而这个关联关系就是文件描述符。所以，选项 D 错误。

【真题 86】 下列关于#include 命令的描述中，正确的是（　　）。
A．一个#include 命令中只能指定一个被包含文件
B．一个#include 命令可以指定包含多个被包含文件
C．一个被包含的文件又可以包含另一个被包含文件
D．#include 命令中，文件名可以用双引号或尖括号括起来
答案：A、C、D。
对于选项 A 与选项 B，一个 include 命令每次只能包含进来一个文件。所以，选项 A 正确，选项 B 错误。
对于选项 C，一个被包含的文件可以通过 include 包含另一个文件。所以，选项 C 正确。
对于选项 D，在#include 命令中，文件名可以用双引号或尖括号括起来。所以，选项 D 正确。
【真题 87】 #include 命令的功能是（　　）。
A．在命令处插入一个头文件　　B．在文件首部插入一个头文件
C．在命令处插入一个文本文件　　D．在文件首部插入一个程序文件
答案：C。
#include 的原理其实很简单，当预处理器发现#include 后，就会寻找指令后面<>或“”中的文件名，并把这个文件的内容包含到当前的文件中，被包含的文件中的文本将替换源代码文件中的#include 指令。
所以，选项 C 正确。

1.8 循环

【真题 88】 下面对于循环知识的描述中，正确的是（　　）。
A．while 循环语句的循环体至少执行 1 次　　B．do–while 循环可以写成 while 循环的格式
C．continue 语句可以出现在各种循环体中　　D．break 语句不可以出现在循环体内
答案：B、C。
本题中，对于选项 A，while 循环语句的执行过程为：判断 while 条件是否成立，如果成立，则进入循环体执行，否则，不进入循环。当第一次判断 while 条件为 false 时，程序就不会进入 while 循环体了，此时循环体执行 0 次。所以，选项 A 错误。
对于选项 B，do–while 循环的执行过程为：首先进入循环体执行一次，然后判断 while 条件是否为 true，如果为 true，则执行下一次循环，否则，退出循环。把 do–while 语句改为 while 语句的最简单方法就是把循环体的内容先在循环体外执行一次，然后把 do–while 循环改成 while 循环即可。所以，选项 B 正确。
对于选项 C，关键字 continue 用来跳过循环体中剩余的代码而进入下一次循环，因此，可以出现在各种循环体内。所以，选项 C 正确。
对于选项 D，关键字 break 一般用于跳出当前循环，因此，它可以出现在循环体内。所以，选项 D 错误。
【真题 89】 break 关键字不能使用在（　　）语法结构中。
A．for 语句　　B．switch 语句　　C．if 语句　　D．while 语句
答案：C。

关键字 break 的作用是跳出当前循环，它不能用于循环语句（如 for 循环或 while 循环）和 switch 语句之外的任何其他语句中。显然，if 语句不在此之列。所以，选项 C 错误。

【真题 90】 有以下程序段：

```
int i = 0;
while(i=1)
    i++;
```

while 循环执行的次数是（　　）。

A．有语法错误，不能执行　　B．无限次

C．执行 1 次　　D．一次也不执行

答案：B。

while(expr)语句括号中是一个逻辑表达式，用以判断 while 循环是否需要继续执行，它可以是赋值语句。不过直接用赋值语句作为 expr 时，编译器可能认为赋值操作符=是等于判断符==的误写，这时编译器可能会报一个警告（warning），编译器是否报警，及是报警告还是错误（error）取决于编译器设置。

本题中，i=1 是赋值语句，但是 while 判断的不是赋值过程，而是内部的值（即 i 的值）；实际执行的顺序是如下两步，就好理解了：

1）i=1；

2）while(i)；

由于 i 的值为 1，所以，while 循环执行的次数时无限次。所以，选项 B 正确。

【真题 91】 t 为 int 类型，进入下面的循环之前，t 的值为 0。

```
while(t=1)
{
  _______
}
```

则以下叙述中正确的是（　　）。

A．循环控制表达式的值为 0　　B．循环控制表达式的值为 1

C．循环控制表达式不合法　　D．以上说法都不对

答案：B。

【真题 92】 do/while 和 while 的区别是什么？

答案：do/while 与 while 都是循环控制语句，只是 do/while 的执行流程是先执行一次循环然后再判断循环条件是否满足，属于“先斩后奏”，而 while 的意思是先进行循环条件判断，然后再根据判断结果决定是否执行循环体内容，属于“先奏后斩”。

【真题 93】 下面对 do-while 结构的描述中，不正确的是（　　）。

A．循环体至少执行一次

B．属于“直到型”循环

C．与 while 语句执行情况相同

D．先执行循环体，再计算终止条件，若结果为 true，重复执行循环体，直到布尔表达式的值为 false

答案：C。

【真题 94】 C 语言中 while 和 do/while 循环的主要区别是（　　）。

A．do/while 的循环体至少无条件执行一次
B．while 的循环控制条件比 do/while 的循环控制条件更严格
C．do/while 允许从外部转到循环体内
D．do/while 的循环体不能是复合语句
答案：A。
【真题 95】 在循环结构中，可以使得同一组语句一次也不执行的是（　　）。
A．当型循环　　B．直到型循环　　C．都不能实现　　D．do-while 语句
答案：A。

【真题 96】 在 for 循环中，是先执行循环体后再判断条件。（　　）。
答案：错误。
在 for 循环中，首先进行条件判断，然后再根据条件判断的结果判断是否执行循环体内容。如果初值不满足条件，则一次都不执行循环体内容。例如：

```
for (i = 100; i < 100; i++)
    printf("hello");
```

上述语句是没有输出的。
【真题 97】 语句 for(;1;)是什么意思？
答案：语句 for(;1;)的意思是执行无限循环，常见于嵌入式软件研发的中断控制中。与其功能类似的写法还有 while(1)。
【真题 98】 假设变量 m 和变量 n 都是 int 类型，那么以下关于 for 循环语句的描述中，正确的是（　　）。

```
for(m=0,n=-1;n=0;m++,n++)
    n++;
```

A．循环体一次也不执行　　B．循环体执行一次　　C．是无限循环
D．有限次循环
答案：A。
for 循环是否继续执行取决于中间的判断条件是否为真，本题中，判断条件为 n 赋值为 0（并非 n 是否等于 0），n=0 的值是 0，也就是说条件一直为假，所以，循环体一次也不会执行。有的读者就不明白了，为什么语句 n=0 的返回值是 0 呢？其实，赋值表达式的值就是赋给的值，例如 n=1 的值就是 1，a=（n=0），a 的值就是赋值表达式的值，而 a 的值会是 0。所以，选项 A 正确。

【真题 99】 有如下代码：

```
int func(int a)
{
    int b;
    switch (a)
    {
        case 1: b = 30;
        case 2: b = 20;
        case 3: b = 16;
        default: b = 0;
    }
    return b;
}
```

那么 func(1)=（　　）。

答案：0。

本题中，因为没有 break 语句，所以，程序不会跳出 switch，程序会一直执行到语句 b=0。

【真题 100】 以下关于关键字 break 的描述中，正确的是（　　）。

A．只中断最外层的循环　　B．只中断最内层的循环

C．借助于标号，可以实现任何外层循环中断　　D．只中断某一层的循环

答案：C。

关键字 break 的作用是直接强行跳出当前循环，不再执行剩余部分代码。当循环中遇到 break 语句时，忽略循环体中任何其他语句和循环条件测试，程序控制在循环后面语句重新开始。所以，当多层循环嵌套，并且 break 语句出现在嵌套循环中的内层循环时，它只是终止了内层循环的执行，而不影响外层循环的执行。

由于 break 只能跳出当前的循环，那么如何才能实现跳出多重循环呢？可以在多重循环的外面定义一个标识，然后在循环体里使用带有标识的 break 语句即可跳出多重循环。

从以上分析可以看出，break 可以借助标记跳出多重循环，选项 C 正确。

【真题 101】 switch()中不允许的数据类型有（　　）。

A．整型　　B．浮点型　　C．字符串　　D．布尔

答案：B、C。

1.9 数据类型

【真题 102】 下面不是 C++的标准数据类型的是（　　）。

A．int　　B．char　　C．bool　　D．real

答案：D。

下表为 C++数据类型的各类性质。

数据类型名称	字节数	别名	取值范围
int	*	signed,signed int	由操作系统决定，即与操作系统的“字长”有关
unsigned int	*	unsigned	由操作系统决定，即与操作系统的“字长”有关
bool	1	无	false 或 true
char	1	signed char	–128 ～127

unsigned char	1	无	0～255
short	2	short int,signed short int	–32,768 ～32,767
unsigned short	2	unsigned short int	0～65,535
long	4	long int,signed long int	由操作系统决定，即与操作系统的“字长”有关
long long	8	none (but equivalent to __int64)	–9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807
unsigned long	4	unsigned long int	由操作系统决定，即与操作系统的“字长”有关
enum	*	无	由操作系统决定，即与操作系统的“字长”有关
float	4	无	3.4E +/– 38 (7 digits)
double	8	无	1.7E +/– 308 (15 digits)
long double	8	无	1.7E +/– 308 (15 digits)

通过上表可知，real 不是 C++语言中的基本数据类型，所以，选项 D 不正确。

【真题 103】C600 编译器对每种数据类型都确认了一个尺寸，下面分配形式错误的是（　　）。

A．char 8bit　　B．short 16bit　　C．int 32bit　　D．long 64bit

答案：　D。

C600 编译器对每种数据类型都确认了一个尺寸，其分配形式见下表：

类型	大小/bit
字符型（char）	8
短型（short）	16
整型（int）	32
长型（long）	40
浮点型（float）	32
双精度浮点型（double）	64

【真题 104】 在 32 位机器上，有如下代码：

```
signed char a = 0xe0;
unsigned int b = a;
unsigned char c = a;
```

以下描述正确的是（　　）。

A．a>0 && c>0 为真　　B．a == c 为真

C．b 的十六进制表示是：0xfffffffe0　　D．上面都不对

答案：C。

本题中，对于第一个有符号字符 a，它的十六进制表示为 0xe0，将该值转换成二进制表示后即为 1110 0000，因为二进制表示的最高位为 1，所以，当其表示的是有符号数时，表示的就是一个负数，当其表示的是无符号数时，表示的就是一个正数。

对于选项 A 与选项 B，变量 a 表示的是一个负数，而变量 c 表示的是一个正数。因此，c<0，且 a 的值不等于 c 的值。所以，选项 A 错误，选项 B 错误。

对于选项 C，b 是一个无符号整型变量，其值为 0xe0，由于此时需要将一个 8 位的有符号字符型转换为一个 32 位的无符号整型，根据数据的扩展原则：有符号的数据类型在向高精度扩展时，总是带符号扩展，而无符号的数据类型在向高精度扩展时，总是无符号扩展，所以，此时高位用 1 补齐，变量 b 的二进制表示形式是 1111 1111 1111 1111 1111 1111 1110 0000，对应的十六进制的表示形式是 0xfffffffe0。所以，选项 C 正确。

对于选项 D，由于前三个选项中，选项 C 是正确的，所以，选项 D 错误。

【真题 105】 有如下代码：

```
unsigned int k = 20;
while (k >= 0)
    --k;
```

则 while 循环执行次数为（　　）。

A．20 次　　B．一次也不执行　　C．死循环　　D．21 次

答案：C。

由于 k 是一个无符号整数，而无符号整数的取值范围是 0–255，所以，k 的值会永远大于 0，所以，while 循环是一个死循环。所以，选项 C 正确。

【真题 106】 设 C 语言中，一个 int 型数据在内存中占 2 个字节，则 unsigned int 型数据的取值范围为（　　）。

答案：0～65535。

两个字节，即 16 位，unsigned int 的取值范围是 0～最大数–1。所以，对于 int 型而言，最高位为符号位，表示范围为–32768～32767，而 unsigned int 无符号位，所以，它可以表示的范围为 0～65535。

【真题 107】 有以下定义和语句：

```
int u=010, v= 0x10, w=10;
printf("%d,%d,%d\n",u,v,w);
```

以上程序的输出结果为（　　）。

A．8,16,10　　B．10,10,10　　C．8,8,10　　D．8,10,10

答案：A。

在计算机中，八进制的数通常以 0 开头，十六进制的数通常以 0x 开头，所以，变量 u 的初始化值为八进制的 010，对应的十进制表示为 8，变量 v 的初始化值为十六进制的 0x10，对应的十进制表示为 16，变量 w 的初始化值为十进制的 10，所以，变量 u、v、w 的值分别为 8、16、10。所以，选项 A 正确。

【真题 108】 关键字 const 的作用是什么？

答案：关键字 const 修饰的变量表示该常量是不可以修改的。

【真题 109】 简述以下各定义的含义：

1）const char *p;

2）char const *p;

3）char * const p;

答案：const char *p；　//指向常量的指针，指向的常量值不可以改（const 修饰 char）

char const * p；// 和 const char *p 一样（const 修饰*p）

char * const p; //常量指针，p 的值不可以修改(const 修饰 p)

由于没有 const *的运算，如果出现 const * 的形式，则 const 实际上是修饰前面的。

例如：char const*p，由于没有 const*运算，则 const 实际上是修饰前面的 char，因此，char const*p 等价于 const char*p，所以，语句 1）和语句 2）中 p 表示的意思是一样的，为指向常量的指针，语句　　3）中 p 表示的意思是指针常量。

【真题 110】 有以下表达式：int a=248; b=4;int const c=21;const int *d=&a; int *const e=&b;int const *f const =&a;请问下列表达式中，哪些会被编译器禁止？为什么？

*c=32;d=&b;*d=43;e=34;e=&a;f=0x321f;
答案：变量 c 是 const 常量，不是指针，所以，*c 是一种非法使用方式，会被编译器禁止。*d 是 const 类型，所以，d=&b;*d=43;是非法使用方式，会被编译器禁止。同理，e 与 f 都是 const 类型，所以，也会被编译器禁止。
【真题 111】 在 C++语言中，X 为一个 C++类，以下语法错误的是（ ）。
A．const X * x B．X const * x C．const X const * x D．X * const x
答案：C。
如果 const 位于*的左侧，则 const 就是用来修饰指针所指向的变量，即指针指向为常量。
如果 const 位于*的右侧，则 const 就是修饰指针本身，即指针本身是常量。
选项 A，const 修饰的是 X（在*左边），表示的是指针 x 指向的对象常量。
选项 B，const 修饰*x（在*左边），与选项 A 是相同的，表示 x 指向的对象是常量。
选项 C，两个 const 表示的同一个意思，因此，这种写法是错误的。如果换成 const X* const x，那么表示 x 这个指针为常量，而且 x 指向的对象也是常量。
选项 D，const 修饰 x（在*的右边），表示 x 这个指针为常量，一旦初始化后就不能指向其他的对象了。
本题中，选项 A、选项 B 和选项 D 符合语法规则，只有选项 C 不符合语法规则。所以，选项 C 错误。
【真题 112】 定义有 int * const ptr，下面说法中正确的是（ ）。
A．ptr 不可修改，*ptr 不可修改 B．ptr 不可修改，*ptr 可修改
C．ptr 可修改，*ptr 可修改 D．ptr 可修改，*ptr 不可修改
答案：B。
当 const 与指针运算符*搭配使用时，*号右边的 const 表示指针不能改变其自身的指向，即常指针，固定指向一个地址，需要在声明的时候初始化。所以，选项 B 正确。
为了更好地解释，举几个简单的例子加以说明。

```
int i = 4, j = 5;          //定义整型变量 a 与 b
int* const ptr;            //错误，未在声明的时候初始化。
int* const ptr = &i;       //正确，声明的时候初始化
ptr = &j;                  //错误，常指针固定指向一个地址，不能更改。
*ptr = 6;                  //正确，内容可以改
```

在《C++ Primer》（第五版）一书中指出，const int*和 int* const 中的 const 修饰的对象不同，const int*修饰值，不能改变指针所指的值；而 int* const 修饰指针，不能改变指针的地址。
【真题 113】 已知 const char * node="ABC"；下列语句合法的是（ ）。
A．node[2] = 'k'; B．*node[2]='k'; C．*node = "xyz"; D．node="xyz";
答案：D。
本题中，node 是一个指针，它指向的内容是 const char 类型。node 可以指向任何标示符，但它指向的内容是不能被改变的，但是选项 A、选项 B 与选项 C 三项均改变了它指向的内容，所以，选项 A、选项 B 与选项 C 错误。只有选项 D 的写法是正确的。
【真题 114】print()函数是一个类的常成员函数，它无返回值，下列表示中，正确的是（ ）。
A．const void print(); B．void const print(); C．void print() const; D．void print(const);
答案： C
常成员函数的说明格式如下：
类型说明符函数名（参数表）const;

【真题 115】 static 关键字的作用是什么？static 全局变量与普通全局变量的区别是什么？static 局部变量与普通变量的区别是什么？static 函数与普通函数的区别是什么？

答案：static 关键字是 C/C++语言中都存在的关键字，中文译为静态的，它具有以下特性：

1）static 全局变量的作用域范围是有限制的，即如果一个变量被声明为静态的，那么该变量可以被模块内所有函数访问，但不能被模块外其他函数访问，它是一个本地的全局变量。而普通全局变量能被其他模块访问。

2）在函数体内，静态变量具有“记忆”功能，即一个被声明为静态的变量在这一函数调用结束后，它的值仍然被保存着，当这个函数下一次被调用的时候，这个静态变量的值仍然是上次调用后的结果（需要注意的是，函数中的静态变量只初始化一次），而函数体内的普通变量没有记忆功能。如下例所示。

```
#include<stdio.h>
void fun1(int i)
{
    static int value = i;
    value+=1;
    printf("%d  ", ++value);
}
void fun2(int i)
{
    int value = i;
    printf("%d  ", ++value);
}
int main()
{
    printf("fun1:");
    fun1(0);
    fun1(4);
    fun1(7);
    printf("\nfun2:");
    fun2(0);
    fun2(4);
    fun2(7);
    return 0;
}
```

程序的运行结果为：

```
fun1:2  7  15
fun2:1  5  8
```

函数 fun1 中把 value 定义为静态变量，因此，它会在第一次调用的时候初始化为 0，由于其具有记忆功能，只会被初始化一次。因此，在调用函数 fun1(0)的时候，函数 fun1 内部的 value 被初始化为 0，语句 printf("%d ",++value);输出结果为 2；当调用函数 fun1(4)的时候，语句 static int value=i;不会再被执行（只能初始化一次），此时 value 的值为 2，因此，打印语句 printf("%d ",++value)的输出结果为 2+4+1=7，同理，第三次调用函数 fun1(7)时，输出的结果为 15。

而对于普通变量而言，没有记忆功能，每次被调用的时候都需要初始化。调用 fun2 函数的时候，value 每次都会被初始化为传入的参数，因此，每次输出++value 的值就是实参+1。

3）如果一个函数被声明为静态的，那么该函数与普通函数的作用域不同，静态函数的作用域仅在本文件中，它只能被这一模块内的其他函数调用，不能被模块外的其他函数调用，也就是说，这个函数被限制在声明它的模块的本地范围内使用。而普通函数可以被其他模块使用。

【真题 116】 下面程序的输出结果是（　　）。

```
#include<iostream>
using namespace std;
int i = 0;
int fun(int n)
{
    static int a = 2;
    a++;
    return a*n;
}
int main()
{
    int k = 5;
    {
        int i = 2;
        k += fun(i);
    }
    k += fun(i);
    cout << k;
    return 0;
}
```

A．13　　B．14　　C．15　　D．11

答案：D。

本题中，整型变量 k 首先被初始化为 5，紧接着在一个大括弧的范围内，k 的值执行了一次与 fun(i)的求和操作，由于函数 fun 中存在一个静态变量 a，而静态变量的生命周期贯穿整个程序的运行过程，它只初始化一次，因此，第一次调用函数 fun 后，a 值变为了 3，k 的值变为了 5+6=11。由于大括号限定了 int i = 2 的作用域，因此，当第 2 次调用函数 fun 时，变量 i 是全局变量，int i = 0，此时 k 的值与 0 相加，仍然为 11，所以，程序输出为 11。所以，选项 D 正确。

【真题 117】 以下关于类中的 static 成员和对象成员的描述中，正确的是（　　）。

A．static 成员变量在对象构造时候生成　　B．static 成员函数在对象成员函数中无法调用

C．虚成员函数不可能是 static 成员函数　　D．static 成员函数不能访问 static 成员变量

答案：C。

对于选项 A，类的 static 成员变量属于该类（而非对象），需要在类定义时初始化，不可以在对象的构造函数中初始化。所以，选项 A 错误。

对于选项 B，static 成员函数在对象成员函数中可以调用，同属于一个类作用域。所以，选项 B 错误。

对于选项 C，static 成员函数不可以声明为 const 和 virtual。所以，虚成员函数不可能是 static 成员函数。所以，选项 C 正确。

对于选项 D，static 成员函数只能够访问 static 成员变量。所以，选项 D 错误。

【真题 118】 在一个 cpp 文件里面，定义了一个 static 类型的全局变量，下面描述正确的是（　　）。

A．只能在该 cpp 所在的编译模块中使用该变量

B．该变量的值是不可改变的
C．该变量不能在类的成员函数中引用
D．这种变量只能是基本类型（例如 int，char），不能是 C++类型
答案：A。
在 C/C++语言中，static 表示静态的意思，由 static 修饰的变量通常存放在内存空间的堆上，它不会因为函数的执行完毕而被销毁。
本题中，定义了一个 static 类型的全局变量，该变量只能在该 cpp 所在的编译模块中被使用。所以，选项 A 正确。
对于选项 B，只有 const 变量是不可以改变的，而 static 变量是可以改变的。所以，选项 B 错误。
对于选项 C，该变量可以在类的成员函数中引用。所以，选项 C 错误。
对于选项 D，static 既可以修饰基本数据类型，例如 int、char，也可以修饰结构体类型、对象类型。所以，选项 D 错误。
【真题 119】 声明常量时通常会用关键字 const 和 static，下面关于两者的区别中，说法正确的是（　　）。
A．static 表示这个变量可以在其他文件中使用
B．static 变量连接期被分配到了 data 段，即使是在函数调用中定义也不会在栈中产生，而是在程序加载期就被加入了内存
C．const 表面含义是个常量，但实际上还是占据一个内存位置的变量，但是它的值一般是在编译时期就决定了
D．上面说法都不正确
答案：B、C。

1.10 运算

【真题 120】 定义有变量 int i = 0; int a = i++; int b=++a; int c = a+b;，那么表达式 a?b:c 的值为（　　）。
A．0　　B．1　　C．2　　D．3
答案：B。
对于变量 a，当使用前置运算符修饰时，即++a 表示的意思是先取 a 的地址，增加它的内容，然后再把值放在寄存器中；当使用后置运算符修饰时，即 a++表示的意思是先取 a 的地址，再把它的值装入寄存器，然后增加内存中 a 的值。一般而言，当涉及表达式计算时，对这两种情况的计算过程区分如下：后置的++运算符是先将其值返回，然后其值增 1；而前置的++运算符，则是先将值增 1，再返回其值。
对于本题而言，语句 int a = i++的执行过程如下：首先，取 i 的值返回并赋值给 a，此时 a 的值变为 0，然后，i 的值变为 1。语句 int b=++a 的执行过程如下：首先，a 执行自增操作，a 的值变为 1，然后，再把自增后的值赋值给 b，变量 b 的值为 1。接着执行语句 c=a+b=0+1=1，对于表达式 a?b:c 的值而言，因为变量 a 的值为 0，等价于 false，所以，a=c=1。因此，选项 B 正确。
【真题 121】 如果在某系统中，等式 15*4=112 成立，则系统采用的是（　　）进制。
A．6　　B．7　　C．8　　D．9

答案：A。
本题可以采用假设法来求解。假设采用的是六进制，那么计算过程为：4*5=20，20/6=3，20%6=2，因此，4*5 的结果等于 2，进位值为 3。接着计算 1*4+3（进位值）=7，由于 7/6=1，7%6=1，因此，计算结果为 1，进位为 1。所以，计算结果为 112。符合题目预期。因此，假设成立。
而选项 B、选项 C 和选项 D 中的三种假设都不满足题意。根据排除法，只有选项 A 正确。
【真题 122】 除了十进制、二进制之外，十六进制表达式在计算机领域中也经常使用（例如各种字符集的定义描述），下式：（2012）10+（AF1）16 的结果是（　　）（请用十进制表示）。
答案：4813。
本题是一道有关进制转换的题目，只需要将十六进制数表示为十进制数，然后再与另外一个十进制数进行求和即可。所以，AF1(16)=10*16*16+15*16+1=2801(10)，二者求和 2012(10)+2801(10)，结果为 4813(10)。
【真题 123】 下面不能被重载的运算符有（　　）。
A．作用域运算符“::”　　B．对象成员运算符“.”
C．指针成员运算符“->”　　D．三目运算符“?:”
答案：A、B、D。
C++语言中能被重载的运算符有如下几种：
1）算术运算符：+、-、*、/、%、++、--。
2）位操作运算符：&、|、～、^(位异或)、<<(左移)、>>(右移)。
3）逻辑运算符：!、&&、||。
4）比较运算符：<、>、>=、<=、==、!=。
5）赋值运算符：=、+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=。
6）其他运算符：[]、()、->、,(逗号运算符)、new、delete、new[]、delete[]、->*。
不能重载的运算符有 5 个，分别为. 、?:、sizeof、:: 、.*。
【真题 124】 给定 3 个 int 类型的正整数 x、y、z，对如下 4 组表达式进行判断，正确的选项是（　　）。

```
int a1=x+y-z; int b1=x*y/z;
int a2=x-z+y; int b2=x/z*y;
int c1=x<<y>>z; int d1=x&y|z;
int c2=x>>z<<y; int d2=x|z&y;
```

A．a1 一定等于 a2　　B．b1 一定等于 b2　　C．c1 一定等于 c2　　D．d1 一定等于 d2
答案：A。
本题中涉及的运算符有+、-、*、/、<<、>>、&、|等。
本题中，对于选项 A，整型变量 a1 与整型变量 a2 的运算只涉及了加法运算符+与减法运算符-，这两个符号的运算满足交换律，表面上看，二者确实是相等的。但是，细心的读者可能会有疑问了，很有可能存在一种特例，就是数据溢出了，交换律是否还成立呢？假设 x 与 y 的和发生了溢出，那么此时 a1 与 a2 还能一定相等吗？回答还是肯定的。即使 x 与 y 的和值发生了溢出，其结果仍然是相等的。
当发生溢出时，假设整型变量 x 与整型变量 y 同号（要么同时为正整数，要么同时为负整数），会存在以下几种情况：情况一：x 与 y 同号，都为正数。针对 x 与 y 都为正整数的情况，当 x 与 y 的和值发生溢出时，那么溢出后的值为最大的负数，这个最大的负数再减去整型数 z，其符号有一次改变。符号改变了两次，相当于最高位的值没有改变。而这个时候 x-z+y 因为同号也不可能符号改变，低位又都是直接相加，所以，x-z+y 与 x+y-z 一样。还有就是 x 和

y 与 z 不同号，这个时候就更容易了，x+y−z 溢出，x−z+y 同样溢出，符号位改变一次，所以，二者也相等。情况二：x 和 y 同号为负数，其实和情况一一样，这里不再赘述。

其实对于计算机而言，任何两个数相加 a+b（包括负数，−1−2 在计算机看来就是−1+(−2)，注意负数在计算机中存放的特殊性），在计算的时候是直接取出表示这个数字的字节，然后从低位到高位相加，最后得出来一个结果，无论是有符号还是无符号都再把这个结果翻译成对应的数即可。

```
#include <iostream>
using namespace std;
int main()
{
    char x = 126;
    char y = 24;
    char z = 65;
    char a1 = x + y − z;   // x + y 有溢出，但是 − z 后可以减回来
    char a2 = x − z + y;
    cout << (int)a1 << endl;
    cout << (int)a2 << endl;
    return 0;
}
```

所以，选项 A 正确。

对于选项 B，涉及整数的除法运算。

其实，C/C++语言中的除法运算符/与求余运算符%，都为二元运算符，具有左结合性。它们通常遵循以下几点规律：

1）整数除法结果的小数部分都被丢弃，这个过程称为“截尾”（truncation）。

2）相同数据类型的数据、变量进行运算，结果保持原有数据类型。当不同数据类型的数据、变量进行运算时，结果为精度高的数据类型。例如，整数与浮点数进行混合除法运算时，其结果都是浮点数。

3）对负数的整数除法，C99 要求使用“趋零截尾”。即除号的正负取舍和一般的算数一样，符号相同为正，相异为负。

4）对负数的取模运算，C99 规定：如果第一操作数为负数，那么得到的模为负数，如果第一操作数为正数，那么得到的模为正数。例如：7/4=1，−7/4=−1，7/−4=−1，−7%4=−3，7%−4=3。

根据以上的分析可知，由于整数除法的截断，b1 和 b2 不一定相等。以如下程序为例。

```
#include<iostream>
using namespace std;
int main()
{
    int x = 3;
    int y = 4;
    int z = 12;
    int b1 = x * y / z;
    int b2 = x / z * y;
    cout << b1 << endl; // 1
    cout << b2 << endl; // 0
    return 0;
}
```

程序的运行结果为：

```
1
0
```

所以，选项 B 不正确。

对于选项 C，左移运算是将一个二进制位的操作数按指定移动的位数向左移位，移出位被丢弃，右边的空位一律补 0。右移运算是将一个二进制位的操作数按指定移动的位数向右移动，移出位被丢弃，左边移出的空位或者一律补 0，或者补符号位，这由不同的机器而定。由于移位会丢弃超出位，所以，c1 和 c2 不一定相等。以如下程序为例。

```
#include<iostream>
using namespace std;
int main()
{
    int x = 2;
    int y = 2;
    int z = 2;
    int c1 = x << y >> z; // 2 先乘以 4，再除以 4
    int c2 = x >> z << y; // 2 先除以 4，再乘以 4
    cout << c1 << endl; // 2
    cout << c2 << endl; // 0
    return 0;
}
```

程序的运行结果为：

```
2
0
```

所以，选项 C 不正确。

对于选项 D，d1 是(x&y)|z，而 d2 是 x|(y&z)，该运算不满足结合律，因此，不一定相等。

程序示例如下：

```
#include<iostream>
using namespace std;
int main()
{
    int x = 1;
    int y = 2;
    int z = 4;
    int d1 = x & y | z;
    int d2 = x | z & y;
    cout << d1 << endl; // 4
    cout << d2 << endl; // 1
    return 0;
}
```

程序的运行结果为：

```
4
1
```

所以，选项 D 不正确。

【真题 125】 有以下程序：

```
void main(void)
{
    int a, b, d = 26;
    a = d / 10 % 9;
    b = a && (-1);
    printf("%d, %d\n", a, b);
}
```

程序运行后的输出结果是（　　）。

A．6，1　　B．2，1　　C．6，0　　D．2，0

答案：B。

本题中，整型变量 d 的值为 26，当计算表达式 a=d/10%9 的值时，由于除号/与取余符%的优先级是一样的，所以，运算顺序是从左向右，先执行 d/10，其运算结果为 2，然后执行 2%9，其运算结果为 2，即 a 的值为 2。&&为逻辑与运算符，当进行与运算时，只要参与运算的数不是 0，都认为是 true(1)，参与运算的数是 0 则认为是 false(0)，2 和−1 都是非 0 值，都被认为真，所以，最后结果为真，对 bool 型变量 true 进行强制整型转换，其结果即为 1。所以，选项 B 正确。

【真题 126】 以下程序运行后的输出结果是 10，20，0。（　　）。

```
void main(void)
{
        int a, b, c;
        a = 10;
        b = 20;
        c = (a%b<1) || (a / b>1);
        printf("%d %d %d\n", a, b, c);
}
```

答案：正确。

变量 a 和变量 b 的值在初始化后就没有被修改过，它们的值分别为 10 和 20。对于变量 c，由于 a%b=10，所以，a%b<1 不满足，结果为 false，表达式 a/b 的值 0，因此，表达式 a/b>1 的值也为 false，所以，(a%b<1) || (a/b>1)的值为 false。在 C/C++语言中，false 表示 0，true 表示 1，因此，c 的值为 0。

【真题 127】 unsigned short A = 10; printf("~A = %u\n", ~A); char c=128; printf("c=%d\n",c); 输出结果是什么？

答案：4294967285, −128。

本题中，变量 A 是一个 unsigned short 类型的变量，值为 10，由于~A＝0xfffffff5，所以，~A 对应的 int 值为−11，但输出的是 uint，所以，输出 4294967285。

由于 c＝0x80，输出的是 int，最高位为 1，是负数，所以，它的值就是 0x00 的补码，也就是 128，所以，输出−128。

【真题 128】 有如下代码：

```
#include<iostream.h>
using namespace std;
void main()
{
    int a = 2;
    int b = ++a;
    cout << a / 6 << endl;
}
```

程序的输出结果是（　　）。

A．0.5　　B．0　　C．0.7　　D．0.6666666

答案：B。

本题中，首先初始化变量 a 的值为 2，然后执行++a 运算，此时 a 的值变为 3，紧接着执行 a/6 运算，由于变量 a 是 int 类型，因此，3/6 的结果会被四舍五入，最终输出结果为 0。所以，选项 B 正确。

【真题 129】 有如下代码：

```
int i, n = 0;
float x = 1, y1 = 2.1 / 1.9, y2 = 1.9 / 2.1;
for (i = 1; i < 22; i++)
    x = x * y1;
while (x != 1.0)
{
    x = x * y2;
    n++;
}
printf("%d\n", n);
```

程序的运行结果是（　　）。

A．21　　B．22　　C．无限循环　　D．程序崩溃

答案：C。

本题中，首先对浮点型变量 x 的值进行了迭代计算，然后判断 x 的值与浮点数 1.0 是否相等，可是此时却忽略了一个事实，即对于浮点数的运算中不允许出现==或者!=。

为什么会是这样的结果呢？其实，这与浮点数在内存中的存储有关。

浮点数指的是小数点在数据中的位置可以左右移动的数据。它通常被表示成如下形式：N=M*RE。其中，M(Mantissa)被称为浮点数的尾数，R(Radix)被称为阶的基数（底），E(Exponent)被称为阶码。计算机中一般规定 R 为 2、8 或 16，它是一个确定的常数，不需要在浮点数中明确表示出来。因此，如果要表示一个浮点数，一是要给出尾数 M 的值，通常用定点小数形式表示，它决定了浮点数的表示精度，即可以给出的有效数字的位数。二是要给出阶码，通常用整数形式表示，它指出的是小数点在数据中的位置，决定了浮点数的表示范围。浮点数也要有符号位。在计算机中，浮点数通常被表示成如下格式：

Ms	E	M

其中，Ms 表示浮点数是正数还是负数，安排在最高一位，0 表示正数，1 表示负数。

E 表示的是阶码，紧跟在符号位之后，占用 m 位，含阶码的一位符号。类似于科学计数法中的 M*10^N 中的 N。

M 表示的是尾数，在低位部分，占用 n 位。

在计算机中，浮点数其实是一种不精确的表示，它存在舍入（rounding）误差。因为表示方法限制了浮点数的范围和精度，浮点运算只能近似地表示实数运算，而判等运算符==表示的是在计算机中的内存表示完全一样，显然，浮点数无法在内存中表示完全一样，所以，不能使用==或者!=来表示两个浮点数是否相等。

既然如此，是否还存在其他的方法来比较浮点数是否相等呢？答案是肯定的。判断两个浮点数是否相等可以通过下面的方法来实现。

const float EPSINON = 0.000001;

if (fabs(a−b)< EPSINON)（其中，a 与 b 表示待比较的两个浮点数）

本题中，由于变量 x 的值几乎永远无法与 1.0 相等，循环条件几乎永远成立，程序会进入无

限循环。所以，选项 C 正确。

【真题 130】 在 16 位机器上，运行下列 foo 函数的结果是（　　）。

```
void foo()
{
    int i = 65536;
    cout << i <<",";
    i = 65535;
    cout << i;
}
```

A．−1,65535　　B．0,−1　　C．−1,−1　　D．0,65535

答案：B。

在 16 位的机器上，int 型数据的取值范围为[−32768, 32767]，由于 65536 的十六进制表示为 0x10000，此时会发生溢出，溢出后会发生数据截断，取其后的 0x0000 为值，对应的十进制表示为 0。而 65535 的十六进制表示为 0xFFFF，最高位为符号位，由于在计算机中负数使用二进制补码表示，此时对该值取反加 1，最终结果为−1。所以，选项 B 正确。

如果是在 32 位的机器上，由于不存在数据溢出的情况，程序的输出就变为 65536 和 65535 了。

【真题 131】 假设 x = 5，y = 6，则 x < y 和 x >= y 的逻辑值分别为（　　）和（　　）。

答案：true，false。

本题中，由于 x=5，y=6，所以，x<y 是成立的，因此，逻辑值为 true，而 x >= y 不成立，因此，逻辑值为 false。

【真题 132】 以下代码的运行结果为（　　）。

```
#include<stdio.h>
int main()
{
    int a = 100;
    while (a > 0)
    {
        --a;
    }
    printf("%d", a);
    return 0;
}
```

A．−1　　B．100　　C．0　　D．死循环

答案：C。

对于 while 循环而言，只要循环条件（while 语句后面括号内的表达式的值）为真，则循环一直进行下去，对于前置自减运算符--而言，当运算结束后，变量的值会减 1。（备注：需要注意区分前置运算符与后置运算符的差别）

本题中，while 循环的条件是变量 a 的值大于 0，由于 a 的初始化值为 100，当代码执行时，会进入 while 循环内执行，由于执行了一次自减操作，于是变量 a 的值变为 99，由于 a 的值此时仍然大于 0，此时循环继续执行，直到 a 的值变为 0，不满足 while 循环的条件，此时循环终止，最终变量 a 的值为 0。因此，选项 C 正确。

【真题 133】 如果 x = 1 , y = 2 , z = 3，则表达式 y+＝z--/++x 的值是（　　）。

A．3　　B．3. 5　　C．4　　D．4.5

答案：A。

在编程的时候，经常会用到变量的自增或自减操作，尤其在循环中用的最多。以自增为例，有两种自增方式：前置与后置，即++i 和 i++，它们的不同点在于 i++是在程序执行完毕后自增，而++i 是在程序开始执行前进行自增。

对于本题而言，在执行语句 y+=z--/++x 的时候，z--是在这个表达式计算结束后才执行自减操作，在这个表达式中，z 的值为 3，在表达式计算完成后，z 的值变为 4，而++x 则不同，会首先执行 x 自增操作，x 的值变为 2，然后参与运算，上面表达式等价于 y+=3/2，显然，3/2=1（运算结果需要被转换为 int 类型，丢弃掉小数部分）。y 的值变为 3。所以，选项 A 正确。

【真题 134】 下面代码的输出结果是什么？

```
#include<stdio.h>
int main()
{
    int a, b, c, d;
    a = 10;
    b = a++;
    c = ++a;
    d = 10 * a++;
    printf("%d,%d,%d", b, c, d);
    return 0;
}
```

答案：10，12，120。

本题中，变量 a 的值被初始化为 10，当执行完语句 b=a++后，变量 a 的值变为 11，而变量 b 的值变为 10。当执行完语句 c=++a 后，a 的值变为 12，而变量 c 的值变为 12。当执行完语句 d=10*a++后，变量 a 的值变为 13，而变量 d 的值变为 120。所以，变量 b、c、d 的值分别为 10、12、120。

【真题 135】 设 int arr[]={6,7,8,9,10};

```
int *ptr=arr;
*(ptr++)+=123;
printf( " %d,%d " ,*ptr,*(++ptr));
```

程序的输出结果为（　　）。

答案：答案是不确定的。因为参数是从左到右执行还是从右到左执行，规范是没有规定的，不同的编译器可能会有不同的实现方式。

目前来讲，大部分的编译器都采用从右到左的执行方式。在这种情况下，在执行完语句 *(ptr++)+=123 后，ptr 指向 7，在执行 printf 语句的时候，由于从右到左执行，因此，先执行 *(++ptr)，显然结果为 8，当输出*ptr 的时候，由于 ptr 已经指向 8 了，因此，也输出 8。所以，对于这类编译器，输出结果为 8，8。

如果碰到从左向右执行的编译器输出结果则为 7，8，因为此种写法的执行过程与编译环境有着非常紧密的联系，所以，在编写代码的时候，尽量避免采用这种写法。

【真题 136】 以下四个表达式写法是否正确？如果正确，写出经过表达式运算后 a 的值。

```
int a = 4;
```

A. a += (a++);　　B. a += (++a) ;　　C. (a++) += a;　　D. (++a) += (a++);

答案：表达式 C 与表达式 D 错误。表达式 C 和表达式 D 等号左边都是个具体的数值，不是一个合法的左值。选项 A 运算结束后 a 的值为 9；选项 B 运算结束后 a 的值为 10。

【真题 137】 假设 i=2，那么表达式 (i++)/3 的值为（　　）。

答案：0。

表达式的值为 2/3（计算结束后，i 的值变为 3），运算结果的值与 i 的类型相关，如果变量 i 为 int 类型，那么结果为 0。

【真题 138】 有如下代码：

```
main()
{
    char a='1',b='2';
    printf("%c, ",b++);
    printf("%d\n",b-a);
}
```

在计算机中，数字字符 0 的 ASCII 值为 48，以上程序运行之后的输出结果是（　　）。

A．3,2　　B．50,2　　C．2,2　　D．2,50

答案：C。

在计算机中，由于数字字符 0 的 ASCII 值是 48，所以，数字 1 的 ASCII 值是 49，数字 2 的 ASCII 值是 50，数字 3 的 ASCII 值是 51。

当执行到语句 printf("%c, ",b++)时，由于是后置++运算，所以，b++返回的还是 b 的值，b 表示字符 2，而打印输出的格式为字符，打印输出为字符 2。

当执行到语句 printf("%d\n",b-a)时，由于经过 b++运算后，b 的 ASCII 值由 50 变为了 51，而此时 a 的 ASCII 值仍然为 49，b-a=2，此时输出的是整数，因此，输出为整数 2。所以，选项 C 正确。

【真题 139】 下列表达式正确的是（　　）。

A．9++　　B．(x+y)++　　C．c+++c+++c++　　D．++(a-b--)

答案：C。

在 C/C++语言中，自增运算符++与自减运算符--都具有对运算量重新赋值的功能，因为常量与表达式在内存中没有存储单元，所以，自增运算符++与自减运算符--只能用在变量（包括整型、实型，还有字符型、枚举型等标准类型）上，而不能作用于常量或表达式上，而且，这两个运算符（++与--）的运算优先级高于加法运算符+和减法运算符-，低于括号运算符()。

本题中，对于选项 A，9 是一个数字常量，显然，对其进行自增运算是非法的。所以，选项 A 错误。

对于选项 B，对表达式 x+y 进行了自增运算，显然，该运算也是非法的。所以，选项 B 错误。

对于选项 C，表达式看似复杂，其实也不难理解，表达式的意思是对变量 c 执行三次自增运算，最后再将它们相加，该过程是合法的。所以，选项 C 正确。

对于选项 D，对表达式 a-b--执行自增运算，显然，该表达式是非法的。所以，选项 D 错误。

【真题 140】 执行 C 语言代码“int a=1; int b=0; int c=0; int d=(++a)*(c=1);”后，变量 a、b、c、d 的值分别为（　　）。

A．2，0，1，2　　B．1，0，1，1　　C．2，0，1，1　　D．2，0，0，2

答案：A。

在解答本题之前，首先要弄懂前置++/--运算符与后置++/--运算符的运算原理以及二者之间的区别。以变量 i 为例，i++与++i 这两种方式都会使得 i 自增 1，不同之处在于其内部实现，++i 运算表示先取 i 的地址，增加它的内容，然后把结果放在寄存器中，而 i++表示首先取 i 的地址，把它的值装入寄存器，然后增加内存中的 i 的值。《Google C++编程风

格》一书中强调，对于迭代器和其他模板对象使用前缀形式（++i）的自增、自减运算符，因为前置自增（++i）通常要比后置自增（i++）效率更高。
本题中，语句 d=(++a)*(c=1)的执行过程如下：执行++a 运算后，变量 a 的值变为 2，执行 c=1 运算后，变量 c 的值变为 1，而 d=2*1=2。由于变量 b 被初始化为 0 后，一直没有被修改，b 的值为 0。所以，选项 A 正确。

【真题 141】 下列操作符中，属于位操作的是（　　）。
A．|　　B．^　　C．&&　　D．<<
答案：A、B、D。
对于选项 A，|是按位或运算符，属于位操作。所以，选项 A 正确。
对于选项 B，^是按位异或运算符，属于位操作。所以，选项 B 正确。
对于选项 C，&&是逻辑与运算符，不属于位操作。所以，选项 C 错误。
对于选项 D，<<是左移运算符，属于位操作。所以，选项 D 正确。
【真题 142】 有以下程序：

```
#include<stdio.h>
main()
{
    char x=2,y=2,z;
    z=(y<<1)&(x>>1);
    printf("%d\n",z);
}
```

程序的输出结果是（　　）。
A．1　　B．0　　C．4　　D．8
答案：B。
左移 1 位相当于乘以 2，右移 1 位相当于除以 2，y<<1 后的结果是 0000 0100，x>>1 后的结果是 0000 0000，二者执行&操作后，结果是 0。所以，选项 B 正确。
【真题 143】 执行 1100|1010，1001^1001，1001&1100 后，其结果分别为（　　）。
A．1110　0000　1000　　B．1000　1001　1000
C．1110　1001　0101　　D．1000　1001　1000
答案：A。
对于或运算符|，有：1 | 1 = 1，1 | 0 = 1，0 | 1 = 1，0 | 0 = 0。对于异或运算符^，有：1 ^ 1 = 0，1 ^ 0 = 1，0 ^ 1 = 1，0 ^ 0 = 0。对于与运算符&，有：1 & 1 = 1，1 & 0 = 0，0 & 1=0，0 & 0 = 0。
本题中，1100|1010=1110，1001^1001=0000，1001&1100=1000。所以，选项 A 正确。
【真题 144】 定义有整数 int i = 0xFE78DA45，int k = 0xAC3189B2，则 i^k 的值为（　　）。
A．0x524953f7　　B．0xAC308800　　C．0xFE79DBF7　　D．0X0000001
答案：A。
按位异或就是在同一位置上，当两者相同时，结果位为 0，当两者不同时，结果位为 1。也就是说：0^0=0，1^1=0，1^0=1，0^1=1。
对于本题而言，可以首先计算高位的异或值：F^A=1111^1010=0101，0101 转换为十进制后，该值为 5，符合条件的答案里面只有选项 A，从而排除了选项 B、选项 C 和选项 D 的可能性。
【真题 145】 有如下代码：

```
int func(int x)
{
    int countx = 0;
    while (x)
    {
        countx++;
        x = x&(x - 1);
    }
    return countx;
}
```

假设 x 的值为 65530，那么 func(x)的返回值是（　　）。

A．20　　B．16　　C．100　　D．14

答案：D。

解答本题的关键在于理解 x=x&(x−1)这条语句的作用。运算符&是一个二进制的运算符号，表示的是二进制的与操作，二进制与操作具有如下性质：只有当参与运算的两位同时为“1”时，其运算结果才为“1”，否则，其运算结果为 0：0&0=0，0&1=0，1&0=0，1&1=1。例如，十进制数 10，其二进制表示为 1010，当它与十进制数 9（二进制表示为 1001）执行&运算时，其结果为 1010 & 1001 = 1000。

对于表达式 x&(x−1)而言，其结果到底是什么呢？x 会不断地与比它小 1 的数进行与运算，每执行一次 x = x&(x−1)操作，会将 x 用二进制表示时最低位的一个 1 变为 0，因为 x−1 的二进制是把 x 的二进制的最低位的 1 变成 0。这段代码的目的是就是计算 x 的二进制表示中 1 的个数。65530 对应的二进制表示为 1111 1111 1111 1010，对应的二进制中有 14 个 1。所以，选项 D 正确。

【真题 146】 有如下代码：

```
int fun(int a)
{
    a = (1 << 5) - 1;
    return a;
}
```

fun(21)的结果是（　　）。

A．10　　B．8　　C．5　　D．31

答案：D。

函数的返回值与传递的参数无关。本题中，语句 1<<5 相当于执行了 2 的 5 次方，值为 32，最终的结果为 32−1= 31。所以，选项 D 正确。

【真题 147】 以下能够实现“判断一个非零整数 x 是否是 2 的幂”功能的表达式是（　　）。

A．x&(～x)　　B．x&(!x)　　C．x&(x−1)　　D．x&(−x)

答案：C。

如何判断一个数是 2 的幂，主要是要找出 2 的幂次方的数的特点。通常，1 个数乘以 2 就是将该数左移 1 位，而 2 的 0 次幂为 1，所以，2 的 n 次幂（就是 2 的 0 次幂 n 次乘以 2）就是将 1 左移 n 位，这样我们知道，如果一个数 n 是 2 的幂，则其对应的二进制数中只有一位是 1，必然有 n & (n − 1)为 0。（在求 1 个数的二进制表示中 1 的个数的时候说过，n&(n−1)去掉 n 的最后一个 1）。因此，判断一个数 n 是否为 2 的幂，只需要判断 n&(n−1)是否为 0 即可。所以，选项 C 正确。

【真题 148】 交换两个变量的值，不使用第三个变量。即 a=3，b=5，交换之后 a=5，b=3。

答案：有两种解法，一种为求和法，一种为异或法。

方法名	方法描述	备注
求和法	a = a + b; b = a − b; a = a − b;	可能会溢出

异或法	a = a^b;
	b = a^b;
	a = a^b;
		该方法只能针对变量类型为 int 和 char 的情况

【真题 149】 有如下代码：

```
int foo(int x)
{
    return x&-x;
}
```

在 32 位系统中，函数 foo(2^31-3)的值是（　　）。

A．0　　B．1　　C．2　　D．4

答案：C。

为了解答本题，需要掌握下面两个知识点：

1）算术运算符比位运算符的运算优先级高，也就是说 x&-x 等价于 x&(-x)，2^31-3 等价于 2^(31-3)。

2）异或操作的规则如下：给定数字 a 和 b，如果 a、b 两个值不相同，则异或结果为 1，否则，异或结果为 0。

本题中 foo(2^31-3)的参数为 2^31-3，x 的值为 2^31-3=2^28，在 32 位系统中，转换为二进制数表示为 0000 0000 0000 0000 0000 0000 0001 1110，-x 的二进制表示为 0111 1111 1111 1111 1111 1111 1110 0010，所以，x&-x 的结果的二进制表示为 0000 0000 0000 0000 0000 0000 0000 0010，将该二进制数转换为十进制数为 2。因此，选项 C 正确。

【真题 150】 有如下代码：

```
int c = 23;
printf("%d\n", c&c);
```

上述代码的运行结果为（　　）。

A．0　　B．46　　C．23　　D．以上都不对

答案：C。

按位与运算符&是一个位运算符，当参与运算的位的值都为 1 时，运算结果为 1，否则，运算结果为 0。

本题中，变量 c 的初值为 23，由于表达式 c&c 中参与运算的两个数的值是一样的，所以，当执行完&运算后，其结果也是 c，即为 23。所以，选项 C 正确。

【真题 151】 &和&&的区别是什么？

答案：以下将分别对&与&&进行说明。

&是按位与操作符，a&b 是把 a 和 b 都转换成二进制数后，然后再进行按位与的运算。而&&为逻辑与操作符，a&&b 就是当且仅当两个操作数均为 true 时，其结果才为 true，只要有一个为 false，a&&b 的结果就为 false。

此外，&&还具有短路的功能，在参与运算的两个表达式中，只有当第一个表达式的返回值为 true 的时候，才会去计算第二个表达式的值，如果第一个表达式的返回值为 false，则此时&&运算的结果就为 false，同时，不会去计算第二个表达式的值，例如：if(i!=0 && i++>10)，当 i 的值为 0 的时候，表达式 i!=0 的返回值为 false，因此，此时将不会执行第二个表达式

i++>10 的判断。

【真题 152】 有如下代码：

```
unsigned short A = 10;
printf("~A = %u\n", ~A);
char c = 128;
printf("c=%d\n", c);
```

程序的输出是（　　）。

A．10,128　　B．−11,−128　　C．4294967285,−128　　D．−11,128

答案：C。

【真题 153】 表达式 4&7 的运算结果是（　　）。

A．4　　B．1　　C．6　　D．7

答案：A。

&（与）是二进制与操作运算符，其功能是参与运算的两数对应的二进位进行与操作。只有对应的两个二进位均为 1 时，结果位才为 1，否则为 0。

本题中，十进制数字 7 的二进制表示为 0111，十进制数字 4 的二进制表示为 0100，这两个制数相与的结果为 0100，对应的十进制数为 4。所以，选项 A 正确。

【真题 154】 x 为整型，请用位运算实现 x%8。

答案：(x<<28)>>28。

有人觉得 x&7 可以，但是当 x 为负数的时候，这种方法就得不到正确的结果了。

【真题 155】 执行“int x=1;int y=~x;”语句后，y 的值为（　　）。

答案：−2。

假设在机器中，int 型变量占用 2 个字节，那么，整数 1 的二进制表示是 0000 0001 ，～为按位取反运算符，所以，对 x 取反后，0000 0001 变为 1111 1110，在计算机中整数用补码形式表示，正数的补码是它本身，而负数的补码则是原数值除符号位按位取反再加一，由补码求原数值也是按位取反再加一，那么 1111 1110 除符号位按位取反再加一变成 1000 0010，即−2。

【真题 156】 有如下代码：

```
int func(int x){
    int count=0;
    while (x)
    {
        count++;
        x=x&(x−1);//与运算
     }
    return count;
}
```

当 x=9999 时，函数的返回值是（　　）。

A．8　　B．9　　C．10　　D．12

答案：A。

一个数与这个数减 1 的结果进行按位&运算，结果为：这个数二进制数最右边的 1 变为 0。所以，该函数的本质为计算一个数的二进制中 1 的个数，由于十进制数 9999 的二进制表示为 10011100001111，共有 8 个 1。所以，选项 A 正确。

1.11 sizeof

【真题 157】 在 32 位计算环境下，定义有语句 int *p=new int[10]，那么 sizeof(p)的值为（ ）。

A．4　　B．10　　C．40　　D．8

答案：A。

sizeof 是 C/C++语言的关键字，它以字节的形式给出了其操作数的存储大小。对于本题而言，p 是一个指针，在 32 位环境下，指针只占用 4 个字节。所以，选项 A 正确。

如果题目改成 int p[10]，那么 sizeof(p)的值则是 40，因为此时 p 是一个数组类型，数组中存放了 10 个 int 类型的元素，在 32 位环境下，每个 int 类型的变量占用 4 个字节，因此，这个数组将会占用 40 个字节。

所以，本题的答案为 A。

引申：在程序员笔试中经常会考察结构体变量的 sizeof 的值，而这类题求职者经常容易做错。下面重点介绍 struct 的 sizeof 值的求解方法。

struct 是一种复合数据类型，其构成元素既可以是基本数据类型，例如：int、double、float、short、char 等，也可以是复合数据类型，例如：数组、struct、union 等数据单元。

一般而言，struct（结构体）的 sizeof 是所有成员对齐后长度相加，而 union（联合体）的 sizeof 取最大的成员变量长度。

在结构体中，编译器为结构体的每个成员按其自然边界（alignment）分配空间。各个成员按照它们被声明的顺序在内存中顺序存储，第一个成员的地址和整个结构体的地址相同。

字节对齐也称为字节填充，它是 C/C++编译器的一种技术手段，主要是为了在空间与复杂度上达到平衡。简单地讲，是为了在可接受的空间浪费的前提下，尽可能地提高对相同运算过程的最少（快）处理。字节对齐的作用不仅便于 CPU 的快速访问，使 CPU 的性能达到最佳，同时合理地利用字节对齐可以有效地节省存储空间。例如 32 位计算机的数据传输值是 4 字节，64 位计算机的数据传输值是 8 字节，这样，在默认的情况下，编译器会对 struct 进行（32 位机）4 的倍数或（64 位机）8 的倍数的数据对齐。对于 32 位机来说，4 字节对齐能够使 CPU 访问速度提高，例如一个 long 类型的变量，如果跨越了 4 字节边界存储，那么 CPU 要读取两次，这样效率就低了，但需要注意的是，如果在 32 位计算机中使用 1 字节或者 2 字节对齐，不仅不会提高效率，反而会使变量访问速度降低。

在默认情况下，编译器为每一个变量或数据单元按其自然边界条件分配空间。一般地，可以通过下面的方法来改变缺省的边界条件：

1）使用伪指令#pragma pack (n)，C 语言编译器将按照 n 个字节对齐。

2）使用伪指令#pragma pack ()，取消自定义字节对齐方式。

3）另外，还有如下的一种方式：_attribute((aligned (n)))，让所作用的结构体成员对齐在 n 字节自然边界上。如果结构体中有成员的长度大于 n，则按照最大成员的长度来对齐。_attribute_ ((packed))，取消结构在编译过程中的优化对齐，按照实际占用字节数进行对齐。

例如以下数据结构：

```
struct test
{
```

```
        char x1;
        short x2;
        float x3;
        char x4;
};
```

由于编译器默认情况下会对 struct 作边界对齐，结构的第一个成员 x1，其偏移地址为 0，占据了第 1 个字节，第二个成员 x2 为 short 类型，其起始地址必须 2 字节对界，因此，编译器在 x2 和 x1 之间填充了一个空字节。结构的第三个成员 x3 和第四个成员 x4 恰好落在其自然边界地址上，在它们前面不需要额外的填充字节。在 test 结构体中，成员 x3 要求 4 字节对界，是该结构所有成员中要求的最大边界单元，因而 test 结构的自然边界条件为 4 字节，所以编译器在成员 x4 后面填充了 3 个空字节。整个结构所占据空间为 12 字节。

再例如有以下数据结构：

```
struct s1
{
    short d;
    int a;
    short b;
}a1;
```

在 32 位机器下，short 型占 2 个字节，int 型占 4 个字节，所以，为了满足字节对齐的要求，变量 d 除了自身所占用的 2 个字节外，还需要再填充 2 个字节，变量 a 占用 4 个字节，变量 b 除了自身占用的 2 个字节，还需要 2 个填充字节，所以，最终 s1 的 sizeof 值为 12。

字节对齐的细节和编译器实现相关，但一般而言，满足以下三个准则：

1）结构体变量的首地址能够被其最宽基本类型成员的大小所整除。

2）结构体每个成员相对于结构体首地址的偏移量（offset）都是成员大小的整数倍，如有需要编译器会在成员之间加上填充字节。

3）结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节。

需要注意的是，基本类型是指前面提到的像 char、short、int、float、double 这样的内置数据类型，这里所说的“数据宽度”就是指其 sizeof 的大小，在 32 位机器上，这些基本数据类型的 sizeof 大小分别为 1、2、4、4、8。由于结构体的成员可以是复合类型，所以，在寻找最宽基本类型成员时，应当包括复合类型成员的子成员，而不是把复合成员看成是一个整体，例如一个结构体中包含另外一个结构体成员，那么此时最宽基本类型成员不是该结构体成员，而是取基本类型的最宽值。但在确定复合类型成员的偏移位置时，则是将复合类型作为整体看待，即复杂类型（如结构体）的默认对齐方式是它最长的成员的对齐方式，这样在成员是复杂类型时，可以最小化长度，达到程序优化的目的。

【真题 158】 在 32 位计算环境下，定义有语句 char str[] = "abcde";，那么 sizeof(str)的值为（　　）。

A．1　　B．4　　C．5　　D．6

答案：D。

sizeof 是 C/C++语言中的关键字，它以字节的形式给出了其操作数的存储大小。

本题中，语句 char str[] = "abcde";在内存中实际存储为一个字符数组：{‘a’,‘b’,‘c’,‘d’,‘e’,‘\0’}，最容易忽视的就是字符串末尾的结束符‘\0’。同时，需要将 strlen 与 sizeof 进行区别，strlen 执行的是一个计数器的工作，它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描，直到碰到第一个字符串结束符‘\0’为止，然后返回计数器值，很显然，本题中，字符串 str 的 strlen 的值为 5，而字符串 str 的 sizeof 的值为 6，

因为除了实际能看到的字符外，数组中还存储了字符串的结束符‘\0’，这个结束符也需要占用一个字节的存储空间。因此，选项 D 正确。

【真题 159】 在某 32 位系统下，C++程序如下所示：

```
char str[] = "http://www.tianya.com"（长度为 21）
char *p = str;
sizeof (str) = ？（1）
sizeof (p) = ？（2）
void foo(char str[100])
{
    sizeof(str) = ？（3）
}
void *p = malloc(100);
sizeof (p) = ？（4）
```

（1）、（2）、（3）和（4）处分别填写的值为（　　）。

A．22, 22, 100, 100　　B．4, 4, 4, 4　　C．22, 4, 4, 4　　D．22, 4, 100, 4

答案：C。

本题中，str 是字符数组类型，数组的长度为 22（包括 21 个字符和字符串结束符‘\0’），sizeof(str)表示的是这个数组类型占用的空间，所以，值是 22。而 p 是一个指向字符的指针，在 32 位的系统下，指针变量占用的存储空间为 4 个字节，因此，sizeof(p)的值为 4；在 C/C++语言中，当数组类型作为参数传递的时候，都会被转变为指针类型，因此，foo 函数中的 str 实际上是一个指向字符的指针（在参数传递的过程中数组会退化为指针），因此，第二个 str 的 sizeof 的值为 4。对于 void* p 而言，p 还是一个指针类型，在 32 位的系统下，任何指针类型的数据都会占用 4 个字节，因此，sizeof(p)=4。所以，选项 C 正确。

【真题 160】 在 C 语言中，有数组定义如下：char array[]="China";，则数组 array 所占用的空间为（　　）。

A．4 个字节　　B．5 个字节　　C．6 个字节　　D．7 个字节

答案：C。

在 C/C++语言中，字符串都是以字符‘\0’作为结束符。因此，字符串“China”在内存中实际存储的值为{‘C’,‘h’, ‘i’,‘n’,‘a’,‘\0’}，一共占用 6 个字节。所以，选项 C 正确。

【真题 161】 有以下程序：

```
void main(void)
{
    char a[7] = "a0\0a0\0";
    int i, j;
    i = sizeof(a);
    j = strlen(a);
    printf("%d, %d\n", i, j);
}
```

程序运行后的输出结果是（　　）。

A．2，2　　B．7，6　　C．6，2　　D．7，2

答案：D。

本题中，sizeof(a)的功能是求出字符串中的字符占用存储空间的大小，本题中，由于字符数组 a 的长度定义为 7，而每个 char 类型的长度在 32 位计算机下为 1 个字节，i 的值为 7。函数 strlen(a)的功能是求出字符串 a 的长度，而每个字符串都以‘\0’为字符串的结束标记，当遇到第一个‘\0’字符时，strlen 就认为字符串结束了，j 的值为 2。所以，选项 D 正确。

【真题 162】 有如下代码：

```
char *string_a = (char *)malloc(100*sizeof(char));
char string_b[100];
```

在 64 位平台机器下，sizeof(string_a)与 sizeof(string_b)的值分别是（　　）、（　　）。

A．8，100　　B．100，8　　C．100，100　　D．8，8

答案：A。

本题中，string_a 是一个字符指针，不管它指向的空间有多大，它本身的空间是固定的。在 64 位机器下，一个指针的大小是 8，单位是字节。而 string_b 是一个字符数组，长度为 100，在 64 位机器下，每个 char 占用 1 个字节，string_b 的 sizeof 为 100。所以，选项 A 正确。

【真题 163】 有如下代码：

```
int b[][3] = {{1},{3,2},{4,5,6},{0}};
```

在 32 位机器下，sizeof(b)的值为（　　）。

A．4　　B．12　　C．28　　D．48

答案：D。

本题中，变量 b 为一个二维整型数组，该二维数组规定了列的数量为 3，而行的数量却没有显式写明，根据数组中内容可知，该数组一共有 4 行，每行又有 3 列。根据约定，当初始化数据的语句中没有写明元素的值时，该值默认为 0。所以，语句 int b[][3] = {{1},{3,2},{4,5,6},{0}} 等价于语句 int b[ ][3] = {{1, 0, 0},{3,2, 0},{4,5,6},{0, 0, 0}}。

在 32 位机器下，每个 int 型变量占 4 个字节，二维数组 b 的 sizeof 值 sizeof(b)=4*3*4=48。所以，选项 D 正确。

【真题 164】 32 位机器上定义如下结构体：

```
struct xx
{
    long long _x1;
    char _x2;
    int _x3;
    char _x4[2];
    static int _x5;
};
int xx::_x5;
```

此时，sizeof(xx)的大小是（　　）。

A．19　　B．20　　C．15　　D．24

答案：D。

求解各种类型数据变量的 sizeof 值是一个常考的题型。对于基本数据类型而言，这种问题较为简单。以下是一些常见的结构类型的 sizeof 大小（备注：在 32 位计算机下）。

```
int x;
cout << sizeof(long long) << endl;//8
cout << sizeof(char) << endl;//1
cout << sizeof(int) << endl;//4
cout << sizeof(long) << endl;//4
```

```
cout << sizeof(float) << endl;//4
cout << sizeof(x) << endl;//4
```

以上类型都属于基本数据类型，但如果遇到复杂结构了，例如结构体、结构体嵌套结构体、派生类、带虚函数、空类等，它们的 sizeof 该如何求解呢？本题中，由于成员变量_x5 是静态变量，所以，可以不用管它，其次是要考虑字节对齐的问题。对于结构体中没有含有结构体变量的情况，有两条原则：

1）结构体变量中成员的偏移量必须是成员大小的整数倍。

2）结构体的最终大小必须是结构体最大的基本类型的整数倍。

x1 的偏移量是 0，长度是 8，符合；x2 的偏移量是 8，长度是 1，符合；x3 的偏移量是 9，长度是 4，不符合，需要在 x2 之后填充 3 字节使得 x3 的偏移量达到 12；x4 的偏移量是 16，长度是 2，符合；此时总长度为（8)+(1+3)+(4)+(2)=18，而最大简单类型为 long long 长度为 8，需要在 x4 之后再填充 6 字节，使得总长度达到 24 可被 8 整除。因此 sizeof(xx)的结果为 24。在 Visual Studio 2010 环境下运行结果也为 24，正好印证了这一结论。所以，选项 D 正确。

【真题 165】 有如下代码：

```
typedef union
{
    long i; int k[5];
    char c;
} DATE;
struct data
{
    int cat;
    DATE cow;
    double dog;
} too;
DATE max;
```

则语句 printf("%d", sizeof(too)+sizeof(max));的执行结果是什么？

答案：52。

本题中，DATE 是一个 union（联合）类型，union 的 sizeof 通常取其成员变量的最大值，同时，地址的对齐方式还得考虑 union 中其他的数据成员。对于本题而言，其成员变量里面最大的变量类型是 int[5]，在 32 位机器下，每个 int 型变量占用 4 个字节空间，int[5]占用 4*5 个字节，即 20 个字节，所以，sizeof(max)的值为 20。由于 union 中有 long 类型的成员存在，而 long 占用 8 个字节，因此，union 的对其方式为 8，为了满足对齐的要求，需要填充 4 个字节，sizeof(max)=24。data 是一个 struct（结构体）类型，在 32 位机器下，每个变量分开占用空间，依次为 int(4) + DATE(24)+ 填充 4 个字节 + double(8) = 40，所以，sizeof(too)+sizeof(max)=24+40=64。

本题中需要注意的一个问题就是字节对齐。本书中已在其他章节对字节对齐进行了详细介绍，此处不再赘述。

1.12 函数

【真题 166】 定义有函数 int func(int i)，它的实现如下：

```
int func(int i)
{
    if(i > 1)
        return i*func(i−1);
    else
        return 1;
}
```

那么调用 f(5)方法的返回值为（　　）阶乘计算。

A．5　　B．15　　C．20　　D．120

答案：D。

在分析递归的问题时，最重要的是找出递归表达式与递归结束的条件。本题中，递归调用的过程为 func(5)=5*func(4)=5*4*func(3)=5*4*3*func(2)=5*4*3*2*func(1)=5*4*3*2*1=120。这个函数实际上是用来求解 i 的阶乘。所以，选项 D 正确。

【真题 167】 以下代码是用来计算 100 以内的素数的个数，请把相应的空填上。

```
struct prime_number_node
{
    int prime_number;
    prime_number_node* next;
};
int calc_prime_number()
{
    prime_number_node* list_head = new prime_number_node();
    list_head−>next = NULL;
    list_head−>prime_number = 2;
    prime_number_node* list_tail = list_head;
    for (int number = 3; number < 100; number++)
    {
        int remainder;
        prime_number_node* cur_node_ptr = list_head;
        while (cur_node_ptr != NULL)
        {
            remainder = number%cur_node_ptr−>prime_number;
            if (remainder == 0)
            {
                //1
            }
            else
            {
                //2
            }
        }
        if (remainder != 0)
```

```
                {
                    prime_number_node* new_node_ptr = new prime_number_node();
                    new_node_ptr−>prime_number = number;
                    new_node_ptr−>next = NULL;
                    list_tail−>next = new_node_ptr;
                    //3
                }
            }
            int result = 0;
            while (list_head != NULL)
            {
                result++;
                prime_number_node* temp_ptr = list_head;
                list_head = list_head−>next;
                //4
            }
            return result;
        }
```

答案：1、2、3、4 四行代码依次为：

```
break;
cur_node_ptr = cur_node_ptr−>next;
list_tail = list_tail−>next;
delete temp_ptr;
```

对于空白 1，一旦满足 remainder==0，说明这个数不是素数，此时没有必要判断这个数能否被其他数整除了，可以直接跳出循环，继续判断下一个数是否为素数，因此，这里的代码需要能够跳出循环，跳出循环的代码为 break。

对于空白 2，本题判断一个数 n 是否为素数的方法为：只需把 n 被小于 n 的每一个素数去除，如果都不能被整除，那么 n 就是一个素数。在 2 处需要取下一个素数，然后判断能否被当前遍历到的数整除，取链表下一个结点的代码为：cur_node_ptr = cur_node_ptr−>next。

对于空白 3，当代码执行到 3 的时候，说明当前遍历到的元素值是素数，需要把这个素数添加到链表的尾部。由于链表增加了一个新的结点，因此，需要更新指向尾结点的指针：list_tail = list_tail−>next 或 list_tail= new_node_ptr。

对于空白 4，由于这个函数只需返回素数的个数，因此，在函数结束的时候，需要把存储素数的链表中结点占用的空间给释放掉。释放结点占用的空间的代码为：delete temp_ptr;。

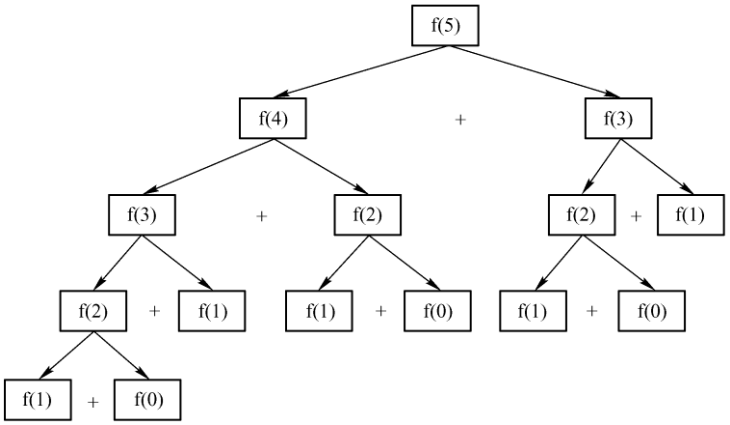
【真题 168】 当 n=5 时，下列函数的返回值是（　　）。

```
int foo(int n){
    if (n<2)
        return n;
    else
        return foo(n − 1) + foo(n − 2);
}
```

A．5　　B．7　　C．8　　D．10

答案：A。

对于递归调用而言，最重要的就是要找到递归调用结束的条件。以本题为例，递归调用的过程如下图所示。



按照上述调用过程执行，当调用到函数 f 的参数满足递归的结束条件 n<2 时，递归结束，此时会返回函数调用的结果。对于本题而言，最后递归调用的结果就是所有叶子结点的和。由于 f(1)=1，f(0)=0，叶子结点总共有 5 个 f(1)，因此，这个函数调用的结果为 5。递归调用结束后程序的运行过程如下图所示。