

首先，我要感谢本书的作者能够选择这样一个备受大家关注的话题作为题材，同时也要感谢电子工业出版社能够将此书大力推广。要知道，程序员和面试可能是现在因特网上大家最为关心的字眼之一了——不，应该是之二。正好，本书详尽地描述了程序员应该学些什么、做些什么，然后应该如何面对烦人的但又必不可少的面试过程。当然，如果您不是程序员，我依然认为本书会对您的职业生涯有所帮助，相信我吧。

哦，忘了介绍我自己了。我是孔文达，毕业于北京某某大学材料系，现任微软（中国）有限公司顾问。咦？怎么读材料的从事上 IT 工作了？这说来可话长了。一句概括的话，就是：努力加机遇。当然，我并不想长篇大论应该如何努力及如何把握机遇，我想说的是和本书密切相关的的话题——面试。

其实，无论是程序员还是其他任何行业的任何职位，面试过程都大同小异，无非就是提交简历、电话面试、面谈、得到 offer 等这一系列过程。当然，这其中每一步都很重要！简历要写得得体、漂亮，尽量突出自己的优势，屏蔽自己的劣势。电话面试还好一些，因为只是电话交谈，所以您也许会更好地把握自己的语言。面谈是最关键的一步，而且如果您准备不充分的话，一定会紧张。紧张，就有可能出现错误。不过还好，大多数面试官都可以接受面试者的紧张，只要不是太过分，问题就不大了。一般来说，中型或大型企业的面试都不止一轮，有些甚至有十几轮。就拿微软来说吧，官方渠道需要 12 轮面试，内部推荐也需要 4 轮，而且是一票否决式。就是说，有一个面试官说你不行，你就没戏了。怎么搞定所有的面试官呢？当然有很多技巧，但最重要的一条就是：面试官是个活生生的人，他/她一定有个人偏好，在你见到面试官时，尽可能在最短的时间内——最好是在他/她了解你之前——了解他/她，因地制宜地与他/她展开对话。最后一点，最好不要极其地、非常地、十分地想得到某个职位，这有可能会使你失态，抱一平常心有时会得到意想不到的效果。

Preface

这本书写得非常好，它非常详尽地描述了作为一名程序员应该为面试准备些什么和注意些什么。也许您现在还用不到它，先看看吧，指不定什么时候就用上了呢！这不是杞人忧天，而是未雨绸缪！

孙立达

Microsoft

技术顾问 漫软全国 TOP3 讲师

【在正式加入微软（中国）有限公司前，
曾任微软外聘顾问及特约讲师 7 年，并在
北京中达金桥科技开发有限公司
(微软在国内最大的技术及培训合作伙伴)

任人力资源部总监及副总裁。】

第二届微软十佳金牌讲师

首届微软十佳金牌讲师

MLC 认证讲师

微软护航专家

CIW 认证讲师 (CIW CI)

CIW 网络安全分析大师 (CIW)

华为网络工程师 (HCNE)

HP-UNIX 系统及网络管理员 (HP-UX Administrator)

Cisco 认证网络专家 (CCNA)

微软认证讲师 (MCT)

微软认证数据库管理员 (MCDBA)

微软认证系统工程师 (MCSE)

微软认证专家 (MCP)

微软销售专员 (MSS)

作为刚毕业的学生和正在找工作的程序员，当你应聘一份程序设计、软件开发方面的工作时，招聘方总会安排一次笔试以考查你的程序设计能力。我们写作这本书的目的就是希望能帮助大家顺利地通过这类面试。

程序设计面试，时间大约是一小时，试题范围包括计算机知识、程序设计、逻辑分析等。与传统的面试不同，程序设计面试题以程序设计题、IQ 智力题及各种与计算机相关的技术性问题为主。我们收集了大量知名企技术类笔试中的常见试题，深入浅出地对试题的解答思路进行了分析和指导，不仅能帮助求职者快速复习有关知识，也对如何给面试官留下一个良好而深刻的印象进行了指导。希望能把在技术面试中取得的宝贵经验毫无保留地传授给读者，以便使求职者对程序设计面试中的常见题型应付自如，从而获得一份真正的高薪工作！

通过对本书的学习，读者能够掌握关键性的面试技巧，发现和完善有关试题的最佳解决方案，以应对不利的局面。读完本书，读者会了解负责招聘工作的 HR 主管对程序设计面试都有哪些想法，公司将依据怎样的标准评估应聘人员在程序设计面试中的表现，公司将出哪方面的题目来测试你，以及不同的公司在程序设计方面的侧重点有何不同。通过对书中代表性例题举一反三地钻研，相信读者无论以后遇见什么样的面试题目，都可以应对自如，在就业路上一帆风顺。

《程序员面试宝典》不同于同类书籍的主要特点是：

◆ 细

国外的面试书籍和中国国情不是很相符。中国的软件企业比较小，涉及的方面比较细、比较基础，比如常会考基础性编程的问题，如 const、sizeof、类型转化等。一些国内公司（北大方正、北大青鸟等）的面试题，多半是浅显的基础的问询。换句话说就是，他们考得很细。本书把面试中国内公司最易考到的基础考点放在“第 2 部分 C/C++ 程序设计”里面，希望能切切实实帮读者解决实际问题。

◆ 专

面试题是通过一道题考查一个专类的能力。从被面试者的角度来讲，你能了解许多关于出题者或监考者的情况。从面试者的角度来讲，一个测试也许能从多方面揭示应试者的素质。本书将考查的方面分类：嵌入

Foreword

式编程类，软件类，面向对象类，模板类……通过面试题目提升你对这些方面知识的掌握能力，以达到有的放矢、举一反三的效果。

◆ 广

求职者应聘的职位一般有3种：Bnet、Btest和Se，分别代表网络工程师、测试工程师和软件开发人员。市面上流行的面试书籍仅侧重第三类软件开发人员而忽略网络工程师和测试工程师。现实情况是诸如趋势科技、华为3COM、Cisco等公司对网络方面的考题日趋增加。本书就这一方面给出了详细论断，并结合大量考题分析题目特点，给出应试方案。

此外，随着全球五百强企业进入中国大陆，外企对UML、设计模式、软件度量等方面试题的喜爱有增无减。本书在这些方面加以改进，以适应市场需求。

◆ 新

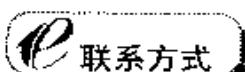
程序员面试或者说应届毕业生面试的题目年年翻新，岁岁不同。比如说现在很少有ASP和VB的面试例题，而新兴语言C#却成为面试热点。同时本书对时下流行的综合面试题、智力测试题、英语面试、电话面试加以分门别类，以大量实际案例总结应对面试的方案，以达到“一册在手，临危不乱”的效果。

◆ 真

本书的所有面试题都是近两年各大公司面试题的汇总，内容非常新，可以算做面试者求职前的一份全真模拟。我们希望营造一种真实的面试氛围，同时把如何写好简历及如何在面试过程中应答的实际感悟融汇在书中，指引读者走上理想的工作岗位。

本书不是一本万能书籍，但却肯定是你工作求职的好助手和好伙伴！

编著者



咨询电话：(010) 68134545 88254160

电子邮件：support@fecit.com.cn

服务网址：<http://www.fecit.com.cn> <http://www.fecit.net>

通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

第1部分 求职过程

求职的过程就是一个提高和认识自我的过程。最后的成功往往在于你本人一丝一毫的努力当中，也许真的像电影《肖申克的救赎》里面说的那样：“得救之道，就在其中。”

第1章 应聘求职	3
1.1 应聘渠道	3
1.2 应聘流程	4
第2章 简历书写	5
2.1 简历注意事项	5
2.2 简历模板	8
第3章 3种考试	13
3.1 笔试	13
3.2 电话面试	15
3.3 面试	16
第4章 职业生涯发展规划	19
4.1 缺乏工作经验的应届毕业生	19
4.2 更换工作的程序员们	21

第2部分 C/C++程序设计

为什么要选择C系的语言呢？这是因为各大公司的编程语言绝大多数是C系的语言，虽然Java也占很大的比重，可是C++相对于Java来说更有区别度——C++是那种为每一个问题提供若干个答案的语言，远比Java灵活。

第5章 程序设计基本概念	25
作为一个求职者或应届毕业生，公司除了对你的项目经验有所询问之外，最好的考量办法就是你的基本功，包括你的编程风格、你对赋值语句、进階语句、类型转换、数据交换等程序设计基本概念的理解。	
5.1 赋值语句	25
5.2 i++	27
5.3 编程风格	29

Contents

5.4 类型转换	30
5.5 螺旋队列	34
5.6 a、b 交换.....	35
5.7 C 和 C++ 的关系.....	36
5.8 程序设计其他问题	37
第 6 章 预处理、const 与 sizeof	39
6.1 宏定义	39
6.2 const	41
6.3 sizeof.....	42
6.4 内联函数和宏定义	55
第 7 章 指针与引用	57
指针是 C 语言的特色，是 C 和 C++ 的精华所在，也是 C 和 C++ 的一个十分重要的概念。	
7.1 指针基本问题	57
7.2 传递动态内存	65
7.3 函数指针	71
7.4 指针数组和数组指针	73
7.5 迷途指针	77
7.6 指针和句柄	79
第 8 章 循环、递归与概率	81
8.1 递归基础知识	81
8.2 典型递归问题	83
8.3 打靶	84
8.4 字符子串	89
8.5 循环语言	91
8.6 0-1 背包	94
8.7 概率	95
第 9 章 STL 模板与容器	97
9.1 向量容器	98
9.2 泛型编程	103

9.3 模板	105
第 10 章 面向对象	109
有这样一句话：“编程是在计算机中反映世界”，我觉得再贴切不过。	
面向对象（Object-Oriented）对这种说法的体现也是最优秀的。	
10.1 面向对象的基本概念	110
10.2 类和结构	111
10.3 成员变量	113
10.4 构造函数和析构函数	117
10.5 拷贝构造函数和赋值函数	119
10.6 多态的概念	123
第 11 章 继承与接口	127
整个 C++程序设计全面围绕面向对象的方式进行。类的继承特性是 C++ 的一个非常重要的机制。这一章的内容是 C++面向对象程序设计的关键。	
11.1 覆盖	128
11.2 私有继承	130
11.3 虚函数继承和虚继承	137
11.4 多重继承	141
11.5 检测并修改不适合的继承	142
11.6 纯虚函数	146
11.7 COM	148
第 12 章 位运算与嵌入式编程	151
12.1 位制转换	151
12.2 嵌入式编程	157

第 3 部分 数据结构和设计模式

随着外企研发机构大量内迁我国，在外企的面试中，软件工程的知识，包括设计模式、UML、敏捷软件开发，以及.NET 技术和完全面向对象语言 C# 的面试题目将会有增无减。

Contents

第 13 章 数据结构基础	167
面试时间一般有 2 小时，其中至少有约 20~30 分钟是用来回答数据结构相关问题的。链表、数组的排序和遍历是必考的内容之一。	
13.1 单链表	167
13.2 双链表	173
13.3 循环链表	176
13.4 队列	177
13.5 堆栈	180
13.6 树	185
13.7 排序	185
第 14 章 字符串	203
14.1 整数字符串转化	203
14.2 字符数组和 strcpy	205
14.3 数组越界	210
14.4 数字流和数组声明	212
14.5 字符串其他问题	213
第 15 章 设计模式	219
“地上本没有路，走的人多了也就成了路”。设计模式如同此理，它是经验的传承，并非体系。它是被前人发现，经过总结形成的一套某一类问题的一般性解决方案，而不是被设计出来的定性规则。	
15.1 设计模式	220
15.2 软件工程	235
15.3 C#基础	237
15.4 C#继承	240
15.5 C#委托	249
15.6 ASP.NET	251

第 4 部分 操作系统、数据库和网络

本部分主要介绍求职面试过程中出现的第三个重要的板块——操作系统、数据库和网络知识。这些内容虽不是面试题目中的主流，但仍然具有重要的意义。

第 16 章 操作系统	257
16.1 进程	257
16.2 图形学	259
16.3 内存管理	261
16.4 DOS、Linux、UNIX	271
第 17 章 数据库与 SQL 语言	273
17.1 数据库理论	273
17.2 SQL 语言	276
17.3 SQL 语言客观题	279
17.4 SQL 语言主观题	282
第 18 章 计算机网络及分布式系统	285
18.1 网络结构	285
18.2 TCP/IP	288
18.3 SNMP	292
18.4 网络其他问题	295

第 5 部分 综合面试题

英语面试、电话面试和智力测试，是除技术面试之外的另三大模块。本部分教你如何精心地为这些内容做好准备，以让你在整个面试过程中的表现更加完美。

第 19 章 英语面试	301
<p>这里的英语面试不同于普通的英语面试。就一个程序员而言，最好能够做到用英文流利地介绍自己的求职经历，这是进外企非常重要的一步。有些问题即使是中文你都很难回答，更何况是用英文去回答。但是求职过程本身就是个准备的过程，精心地准备等待机会，机会总是垂青于那些精心准备的人。</p>	
19.1 面试过程和技巧	301
19.2 关于工作 (About Job)	303
19.3 关于个人 (About Person)	308
19.4 关于未来 (About Future)	311

Contents

19.5 其他建议（Other Tips）	312
19.6 英文面试常用词汇	313
19.6.1 工作经历相关词汇	313
19.6.2 个人资料相关词汇	314
19.6.3 个人品质相关词汇	315
19.6.4 学历相关词汇	316
19.6.5 离职原因相关词汇	318
第 20 章 电话面试	319
20.1 电话面试之前的准备工作	319
20.2 电话面试交流常见问题	320
第 21 章 智力测试	329
智力测试是企业招聘时有可能出现的一个环节，事实上，IT 企业求职招聘还是主要以基本的程序设计及数据结构为主。智力测试是考验人的综合智商、逻辑能力的过程，本身是很难复习和准备的。这些年来，智力测试的一个新的趋势是和编程及算法结合起来。	
21.1 关于数字的智力测试	329
21.2 关于推理的智力测试	332
21.3 关于时间的智力测试	333
附录 A 简历模板	339
附录 B 面试经历总结	351

第1部分

求职过程

The procedure of applying for a job

本部分将详述作为一个计算机相关专业的应届毕业生或程序员，在求职面试中应该注意的一些问题。

古人云：凡事预则立，不预则废。机会都是垂青有准备的人的。为了得到一份满意的工作，大家一定要对整个求职过程有清醒的了解。把能够预见的、必须做的事情早一些做完，这样在大规模招聘开始的时候就可以专心地为面试做准备。求职过程中会发生很多预料不到的事情，当你的计划被这些事情打乱之后，要做的事会越堆越多，一步落后，步步落后。如果能够尽早把能做的事做完，即便有计划外事件发生，也不会产生太严重的影响。努力地使事态的发展处在自己能控制的范围之内，这样无论发生任何事都能有应对之策。

第 1 章

应聘求职

每年的二三月份，都是应届生求职、在职人员跳槽的高峰期。对于即将成为程序员的应届毕业生们，在求职过程中怎样确定目标公司和目标职位；对于已经是程序员的跳槽大军，是按照技术路线发展自己的职业生涯，还是走向管理岗位继续自己的职业道路，或者是改变自己的发展轨迹；大家在求职过程中要注意哪些细节？这些都是大家所关心的话题。

国内的 IT 业比国外兴起得晚，而且目前还没有权威的适合中国本土程序员的职业生涯发展规划。因此，国内流行的“35 岁退休说”其实是一种误解，只要我们好好规划自己的职业生涯，提高自己的技术水平、沟通技巧和管理能力，就能够获得更高，更好的职位，完全可以像国外程序员一样工作到 60 岁再退休。

让我们先从应聘流程中的注意事项，这个轻松却又容易被人忽略的话题开始吧。

1.1 应聘渠道

对于应届生而言，可以选择参加校园宣讲会的形式投递简历。如下图所示，这是 EMC 公司 2006 年校园宣讲会日程表。我们可以选择就近的城市参加它的宣讲会并投递简历。

第 1 章

应聘求职

每年的二三月份，都是应届生求职、在职人员跳槽的高峰期。对于即将成为程序员的应届毕业生们，在求职过程中怎样确定目标公司和目标职位；对于已经是程序员的跳槽大军，是按照技术路线发展自己的职业生涯，还是走向管理岗位继续自己的职业道路，或者是改变自己的发展轨迹；大家在求职过程中要注意哪些细节？这些都是大家所关心的话题。

国内的 IT 业比国外兴起得晚，而且目前还没有权威的适合中国本土程序员的职业生涯发展规划。因此，国内流行的“35 岁退休说”其实是一种误解，只要我们好好规划自己的职业生涯，提高自己的技术水平、沟通技巧和管理能力，就能够获得更高，更好的职位，完全可以像国外程序员一样工作到 60 岁再退休。

让我们先从应聘流程中的注意事项，这个轻松却又容易被人忽略的话题开始吧。

1.1 应聘渠道

对于应届生而言，可以选择参加校园宣讲会的形式投递简历。如下图所示，这是 EMC 公司 2006 年校园宣讲会日程表。我们可以选择就近的城市参加它的宣讲会并投递简历。

· EMC 2006 校园宣讲会行程表

日期	时间	城市	学校	地点
2006-04-02	19:00-21:00	北京	北京大学	演讲交流中心报告厅
2006-04-04	19:00-21:00		清华大学	就业指导中心多功能厅
			复旦大学	复旦体育馆
2006-04-05	19:00-21:00	南京		
	19:00-21:00	成都	成都电子科技大学	图书馆
2006-04-06	19:00-21:00	西安	西安交通大学	就业指导中心报告厅
	19:00-21:00	上海	上海交通大学	光南路4号行政楼3
2006-04-07	19:00-21:00		上海复旦大学	复旦体育馆

招聘会投递的简历是“纸”的简历。尽管现在网上投递电子简历的方式大行其道，但是“纸”的简历仍然有着其无可比拟的优势。HR（人力资源经理）拿到“纸”的简历，相比一份电子简历更有一种亲切感，重视程度也较电子简历高一些。

第二种方式是投递电子简历，可以通过公司的电子信箱和公司网站招聘信息栏（数据库），以及各大招聘门户网站，如 ChinaHR 或者智联招聘等，来投递自己的电子简历。

1.2 应聘流程

应聘时的一个完整流程如下图所示。



通常一个外企的应聘流程是一个很长的过程，有时甚至可以达到一二个月。还是以 EMC 公司为例，如下图所示，让我们看一下他们的应聘流程。

· EMC 2006 校园招聘流程计划安排

流程	简历筛选 及面试	预面 宣讲会	现场 宣讲会	笔试	第一轮 面试	第二轮 面试	Offer
2006年1月中旬							
2006年2月下旬							
2006年3月上旬							
2006年4月中旬							
2006年4月下旬							
2006年5月上旬							
2006年5月中旬							
2006年5月下旬							

由上图可知，比较正规的企业的应聘流程一般分为 5 个部分：简历筛选，笔试，第一轮面试（含电话面试或邮件面试），第二轮面试，发 offer。下面的章节我们会就这 5 个步骤做详细的阐述。

第 2 章

简历书写

据 统计，80%的简历都是不合格的。不少人事管理者抱怨收到的许多简历在格式上很糟糕。简历应该如何做到在格式上简洁明了，重点突出？求职信应该如何有足够的内容推销自己？如何控制长度，言简意赅？相信读了本章你会对简历的撰写有一个新的认识。

2.1 简历注意事项

1. 简历不要太长

一般的简历普遍都太长。其实简历内容过多反而会淹没一些有价值闪光点。而且，每到招聘的时候，一个企业，尤其是大企业会收到很多份简历，工作人员不可能都仔细研读，一份简历一般只用1分钟就看完了，再长的简历也超不过3分钟。所以，简历要尽量短。我们做过一个计算，一份中文简历压缩在2页左右就可以把所有的内容突出了。1页显得求职者过于轻浮，三四页就太多了。

简历过长的一个重要原因是有的人把中学经历都写上去，其实这完全没有必要，除非你中学时代有特殊成就，比如在奥林匹克竞赛中获过奖。一般来说，学习经历应该从大学开始写起。

很多学生的求职简历都附了厚厚一摞成绩单、荣誉证书的复印件，其实简历上可以不要这些东西，只需要在简历上列出所获得的比较重要的荣誉。如果企业对此感兴趣，会要求求职者在面试时把这些带去。

2. 简历一定要真实客观

求职简历一定要按照实际情况填写，任何虚假的内容都不要写。即使有的人靠含有水分的简历得到面试的机会，面试时也会露出马脚的。千万不要为了得到一次面试机会就编写虚假简历。被招聘方发现后，你几乎就再也没有机会进入这家公司了。而且对于应届生来说，出现这种情况后，还有可能影响到同校的其他同学。

北京某高校一位计算机专业本科毕业的女孩子，简历上写的是 04 年毕业，但面试中被发现她是 05 年毕业的，而且没有任何工作经验。这女孩儿比较诚实，说是同学教她这样做的。

她这种编制虚假简历的做法应该否定，因为谁都不希望被骗。作为面试官来说，首先希望应聘者是一个诚实的人。我希望她在听到同学那个不明智的建议时，首先不应选择这种做法，其次要尽力阻止其他人这样做。因为，就像面试官代表公司形象一样，她在某种程度上也代表了她所毕业的学校来参加面试！最起码在她传达给 HR 的信息中，与她同专业应届生的简历可信度较差。

3. 不要过分谦虚

简历中不要注水并不等于把自己的一切，包括弱项都要写进去。有的学生在简历里特别注明自己某项能力不强，这就是过分谦虚了，实际上不写这些并不代表说假话。有的求职学生在简历上写道：“我刚刚走入社会，没有工作经验，愿意从事贵公司任何基层工作。”这也是过分谦虚的表现，这会让招聘者认为你什么职位都适合，其实也就是什么职位都不适合。

4. 简历要写上求职的职位

求职简历上一定要注明求职的职位。每份简历都要根据你所申请的职位来设计，突出你在这方面的优点，不能把自己说成是一个全才，任何职位都适合。**不要只准备一份简历，要根据工作性质有侧重地表现自己。**如果你认为一家单位有两个职位都适合你，可以向该单位同时投两份简历。

在我曾看到的一些简历中，经常有如下的错误：简历上描述的多为 Windows 操作系统下 C/C++ 开发经验，但申请的目标职位为“Linux 操作系统下的 C/C++ 开发工程师”。这样当然不容易得到应聘职位的面试

机会。还有就是去应聘 ERP、CRM 方面的职位，而简历里却大肆强调自己在嵌入式编程方面的优势。就算你非常优秀，你对这个企业还是没有用处。

有些简历里面没有详细的项目描述及责任描述，在责任描述栏仅仅填写“软件开发”或者在工作业绩栏仅仅填写“可以”两字。这样的信息传达无疑是不成功的。

作为求职的开始，我们要编写一份或者几份有针对性的简历，也就是按照对方的要求突出自己相关经历。只要你的优势与招聘方的需要吻合，并且比其他应聘者突出的话，你就胜利了。

5. 在文字、排版、格式上不要出现错误

用人单位最不能容忍的事是简历上出现错别字或是在格式、排版上有技术性错误，以及简历被折叠得皱巴巴、有污点，这会让用人单位认为你连自己求职这样的事都不用心，那工作也不会用心。

6. 简历不必做得太花哨

一般来说简历不必做得太花哨，用质量好一些的白纸就可以了，尽量用 A4 规格的纸。曾看到过一份简历封面上赫然写着 4 个大字“通缉伯乐”，给人的感觉就像是在威胁用人单位。现在学生简历中比较流行做封面的形式，其实没有必要，这会增加简历的厚度，实际上完全可以不用封皮。

7. 简历言辞要简洁直白

大学生的求职简历很多言辞过于华丽，形容词、修饰语过多，这样的简历一般不会打动招聘者。简历最好多用动宾结构的句子，简洁直白。

8. 不要写上对薪水的要求

在简历上写上对工资的要求要冒很大的风险，最好不写。如果薪水要求太高，会让企业感觉雇不起你；如果要求太低，会让企业感觉你无足轻重。对于刚出校门的大学生来说，第一份工作的薪水不重要，不要在这方面费太多脑筋。

9. 不要写太多个人情况

不要把个人资料写得如此详细，姓名、电话是必需的，出生年月可

有可无。如果应聘国家机关、事业单位，应该写政治面貌。如果到外企求职，这一项也可省去，其他都可不写。

10. 不要用怪字怪体

我见过一份简历，用中空字体，还有斜体字。这些都是很忌讳的。试想一个 HR 挑了一天的简历，很累了，还要歪着头看你的简历。你想你的胜算能有多大？其实用简单的宋体 5 号字就很好了，不用标新立异。

2.2 简历模板

一份合格的求职简历应该包括以下内容。

姓名、电话（或其他联系方式）等个人资料应该放在简历的最上面，这主要是为了方便用人单位与求职者及时取得联系。紧接着是毕业的学校、专业和时间。下面应该注明应聘的职位和目标。

接下去就是简历上最重要的部分：工作经历。对于初出茅庐的大学生来说，这部分包括勤工助学、课外活动、义务工作、参加各种各样的团体组织、实习经历和实习单位的评价，等等。这部分内容要写得详细些，指明你在社团中、在活动中做了哪些工作，取得了什么样的成绩。用人单位要通过求职者的这些经历考查其团队精神、组织协调能力等。

兴趣爱好也最好列上两三项，用人单位可就此观察求职者的工作、生活态度。

如果应聘外资企业、大的跨国公司，一定要附上英文简历，而且要把最近的经历放在最前面，简历前面最好附一封推荐信。一定要认真对待英文简历的编写，因为它会泄露你的实际英文水平。

下面是作者的一份简历模板。

求职简历

个人介绍:

姓名: 欧立奇

性别: 男

出生日期: 1980/07/03

学校及专业: 西北大学计算机系软件与理论专业

学历: 硕士

移动电话: 13096964884

电子邮件: jinder24@263.net

IT && 英语技能:

1. 软件结构设计, 需求分析能力
2. 精通 C/C++、C#, 精通 SQL
3. 熟悉 Windows 开发平台, 精通.NET 体系
4. 熟悉 Delphi 开发工具, 熟悉 UML 统一建模语言
5. 深入理解面向对象的思想, 并能熟练地应用于具体的软件设计开发工作中
6. 英语水平: 国家六级

项目经验(近期):

2005/5—2005/9

由×××出版社约稿编写《Visual C# .NET 案例×××》，出任第一作者，目前该书已经出版 (ISBN 7-121-01722-9)。本书主要通过具体的实例介绍如何运用 Visual C# 编程工具开发实际的应用程序，从基本应用到高级处理都有涉及，包括基础设计、图像处理、多媒体应用、系统文件处理、数据库基本处理、网络处理、网络与数据库高级应用、综合实例共 8 个章节。

2005/7—2005/9

与实验室人员合作，在基于 ASP.NET+SQL 2000 的平台上开发西北大学网络选课系统程序。目前已交付使用。

2005/8—2006/2

与别人合作编写《×××指南: C# 设计师之路》，出任第二作者，目前全书已经完成。完成“第 5 章 面向对象的编程”、“第 6 章 接口”、“第 7 章 数组”、“第 9 章 线程”的编写工作。

2004/9—2005/1

与实验室人员合作，在基于 Delphi+SQL 2000 的平台上开发西北大学人事管理系
统程序。该项目是西北大学基金项目，目的为完成西北大学教职员信息的统一规范化
管理。系统分为教师科、劳资科、人事科、人才交流中心等几部分，实现了各个部
门之间的信息统一化协调管理。系统由 Delphi、Power Designer、MS SQL 开发完成。

工作经验：

2005/9—2005/12

北大青鸟 ACCP 培训师

2003/7—至今 西北大学

就读于西北大学计算机系，在学期间，与实验室小组合作完成网上选课系统（ASP.NET+SQL 2000）和人事管理系统（Delphi+SQL 2000）的研发工作，以及教务管理系统（PowerBuilder 8+SQL 2000）的测试工作。

应聘西北工业大学计算机三级网络任课老师，教授国家计算机三级相关内容。
北大青鸟 ACCP 讲师。

建立个人主页蓝色菲林（www.bluefeeling.cn），整个网站包括“我的技术社区”和“我的项目介绍”，完全由 C# 编写完成。

2003/02—2003/08

天津 LG 电子科技有限公司吸尘器部门研发部技术人员，并从事吸尘器量产管理工作。

2002/07—2003/01

重庆信息产业部 44 所 7 研究室研究人员，并从事 CCD&CMOS 生产检测工作。

1998/09—2002/07

就读于西北大学物理系，获得理学学士学位。

奖学金：

中国石油奖学金优秀学生一等奖。

其他特长：

文学和美术功底较好，擅长网页制作，Photoshop 和 Dreamweaver 水平较好。擅长表述，能够胜任教学工作。

个人评价：

我无法掩饰我对这份工作的渴望——一份有科研挑战的职位。

我一向认为理想分为两类，一类是实现自己的理想，另一类理想则通过自身得到实现。理想之于我则两者兼而有之，并稍稍倾向后者。作为老师，我喜欢传道、授业、解惑，形成一套自己的理论并潜移默化我的学生，在写书时常为一个理论如何能够清楚地讲述而煞费苦心；同样，作为科研工作者，我也被 C++ 的华贵多彩而吸引，那是真正的逻辑之美。此外，很多时候为了项目的完满，必须具备一种不破楼兰终不还的决心和不积跬步无以至千里的恒心。

最后，我谦和、谨慎，富于团队精神。希望您能给我这样一个机会展示自己。
谢谢。

欧立奇

2006.3.18

Resume**Information**

Name: LiQi Ou

Gender: Male

School & Major: Northwest University Master of Computer Software and Theory

Mobile phone: 13096964884

Email: jinder24@263.net

Education: Master

Career Objective

To obtain a challenging position as a software engineer with an emphasis in software design and development.

Computer Skills && English Skills

Languages: C, C++, C#, Delphi, XML, UML

Systems: Windows, .NET

Database: MS SQL Server

English Skills: CET-6

Recent Experience

09/2004--01/2005 Northwest University Personnel Managing System

Northwest University Personnel Managing System is a system of auto-manage the personnel information. It is designed by PowerDesigner. The project is a C/S architecture system. It is based on a Microsoft SQL database, and the UI is developed by Delphi 7. In this project, I designed the schema of database, programmed database connectivity using Delphi 7 and ADO.

05/2005---09/2005 Publishing House Of × × × ×

VC# Casus Design was published by Publishing House Of × × × × in 2005.10. As the author I finished most of leading articles, including GDI+, Multi-Media, Database, Web System, and so on. It's composed of 8 chapters, 50 programs. I concentrated my energy on The book. In addition, I attained much pleasure in the process of writing. Now I'm going to write another book named **White-collar Obtain Employment Guiding-- Flying with C#**.

04/2004--09/2004 Northwest Network Course-selected System

Based on ASP.NET+SQL 2000 we finished Northwest Network Course-selected System. Everybody in this University can select, cancel, query course in network. The project is a B/S architecture system; the code is developed by Visual C#, and runs on

the .NET plat. In this project, I used the ADO interface which is provided by the database program. And after that, I joined the testing of whole system.

Self Assessment

I am an active, innovative man, a good team-worker, with rich IT knowledge and developing experience. I am fit for a job of programming in an IT company.

第 3 章

3 种 考 试

笔试，电话面试，面试，是顺利求职的 3 个过程。三关全过才能顺利签约，只要有一关没能通过，就会被“刷”掉。

3.1 笔试

我认为笔试是程序员面试 3 个过程中最重要的一个环节，也是最难以提升的一个环节。本书中主要叙述的也是程序员的笔试经历。不论你有多么大的才干，多么广博的知识，如果未能通过笔试，则无缘下面的进程。下面是一个表，描述了各种 IT 公司笔试所考题目的类型。

公司名称	公司类型	笔试内容
Trend	网络公司	C++ 或 Java、网络、数据库、设计模式、智力测试、英语阅读
SAP	软件咨询、ERP、CRM	C++、概率问题、设计模式、智力测试
Advantech	硬件、自动化公司	C++（尤其是指针问题），嵌入式编程
Synopsys	电子类公司	C++（尤其是指针问题），数据结构
NEC	综合软件公司	C、数据结构
金山	综合软件公司	C++或 PHP、数据库、数据结构、设计模式
华为	通信公司	C++或 Java、数据结构、数据库
中兴	通信公司	C++或 Java、数据结构、数据库
VIA	硬件公司	C++（尤其是指针问题），嵌入式编程
华为 3COM	网络公司	C++、网络
SPSS	数据统计软件公司	C++（尤其是继承、多态问题），数据结构

(续表)

公司名称	公司类型	笔试内容
Sybase	数据库公司	C++、Linux、UNIX
Motorola	网络公司	C++、网络
IBM	综合软件公司	C++或Java
Oracle	数据库公司	Java、数据库
HP	综合软件公司	C++
腾讯	综合软件公司	C++
Yahoo	综合软件公司	C++或Java或C#
微软	综合软件公司	C++、数据结构、智力测试
神州数码	金融软件公司	C++或Java、数据结构、数据库（SQL）
大唐移动	通信公司	C++
Siemens	数据通信公司	C++、设计模式
Grapecity	软件公司	C++、C#、智力测验

根据上表，对各大IT公司的笔试题目和所考的内容，我们可以窥见一斑，并得出以下几个结论。

1. 语言的偏向性

综合上表所示，IT公司笔试在编程语言上有一定偏向性，以C、C++为主或者是以Java为主。语言本身并没有什么高低贵贱之分，但相对来说，考到Delphi或者VB的可能性很小。作为应届毕业生，如果只是学过VB、VF却从来没有接触过C系语言，则在笔试中是比较吃亏的。

2. 英语的重要性

我所经历过的外企的笔试卷子基本上都是英语试卷，无论从出题到解答，都是让你用英文去回答，所以必须有很好的英文阅读能力，这也是外企招人对英语非常看重的原因。其实也不需要一定通过六级，但一定要有相对多的单词量，能够看懂考题的意思。然后按自己的想法组织语言来描述就可以。

国内企业一般对外语要求不是很看重，题目也是中文的。如果不想要进外企的话，也不用特别准备英语。

3. 淡看智力测试

之所以要强调这一点，是和市面上过度强调外企智力测试有关。实际上笔者参加过的微软等外企笔试，智力测试只占很小的比例，约3%~

5%左右。而华为、神州数码等国内IT企业基本上没有智力测试，完全是技术考试。所以奉劝大家不要把精力都投在所谓的外企智力测试上面，还是应该以准备技术方面的笔试为主。

4. 有的放矢准备简历

不同的公司会考不同的内容，这就像高中时准备不同科目考试的差别。比如说神州数码不会考嵌入式编程，而VIA考设计模式的可能性很小。一般有点儿偏“硬”的IT公司会对C++中指针的用法、数据结构考得比较多。偏“软”的企业会对设计模式、模板着重一些。所以本书分得很细，力求对各种IT公司的笔试题目做一个详尽的阐述。

作为求职者，**笔试前你要首先搞清这个公司基本情况，它是做什么的，它有什么产品，你是学什么方面的。有的放矢才能折桂。**

5. 纸上写程序

搞计算机的肯定不习惯在纸上写程序，然而技术面试的时候这是面试官最常用的一招。让写的常见程序有：数据结构书上的程序，经典C程序（strcmp、strcpy、atoi……），C++程序（表现C++经典特性的）。第一次在面试官眼皮底下在纸上写程序，思路容易紊乱。建议大家事先多练习，找个同学坐在边上，在他面前写程序，把该同学当成面试官。经过多次考验，在纸上写程序就基本不慌了。

每次面试总会有些问题回答得不好，回来之后一定要总结，把不懂的问题搞明白。一个求职者就碰到两家公司问了同样的问题，第一次答不出，回去没查，第二次又被问到，当然这是很郁闷的事情。

3.2 电话面试

电话面试主要是对简历上一些模糊信息的确认、之前经历的验证、针对应聘职位简单技术问题的提问，以及英文方面的考查。

由于模式的限制，电话面试时间不会很长。在这个环节中，**一定要表现得自信、礼貌、认真、严肃**，这样会在声音上给对方一个良好的印象。如果声音慵懒，语气生硬，除非是技术题目及英文方面表现得足够好，否则很难予以平衡。

在回答电话面试的问题时，不要过于紧张，要留心对方的问题，这些问题也许在当面的面试中还会再出现。如果对方在电话面试中要求你做英文的自我介绍，或者干脆用英文和你对话，那在电话面试结束后一定要好好准备英文面试的内容。

笔者曾经参加过 Thoughtworks、Sybase、SAP、麒麟原创等公司的电话面试。外企一般都会要求你做一个英文自我介绍和一些小问题，总的来说不会太过涉及技术方面，因为用英语来描述技术对国人而言还是有一定困难的。国企会问到技术问题，我就曾被问到如何在 C++ 中调用 C 程序、索引的分类等技术问题，回答基本上要靠平时的积累和对知识的掌控能力。电话面试的具体内容可参见第 18 章。

3.3 面试

一个比较好的面试是能够问出求职者擅长哪方面而哪方面不足的面试。如果面试官针对求职者不足之处穷追猛打，或是炫耀自己的才能，这是不足取的。

对于求职者而言，面试是重点环节，要守时是当然的了。如果不能按时参加面试，最好提前通知对方。着装上不需要过分准备，舒服、干净就好了。一般的 IT 公司对技术人员都不会有很高的着装要求。虽然着装不要求，但精神状态一定要好。饱满的精神状态会显得你很自信。

有笔试的话（有时笔试和面试是同时进行的，即面试官会在提问后请你回答并写下详细描述），也无非是与应聘职位相关的技术考查或者英文考查，如英汉互译等。应视你应聘职位的等级进行准备。

应聘初级职位，会针对你的编程能力和以往的项目经验进行重点的考查。如果面试官针对你做的某个项目反复提问，那么你就需要注意了，要么面试官在这个方面特别精通，要么就是未来的职位需要用到这方面的技术。我们应该抱着一种诚恳的态度来回答，对熟悉的技术点可以详细阐述，对于不熟悉的部分可以诚实地告诉面试官，千万不要不懂装懂。不过，我们同意可以引导与面试官的谈话，把他尽量引导到我们所擅长的领域。在 SPSS 公司面试时，在回答完面试客单链表逆置和拷贝构造函数问题之后，我把话题引入了我所擅长的设计模式方面，这是一种谈话

的艺术。

应聘中级职位，不但会考查代码编写，而且会对软件架构或相关行业知识方面进行考查。代码编写方面，主要以考查某种编程技巧来判断你对代码的驾驭能力。比如某国际知名软件公司经常会让面试者编写 malloc 或 atoi 函数。越是简单的函数越能考验应聘者的编码能力。你不但要实现功能，而且还要对可能出现的错误编写防御性代码，这些经验都需要在实际编程过程中积累。

应聘高级职位，应聘者肯定对技术或某个行业有相当程度的了解，这时主要是看你与职位的契合程度、企业文化的配比性（即将人力资源及成本配比作为服务体系的重要组成部分，将公司企业文化中核心理念及价值观作为客户服务的重要媒介）及整体感觉。应聘管理职位的话，考查的更多是管理技巧、沟通技巧和性格因素。架构师一般会考查行业背景与软件架构方面的知识，比如 UML 或建模工具的使用等；技术专家的职位则会针对相关技术进行深度考查，而不会再考查一般性的编码能力。

面谈的时候，要与面试官保持目光接触，显示出你的友好、真诚、自信和果断。如果你不与对方保持目光接触，或者习惯性地瞟着左上角或者右上角的话，会传达给对方你对目前话题表现冷淡、紧张、说谎或者缺乏安全感的感觉。

如果对方向到的某个问题你不是很熟悉，有一段沉默的话，请不要尴尬和紧张。面试过程中允许沉默，你完全可以用这段时间来思考。可

的第一感觉填写就可以了。在几十道甚至上百道题目中，都有几道题是从不同角度考查一个方向的，凭猜测答题反而会前后有悖。

当然，要先看清楚题目，搞清楚是选择一个最适合你自己的，还是描述得最不恰当的。在通过面试之后，如果有多家公司和职位的 Offer 可以选择的话，我们可以将公司的行业排名、公司性质、人员规模、发展前景、企业文化、培训机制，结合自身的生活水平、职业生涯发展规划来进行排列，选出最适合自己的公司和职位。

建议准备一个日程本，记录每一次宣讲会、笔试和面试的时间，这样一旦公司打电话来预约面试，可以马上查找日程本上的空闲时间，不至于发生时间上的冲突。每投一份简历，记录下公司的职位和要求，如果一段时间以后（1个月或更长）有面试机会，可以翻出来看看，有所准备。**根据不同的公司，准备不同的简历**，千万不要一概而论，不同的公司 care（在意）的东西不一样。每参加完一次笔试或面试，把题目回忆一下，核对一下答案，不会做的题目更要好好弄懂。同学们之间信息共享，总有人有你没有的信息。如果投了很多份简历，一点儿回音都没有，你得好好看看简历是否有问题，增加一些吸引 HR 眼球的东西。

第 4 章

职业生涯发展规划

在一般情况下，我们工作一年之后，对自己的喜好及擅长都有了更加深刻的了解，这时会有较为明确的职业发展规划。

4.1 缺乏工作经验的应届毕业生

即将毕业的学生们自己的目标职位很模糊，只要是计算机相关的工作都想试一下。但是现在公司看重的除了学生的基本素质，即沟通能力、团队协作、学习能力、外语水平等之外，也会关注应届毕业生在校及实习经历中与目标职位相关的经验。假设与导师做的课题或者实习中接触到 J2EE 企业级开发，那么在应聘时寻找一份相关要求的工作就更为容易。而这样的经历去找一份 C/C++ 开发的职位可能就略微难些。

上海某高校的一位学生在课余时间开发了一个基于校园网内部的搜索引擎。比起商用的搜索引擎，其搜索效率、数据量不算出色，但是该生通过编写自己的搜索引擎，详细了解了网络编程、网页爬虫等领域的知识。这个搜索引擎也表现出了他专业技能的水平，从而为他赢得了前往国际某著名网络公司应聘的机会。

所以，在大学期间，我们可以通过参加创新杯比赛、著名软件公司举行的各种编程大赛、各种技术社团的活动来增加编程经验，以获取公司对你专业技能的肯定。各种编程大赛中获得的名次、实践大赛中的作品，都可以作为工作经验的替代。

通过校园招聘招人的大公司，一份有分量的简历只是第一步。有分

量指的是成绩尚可，有让他们感兴趣的实习经历，有一定的获奖经历，担任过一定的职务，英语能力还行。这仅仅是第一步。它能让你从众多应聘者中被选出来参加初试，接下来就看你的真正功力和造化了。

初试的要点是基本功扎实，自信乐观，英语交流能力不错，够聪明，够机灵。基本功扎实并且聪明尤为重要。某位毕业生参加 Sybase 公司面试，过了印度技术官的英语技术面试，第二天参加他们的 Aptitude Test（智商测试），误认为是态度测试（Attitude Test），结果没发挥好。智商测试通常让你在很短的时间内做大量的逻辑题和智力题。不要在前面的题目上浪费太多时间，后面的题目往往更加简单。

另一位求职者通过了微软公司的笔试和电话面试，后来去参加了正式的面试。一连 3 轮，面试官全都是微软高级技术经理。面对这么高级别的面试官，求职者难免紧张。3 轮全英文面试，写了 6 个程序，不算难，但是考得很细，注重求职者的逻辑思维能力、反应能力和编程技巧。写完程序之后马上设计测试用例。或许是是没有参加过特别正规的项目开发的缘故，他表现一般，有几个程序有疏漏，面试官加以提醒，虽然最后能够改正，但是加重了他的紧张情绪，没能闯入下一轮。

没必要因为自己的学校而显得不够自信，只要打好技术功底，多参加正规的实践项目，找工作的時候自然会顺利。此外在求职过程中，整体形势和个人形势没有必然联系。整体形势好了，个人形势未必好。往往整体形势好了，个人容易盲目乐观，在准备不充分的情况下，很容易被莫名其妙地淘汰。即使明年的整体形势比今年还要好，招人的公司比今年还要多，还是建议大家脚踏实地，做好充分的知识储备和心理准备，找工作绝对是一场硬仗。

有一种说法：80% 的 Offer 掌握在 20% 的牛人手中。每年 10 月、11 月应届毕业生刚刚开始找工作的時候，正是牛人们发威的時候，笔试、面试都有他们的份，到了发 Offer 的時候他们手中集中了很多好 Offer。这时候我们得摆正心态，尽自己最大努力，发挥出自己的最好水平就行了，不用太在意结果。晚些时候往往反而会有好 Offer。写论文的同时抽空复习一下基础的课程：数据结构、C、C++、TCP/IP、操作系统、计算机网络、UML、OOA&OOP、自己做过的项目知识，等等。不要怕笔试和面试，笔试得多了，感觉就来了。

可能你找到了工作，并且不止一个。但手头的 Offer 再多，也只能跟一家公司签约，面对的诱惑再多，也只能选择一个。不用羡慕那些手头有很多好 Offer 的人，他们其实很痛苦，这是一种甜蜜的烦恼。罗列出你最在意的方面，把几家公司做详细的比较（见下表）。做选择有时候很感性，理性的数据往往不如公司的一名普通员工给你的印象更能影响你的决定。

比较内容	权重指数	A公司	B公司
发展机会	20%		
公司前景	10%		
技术方向	10%		
培训机制	5%		
公司性质	5%		
人员规模	3%		
企业文化	5%		
行业排名	5%		
薪酬	20%		
...			

4.2 更换工作的程序员们

如果你是跳槽者中的一员，我们要明白**频繁跳槽对职业生涯发展是有害无益的**，招聘方也十分关注求职者的稳定性。一般来说，每份工作都要维持一年以上，能够在某家公司工作满 3 年，才会对公司所在行业及这家公司有比较深入的了解。决定更换工作时，我们要先问问自己要在哪个方向继续自己的职业生涯。假设目前你是某家公司的开发人员，要应聘更大规模公司的同等职位，我们应该注意下面两点。

首先，比起创业型公司，大公司的开发流程要求会更加规范和严格，有的时候我们必须放弃一些编程的习惯。严格的开发流程对文档的依赖性很大，我们必须做到文档优先。这样的一种环境，可能是初入大公司的程序员最难接受的一点。

其次，小公司里那种 Superman 型的程序员在大公司里很少见到。我

曾经听一个程序员朋友抱怨他们公司的架构师连 ASP 代码都不会写，其实这是很正常的事情。架构师的工作是将业务需求变成计算机软件的模块和类，他们不需要了解具体代码的编写，只需要分析几种软件平台之间的实现难度和效率差异就够了。当然，大公司也有所谓的技术高手，但这种技术高手并不是精通几种开发语言的“万能钥匙”，而是对某种技术有深入理解，能够解决深层次问题的人。

中国的 IT 界，“技则优而仕”的比较多。很多技术出身的人员做到管理岗位后，关注的仍然是技术细节。但实际上，人员的管理也是一门很大的学问。技术主管的个人风格会影响整个团队的氛围。如果主管不善沟通、只关心 Dead Line，那么整个团队将会毫无活力，主管的技术再高超也不会得到信服。如果主管善于沟通、关心下属，那么整个团队就会生机勃勃，即使加班也有劲头。

假设你已不想再做开发，想要转向测试或其他相关岗位，如实施、技术支持，甚至培训、售前等，那你一定要认真向目前在做这份工作的人员了解他们的实际职责与相关要求，确认是否可以接受转换岗位后带来的挑战。如果确定，则可以选择具有相同行业背景的目标职位，并且调整好自己的心理状态，给自己一段较长的时间来适应这种改变。刚开始时感觉无从下手或者有较大落差是很正常的，最起码要在半年之后才能证实你和这个岗位的匹配度。

如果你现在已经有了较为明确的职业生涯发展规划，推荐大家使用倒推法使之切合实际并行之有效。以一个普通程序员为例，我们可以首先为自己的目标设置一个年限，并列出实现这个目标所需要的专业技能，然后使用倒推法确定我们的阶段目标，直至将这个阶段目标倒推至一个月后，那它就会是一个很具体的目标了。只要你坚持去做，就会逐步实现自己的最终目标。

当然，除此之外，你还要时时关注业界动态，尽可能多地参加在职培训并且补充外语方面的技能。这样才能保持你继续前进的步伐。

当然，**最重要的是我们要把握好自己，把握好自己要走的路**。其实任何一个职位都需要我们努力工作，任何一份工作都无法“钦定”我们的终身。求职“just a job”而已。找不到，不用悲悲切切，找到了也不用狂喜。这只是人生中众多历练之一。

第 2 部分

C/C++程序设计

C / C + + p r o g r a m d e s i g n

本部分主要以 C/C++设计语言为基础，通过大量实际的例子分析各大公司 C/C++面试题目，从技术上分析问题的内涵。为什么要选择 C 系的语言呢？这是因为各大公司的编程语言绝大多数是 C 系语言，虽然 Java 也占很大的比重，可是 C++相对于 Java 来说更有区分度——C++是那种为每一个问题提供若干个答案的语言，远比 Java 灵活，所以面试考题绝大多数以 C/C++为主（或者是两套试题，C++ 或 Java，面试者可以选择）。

许多面试题看似简单，却需要深厚的基本功才能给出完美的解答。企业要求面试者写一个最简单的 strcpy 函数就可看出面试者在技术上究竟达到了怎样的水平。我们能真正写好一个 strcpy 函数吗？我们都觉得自己能，可是我们写出的 strcpy 很可能只能拿到 10 分中的 2 分。读者可从本部分中关于 C++的几个常用考点，看看自己属于什么样的层次。

第 5 章

程序设计基本概念

作为一个求职者或是应届毕业生，公司除了对你的项目经验有所问询之外，最好的考量办法就是你的基本功，包括你的编程风格，你对赋值语句、递增语句、类型转换、数据交换等程序设计基本概念的理解。当然，在考试之前你最好对你所掌握的程序概念知识有所复习，尤其是各种细致的考点要尤其加以重视。以下的考题来自真实的笔试资料，希望读者先不要看答案，自己解答后再与答案加以比对，找出自己的不足。

5.1 赋值语句

面试例题 1：What does the following program print? (下面程序的结果是多少?) [中国台湾某著名计算机硬件公司 2005 年 12 月面试题]

```
#include <iostream>
using namespace std;
int main()
{
    int x=2,y,z;
    x *=(y=z=5); cout << x << endl;
    z=3;

    x == (y=2); cout << x << endl;
    x =(y==z); cout << x << endl;
    x =(y&z); cout << x << endl;
    x =(y&&z); cout << x << endl;
    y=4;
    x=(y|z); cout << x << endl;
    x=(y||(z)); cout << x << endl;
```

```

    return 0;
}

```

解析：

$x *= (y=z=5)$ 的意思是说 5 赋值给 z，z 再赋值给 y， $x=x*y$ ，所以 x 为 $2*5=10$ 。

$x == (y=z)$ 的意思是说 z 赋值给 y，然后看 x 和 y 相等否？不管相等不相等，x 并未发生变化，仍然是 10。

$x = (y==z)$ 的意思是说首先看 y 和 z 相等否，相等则返回一个布尔值 1，不等则返回一个布尔值 0。现在 y 和 z 是相等的，都是 3，所以返回的布尔值是 1，再把 1 赋值给 x，所以 x 是 1。

$x = (y&z)$ 的意思是说首先使 y 和 z 按位与。y 是 3，z 也是 3。y 的二进制数位是 0011，z 的二进制数位也是 0011。按位与的结果如下表所示。

y	0	0	1	1
z	0	0	1	1
y&z	0	0	1	1

所以 y&z 的二进制数位仍然是 0011，也就是还是 3。再赋值给 x，所以 x 为 3。

$x = (y&&z)$ 的意思是说首先使 y 和 z 进行与运算。与运算是指如果 y 为真，z 为真，则 $(y \& \& z)$ 为真，返回一个布尔值 1。这时 y、z 都是 3，所以为真，返回 1，所以 x 为 1。

$x = (y|z)$ 的意思是说首先使 y 和 z 按位或。y 是 4，z 是 3。y 的二进制数位是 0100，z 的二进制数位也是 0011。与的结果如下表所示。

y	0	1	0	0
z	0	0	1	1
y z	0	1	1	1

所以 y|z 的二进制数位是 0111，也就是 7。再赋值给 x，所以 x 为 7。

$x = (y||z)$ 的意思是说首先使 y 和 z 进行或运算。或运算是指如果 y 和 z 中有一个为真，则 $(y || z)$ 为真，返回一个布尔值 1。这时 y、z 都是真，所以为真，返回 1。所以 x 为 1。

答案： 10, 10, 1, 3, 1, 7。

面试例题 2： What does the following program print? (下面程序的结果是

多少?) [中国某著名计算机金融软件公司 2005 年 12 月面试题]

```
#include <iostream>
using namespace std;

int Vac=3;
int main() {
    int Vac=10;
    ::Vac++;
    cout<<::Vac << endl;
    cout<<Vac << endl;
    return 0;
}
```

- A. 11 11 B. 11 4 C. 10 4 D. 4 10

解析: 这是一种全局变量和局部变量对比的问题。

答案: D。

5.2 i++

面试例题 1: What will be the output of the following code (assume the necessary include files are present)? (下面程序的结果是什么?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

```
int i=3, j=4;
i?i++:++j;
printf("%d %d\n", i, j);
```

- A. 3 3 B. 4 4 C. 3 4 D. 4 3

解析: i?i++:++j 的意思是说如果问号前面的 i 是真，则进行 i++，否则进行 ++j。因为 i 是 3，所以是真，执行 i++，j 不变化。i++后的结果是 4，j 不变仍然是 4。所以答案是 4 和 4。

如果 i?i++:++j 换做 0?i++:++j，则答案就换做 3 和 5 了。

答案: B。

面试例题 2: 以下代码的输出结果是_____。[中国某著名计算机金融软件公司 2005 年面试题]

```
int i=1, j=2;
int k = i++ + j;
cout << k << endl;
```

- A. 2 B. 3 C. 4 D. 5

解析： $i++ + j$ 是首先结合 $i++$ ，然后再 $+j$ ，但是 $i++$ 是事后计算，也就是说先算 $i+j$ 然后再 $i++$ ，所以 k 的和是 $1+2=3$ 。然后 i 才自增到2。如果是 $\text{int } k = ++i + j;$ ，则 i 先自增，然后 $+j$ ，结果为4。

答案：B

面试例题 3： $x=x+1$, $x+=1$, $x++$, 哪个效率最高? 为什么?

解析：

$x=x+1$ 最低，因为它的执行过程如下：

- (1) 读取右 x 的地址。
- (2) $x+1$ 。
- (3) 读取左 x 的地址。
- (4) 将右值传给左边的 x （编译器并不认为左右 x 的地址相同）。

$x+=1$:其次，其执行过程如下：

- (1) 读取右 x 的地址。
- (2) $x+1$ 。
- (3) 将得到的值传给 x （因为 x 的地址已经读出）。

$x++$;效率最高，其执行过程如下：

- (1) 读取右 x 的地址。
- (2) x 自增1。

答案： $x++$ 效率最高。

面试例题 4：What will be the output of the following C code? (以下代码的输出结果是什么?) [中国台湾某著名CPU生产公司 2005 年面试题]

```
#define product(x) (x*x)
int main()
{
    int i=3,j,k;
    j=product(i++);
    k=product(++i);
    printf("j=%d,k=%d",j,k);
    return 0;
}
```

解析： $\text{product}(i++) = i++ * i++$; $i=3$ ，所以 j 等于9，此时 i 已经累积为5。 $\text{product}(++i)$ 要求先累加 i ，累积后 i 等于7，所以 $\text{product}(++i)$ 的结果是49。

答案：9, 49。

面试例题 5：If there are "int a=5, b=3; ", the values of a and b are _____ and _____ after execute "!a&&b++;". (如果有“int a=5, b=3;”，则在执行“!a&&b++;”后 a 和 b 的值是_____和_____。) [中国某著名综合软件公司 2005 年面试题]

- A. 5, 3 B. 0, 1 C. 0, 3 D. 5, 4

解析：这是表达式运算问题。因为“!a”运算结束后，整个表达式的值已肯定为假，所以不必再去计算后面的式子。

答案：A。

5.3 编程风格

面试例题 1：We have two pieces of code , which one do you prefer, and tell why. (下面两段程序有两种写法，请告诉我你青睐哪种，为什么？) [美国某著名计算机嵌入式公司 2005 年 10 月面试题]

A.

```
// a is a variable
```

写法 1：

```
if( 'A'==a ) {
    a++;
}
```

写法 2：

```
if( a=='A' ) {
    a++;
}
```

B.

写法 1：

```
for(i=0;i<8;i++) {
    X= i+Y+J*7;
    printf("%d",x);
}
```

写法 2：

```
S= Y+J*7;
for(i=0;i<8;i++) {
    printf("%d",i+S);
}
```

答案：

A. 第一种写法'A'==a 比较好一些。这时如果把“==”误写做“=”的话，因为编译器不允许对常量赋值，就可以检查到错误。

B. 第二种写法好一些，将部分加法运算放到了循环体外，提高了效率。缺点是程序不够简洁。

5.4 类型转换

面试例题 1：下面程序的结果是什么？[中国台湾某著名 CPU 生产公司 2005 年面试题]

```
char foo(void)
{
    unsigned int a=6;
    int b=-20;
    char c;
    (a+b>6) ? (c=1) : (c=0);
    return c;
}
```

解析：unsigned int 类型的数据与 int 类型的数据相运算后，自动转化为 unsigned int 类型。因此 a+b 的值不是 -14，而是一个 unsigned int 类型的数 4294967382。

因此返回值是 1，与实际我们想要得到的结果不符。若两个操作数都不是 long 型而其中一个是 unsigned int 型，则另一个也被转换成 unsigned int 型，否则两个操作数一定都要是 int 型。

可以定义一个 int 类型的数接收 a+b 的值，如 int c=a+b。或者是对相加结果进行强制类型转换，如 int(a+b)。

这个问题测试你是否懂得 C 语言中的整数自动转换原则。我发现有些开发者极少懂得这些东西。无论如何，这道无符号整型问题的答案是输出 “>6”。原因是当表达式中存在有符号类型和无符号类型时。所有的操作数都自动转换为无符号类型。因此 -20 变成了一个非常大的正整数，所以该表达式计算出的结果大于 6。这一点对于频繁用到无符号数据类型的嵌入式系统来说是非常重要的。如果你答错了这个问题，你也就到了得不到这份工作的边缘。

答案：1。

扩展知识

C++ 定义了一组内置类型对象之间的标准转换，在必要时它们被编译器隐式地应用到对象上。

隐式类型转换发生在下列这些典型情况下。

1. 在混合类型的算术表达式中

在这种情况下最宽的数据类型成为目标转换类型，这也被称为算术转换（Arithmetic Conversion），例如：

```
int ival = 3;
double dval = 3.14159;
// ival 被提升为 double 类型: 3.0
ival + dval;
```

2. 用一种类型的表达式赋值给另一种类型的对象

在这种情况下目标转换类型是被赋值对象的类型。例如在下面第一个赋值中文字常量 0 的类型是 int。它被转换成 int* 型的指针表示空地址。在第二个赋值中 double 型的值被截取成 int 型的值。

```
// 0 被转换成 int* 类型的空指针值
int *pi = 0;
// dval 被截取为 int 值 3
ival = dval;
```

3. 把一个表达式传递给一个函数，调用表达式的类型与形式参数的类型不相同

在这种情况下目标转换类型是形式参数的类型。例如：

```
extern double sqrt( double );
// 2 被提升为 double 类型: 2.0
cout << "The square root of 2 is " << sqrt( 2 ) << endl;
```

4. 从一个函数返回一个表达式的类型与返回类型不相同

在这种情况下返回的表达式类型自动转换成函数类型。例如：

```
double difference( int ival1, int ival2 )
{
    // 返回值被提升为 double 类型
    return ival1 - ival2;
}
```

算术转换保证了二元操作符，如加法或乘法的两个操作数被提升为共同的类型，然后再用它表示结果的类型。两个通用的指导原则如下：

①为防止精度损失，如果必要的话，类型总是被提升为较宽的类型。

②所有含有小于整型的有序类型的算术表达式在计算之前其类型都会被转换成整型。

规则的定义如上面所述，这些规则定义了一个类型转换层次结构。我们从最宽的类型 long double 开始。

如果一个操作数的类型是 long double，那么另一个操作数无论是什么类型都将被转换成 long double。例如在下面的表达式中，字符常量小写字母 a 将被提升为 long double，它的 ASC 码值为 97，然后再被加到 long double 型的文字常量上：

```
3.14159L + 'a';
```

如果两个操作数都不是 long double 型，那么若其中一个操作数的类型是 double 型，则另一个就将被转换成 double 型。例如：

```
int ival;
float fval;
double dval;
// 在计算加法前 fval 和 ival 都被转换成 double
dval + fval + ival;
```

类似地，如果两个操作数都不是 double 型而其中一个操作数是 float 型，则另一个被转换成 float 型。例如：

```
char cval;
int ival;
float fval;
// 在计算加法前 ival 和 cval 都被转换成 double
cval + fval + ival;
```

否则如果两个操作数都不是 3 种浮点类型之一，它们一定是某种整值类型。在确定共同的目标提升类型之前，编译器将在所有小于 int 的整值类型上施加一个被称为整值提升（integral promotion）的过程。

在进行整值提升时类型 char、signed char、unsigned char 和 short int 都被提升为类型 int。如果机器上的类型空间足够表示所有 unsigned short 型的值，这通常发生在 short 用半个字而 int 用一个字表示的情况下，则 unsigned short int 也被转换成 int，否则它会被提升为 unsigned int。wchar_t 和枚举类型被提升为能够表示其底

层类型 (underlying type) 所有值的最小整数类型。例如已知如下枚举类型：

```
enum status { bad, ok };
```

相关联的值是 0 和 1。这两个值可以但不是必须存放在 char 类型的表示中。当这些值实际上被作为 char 类型来存储时，char 代表了枚举的底层类型，然后 status 的整值提升将它的底层类型转换为 int。

在下列表达式中：

```
char cval;
bool found;
enum mumble { m1, m2, m3 } mval;
unsigned long ulong;
cval + ulong; ulong + found; mval + ulong;
```

在确定两个操作数被提升的公共类型之前，cval found 和 mval 都被提升为 int 类型。

一旦整值提升执行完毕，类型比较就又一次开始。如果一个操作数是 unsigned long 型，则第二个也被转换成 unsigned long 型。在上面的例子中所有被加到 ulong 上的 3 个对象都被提升为 unsigned long 型。如果两个操作数的类型都不是 unsigned long 而其中一个操作数是 long 型，则另一个也被转换成 long 型。例如：

```
char cval;
long lval;
// 在计算加法前 cval 和 1024 都被提升为 long 型
cval + 1024 + lval;
```

long 类型的一般转换有一个例外。如果一个操作数是 long 型而另一个是 unsigned int 型，那么只有机器上的 long 型的长度足以存放 unsigned int 的所有值时（一般来说，在 32 位操作系统中 long 型和 int 型都用一个字长表示，所以不满足这里的假设条件），unsigned int 才会被转换为 long 型，否则两个操作数都被提升为 unsigned long 型。若两个操作数都不是 long 型而其中一个是 unsigned int 型，则另一个也被转换成 unsigned int 型，否则两个操作数一定都是 int 型。

尽管算术转换的这些规则带给你的困惑可能多于启发，但是一般的思想是尽可能地保留多类型表达式中涉及到的值的精度。这正是通过把不同的类型提升到当前出现的最宽的类型来实现的。

5.5 螺旋队列

面试例题 1：

```

    21  22……
    20   7   8   9   10
    19   6   1   2   11
    18   5   4   3   12
    17   16  15  14  13
  
```

看清以上数字排列的规律，设 1 点的坐标是(0,0)，x 方向向右为正，y 方向向下为正。例如，7 的坐标为(-1, -1)，2 的坐标为(0,1)，3 的坐标为(1,1)。编程实现输入任意一点坐标(x,y)，输出所对应的数字。[芬兰某著名通信设备公司 2005 年面试题]

答案：

代码如下：

```

#include <stdio.h>
#define max(a,b) ((a)<(b)?(b):(a))
#define abs(a) ((a)>0?(a):-(a))
int foo(int x, int y)
{
    int t = max(abs(x), abs(y));
    int u = t + t;
    int v = u - 1;
    v = v * v + u;
    if (x == -t)
        v += u + t - y;
    else if (y == -t)
        v += 3 * u + x - t;
    else if (y == t)
        v += t - x;
    else
        v += y - t;
    return v;
}

int main()
{
    int x, y;

    for (y=-4;y<=4;y++)
    {
        for (x=-4;x<=4;x++)
    }
  
```

```

        printf("%5d", foo(x, y));
        printf("\n");
    }
    while (scanf("%d%d", &x, &y) == 2)
        printf("%d\n", foo(x, y));

    return 0;
}

```

5.6 a、b 交换

面试例题 1：There are two int variables: a and b, don't use "if", "? ;", "switch" or other judgement statements, find out the biggest one of the two numbers. (有两个变量 a 和 b，不用 “if”、“?:”、“switch” 或其他判断语句，找出两个数中间比较大的。) [美国某著名网络开发公司 2005 年面试题]

答案：

方案一：

```
int max = ((a+b)+abs(a-b)) / 2
```

方案二：

```
int c = a -b;
char *strs[2] = {"a 大", "b 大"};
c = unsigned(c) >> (sizeof(int) * 8 - 1);
```

面试例题 2：如何将 a、b 的值进行交换，并且不使用任何中间变量？

解析：

简而言之，用异或语句比较容易，不用担心超界的问题。

如果采用：

```
a=a+b;
b=a-b;
a=a-b;
```

这样做的缺点就是如果 a、b 都是比较大的两个数， $a=a+b$ 时就会超界。

而采用：

```
a=a^b;
b=a^b;
a=a^b;
```

无须担心超界的问题，这样就比较好。

这样做的原理是按位异或运算。按位异或运算符“`^`”是双目运算符，其功能是参与运算的两数各对应的二进制位相异或，当对应的二进制位相异时，结果为1。参与运算数仍以补码形式出现。例如`9 ^ 5`可写成算式如下：

```
00001001^00000101 00001100 (十进制数为12)
main(){
    int a=9;
    a=a^5;
    printf("a=%d\n",a);
}
00001001^00000101 得到00001100。
00001001^00001100 得到00000101。
00000101^00000101 得到00001001。
```

答案：

```
a=a^b;
b=a^b;
a=a^b;
```

5.7 C 和 C++ 的关系

面试例题 1：在 C++ 程序中调用被 C 编译器编译后的函数，为什么要加 `extern "C"`？

答案：C++ 语言支持函数重载，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 语言的不同。假设某个函数的原型为：`void foo(int x, int y)`。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。

C++ 提供了 C 连接交换指定符号 `extern "C"` 解决名字匹配问题。

面试例题 2：头文件中的 `#ifndef/define/endif` 是干什么用的？

答案：防止该头文件被重复引用。

面试例题 3：`#include <filename.h>` 和 `#include "filename.h"` 有什么区别？

答案：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`。

面试例题 4：如何判断一段程序是由 C 编译程序还是由 C++ 编译程序编译的？[美国某著名网络开发公司 2005 年面试题]

答案：

C++ 编译时定义了`_cplusplus`。

C 编译时定义了`_STDC_`。

5.8 程序设计其他问题

面试例题 1：main 主函数执行完毕后，是否可能会再执行一段代码？给出说明。[美国某著名网络开发公司 2005 年面试题]

答案：如果需要加入一段在 main 退出后执行的代码，可以使用`atexit()`函数注册一个函数，代码如下：

```
#include <stdlib.h>
int atexit(void (*function)(void));
#include <stdlib.h>
#include <stdio.h>

void fn1( void ), fn2( void ), fn3( void ), fn4( void );

int main( void )
{
    atexit( fn1 );
    atexit( fn2 );
    atexit( fn3 );
    atexit( fn4 );
    printf( "This is executed first.\n" );
}

void fn1()
{
    printf( "next.\n" );
}
```

```
void fn2()
{
    printf( "executed " );
}

void fn3()
{
    printf( "is " );
}

void fn4()
{
    printf( "This " );
}
```

第 6 章

预处理、const 与 sizeof

预 处理问题、const 问题和 sizeof 问题是 C++ 设计语言中的三大难点，也是各大企业面试中反复出现的问题。就 sizeof 问题而言，我们曾在十几家公司、几十套面试题目中发现它的存在。所以本章把这三大问题单独提出来，并结合详细的分析和解释来阐述各个知识点。

6.1 宏定义

面试例题 1：What is the output of the following code? (下面代码的输出结果是什么?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

```
#define SQR(x) (x*x)
main()
{
    int a,b=3;
    a=SQR(b+2);
    printf("\n%d",a);
}
```

- A. 25 B. 11
C. Would vary from compiler to compiler D. Error

解析：

完整的源代码如下，执行后输出 11。

```
#include <cstdio>
#define SQR(x) (x*x)
int main()
{
    int a,b=3;
    a=SQR(b+2);
```

```

        printf("\n%d", a);
    return 0;
}

```

宏定义展开时容易造成二义性问题。 $a=\text{SQR}(b+2)$ 这一语句展开后为“ $b+2*b+2$ ”，而并不是想象中的“ $(b+2)*(b+2)$ ”。要是想得到这个结果，必须把宏定义语句改为如下的形式：

```
#define SQR(x) ((x)*(x))
```

这样输出的结果就是 25。

答案：B。

面试例题 2：用预处理指令#define 声明一个常数，用以表明 1 年中有多少秒（忽略闰年问题）。[美国某著名计算机嵌入式公司 2005 年面试题]

解析：

通过这道题面试官想考几件事情：

- #define 语法的基本知识（例如，不能以分号结束，括号的使用，等等）。
- 要懂得预处理器将为你计算常数表达式的值，因此，写出你是如何计算一年中有多少秒而不是计算出实际的值，会更有意义。
- 意识到这个表达式将使一个 16 位机的整型数溢出，因此要用到长整型符号 L，告诉编译器这个常数是长整型数。

如果你在你的表达式中用到 UL（表示无符号长整型），那么你就有了一个好的起点。记住，第一印象很重要。

答案：#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)UL

面试例题 3：写一个“标准”宏 MIN，这个宏输入两个参数并返回较小的一个。[美国某著名计算机嵌入式公司 2005 年面试题]

解析：

这个测试是为下面的目的而设的：

- 标识#define 在宏中应用的基本知识。这是很重要的，因为直到嵌入（inline）操作符变为标准 C 的一部分，宏都是方便地产生嵌入代码的唯一方法。对于嵌入式系统来说，为了能达到要求的性能，嵌入代码经常是必须的方法。

- 三重条件操作符的知识。这个操作符存在 C 语言中的原因是它使得编译器能产生比 if-then-else 更优化的代码，了解这个用法是很重要的。
- 懂得在宏中小心地把参数用括号括起来。

答案：#define MIN(A,B) ((A) <= (B) ? (A) : (B))

6.2 const

面试例题 1：What does the keyword "const" means in C program? Please at least make two examples about the usage of const. (const 有什么用途？请至少说明两种。) [中国台湾某著名 CPU 生产公司 2005 年面试题]

解析：在 C 程序中，const 的用法主要有定义常量、修饰函数参数、修饰函数返回值等 3 个用处。在 C++ 程序中，它还可以修饰函数的定义体，定义类中某个成员函数为恒态函数，即不改变类中的数据成员。

答案：(1) 可以定义 const 常量。(2) const 可以修饰函数的参数和返回值，甚至函数的定义体。被 const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

面试例题 2：const 与#define 相比有什么不同？

答案：C++ 语言可以用 const 定义常量，也可以用#define 定义常量，但是前者比后者有更多的优点：

(1) const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查，而对后者只进行字符替换，没有类型安全检查，并且在字符替换中可能会产生意想不到的错误（边际效应）。

(2) 有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。在 C++ 程序中只使用 const 常量而不使用宏常量，即 const 常量完全取代宏常量。

扩展知识

常量的引进是在早期的 C++ 版本中，当时标准 C 规范正在制订。那时，常量被看做一个好的思想而被包含在 C 中。但是，C

中的 `const` 的意思是“一个不能被改变的普通变量”。在 C 中，它总是占用内存，而且它的名字是全局符。C 编译器不能把 `const` 看成一个编译期间的常量。在 C 中，如果写：

```
const bufsize=100;
char buf[bufsize];
```

尽管看起来好像做了一件合理的事，但这将得到一个错误的结果。因为 `bufsize` 占用内存的某个地方，所以 C 编译器不知道它在编译时的值。在 C 语言中可以选择这样书写：

```
const bufsize;
```

这样写在 C++ 中是不对的，而 C 编译器则把它作为一个声明，这个声明指明在别的地方有内存分配。因为 C 默认 `const` 是外部连接的，C++ 默认 `const` 是内部连接的，这样，如果在 C++ 中想完成与 C 中同样的事情，必须用 `extern` 把内部连接改成外部连接：

```
extern const bufsize;//declaration only
```

这种方法也可用在 C 语言中。在 C 语言中使用限定符 `const` 不是很有用，即使是在常数表达式里（必须在编译期间被求出）想使用一个已命名的值，使用 `const` 也不是很有用的。C 迫使程序员在预处理器里使用 `#define`。

6.3 sizeof

面试例题 1：What is the output of the following code? （下面代码的输出结果是什么？） [美国某著名计算机软硬件公司面试题]

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
struct{
    short a1;
    short a2;
    short a3;
}A;
struct{
    long a1;
    short a2;
}B;
```

```

int main()
{
    char* ss1 = "0123456789";
    char ss2[] = "0123456789";
    char ss3[100] = "0123456789";
    int ss4[100];
    char q1[]="abc";
    char q2[]="a\n";
    char* q3="a\n";
    char *str1 = (char *)malloc(100);

    void *str2 = (void *) malloc(100);

    cout << sizeof(ss1) << " ";
    cout << sizeof(ss2) << " ";
    cout << sizeof(ss3) << " ";
    cout << sizeof(ss4) << " ";
    cout << sizeof(q1) << " ";
    cout << sizeof(q2) << " ";
    cout << sizeof(q3) << " ";
    cout << sizeof(A) << " ";
    cout << sizeof(B) << " ";
    cout << sizeof(str1) << " ";
    cout << sizeof(str2) << " ";

    return 0;
}

```

解析：

ss1 是一个字符指针，指针的大小是一个定值，就是 4，所以 sizeof(ss1) 是 4。

ss2 是一个字符数组，这个数组最初未定大小，由具体填充值来定。填充值是“0123456789”。1 个字符所占空间是 1 位，10 个就是 10 位，再加上隐含的“\0”，所以一共是 11 位。

ss3 也是一个字符数组，这个数组开始预分配 100，所以它的大小一共是 100 位。

ss4 也是一个整型数组，这个数组开始预分配 100，但每个整型变量所占空间是 4，所以它的大小一共是 400 位。

q1 与 ss2 类似，所以是 4。

q2 里面有一个“\n”，“\n”算做一位，所以它的空间大小是 3。

q3 是一个字符指针，指针的大小是一个定值，就是 4，所以 sizeof(q3) 是 4。

A 和 B 是两个结构体。在默认情况下，为了方便对结构体内元素的访问和管理，当结构体内的元素的长度都小于处理器的位数的时候，便以结构体里面最长的数据元素为对齐单位，也就是说，结构体的长度一定是最长的数据元素的整数倍。如果结构体内存在长度大于处理器位数的元素，那么就以处理器的位数为对齐单位。但是结构体内类型相同的连续元素将在连续的空间内，和数组一样。

结构体 A 中有 3 个 short 类型变量，各自以 2 字节对齐，结构体对齐参数按默认的 8 字节对齐，则 a1、a2、a3 都取 2 字节对齐，`sizeof(A)` 为 6，其也是 2 的整数倍。B 中 a1 为 4 字节对齐，a2 为 2 字节对齐，结构体默认对齐参数为 8，则 a1 取 4 字节对齐，a2 取 2 字节对齐；结构体大小 6 字节，6 不为 4 的整数倍，补空字节，增到 8 时，符合所有条件，则 `sizeof(B)` 为 8。

CPU 的优化规则大致原则是这样的：对于 n 字节的元素 ($n=2,4,8,\dots$)，它的首地址能被 n 整除，才能获得最好的性能。设计编译器的时候可以遵循这个原则：对于每一个变量，可以从当前位置向后找到第一个满足这个条件的地址作为首地址。例子比较特殊，因为即便采用这个原则，得到的结果也应该为 6 (long 的首地址偏移量 0000，short 首地址偏移量 0004，都符合要求)。但是结构体一般会面临数组分配的问题。编译器为了优化这种情况，干脆把它的大小设为 8，这样就没有麻烦了，否则的话，会出现单个结构体的大小为 6，而大小为 n 的结构体数组大小却为 $8 \times (n - 1) + 6$ 的尴尬局面。IBM 出这道题并不在于考查理解语言本身和编译器，而在于应聘者对计算机底层机制的理解和设计程序的原则。也就是说，如果让你设计编译器，你将怎样解决内存对齐的问题。

答案：

4, 11, 100, 400, 4, 3, 4, 6, 8, 4, 4。

扩展知识（内存中的数据对齐）

数据对齐，是指数据所在的内存地址必须是该数据长度的整数倍。DWORD 数据的内存起始地址能被 4 除尽，WORD 数据的内存起始地址能被 2 除尽。`x86` CPU 能直接访问对齐的数据，当它试图访问一个未对齐的数据时，会在内部进行一系列的调整。

这些调整对于程序来说是透明的，但是会降低运行速度，所以编译器在编译程序时会尽量保证数据对齐。同样一段代码，我们来看看用 VC、Dev C++ 和 LCC 这 3 个不同的编译器编译出来的程序的执行结果：

```
#include <stdio.h>

int main()
{
    int a;
    char b;
    int c;
    printf("0x%08x ", &a);
    printf("0x%08x ", &b);
    printf("0x%08x ", &c);
    return 0;
}
```

这是用 VC 编译后的执行结果：

```
0x0012ff7c
0x0012ff7b
0x0012ff80
```

变量在内存中的顺序：b（1字节）—a（4字节）—c（4字节）。

这是用 Dev C++ 编译后的执行结果：

```
0x0022ff7c
0x0022ff7b
0x0022ff74
```

变量在内存中的顺序：c（4字节）—中间相隔3字节—b（占1字节）—a（4字节）。

这是用 LCC 编译后的执行结果：

```
0x0012ff6c
0x0012ff6b
0x0012ff64
```

变量在内存中的顺序：同上。

3 个编译器都做到了数据对齐，但是后两个编译器显然没 VC “聪明”，让一个 char 占了 4 字节，浪费内存。

面试例题 2：What is the output of the following code? (下面代码的输出结果是什么?) [德国某著名电子/通信/IT 企业 2005 年 11 月面试题]

```
#include <iostream>
```

```

using namespace std;

class A {
};

class A2 {
    char d,e;
};

struct B {

};

struct C {
    char b,c;
};

struct D {
    int x,y;
};

int main() {

    cout<<sizeof(A)<<endl;
    cout<<sizeof(A2)<<endl;
    A *p1=new A();
    A p2;
    A *p3;
    cout<<sizeof(p1)<<endl;
    cout<<sizeof(p2)<<endl;
    cout<<sizeof(p3)<<endl;
    cout<<sizeof(B)<<endl;
    cout<<sizeof(C)<<endl;
    cout<<sizeof(D)<<endl;
    //cout<<sizeof(&B)<<endl;

    return 0;
}

```

解析：对于一个类而言，即便它是一个空的类，编译器仍然要给它一个空间，所以类 A 即便什么都没有，它的空间大小依然是 1。

而类 A2 大小是类中的两个字符 d、e 之和，所以它的空间大小依然是 2。

至于 p1、p2、p3，p1 和 p3 是指针，所以它们的大小是一定的，因此是 4；p2 是类 A 的对象，所以它的大小和类 A 相等，也是 1。

B 和 C 的解释同理于 A、A2，不再赘述。至于 D，它和 C 的不同点在于，结构体内有两个整型变量，每个整型变量所占空间为 4，所以 D 所占空间大小为 8。

答案：1, 2, 4, 1, 4, 1, 2, 8。

面试例题3：求解下面程序的结果。[中国某著名通信企业H面试题]

```
#include <iostream>
using namespace std;

class A1
{
public:
    int a;
    static int b;

    A1();
    ~A1();
};

class A2
{
public:
    int a;
    char c;
    A2();
    ~A2();
};

class A3
{
public:
    float a;
    char c;
    A3();
    ~A3();
};

class A4
{
public:
    float a;
    int b;
    char c;
    A4();
    ~A4();
};

class A5
{
public:
    double d;
    float a;
    int b;
    char c;
```

```

A5();
~A5();
};

int main() {
    cout<<sizeof(A1)<<endl;
    cout<<sizeof(A2)<<endl;
    cout<<sizeof(A3)<<endl;
    cout<<sizeof(A4)<<endl;
    cout<<sizeof(A5)<<endl;
    return 0;
}

```

解析：因为静态变量是存放在全局数据区的，而 sizeof 计算栈中分配的大小，是不会计算在内的，所以 sizeof(A1) 是 4。

为了照顾数据对齐，int 大小为 4，char 大小为 1，所以 sizeof(A2) 是 8。

为了照顾数据对齐，float 大小为 4，char 大小为 1，所以 sizeof(A3) 是 8。

为了照顾数据对齐，float 大小为 4，int 大小为 4，char 大小为 1，所以 sizeof(A3) 是 12。

为了照顾数据对齐，double 大小为 8，float 大小为 4，int 大小为 4，char 大小为 1，所以 sizeof(A3) 是 24。

答案：4, 8, 8, 12, 24

面试例题 4：说明 sizeof 和 strlen 之间的区别。

解析：

由以下几个例子我们说明 sizeof 和 strlen 之间的区别。

第一个例子：

```
char* ss = "0123456789";
```

sizeof(ss) 结果为 4，ss 是指向字符串常量的字符指针。

sizeof(*ss) 结果为 1，*ss 是第一个字符。

第二个例子：

```
char ss[] = "0123456789";
```

sizeof(ss) 结果为 11，ss 是数组，计算到 “\0” 位置，因此是 (10 + 1)。

sizeof(*ss) 结果为 1，*ss 是第一个字符

第三个例子：

```
char ss[100] = "0123456789";
```

`sizeof(ss)` 结果为 100，`ss` 表示在内存中预分配的大小， 100×1 。

`strlen(ss)` 结果为 10，它的内部实现是用一个循环计算字符串的长度，直到“\0”为止。

第四个例子：

```
int ss[100] = "0123456789";
```

`sizeof(ss)` 结果为 400，`ss` 表示在内存中的大小， 100×4 。

`strlen(ss)` 错误，`strlen` 的参数只能是 `char*`，且必须是以“\0”结尾的。

第五个例子：

```
class X
{
    int i;
    int j;
    char k;
};

X x;
```

`cout << sizeof(X) << endl;` 结果为 12，内存补齐。

`cout << sizeof(x) << endl;` 结果为 12，理由同上。

答案：

通过对 `sizeof` 与 `strlen` 的深入理解，得出两者区别如下：

(1) `sizeof` 操作符的结果类型是 `size_t`，它在头文件中的 `typedef` 为 `unsigned int` 类型。该类型保证能容纳实现所建立的最大对象的字节大小。

(2) `sizeof` 是算符，`strlen` 是函数。

(3) `sizeof` 可以用类型做参数，`strlen` 只能用 `char*` 做参数，且必须是以“\0”结尾的。`sizeof` 还可以用函数做参数，比如：

```
short f();
printf("%d\n", sizeof(f()));
```

输出的结果是 `sizeof(short)`，即 2。

(4) 数组做 `sizeof` 的参数不退化，传递给 `strlen` 就退化为指针。

(5) 大部分编译程序在编译的时候就把 `sizeof` 计算过了，是类型或是变量的长度。这就是 `sizeof(x)` 可以用来定义数组维数的原因：

```
char str[20] = "0123456789";
int a = strlen(str); // a=10;
int b = sizeof(str); // 而 b=20;
```

(6) `strlen` 的结果要在运行的时候才能计算出来，用来计算字符串的长度，而不是类型占内存的大小。

(7) `sizeof` 后如果是类型必须加括号，如果是变量名可以不加括号。这是因为 `sizeof` 是个操作符而不是个函数。

(8) 当使用了一个结构类型或变量时，`sizeof` 返回实际的大小。当使用一静态的空间数组时，`sizeof` 返回全部数组的尺寸。`sizeof` 操作符不能返回被动态分配的数组或外部的数组的尺寸。

(9) 数组作为参数传给函数时传的是指针而不是数组，传递的是数组的首地址，如：`fun(char [8])`、`fun(char [])`都等价于 `fun(char *)`。在 C++ 里传递数组永远都是传递指向数组首元素的指针，编译器不知道数组的大小。如果想在函数内知道数组的大小，需要这样做：进入函数后用 `memcpy` 将数组拷贝出来，长度由另一个形参传进去。代码如下：

```
fun (unsigned char *p1, int len)
{
    unsigned char* buf = new unsigned char[len+1]
    memcpy(buf, p1, len);
}
```

(10) 计算结构变量的大小就必须讨论数据对齐问题。为了使 CPU 存取的速度最快（这同 CPU 取数操作有关，详细的介绍可以参考一些计算机原理方面的书），C++ 在处理数据时经常把结构变量中的成员的大小按照 4 或 8 的倍数计算，这就叫数据对齐（data alignment）。这样做可能会浪费一些内存，但在理论上 CPU 速度快了。当然，这样的设置会在读写一些别的应用程序生成的数据文件或交换数据时带来不便。MS VC++ 中的对齐设定，有时候 `sizeof` 得到的与实际不等。一般在 VC++ 中加上 `#pragma pack(n)` 的设定即可。或者如果要按字节存储，而不进行数据对齐，可以在 Options 对话框中修改 Advanced Compiler 选项卡中的“Data Alignment”为按字节对齐。

(11) `sizeof` 操作符不能用于函数类型、不完全类型或位字段。不完全类型指具有未知存储大小数据的数据类型，如未知存储大小的数组类型、未知内容的结构或联合类型、`void` 类型等。

面试例题 5：说明 `sizeof` 的使用场合。

答案：

(1) sizeof 操作符的一个主要用途是与存储分配和 I/O 系统那样的例程进行通信。例如：

```
void *malloc (size_t size),
size_t fread(void * ptr, size_t size, size_t nmemb, FILE * stream)
```

(2) 用它看看某种类型的对象在内存中所占的单元字节。例如：

```
void * memset (void * s, int c, sizeof(s))
```

(3) 在动态分配对象时，可以让系统知道要分配多少内存。

(4) 便于一些类型的扩充。在 Windows 中有很多结构类型就有一个专用的字段用来存放该类型的字节大小。

(5) 由于操作数的字节数在实现时可能出现变化，建议在涉及到操作数字节大小时用 sizeof 代替常量计算。

(6) 如果操作数是函数中的数组形参或函数类型的形参， sizeof 给出其指针的大小。

面试例题 6: How many bytes will be occupied for the variable (definition: int **a[3][4])? (这个数组占据多大空间？) [中国某著名计算机金融软件公司 2005 年面试题]

- A. 64 B. 12 C. 48 D. 128

解析： sizeof 问题， $3 \times 4 \times 4 = 48$ 。

答案： C。

面试例题 7: Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

```
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char* argv[]) {
    //To output "TrendMicroSoftUSCN"

    string strArr1[]{"Trend", "Micro", "Soft"};
    string *pStrArr1=new string[2];
    pStrArr1[0]="US";
    pStrArr1[1]="CN";
```

```
    for(int i=0;i<sizeof(strArr1)/sizeof(string);i++)
        cout<<strArr1[i];
    for(int j=0;j<sizeof(pStrArr1)/sizeof(string);j++)
        cout<<pStrArr1[j];

    return 0;
}
```

解析： sizeof 问题。

程序运行后输出：TrendMicroSoftUS。这是因为 sizeof(pStrArr1) 运算得出的结果是指针 pStrArr1 的大小，即 4。这样就不能正确地输出“USCN”。而字符串 strArr1 是由 3 段构成的，所以 sizeof(strArr1) 大小是 12。

答案：

正确的程序如下：

```
#include <iostream>
#include <string>
using namespace std;

int main(int argc,char *argv[])
{
    string strArr1[] = {"Trend","Micro","Soft"};
    string *pStrArr1 = new string[2];
    pStrArr1[0] = "US";
    pStrArr1[1] = "CN";
    cout << sizeof(strArr1) << endl;
    cout << sizeof(string) << endl;
    for(int i =0; i< sizeof(strArr1)/sizeof(string);i++)
        cout << strArr1[i];
        cout << endl;
    for(int j =0; j< sizeof(pStrArr1)*2/sizeof(string);j++)
        cout << pStrArr1[j];

    return 0;
}
```

面试例题 8：写出下面 sizeof 的答案。[德国某著名软件咨询企业 2005 年面试题]

```
#include <iostream>
#include <complex>
using namespace std;
class Base
{

public:
    Base() { cout<<"Base-ctor"<<endl; }
    ~Base() { cout<<"Base-dtor"<<endl; }
```

```

virtual void f(int) { cout<<"Base::f(int)"<<endl; }
virtual void f(double) { cout<<"Base::f(double)"<<endl; }
virtual void g(int i = 10) { cout<<"Base::g()"<<i<<endl; }
void g2(int i = 10) { cout<<"Base::g2()"<<i<<endl; }
};

class Derived: public Base
{
public:
    Derived() { cout<<"Derived-ctor"<<endl; }
    ~Derived() { cout<<"Derived-dtor"<<endl; }
    void f(complex<double>) { cout<<"Derived::f(complex)"<<endl; }
    virtual void g(int i = 20) { cout<<"Derived::g()"<<i<<endl; }
};

int main()
{
    Base b;
    Derived d;

    Base* pb = new Derived;
    //Select the correct one from the four choices:
    cout<<sizeof(Base)<<"tt"<<endl;
    //A. 4   B.32   C.20   D.Platform-dependent
    cout<<sizeof(Derived)<<"bb"<<endl;
    //A. 4   B.8 C.36   D.Platform-dependent
}

```

解析：求类 base 的大小。因为类 base 只有一个指针，所以类 base 的大小是 4。Derive 大小与 base 类似，所以也是 4。

答案：A, A。

面试例题 9：以下代码的输出结果是_____。[中国某著名计算机金融软件公司 2006 年面试题]

```

char var[10]
int test(char var[])
{
    return sizeof(var)
};

```

- A. 10 B. 9 C. 11 D. 4

解析：因为 var[] 等价于 *var，已经退化成一个指针了，所以大小是 4。

答案：D。

面试例题 10：以下代码的输出结果是_____。[美国某著名防毒软件公司 2006 年面试题]

```
class B
{
    float f;
    char p;
    int adf[3];
};

cout << ""<< sizeof(B);
```

解析：float f 占 4 个字节，char p 占 1 个字节，int adf[3] 占 12 个字节，总共是 17 个字节。根据内存对齐原则，要选择 4 的倍数，是 20 个字节。

答案：20。

面试例题 11：一个空类占多少空间？多重继承的空类呢？[英国某著名计算机图形图像公司面试题]

解析：我们用程序来实现一个空类和一个多重继承的空类。看看它们的大小是多少。代码如下：

```
#include <iostream>
#include <memory.h>
#include<assert.h>

using namespace std;
class A
{
};

class A2
{
};

class B : public A
{
};

class C : public virtual B
{
};

class D : public A,public A2
{
};

int main(int argc,char *argv[])
```

```

{
    cout << "sizeof(A)：" << sizeof(A) << endl;
    cout << "sizeof(B)：" << sizeof(B) << endl;
    cout << "sizeof(C)：" << sizeof(C) << endl;
    cout << "sizeof(D)：" << sizeof(D) << endl;
    return 0;
}

```

以上答案分别是：1，1，4，1。这说明：空类所占空间为1，单一继承的空类空间也为1，多重继承的空类空间还是1。但是虚继承涉及到虚表（虚指针），所以 sizeof(C)的大小为4。

答案：一个空类所占空间为1，多重继承的空类所占空间还是1。

6.4 内联函数和宏定义

面试例题1：内联函数和宏的差别是什么？

答案：内联函数和普通函数相比可以加快程序运行的速度，因为不需要中断调用，在编译的时候内联函数可以直接被镶嵌到目标代码中。而宏只是一个简单的替换。

内联函数要做参数类型检查，这是内联函数跟宏相比的优势。

inline是指嵌入代码，就是在调用函数的地方不是跳转，而是把代码直接写到那里去。对于短小的代码来说，inline 可以带来一定的效率提升，而且和C时代的宏函数相比，inline 更安全可靠。可是这个是以增加空间消耗为代价的。至于是否需要 inline 函数，就需要根据你的实际情况取舍了。

inline一般只用于如下情况：

- (1) 一个函数不断被重复调用。
- (2) 函数只有简单的几行，且函数内不包含 for、while、switch 语句。

一般来说，我们写小程序没有必要定义成 inline，但是如果要完成一个工程项目，当一个简单函数被调用多次时，则应该考虑用 inline。

宏在 C 语言里极其重要，而在 C++里用得就少多了。关于宏的第一规则是：绝不应该去使用它，除非你不得不这样做。几乎每个宏都表明了程序设计语言里或者程序里或者程序员的一个缺陷，因为它将在编译器看到程序的正文之前重新摆布这些正文。宏也是许多程序设计工具的主要麻烦。所以，如果你使用了宏，你就应该准备着只能从各种工具（如

排错系统、交叉引用系统、轮廓程序等) 中得到较少的服务。

宏是在代码处不加任何验证的简单替代，而内联函数是将代码直接插入调用处，而减少了普通函数调用时的资源消耗。

宏不是函数，只是在编译前(编译预处理阶段)将程序中有关字符串替换成宏体。

inline 函数是函数，但在编译中不单独产生代码，而是将有关代码嵌入到调用处。

```
inline fac(float i) {return i * i;}//没有写返回值的  
printf("bb= %d", fact(8));//调用时就是执行printf("bb= %d", 8*8);
```

第 7 章

指针与引用

指针是 C 系语言的特色。指针是 C++ 提供的一种颇具特色的数据类型，允许直接获取和操纵数据地址，实现动态存储分配。

指针是 C 和 C++ 的精华所在，也是 C 和 C++ 的一个十分重要的概念。一个数据对象的内存地址称为该数据对象的指针。

指针可以表示各种数据对象，如简单变量、数组、数组元素、结构体，甚至函数。

换句话说：指针具有不同的类型，可以指向不同的数据存储体。

指针问题，包括常量指针、数组指针、函数指针、this 指针、指针传值、指向指针的指针等问题也是各大公司的常备考点。本章不对指针基本知识做回顾和分析（请参考 C++ 其他经典著作），而是通过对各公司面试题目进行全面、仔细的解析帮助读者解决其中的难点。

7.1 指针基本问题

面试例题 1：指针和引用的差别？

答案：

(1) 非空区别。在任何情况下都不能使用指向空值的引用。一个引用必须总是指向某些对象。因此如果你使用一个变量并让它指向一个对象，但是该变量在某些时候也可能不指向任何对象，这时你应该把变量声明为指针，因为这样你可以赋空值给该变量。相反，如果变量肯定指

向一个对象，例如你的设计不允许变量为空，这时你就可以把变量声明为引用。不存在指向空值的引用这个事实意味着使用引用的代码效率比使用指针要高。

(2) 合法性区别。在使用引用之前不需要测试它的合法性。相反，指针则应该总是被测试，防止其为空。

(3) 可修改区别。指针与引用的另一个重要的不同是指针可以被重新赋值以指向另一个不同的对象。但是引用则总是指向在初始化时被指定的对象，以后不能改变，但是指定的对象其内容可以改变。

(4) 应用区别。总的来说，在以下情况下你应该使用指针：一是你考虑到存在不指向任何对象的可能（在这种情况下，你能够设置指针为空），二是你需要能够在不同的时刻指向不同的对象（在这种情况下，你能改变指针的指向）。如果总是指向一个对象并且一旦指向一个对象后就不会改变指向，那么你应该使用引用。

面试例题 2: Please check out which of the following statements are wrong? (看下面的程序哪里有错?) [中国台湾某著名计算机硬件公司 2005 年 12 月面试题]

```
#include <iostream>
using namespace std;
int main()
{
    int iv;                                //1
    int iv2=1024;                           //2
    int iv3=999;                            //3
    int &reiv;                             //4
    int &reiv2 = iv;                         //5
    int &reiv3 = iv;                         //6
    int *pi;                               //7
    *pi = 5;                               //8
    pi=&iv3;                             //9
    const double di;                         //10
    const double maxWage =10.0;              //11
    const double minWage =0.5;
    const double *pc =&maxWage;             //12

    cout << pi;
    return 0;
}
```

答案：

1 正确，很正常地声明了一个整型变量。

2 正确，很正常地声明了一个整型变量，同时初始化这个变量。

- 3 正确，理由同上。
- 4 错误，声明了一个引用，但引用不能为空，必须同时初始化。
- 5 正确，声明了一个引用 reiv2，同时初始化了，也就是 reiv2 是 iv 的别名。
- 6 正确，理由同上。
- 7 正确，声明了一个整数指针，但是并没有定义这个指针所指向的地址。
- 8 错误，整数指针 pi 并没有指向实际的地址。在这种情况下就给它赋值是错误的，因为赋的值不知道该放到哪里去，从而造成错误。
- 9 正确，整数指针 pi 指向 iv3 实际的地址。
- 10 错误，const 常量赋值时，必须同时初始化。
- 11 正确，const 常量赋值并同时初始化。
- 12 正确，const 常量指针赋值并同时初始化。

面试例题 3： Which of the following is NOT true about the "this" pointer of class X? (下面关于“this”指针的叙述哪个是不正确的？) [美国某著名分析软件公司 2005 年面试题]

- A. It lets each object of class X to access its address. (让 X 类的每一个对象指向它的地址。)
- B. It will be implicitly passed as argument of every non-static member function of class X. (可以隐性传递 this 指针。)
- C. It can not be used explicitly in member function of class X. (不能在类的成员函数里明确地声明。)
- D. Its type is const X* in const member function of class X. (它是常量函数中的一个常量指针。)

解析：这是类的 this 指针问题。this 指针是在实例化一个对象后产生的，并且指向对象本身。比如实例化一个对象 pt，那么 this = &pt；用“&”取地址符来取对象的地址。同样，如果定义对象 pt 这个类中，有一个 public 变量 x，那么就可以用 this->x=0 来定义 x 的值，等同于 pt.x=0。这有一个范例程序：

```
#include<iostream>
using namespace std;
```

```
class Point
{
public:
    Point()
    {
        cout<<this<<endl;
        cout<<this+1<<endl;
        cout<<this-1<<endl;
    }
};

int main()
{
    Point a;
    cout<<&a<<endl;
    return 0;
}
```

答案：C。

扩展知识

类的 this 指针有以下特点：

(1) this 只能在成员函数中使用。

全局函数、静态函数都不能使用 this。

实际上，成员函数默认第一个参数为 T* const this。

如：

```
class A{public: int func(int p){}};
```

其中，func 的原型在编译器看来应该是：

```
int func(A* const this, int p);
```

(2) 由此可见，this 在成员函数的开始前构造，在成员的结束后清除。

这个生命周期同任何一个函数的参数是一样的，没有任何区别。

当调用一个类的成员函数时，编译器将类的指针作为函数的 this 参数传递进去。如：

```
A a;
a.func(10);
```

此处，编译器将会编译成：

```
A::func(&a, 10);
```

看起来和静态函数没差别，对吗？不过，区别还是有的。编译器通常会对 this 指针做一些优化，因此，this 指针的传递效率比较高——如 VC 通常是通过 ecx 寄存器传递 this 参数的。

(3) 几个 this 指针的易混问题。

1) this 指针是什么时候创建的？

this 在成员函数的开始执行前构造，在成员的执行结束后清除。

但是如果 class 或者 struct 里面没有方法的话，它们是没有构造函数的，只能当做 C 的 struct 使用。采用 TYPE xx 的方式定义的话，在栈里分配内存，这时候 this 指针的值就是这块内存的地址。采用 new 的方式创建对象的话，在堆里分配内存，new 操作符通过 eax 返回分配的地址，然后设置给指针变量。之后去调用构造函数（如果有构造函数的话），这时将这个内存块的地址传给 ecx，之后构造函数里面怎么处理请看上面的回答。

2) this 指针存放在何处？堆、栈、全局变量，还是其他？

this 指针会因编译器不同而有不同的放置位置。可能是栈，也可能是寄存器，甚至全局变量。在汇编级别里面，一个值只会以 3 种形式出现：立即数、寄存器值和内存变量值。不是存放在寄存器就是存放在内存中，它们并不是和高级语言变量对应的。

3) this 指针是如何传递给类中的函数的？绑定？还是在函数参数的首参数就是 this 指针？那么，this 指针又是如何找到“类实例后函数”的？

大多数编译器通过 ecx 寄存器传递 this 指针。事实上，这也是一個潜规则。一般来说，不同编译器都会遵从一致的传参规则，否则不同编译器产生的 obj 就无法匹配了。

在 call 之前，编译器会把对应的对象地址放到 eax 中。this 是通过函数参数的首参数来传递的。this 指针在调用之前生成，至于“类实例后函数”，没有这个说法。类在实例化时，只分配类中的变量空间，并没有为函数分配空间。自从类的函数定义完成后，它就在那儿，不会跑的。

4) this 指针是如何访问类中的变量的？

如果不是类，而是结构的话，那么，如何通过结构指针来访

问结构中的变量呢？如果你明白这一点的话，就很容易理解这个问题了。

在 C++ 中，类和结构是只有一个区别的：类的成员默认是 private，而结构是 public。

this 是类的指针，如果换成结构，那 this 就是结构的指针了。

5) 我们只有获得一个对象后，才能通过对象使用 this 指针。如果我们知道一个对象 this 指针的位置，可以直接使用吗？

this 指针只有在成员函数中才有定义。因此，你获得一个对象后，也不能通过对象使用 this 指针。所以，我们无法知道一个对象的 this 指针的位置（只有在成员函数里才有 this 指针的位置）。当然，在成员函数里，你是可以知道 this 指针的位置的（可以通过&this 获得），也可以直接使用它。

6) 每个类编译后，是否创建一个类中函数表保存函数指针，以便用来调用函数？

普通的类函数（不论是成员函数，还是静态函数）都不会创建一个函数表来保存函数指针。只有虚函数才会被放到函数表中。

但是，即使是虚函数，如果编译器能明确知道调用的是哪个函数，编译器就不会通过函数表中的指针来间接调用，而是会直接调用该函数。

面试例题 4：将下面程序中的 this 指针改成非 this 指针的传递方式，编写程序实现。[美国某著名分析软件公司 2005 年面试题]

```
#include <string>
#include <iostream>
using namespace std;

struct X {
private:
    int len;
    char *ptr;
public:
    int GetLen() {
        return len;
    }
    char * GetPtr() {
        return ptr;
    }
    X& Set(char *);
};
```

```

X& Cat(char *);
X& Copy(X&);
void Print();
};

X& X::Set(char *pc) {
    len = strlen(pc);
    ptr = new char[len];
    strcpy(ptr, pc);
    return *this;
}

X& X::Cat(char *pc) {
    len += strlen(pc);
    strcat(ptr, pc);
    return *this;
}

X& X::Copy(X& x) {
    Set(x.GetPtr());
    return *this;
}

void X::Print() {
    cout << ptr << endl;
}

int main() {
    X xobj1;
    xobj1.Set("abcd")
        .Cat("efgh");

    xobj1.Print();
    X xobj2;
    xobj2.Copy(xobj1)
        .Cat("ijkl");

    xobj2.Print();
}

```

解析：这里用一个自己构建的 THIS 模拟系统默认的 this 指针，可以达到同样的效果。this 指针不是全局变量，也不是局部变量，它是成员函数的一个参数。因此，它只在成员函数中有效。this 其实就是将对象的地址传递给了对象的成员函数。

答案：

修改后的程序如下：

```

#include <string>
#include <iostream>
using namespace std;

```

```
struct X {
    private:
        int len;
        char *ptr;
    public:
        int GetLen (X* const THIS) {
            return THIS->len;
        }
        char * GetPtr (X* const THIS) {
            return THIS->ptr;
        }
        X& Set(X* const, char *);
        X& Cat(X* const, char *);
        X& Copy(X* const, X&);
        void Print(X* const);
};

X& X::Set(X* const THIS, char *pc) {
    THIS->len = strlen(pc);
    THIS->ptr = new char[THIS->len];
    strcpy(THIS->ptr, pc);
    return *THIS;
}

X& X::Cat(X* const THIS, char *pc) {
    THIS->len += strlen(pc);
    strcat(THIS->ptr, pc);
    return *THIS;
}

X& X::Copy(X* const THIS, X& x) {
    THIS->Set(THIS, x.GetPtr(&x));
    return *THIS;
}

void X::Print(X* const THIS) {
    cout << THIS->ptr << endl;
}

int main() {
    X xobj1;
    xobj1.Set(&xobj1 , "abcd")
    .Cat(&xobj1 , "efgh");

    xobj1.Print(&xobj1);
    X xobj2;
    xobj2.Copy(&xobj2 , xobj1)
    .Cat(&xobj2 , "ijkl");

    xobj2.Print(&xobj2);
}
```

7.2 传递动态内存

面试例题 1: What will happen after running the "Test"? (这个程序测试后会有什么结果?) [美国某著名计算机嵌入式公司 2005 年 9 月面试题]

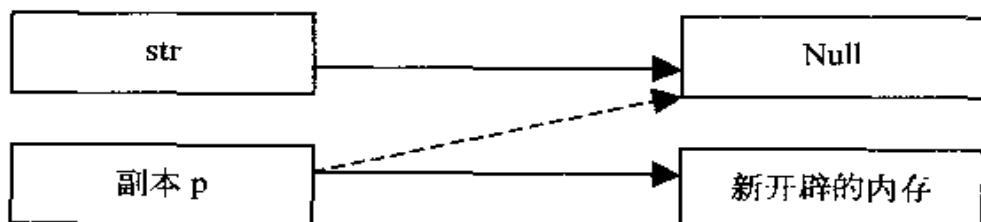
```
#include <iostream>

void GetMemory(char *p, int num)
{
    p = (char *)malloc(sizeof(char) * num);
}

int main()
{
    char *str = NULL;

    GetMemory(str, 100);
    strcpy(str, "hello");
    return 0;
}
```

解析:毛病出在函数 GetMemory 中。void GetMemory(char *p, int num) 中的*p 实际上是主函数中的 str 的一个副本,编译器总是要为函数的每个参数制作临时副本。在本例中, p 申请了新的内存,只是把 p 所指的内存地址改变了,但是 str 丝毫未变。因为函数 GetMemory 没有返回值,因此 str 并不指向 p 所申请的那段内存,所以函数 GetMemory 并不能输出任何东西,如下图所示。事实上,每执行一次 GetMemory 就会申请一块内存,但申请的内存却不能有效释放,结果是内存一直被独占,最终造成内存泄露。



如果一定要用指针参数去申请内存,那么应该采用指向指针的指针,传 str 的地址给函数 GetMemory。代码如下:

```
#include <iostream>

void GetMemory(char **p, int num)
{
```

```

    *p = (char *)malloc(sizeof(char) * num);
};

int main()
{
    char *str = NULL;

    GetMemory(&str, 100);
    strcpy(str, "hello");
    cout << *str << endl;
    cout << str << endl;
    cout << &str << endl;
    return 0;
}

```

这样的话，程序就可以运行成功。字符串是一个比较特殊的例子。我们分别打印*str、str、&str可以发现，结果分别是：h，hello，0*22f7c。str就是字符串的值；*str是字符串中某一字符的值，默认的是首字符，所以是h；&str是字符串的地址值。

由于“指向指针的指针”这个概念不容易理解，我们可以用函数返回值来传递动态内存。这种方法更加简单，代码如下：

```

#include <iostream>
using namespace std;

char *GetMemory(char *p, int num)
{
    p = (char *)malloc(sizeof(char) * num);
    return p;
}

int main()
{
    char *str = NULL;

    str=GetMemory(str,100);
    strcpy(str,"hello");
    return 0;
}

```

我们可以对这道题推而广之，看一下整型变量是如何传值的，代码如下：

```

#include <iostream>
using namespace std;
void GetMemory2(int *z)
{
    *z=5;
};

int main()
{

```

```

    int v;
    GetMemory2(&v);
    cout << v << endl;
    return 0;
}

```

GetMemory2 把 v 的地址传了进来，*z 是地址里的值，是 v 的副本。通过直接修改地址里的值，不需要有返回值，也把 v 给修改了，因为 v 所指向地址的值发生了改变。

答案：程序崩溃。因为 GetMemory 并不能传递动态内存，Test 函数中的 str 一直都是 NULL。

面试例题 2：What will happen after running the "Test"? (这个程序测试后会有什么结果?) [美国某著名计算机嵌入式公司 2005 年 9 月面试题]

```

#include <iostream>
using namespace std;
char *GetMemory(void)
{
    char p[] = "hello world";
    return p;
}
int main()
{
    char *str = NULL;
    str = GetMemory();
    cout << str;
    return 0;
}

```

答案：可能是乱码，也有可能是正常输出。因为 GetMemory 返回的是指向“栈内存”的指针，该指针的地址不是 NULL，但其原来的内容已经被清除，新内容不可知。

面试例题 3：What is the output of the following code? (下列代码的输出是什么?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

```

#include <iostream>
using namespace std;
void change(int* a, int &b, int c)
{
    c=*a;
    b=3;
    *a=2;
}

int main()

```

```

    {
        int a=1,b=2,c=3;
        change(&a,b,c);
        cout<< a <<b << c;
        return 0;
    }

```

- A. 2 3 3 B. 1 2 3 C. 2 3 1 D. 1 3 3

解析：这是指针与引用参数传递问题。

a是传了一个地址进去，修改地址后直接修改了内容，所以a最后是2。

b是传的值。但是函数体内，做了一个b的引用&b，修改了函数体内b的值，就是修改b本身。举个形象的例子而言，比如说b是一个人，叫王富贵，把这个名字传进函数体，函数首先给王富贵取了个别名叫王二，然后给王二发了500块钱，这也就相当于给王富贵发了500块钱。

c传的是值，在函数体内被修改了，但没有返回。所以c不变，还是3。

答案：A。

面试例题4：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司2005年10月面试题]

```

#include <stdio.h>

void swapxy(char* a, char* b) {
    int x=*a, y=*b;

    x=x+y;
    y=x-y;
    x=x-y;

    *a=x, *b=y;

    return;
}

int main() {
    char a='a',b='B';
    char &x=a, &y=b;

    printf("a='%c' and b='%c' before swaping.\n",a,b);
    swapxy(x,y);
    printf("a='%c' and b='%c' after swaping.\n",a,b);

    return 0;
}

```

解析：这道程序体存在着引用的使用问题。

答案：

正确的程序如下：

```
#include <cstdio>

void swapxy(char& a, char& b) { //错误
    int x=a, y=b;

    x=x+y;
    y=x-y;
    x=x-y;

    a=x, b=y;

    return;
}

int main() {
    char a='a',b='B';
    char &x=a, &y=b;

    printf("a='%c' and b='%c' before swaping.\n",a,b);
    swapxy(x,y);
    printf("a='%c' and b='%c' after swaping.\n",a,b);

    return 0;
}
```

面试例题 5：关于函数的参数在调用和返回过程中的值，下列哪个说法是正确的。（ ） [中国某著名软件企业 2005 年面试题]

- A. 实参不会改变
- B. 实参可能会改变
- C. 如果是指针，肯定不会改变
- D. 如果不是指针，可能会改变

解析：实参如果传的是指针的话很有可能会改变。不是指针，如果是无返回值的，则不会改变。因为指针传的是地址，直接给地址赋值。而计算机中的地址是独一无二的，所以很有可能改变。

答案： B。

面试例题 6：What results after run the following code? (下列代码的运行结果是什么？) [中国台湾某著名 CPU 生产公司 2005 年面试题]

```
int *ptr;
ptr=(int*)0x8000;
*ptr=oxaabb;
```

解析：指针问题。

答案：这样做会导致运行时错误，因为这种做法会给一个指针分配一个随意的地址，这是非常危险的。不管这个指针有没有被使用过，这么做都是不允许的。

面试例题 7：写出下列程序的输出结果。[中国某著名软件外包企业 2005 年面试题]

```
#include <iostream>
using namespace std;
class A
{
public:virtual void print(void)
{
    cout<<"A::print()"<<endl;
}
class B:public A
{
public:virtual void print(void)
{
    cout<<"B::print()"<<endl;
}
class C:public A
{
public:void print(void)
{
    cout<<"C::print()"<<endl;
}
void print(A a)
{
    a.print();
}
int main(void)
{
    A a, *pa,*pb,*pc;
    B b;
    C c;

    pa=&a;
    pb=&b;
    pc=&c;

    a.print();
    b.print();
    c.print();

    pa->print();
}
```

```

pb->print();
pc->print();

print(a);
print(b);
print(c);
return 0;
}

```

解析：指针问题。

答案：

```

A::print()
B::print()
C::print()
A::print()
B::print()
C::print()
A::print()
A::print()
A::print()

```

7.3 函数指针

面试例题 1：写出函数指针、函数返回指针、const 指针、指向 const 的指针、指向 const 的 const 指针。

答案：

```

void (*f)()
void* f()
const int*
int* const
const int* const

```

面试例题 2：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

```

//The program need output the maximum value among three integer.
#include <stdio.h>

int max(int x,int y) {
    return x>y?x:y;
}

int main() {
    int max(x,y);
}

```

```
int *p=&max;
int a,b,c,d;
printf("Please input three integer \n");
scanf("%d%d%d",a,b,c);
d=(*p)((*p)(a,b),c);
printf("Among %d, %d, and %d, the maximal integer is
      %d\n", a,b,c,d);
return 0;
}
```

解析：这道程序体存在着函数指针的错误使用问题。

答案：

正确的程序如下：

```
#include <stdio.h> //最好使用 <cstdio>

int max(int x,int y) {
    return x>y?x:y;
}

int main() {
    int max(int,int); //错误1
    int (*p)(int,int)=&max; //错误2
    int a,b,c,d;
    printf("Please input three integer \n");
    scanf("%d%d%d",&a,&b,&c); //错误3
    d=(*p)((*p)(a,b),c);
    printf("Among %d, %d, and %d, the maximal integer is
          %d\n", a,b,c,d);
    return 0;
}
```

面试例题 3： Write in words the data type of the identifier involved in the following definitions. (下面的数据声明都代表什么？) [美国某著名计算机嵌入式公司 2005 年 9 月面试题]

- (1) float(**def)[10];
- (2) double*(*gh)[10];
- (3) double(*f[10])();
- (4) int*((*b)[10]);
- (5) Long (* fun)(int)
- (6) Int (*(*F)(int,int))(int)

解析：函数指针的问题。

就像数组名是指向数组第一个元素的常指针一样，函数名也是指向函数的常指针。可以声明一个指向函数的指针变量，并且用这个指针来

调用其他函数——只要这个函数和你的函数指针在签名、返回、参数值方面一致即可。

```
Long (* fun)(int)
```

上面就是一个函数指针——指向函数的指针，这个指针返回值是 long，所带的参数是 int。如果去掉(* fun)的“()”它就是指针函数，是一个带有整数参量并返回一个长整型变量的指针的函数。

```
Int (*(*F)(int,int))(int)
```

如上所示，F 是一个指向函数的指针，它指向一种这样的函数（该函数参数为 int, int 返回值为一个指针），返回的这个指针指向的是另外一个函数（参数类型为 int，返回值为 int 类型的函数）。

答案：

(1) float(**def)[10];

def 是一个二级指针，它指向的是一个一维数组的指针，数组的元素都是 float。

(2) double*(*gh)[10];

gh 是一个指针，它指向一个一维数组，数组元素都是 double*。

(3) double(*f[10])();

f 是一个数组，f 有 10 个元素，元素都是函数的指针，指向的函数类型是没有参数且返回 double 的函数。

(4) int*((*b)[10]);

就跟 “int* (*b)[10]” 是一样的，b 是一维数组的指针。

(5) Long (* fun)(int)

函数指针。

(6) Int (*(*F)(int,int))(int)

F 是一个函数的指针，指向的函数的类型是有两个 int 参数并且返回一个函数指针的函数，返回的函数指针指向有一个 int 参数且返回 int 的函数。

7.4 指针数组和数组指针

面试例题 1：

```
int a[9];
```

```
int *p;  
p = a;
```

请问哪一个不能表示 `a[1];`? [中国某著名计算机金融软件公司 2005 年面试题]

- A. `p+1` B. `p++` C. `a++` D. `a+1`

解析：数组名 `a` 作为代表数组的首地址，是一个常量指针，既然是常量当然是不容修改的，所以 `a++` 是错误的。`p=a;` 中的 `p`、`a` 表示相同的地址。但是，`int* p;` 说明 `p` 是指针变量，既然是变量当然就可以修改了，所以 `p++` 可以表示 `a[1];`。

答案：C。

面试例题 2： What is the output of these statements? (下面程序的输出结果是什么?) [中国台湾某著名杀毒软件公司面试题]

```
static int a[3][3]={1,3,5,7,9,11,13,15,17},y,x,*p=&a[2][2];  
for(x=0;x<3;x++)  
    y+=*(p+4*x);  
printf("\n%d",y);
```

- A. 45 B. 33 C. 17 D. 27

解析：数组指针问题。

3 次循环相加的数分别是 $17+9+1=27$ 。

答案：D。

面试例题 3： 一个指向整型数组的指针的定义为：()。

- A. `int (*ptr)[]` B. `int *ptr[]` C. `int *(ptr[])` D. `int ptr[]`

解析：

`int (*ptr)[]` 是一个指向整型数组的指针。

`int *ptr[]` 是指针数组，`ptr[]` 里面存的是地址。它指向位置的值就是 `*ptr[0]`、`*ptr[1]`、`*ptr[2]`、`*ptr[3]`。不要存 `*ptr[0]=5;`、`*ptr[1]=6;`，因为这里面没有相应的地址。

`int *(ptr[])` 与 B 相同。

`int ptr[]` 是一个普通的数组。

答案：A。

扩展知识

a 是指针数组，是指一个数组里面装着指针。

b 是指向数组的指针，代表它是指针，指向整个数组。

以下是指针数组 a:

```
#include<iostream>
using namespace std;
int main()
{
    static int a[2]={1,2};

    int *ptr[5]; //指针数组
    int p=5,p2=6,*page,*page2;
    page = &p;
    page2 = &p2;

    ptr[0]=&p;
    ptr[1]=page2;

    cout << *ptr[0] << endl;
    cout << *page << endl;
    cout << *ptr[1] << endl;
    return 0;
}
```

下面是数组指针 b:

```
#include<stdio.h>
#include<iostream>
using namespace std;
int main()
{
    static int a[2]={1,2};

    int p=5,p2=6,*page,*page2;
    int Test[2][3] = {{1,2,3}, {4,5,6}};//测试用二维数组
    int Test2[3] = {1,2,3};//测试用一维数组
    page = &p;
    page2 = &p2;

    ptr[0]=&p;
    ptr[1]=page2;
    //int (*A)[3] = &Test[1];
    int (*A)[3],(*B)[3]; //数组指针
    A = &Test[1];
    B = &Test2;
    cout << *page << endl;
    cout << (*A)[0] << (*A)[1] << (*A)[2] << endl;
    cout << (*B)[3] << endl;
```

```

    return 0;
}

```

面试例题 4：用变量 a 给出下面的定义：[中国台湾某著名 CPU 生产公司 2005 年面试题]

- a) 一个整型数 (An integer)
- b) 一个指向整型数的指针 (A pointer to an integer)
- c) 一个指向指针的指针，它指向的指针是指向一个整型数 (A pointer to a pointer to an integer)
- d) 一个有 10 个整型数的数组 (An array of 10 integers)
- e) 一个有 10 个指针的数组，该指针是指向一个整型数的 (An array of 10 pointers to integers)
- f) 一个指向有 10 个整型数数组的指针 (A pointer to an array of 10 integers)
- g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数 (A pointer to a function that takes an integer as an argument and returns an integer)
- h) 一个有 10 个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数 (An array of 10 pointers to functions that take an integer argument and return an integer)

解析：这道面试例题是嵌入式编程和指针运用中经常考到的问题。这又是那种要翻一下书才能回答的问题。当我写这本书时，为了确定语法的正确性，我的确查了一下书。但是当我被面试的时候，我期望被问到这个问题（或者相近的问题）。因为在被面试的这段时间里，我确定我知道这个问题的答案。应试者如果不知道所有的答案（或至少大部分答案），那么也就没有为这次面试做好准备。如果该面试者没有为这次面试做好准备，那么他又能为什么做好准备呢？

答案：

- a) int a; // An integer
- b) int *a; // A pointer to an integer
- c) int **a; // A pointer to a pointer to an integer
- d) int a[10]; // An array of 10 integers

- e) int *a[10]; // An array of 10 pointers to integers
- f) int (*a)[10]; // A pointer to an array of 10 integers
- g) int (*a)(int); // A pointer to a function that takes an integer argument and returns an integer
- h) int (*a[10])(int); //An array of 10 pointers to functions that take an integer argument and return an integer

7.5 迷途指针

面试例题 1：以下代码有什么错误？会造成什么问题？[美国某著名 CPU 生产公司面试题]

```

typedef unsigned short int USHORT;
#include <iostream.h>
int main()
{
    USHORT * pInt = new USHORT;
    *pInt = 10;
    cout << "*pInt: " << *pInt << endl;
    delete pInt;
    pInt = 0;
    long * pLong = new long;
    *pLong = 90000;
    cout << "*pLong: " << *pLong << endl;

    *pInt = 20;      // uh oh, this was deleted!

    cout << "*pInt: " << *pInt << endl;
    cout << "*pLong: " << *pLong << endl;
    delete pLong;
    return 0;
}

```

解析：编程中一种很难发现的错误是迷途指针。迷途指针也叫悬浮指针、失控指针，是当对一个指针进行 `delete` 操作后——这样会释放它所指向的内存——并没有把它设置为空时产生的。而后，如果你没有重新赋值就试图再次使用该指针，引起的结果是不可预料的。程序崩溃算你走运。

这就如同一家水果公司搬家了，但你使用的仍然是它原来的电话号码。这可能不会导致什么严重的后果——也许这个电话号码是放在一个无

人居住的房子里面。另一方面，这个号码也可能被重新分配给一个军工厂，你的电话可能引起爆炸，把整个城市炸毁。

总之，在删除指针后小心不要再使用它。虽然这个指针仍然指向原来的内存区域，但是编译器已经把这块内存区域分配给了其他的数据。再次使用这个指针会导致你的程序崩溃。更糟糕的情况是，程序可能表面上运行得很好，过不了几分钟就崩溃了。这被称为定时炸弹，可不是开玩笑。为了安全起见，在删除一个指针后，把它设置为空指针（0）。这样就可以消除它的危害。

在本题中，首先声明了一个指针 pInt，然后打印。打印后使用 delete 将其删除。那么现在 pInt 就是一个迷途指针，或者说是悬浮指针。

第二步，声明了一个新的指针 pLong，把 90 000 赋值给它，然后打印。

第三步，把值 20 赋给 pInt 所指向的内存区域，但此时 pInt 不指向任何有效的空间。因为它原来所指向的内存空间已经被释放了，所以这样做会给内存区域带来很坏的结果。

第四步，打印 pInt 的值，结果是 20。再打印 pLong 的值，发现它变成了 65556 而不是 90 000 了。这是因为把 90 000 赋给*pLong 后，它的实际存储为 5F 90 00 01。把 20（也就是十六进制的 00 14）赋给指针 pInt，因为指针 pInt 仍然指向相同的地址，因此 pLong 的前两个字节被覆盖了，变成了 00 14 00 01，所以打印的结果变成了 65 556。

答案：以上程序使用了迷途指针并重新赋值，会造成系统崩溃。

面试例题 2：空指针和迷途指针的区别是什么？

答案：当 delete 一个指针的时候，实际上仅是让编译器释放内存，但指针本身依然存在。这时它就是一个迷途指针。

当使用以下语句时，可以把迷途指针改为空指针：

```
MyPtr=0;
```

通常，如果在删除一个指针后又把它删除了一次，程序就会变得非常不稳定，任何情况都有可能发生。但是如果你只是删除了一个空指针，则什么事都不会发生，这样做非常安全。

使用迷途指针或空指针（如 MyPtr=0）是非法的，而且有可能造成

程序崩溃。如果指针是空指针，尽管同样是崩溃，但它同迷途指针造成的崩溃相比是一种可预料的崩溃。这样调试起来会方便得多。

面试例题3：C++中有了 malloc/free，为什么还需要 new/delete？

答案：malloc 与 free 是 C++/C 语言的标准库函数，new/delete 是 C++ 的运算符。它们都可用于申请动态内存和释放内存。

对于非内部数据类型的对象而言，光用 malloc/free 无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于 malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于 malloc/free。

因此 C++ 语言需要一个能完成动态内存分配和初始化工作的运算符 new，以及一个能完成清理与释放内存工作的运算符 delete。new/delete 不是库函数，而是运算符。

7.6 指针和句柄

面试例题1：句柄和指针的区别和联系是什么？[英国某著名计算机图形图像公司面试题]

解析：句柄是一个 32 位的整数，实际上是 Windows 在内存中维护的一个对象（窗口等）内存物理地址列表的整数索引。因为 Windows 的内存管理经常会将当前空闲对象的内存释放掉，当需要时访问再重新提交到物理内存，所以对象的物理地址是变化的，不允许程序直接通过物理地址来访问对象。程序将想访问的对象的句柄传递给系统，系统根据句柄检索自己维护的对象列表就能知道程序想访问的对象及其物理地址了。

句柄是一种指向指针的指针。我们知道，所谓指针是一种内存地址。应用程序启动后，组成这个程序的各对象是驻留在内存的。如果简单地理解，似乎我们只要获知这个内存的首地址，那么就可以随时用这个地址访问对象。但是，如果您真的这样认为，那么您就大错特错了。我们知道，Windows 是一个以虚拟内存为基础的操作系统。在这种系统环境下，Windows 内存管理器经常在内存中来回移动对象，以此来满足各种

应用程序的内存需要。对象被移动意味着它的地址变化了。如果地址总是如此变化，我们该到哪里去找该对象呢？为了解决这个问题，Windows 操作系统为各应用程序腾出一些内存地址，用来专门登记各应用对象在内存中的地址变化，而这个地址（存储单元的位置）本身是不变的。Windows 内存管理器移动对象在内存中的位置后，把对象新的地址告知这个句柄地址来保存。这样我们只需记住这个句柄地址就可以间接地知道对象具体在内存中的哪个位置。这个地址是在对象装载（Load）时由系统分配的，当系统卸载时（Unload）又释放给系统。句柄地址（稳定）→记载着对象在内存中的地址→对象在内存中的地址（不稳定）→实际对象。但是，必须注意的是，程序每次重新启动，系统不能保证分配给这个程序的句柄还是原来的那个句柄，而且绝大多数情况下的确不一样。假如我们把进入电影院看电影看成是一个应用程序的启动运行，那么系统给应用程序分配的句柄总是不一样，这和每次电影院售给我们的门票总是不同的座位是一样的道理。

HDC 是设备描述表句柄。

CDC 是设备描述表类。

用 GetSafeHwnd 和 FromHandle 可以互相转换。

答案：句柄和指针其实是两个截然不同的概念。Windows 系统用句柄标记系统资源，用句柄隐藏系统的信息。你只要知道有这个东西，然后去调用就行了，它是个 32bit 的 uint。指针则标记某个物理内存地址，是不同的概念。

第 8 章

循环、递归与概率

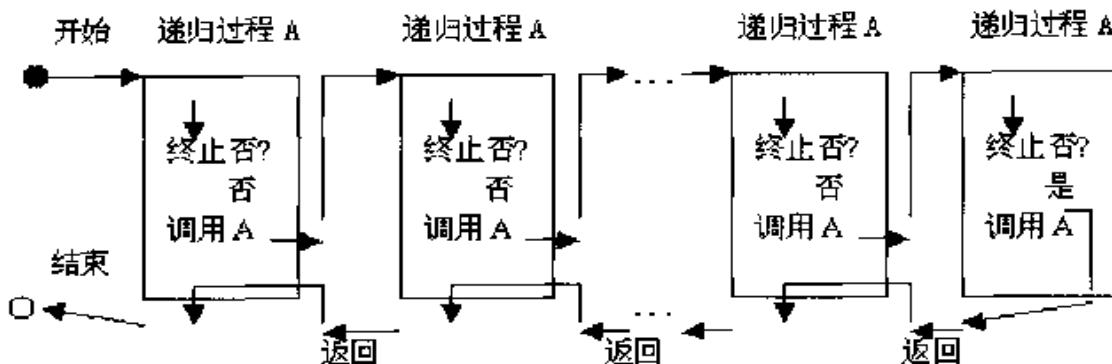
递归问题是求职笔试中最为复杂的地方，也是本书的难点之一。由递归衍生出的相关问题，诸如迭代问题、概率问题、循环问题也是企业经常重复的考点。在阅读本章之前，请读者参考数据结构经典书籍对递归基础知识做简要复习。本章将通过对各公司面试题目进行全面仔细的解析，帮助读者解决其中的难点。

8.1 递归基础知识

递归是程序设计中的一种算法。一个过程或函数直接调用自己本身或通过其他的过程或函数调用语句间接地调用自己的过程或函数，称为递归过程或函数。递归是计算机语言中的一种很有用的工具，很多数学公式用到递归定义，例如 $N!$ ：当 $n > 0$ 时， $f(n) = n \times f(n - 1)$ 。

有些数据结构（如二叉树），其结构本身就有递归的性质。有些问题本身没有明显的递归结构，但用递归求解更简单。

递归是较难理解的算法之一。简单地说，递归就是编写这样一个特殊的过程，该过程体中有一个语句用于调用过程自身（称为递归调用）。递归过程由于实现了自我的嵌套执行，使这种过程的执行变得复杂起来，其执行的流程如下图所示。



递归过程的执行总是一个过程体未执行完，就带着本次执行的结果又进入另一轮过程体的执行，……，如此反复，不断深入，直到某次过程的执行时终止递归调用的条件成立，则不再深入，而执行本次的过程体余下的部分，然后又返回到上一次调用的过程体中，执行余下的部分，……，如此反复，直到回到起始位置上，才最终结束整个递归过程的执行，得到相应的执行结果。递归过程程序设计的核心就是参照这种执行流程，设计出一种适合“逐步深入，而后又逐步返回”的递归调用模型，以解决实际问题。

递归算法应该包括两种情况：递归情况和基底情况。递归情况演变到最后必须达到一个基底。

在程序设计面试中，一个能够完成任务的解决方案是最重要的，解决方案的执行效率要放在第二位考虑。因此，除非试题另有要求，你应该从最先想到的解决方案入手。如果它是一个递归性的方案，你不妨向面试官说明一下递归算法天生的低效率问题——表示你知道这些事情。有时候你同时想到两个解决方案：递归的和循环的，并且实现方式差不多，你可以把两个都向考官介绍一下。比如 $N!$ 问题，利用循环语句解释就是：

```

int find(int i)
{
    int n, val=1;
    for(n=i; n>1; n--)
        val*=n;
    return val;
}
  
```

不过，如果用循环语句做的改进算法实现起来要比递归复杂得多的话——大幅度增加了复杂度而在执行效率上得不到满意的回报，我们建议还是优先选择递归来解决问题。

8.2 典型递归问题

面试例题 1：利用递归调用手段编程计算 $N!$ 。[日本某著名电子/数码/IT 企业面试题]

解析：根据数学知识， $1!=1$ ，正整数 N 的阶乘为： $N \times (N - 1) \times (N - 2) \times \dots \times 2 \times 1$ ，该阶乘序列可转换为求 $N \times (N - 1)!$ ，而 $(N - 1)!$ 可以转换为 $(N - 1) \times (N - 2)!$ ，……，直至转换为 $2 \times 1!$ ，而 $1!=1$ 。

在函数 `find` 中，当 $N > 1$ 时，又调用过程函数，参数为 $N - 1$ ，……，这种操作一直持续到 $N = 1$ 为止。例如当 $N = 5$ 时，`find(5)` 的值变为 $5 * \text{find}(4)$ ，求 `find(4)` 又变为 $4 * \text{find}(3)$ ，……，当 $N = 1$ 时递归停止，逐步返回到第一次调用的初始处，返回结果为 $5 \times 4 \times 3 \times 2 \times 1$ ，即 $5!$ 。

答案：

代码如下：

```
int n,t;
int find(int n);
{
    if (n==1) t=1;
    else
    {
        return find(n-1)*n;
    }
}
main()
{
    cin >> n;
    find(n);
    cout ('N!=' ,t:1:0);
}
```

面试例题 2：试用递归的方法写一下计算斐波那契数列的通项 $f(n)$ ，已知 $f_1=1$, $f_2=1$ ，以后每项都是前两项的和。[中国某著名综合软件公司面试题]

解析：

递归问题。

答案：

代码如下：

```
int f(int n)
{
```

```

if(n==1||n==2)
    return 1;
else
    return f(n-1)+f(n-2);
}

```

8.3 打靶

面试例题 1：一个射击运动员打靶，靶一共有 10 环，连开 10 枪打中 90 环的可能性有多少种？请用递归算法编程实现。[中国某著名通信企业 H 面试题]

解析：靶上一共有 10 种可能——1 环到 10 环，还有可能脱靶，那就是 0 环，加在一起共 11 种可能。这是一道考循环和递归的面试题。我们在这个程序中将利用递归的办法实现打靶所有可能的演示，并计算出结果。读者会问，难道一定要使用递归？当然不是。我们也可以连续用 10 个循环语句来表示程序，代码如下：

```

for (i1=0;i1<=10;i1++)
{
    for (i2=0;i2<=10;i2++)
    {
        for (i3=0;i3<=10;i3++)
        {
            .....
            for (i10=0;i10<=10;i10++)
            {
                if(i1+i2+i3+...+i10=90)
                    Print();
            }
            .....
        }
    }
}

```

但是，上面的循环程序虽然解决了问题，但时间复杂度和空间复杂度无疑是很高的。比较好的办法当然是采用递归的方式，事实上公司也就是这么设计的。递归的条件由以下 4 步完成：

(1) 如果出现这种情况，即便后面每枪都打 10 环也无法打够总环数 90，在这种情况下就不用再打了，则退出递归。代码如下：

```

if(score < 0 || score > (num+1)*10) //次数num为0~9
{
    return;
}

```

(2) 如果满足条件且打到最后一次(因为必须打10次), 代码如下:

```
if(num == 0)
{
    store2[num] = score;
    Output(store2);
    return;
}
```

(3) 如果没有出现以上两种情况则执行递归, 代码如下:

```
for(int i = 0; i <= 10; ++i)
{
    //这里实际上为了方便把顺序倒了过来, store2[9]是第1回
    //store2[8]是第2回……store2[0]是第10回
    store2[num] = i;
    Cumput(score - i, num - 1, store2);
}
```

(4) 打印函数, 符合要求的则把它打印出来。代码如下:

```
public static void Output(int[] store2)
{
    for(int i = 9; i>=0; --i)
    {
        Console.Write(" {0}", store2[i]);
    }
    Console.WriteLine();
    sum++;
}
```

答案:

用C#编写的完整代码如下:

```
using System;

public class M
{
    //public static int[] store;
    //相当于设置了全局变量
    //这个全局变量sum是包含在M类中的
    public static int sum;
    public M()
    {
        int sum = 0;
        // int[] store = {1,2,3,4,5,6,7,8,9,0};
    }

    //打印函数
    //符合要求的则把它打印出来
    public static void Output(int[] store2)
```

```

    {
        for(int i = 9; i>=0; --i)
        {
            Console.WriteLine(" {0}",store2[i]);
        }
        Console.WriteLine();
        sum++;
    }

    //计算总数，返回 sum 值
    public static int sum2()
    {
        return sum;
    }

    public static void Cumput(int score, int num, int[] store2 )
    {

        //如果总的成绩超过了 90 环（也就是 score<0），或者如果剩下要打靶
        //的成绩大于 10 环乘以剩下要打的次数，也就是说即便后面的都打 10 环
        //也无法打够次数，则退出递归
        if(score < 0 || score > (num+1)*10 ) //次数 num 为 0~9
        {
            return;
        }

        //如果满足条件且达到最后一层
        if(num == 0)
        {
            store2[num] = score;
            Output(store2);
            return;
        }

        for(int i = 0; i <= 10; ++i)
        {
            store2[num] = i;
            Cumput(score - i, num - 1,store2);
        }
        //Console.WriteLine(" {0}",store2[5]);
    }
}

public class myApp
{
    public static void Main( )
    {
        int[] store;
        store = new int[10];
        int sum = 0;
        //int a=90;
        //int b=9;
        //Output();
    }
}

```

```

        M.Cumput(90, 9, store);
        sum = M.sum2();

        //M.Cumput2(a, b, store);
        //Console.WriteLine(" {0}", store[3]);
        //cout<<"总数:"<<sum<<endl;
        Console.WriteLine(" 总数: {0}", sum);

    }
}

```

程序结果一共有 92 378 种可能。

也可以用 C++ 编写，代码如下：

```

#include <iostream>
using namespace std;
int sum;
int store[10];
void Output()
{
    for(int i = 9; i>=0; --i)
    {
        cout<<store[i]<<" ";
    }
    cout<<endl;
    ++sum;
}

void Cumput(int score, int num)
{
    if(score < 0 || score > (num+1)*10) //次数num为0~9
        return;
    if(num == 0)
    {
        store[num] = score;
        Output();
        return;
    }
    for(int i = 0; i <= 10; ++i)
    {
        store[num] = i;
        Cumput(score - i, num - 1);
    }
}

int main(int argc, char* argv[])
{
    Cumput(90, 9);
    cout<<"总数:"<<sum<<endl;
    return 0;
}

```

面试例题 2：八皇后问题是一个古老而著名的问题，是回溯算法的典型例题。该问题是 19 世纪著名的数学家高斯 1850 年提出：在 8×8 格的国际

象棋盘上摆放 8 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。[英国某著名计算机图形图像公司面试题]

解析：递归实现 n 皇后问题。

算法分析：

数组 a、b、c 分别用来标记冲突，a 数组代表列冲突，从 a[0]~a[7] 代表第 0 列到第 7 列。如果某列上已经有皇后，则为 1，否则为 0。

数组 b 代表主对角线冲突，为 b[i-j+7]，即从 b[0]~b[14]。如果某条主对角线上已经有皇后，则为 1，否则为 0。

数组 c 代表从对角线冲突，为 c[i+j]，即从 c[0]~c[14]。如果某条从对角线上已经有皇后，则为 1，否则为 0。

代码如下：

```
#include <stdio.h>

static char Queen[8][8];
static int a[8];
static int b[15];
static int c[15];
static int iQueenNum=0; //记录总的棋盘状态数

void qu(int i); //参数 i 代表行

int main()
{
    int iLine, iColumn;
    //棋盘初始化，空格为*，放置皇后的地方为@
    for(iLine=0; iLine<8; iLine++)
    {
        a[iLine]=0; //列标记初始化，表示无列冲突
        for(iColumn=0; iColumn<8; iColumn++)
            Queen[iLine][iColumn]='*';
    }

    //主、从对角线标记初始化，表示没有冲突
    for(iLine=0; iLine<15; iLine++)
        b[iLine]=c[iLine]=0;

    qu(0);
    return 0;
}

void qu(int i)
{
    int iColumn;
    for(iColumn=0; iColumn<8; iColumn++)

```

```

{
    if(a[iColumn]==0&&b[i-iColumn+7]==0&&c[i+iColumn]==0)
        //如果无冲突
    {
        Queen[i][iColumn]='@';           //放皇后
        a[iColumn]=1;                   //标记，下一次该列上不能放皇后
        b[i-iColumn+7]=1;              //标记，下一次该主对角线上不能放皇后
        c[i+iColumn]=1;                //标记，下一次该从对角线上不能放皇后
        if(i<7) qu(i+1);             //如果行还没有遍历完，进入下一行
        else //否则输出
        {
            //输出棋盘状态
            int iLine,iColumn;
            printf("第%d 种状态为: \n",++iQueenNum);
            for(iLine=0;iLine<8;iLine++)
            {
                for(iColumn=0;iColumn<8;iColumn++)
                    printf("%c ",Queen[iLine][iColumn]);
                printf("\n");
            }
            printf("\n\n");
        }

        //如果前次的皇后放置导致后面的放置无论如何都不能满足要求，则回溯，重置
        Queen[i][iColumn]='*';
        a[iColumn]=0;
        b[i-iColumn+7]=0;
        c[i+iColumn]=0;
    }
}

```

8.4 字符子串

面试例题 1：请编写一个函数，把字符串中的所有字符子串的各种组合形式全部都显示出来。字符子串的长度范围是从一个字符到字符串的长度。不管排列顺序如何，只要两种组合中的字符完全一样，它们就是同一种组合。比如：给定字符串“hart”，则“ha”和“har”是不同的组合，而“ha”和“ah”是相同的组合。

解析：我们首先分析一下该题的解答思路。先手写一个例子，看看能说明什么，因为通过一个例子可以推导出相应的算法来。我们使用“hart”作为字符串，按照字符串的长度进行分类，得出下面的各种组合：

h	ha	har	hart
a	hr	hat	
r	ht	hrt	
t	ar	art	
	at		
	rt		

似乎有一定的规律，但是不明朗。我们修改一下它的排列方式——不是以字符串数量多少做排列区分，而是以是否存在某一字符来确定字符串：

h	ha	har	hart	hat	hr	hrt	ht
a	ar	art	at				
r	rt						
t							

大家看上面分类的规律：第一行全是有 h 的，第二行全是没有 h 但有 art 的，第三行全是没有 ha 但有 rt 的，第四行全是没有 har 但有 t 的。我们再分析第一行：

h	ha	har	hart	hat	hr	hrt	ht
---	----	-----	------	-----	----	-----	----

如果把所有 h 去掉，可以得到：

a	ar	art	at
r	rt		
t			

这是一个非常有趣的现象，以上得到的结果正是原来后 3 个字符 art 的组合，就好像是 art 先组合好，再“粘”上一个 h 就变成了新的组合。根据这样的特点，我们可以设计一个递归算法。

算法分为 4 步：

- (1) 从首字符到尾字符的各个字符，循环。
- (2) 输出被循环到的字符 i。
- (3) 如果循环到的字符 i 后面还有字符，递归，以后面的字符为起始字符，重复步骤 (2)、步骤 (3)。
- (4) 如果被循环到的字符 i 后面没有字符，跳出循环。

答案：

程序源代码如下：

```
#include <iostream>
using namespace std;
int sum=0;
char str[]="hart";
int length;
char *out;
```

```

void DoCombine(char in[],char out[],int length,int rec,int start)
{
    int i;
    for(int i = start; i < length; i++)
    {
        out[rec] = in[i];
        out[rec+1] = 0;
        printf("%s\n",out);
        if(i< length-1)
            DoCombine(in,out,length,rec+1,i+1);
    }
}

int main(int argc, char* argv[])
{
    length = strlen(str);
    out = (char *)malloc(length +1);
    DoCombine(str,out,length,0,0);
    return 0;
}

```

8.5 循环语言

面试例题 1：完成下列程序。[日本某著名电子/数码/IT 企业面试题]

```

*
*.*.
*...*..*..
*...*...*...*...
*....*...,*...,*...,*...
*....*...,*...,*...,*...,*...
*....*...,*...,*...,*...,*...,*...
*....*...,*...,*...,*...,*...,*...
*....*...,*...,*...,*...,*...,*...
#include <stdio.h>
#define N 8
int main()
{
    int i;
    int j;
    int k;
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            for(k=0;k<N;k++)
            {
                if((i+j+k)%3==0)
                    printf("*");
                else
                    printf(" ");
            }
            printf(" ");
        }
        printf("\n");
    }
}

```

```

    return 0;
}

```

解析：分析这种循环问题首先要找到规律。为了打印以上图案，我们查找其规律，可以发现：

- (1) 该图案一共有 8 行。
- (2) 第 1 行 1 个 “*”，第 2 行 2 个 “*”，第 3 行 3 个 “*” ……
- (3) 第 1 行 0 个 “.”，第 2 行 1 个 “.”，第 3 行 2 个 “.” ……

答案：

根据以上规律，我们可以编写程序如下：

```

#include<stdio.h>
#include<conio.h>
#define N 8

int main()
{
    int i;
    int j;
    int k;
    clrscr();
    for(i = 1; i <= N; i++)
    {
        for(j=1; j <= i; j++)
        {
            printf("*");
            for(k = 1; k < i; k++)
            {
                printf(".");
            }
        }
        printf("\n");
    }
    getch();
    return 0;
}

```

面试例题 2：如何利用筛选法查找 1 000 以内的素数。[英国某著名计算机图形图像公司面试题]

解析：首先定义 $a[i]=1$ ，初始化整个数组，全部初始化为 1。第 2 步双重循环，从 2 开始，所有 2 的倍数都标记为 0。然后所有 3 的倍数都标记为 0。然后是 4，但因为 4 已经被标记为 0 了，跳过。然后是 5……直到所有数都循环一遍。

答案：

代码如下：

```
#include<iostream>
using namespace std;
int main()
{
    int a[101], i, j;
    for (i=1; i<101; i++)
        a[i]=1;
    //根据筛选法求出 100 以内的所有素数。所谓筛选法是指从小到大筛去一个已知素数
    //的所有倍数。例如，根据 2 我们可筛去 4、6、8、……、98、100 等数
    //然后根据 3 可筛去 9、15、……、
    //99 等数（注意此时 6、12 等数早就被筛去了）
    //由于 4 被筛去了，下一个用于筛选的素数是 5……依次类
    //推，最后剩余的就是 100 以内的素数
    for ( i=2; i<101; i++)
    {
        if (a[i]!=0)
            for (j=i+i; j<101;)
            {
                if (j%i==0)
                    a[j]=0;
                j=j+i;
            }
    }
    for(i=2;i<101;i++)
        if (a[i]!=0)
            cout << i << " ";
    return 0;
}
```

扩展知识

求素数的方法不止一种，下面是用开根号的办法求值，也可以达到目的。代码如下：

```
#include <math.h>
#include <stdio.h>
using namespace std;
int main()
{
    int a[101], i, j, k;
    for(i=1;i<100;i++)
    {
        k=(int)sqrt(i);
        for(j=2;j<=k;j++)
        {
```

```

        if(i%j==0)
            break;
    }
    if(j>k)
        printf("%5d",i);

    }
return 0;
}

```

8.6 0-1 背包

面试例题 1：输入两个整数 n 和 m ，从数列 1、2、3、…、 n 中随意取几个数，使其和等于 m ，要求将其中所有可能的组合列出来，编程求解。[中国某著名通信企业乙面试题]

解析：

这是一道 0-1 背包问题。原来的 0-1 背包问题是这样的：一个容积为 5 的箱子，现在要装入物品，物品一共有 4 个，体积分别是 1、2、3、4，问有几种填充方法。

这个题很简单，一共有两种填充方法：1、4 填充和 2、3 填充。0-1 背包问题与这个面试题是十分相似的。

这里的算法一共有以下 4 步。

(1) 比较 n 和 m 哪个大？如果 $n > m$ ，则把 m 赋值给 n 。因为如果 $n > m$ ，则那些 m 和 n 之间的数，是没有意义的。比如 $m=4$, $n=7$ ，那么既然 n 中的 5、6、7 要比 m 大，所以是没有意义的。于是把 m 赋值给 n ，这样 n 也等于 4，开始比较。

(2) 现在我们开始填充箱子，先把体积最大的物品放入，看是否与箱子所剩容积相等。如果相等的话，则说明合适，打印物品。

(3) 如果与箱子所剩容积不等，则用容积减去最大物品容积，看次大物品容积是否与箱子所剩容积相等，递归开始，直到物品全部比较完为止。

(4) 这里要设计一个 0-1 背包：0 代表没有装入该物品，1 代表装入该物品。比如一个容积为 5 的箱子，放入 4，可以装入，则 $f[4]=1$ ，代表 4 是装入了；然后装 3、2，装不下，所以 $f[3]=0$ $f[2]=0$ ，1 可以装入，则 $f[1]=1$ 。

答案：

程序源代码与解释如下：

```
#include <iostream>
#include <string>
using namespace std;
int mVal,nVal;
int * pOut;
void calFun( int m, int n )
{
    if ( m < 1 || n < 1 || ( n == 1 && m != 1 ) )
        return;

    //看是否与箱子所剩容积相等
    if ( m == n )
    {
        pOut[n] = 1;
        for ( int i = 1; i <= nVal; ++i )
        {
            if( pOut[i] )
                cout << i << " ";
        }
        cout<<endl;
        pOut[n] = 0;
    }

    calFun( m, n - 1 ); //不取n

    pOut[n] = 1;
    calFun( m - n, n - 1 );//取n
    pOut[n] = 0;
}

int main(int argc, char* argv[])
{
    cout<<"m:";
    cin>>mVal;
    cout<<"n:";
    cin>>nVal;

    if ( mVal < nVal )nVal = mVal; //比较n和m哪个大
    pOut = new int[nVal+1];
    memset( pOut, 0, (nVal+1)*sizeof(int) );

    calFun( mVal, nVal );
    delete []pOut;
    return 0;
}
```

8.7 概率

面试例题 1：Please write out the program output. (写出下面程序的运行结果。) [德国某著名软件咨询企业 2005 年 10 月面试题]

```

#include <stdlib.h>
#define LOOP 1000
void main()
{
    int rgnC=0;
    for(int i=0;i<LOOP;i++)
    {
        int x=rand();
        int y=rand();
        if(x*x+y*y < RAND_MAX*RAND_MAX) rgnC++;
    }
    printf("%d\n",rgnC);
}

```

解析：这是我所见到的概率面试例题中出得非常好的一道。

从表面上看，你完全无法看出它是一个概率问题。这里暗含的思想是一个 $1/4$ 圆和一个正方形比较大小的问题，如右图所示。

`RAND_MAX` 是随机数中的最大值，也就是相当于最大半径 R 。 x 和 y 是横、纵坐标上的两点，它们的平方和开根号就是原点到该点 (x,y) 的距离，当然这个距离有可能大于 R ，如 b 点，还有可能小于 R ，如 a 点。整个题目就蜕化成这样一个问题：随机在正方形里落 1 000 个点，落在半径里面的点有多少个。如果落在里面一个点，则累积一次。

那这个问题就很好解决了，求落点可能性之比，就是求一个 $1/4$ 圆面积和一个正方形面积之比。

$$\text{1/4 圆面积} = (1/4) \times \pi \times r \times r$$

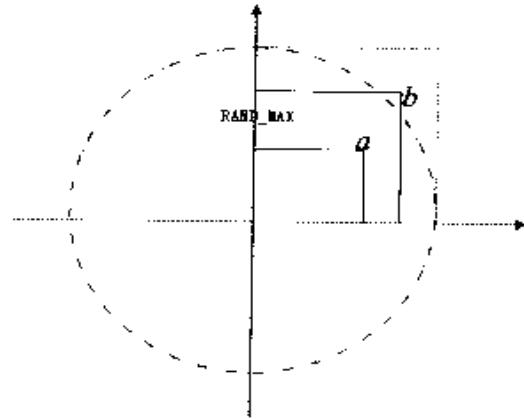
$$\text{正方形面积} = r \times r$$

$$\text{两者之比} = \pi/4$$

$$\text{落点数} = \pi/4 \times 1000 = 250 \times \pi \approx 785$$

答案：出题者的意思显然就是要求你得出一个大概值，也就是 $250 \times \pi$ 就可以了。实际上呢，你只要回答 700~800 之间都是正确的。

我们算的是落点值，落点越多，越接近 $250 \times \pi$ ，落 10 个点、100 点都是很不准确的，所以该题落了 1 000 个点。读者可以上机验证该程序，将 Loop 改成 10 000、100 000 来试验，会发现结果越来越接近 $250 \times \pi$ 。



第 9 章

STL 模板与容器

标 准模板库 (STL, Standard Template Library) 是当今每个从事 C++ 编程的人需要掌握的一项有用的技术。我发现最近各种外企的面试题中，它的比例逐渐增大。想学习 STL 的人应该花费一段时间来熟悉它，有一些命名是不太容易凭直觉就能够记住的。然而如果一旦你掌握了 STL，就不会觉得头痛了。和 MFC 相比，STL 更加复杂和强大。

STL 有以下优点：

- 可以方便、容易地实现搜索数据或对数据排序等一系列的算法。
- 调试程序时更加安全和方便。
- 即使是人们用 STL 在 UNIX 平台下写的代码，你也可以很容易地理解（因为 STL 是跨平台的）。

STL 中一些基础概念的定义如下。

- 模板 (Template) 类 (及结构等各种数据类型和函数) 的宏 (macro)。有时叫做甜饼切割机 (cookie cutter)，正规的名称应叫做泛型 (generic)。一个类的模板叫做泛型类 (generic class)，而一个函数的模板也自然而然地被叫做泛型函数 (generic function)。
- STL 标准模板库，一些聪明人写的一些模板，现在已成为每个人所使用的标准 C++ 语言中的一部分。
- 容器 (Container) 可容纳一些数据的模板类。STL 中有 vector、set、map、multimap 和 deque 等容器。
- 向量 (Vector) 基本数组模板，这是一个容器。

- 游标 (Iterator) 这是一个奇特的东西，它是一个指针，用来指向 STL 容器中的元素，也可以指向其他的元素。

9.1 向量容器

面试例题 1：介绍一下 STL 和包容器。如何实现？举例实现 vector。[美国某著名移动通信企业面试题]

答案：C++的一个新特性就是采用了标准模板库 (STL)。所有主要编译器销售商现在都把标准模板库作为编译器的一部分进行提供。标准模板库是一个基于模板的容器类库，包括链表、列表、队列和堆栈。标准模板库还包含许多常用的算法，包括排序和查找。

标准模板库的目的是提供对常用需求重新开发的一种替代方法。标准模板库已经经过测试和调试，具有很高的性能并且是免费的。最重要的是，标准模板库是可重用的。当你知道如何使用一个标准模板库的容器以后，就可以在所有的程序中使用它而不需要重新开发了。

容器是包容其他对象的对象。标准 C++ 库提供了一系列的容器类，它们都是强有力的工具，可以帮助 C++ 开发人员处理一些常见的编程任务。标准模板库容器类有两种类型：顺序和关联。顺序容器可以提供对其成员的顺序访问和随机访问。关联容器则经过优化关键值访问它们的元素。标准模板库在不同操作系统间是可移植的。所有标准模板库容器类都在 namespace std 中定义。

举例实现 vector 如下：

```
#include <iostream>
#include <vector>
using namespace std;

void print(vector<int>);

int main()
{
    vector<int> vec;
    vec.push_back(34);
    vec.push_back(23);
    print(vec);
    vector<int>::iterator p;
    p=vec.begin();
    *p=68;
    *(p+1)=69;
    /**(p+2)=70;
```

```

print(vec);
vec.pop_back();
print(vec);
vec.push_back(101);
vec.push_back(102);

int i=0;
while(i<vec.size())
cout << vec[i++] << " ";
cout << endl;
vec[0] = 1000;
vec[1] = 1001;
vec[2] = 1002;
//vec[3] = 1002;
i=0;
while(i<vec.size())
cout << vec[i++] << " ";

print(vec);
return 0;
}

void print(vector<int> v)
{
    cout << "\n vector size is: " << v.size() << endl;
    vector<int>::iterator p = v.begin();
}

```

或者是：

```

#include <iostream>
#include <vector>
using namespace std;

int sum(vector<int>vec)
{
    int result = 0;
    vector<int>::iterator p = vec.begin();
    while(p!=vec.end())
        result +=*p++;
    return result;
}

//void print(vector<int>);

int main()
{
    vector<int> v1(100);
    cout << v1.size() << endl; //100
    cout << sum(v1) << endl; //0
    v1.push_back(23);
    cout << v1.size() << endl; //101
    cout << sum(v1) << endl; //23
    v1.reserve(1000); //
    v1[900]=900;
    cout << v1[900] << endl; //900
    cout << v1.front() << endl; //0
    cout << v1.back() << endl; //23
    v1.pop_back();
    cout << v1.back() << endl; //0
    //vector<int>::iterator p2 = v1[0];
    // vector<int>::iterator p2 = &(v1[0]);
}

```

```

        return 0;
    }
}

```

面试例题 2: Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```

#include <vector>
using namespace std;

class CDemo {
public:
    CDemo():str(NULL){}
    ~CDemo() { if(str) delete[] str; }
    char* str;
};

int main(int argc, char** argv) {
    CDemo d1;
    d1.str=new char[32];
    strcpy(d1.str,"trend micro");

    vector<CDemo> *a1=new vector<CDemo>();
    a1->push back(d1);
    delete a1;

    return 0;
}

```

解析: vector 对象指针能够自动析构，所以不需要调用 delete a1，否则会造成两次析构对象。

答案：

正确程序如下：

```

#include <vector>
#include <iostream>
using namespace std;

int i = 0;
int j = 0;
class CDemo {
public:
    CDemo():str(NULL)
    {
        cout<<"constructor:"<< i++<< endl;
    };
    CDemo(const CDemo &cd)
    {
        cout<<"copy constructor:"<< i++<< endl;
        this->str = new char[strlen(cd.str)+1];
        strcpy(str,cd.str);
    };
    ~CDemo()
    {
        if(str)
        {

```

```

        cout<<"destructor:"<< j++<< endl;
        delete[] str;
    }
};

char* str;
};

int main(int argc, char** argv) {
    CDemo d1;
    d1.str=new char[32];
    strcpy(d1.str,"trend micro");

    vector<CDemo> *al=new vector<CDemo>();
    al->push back(d1);
    delete al;

    return 0;
}

```

面试例题 3：以下代码有什么问题？如何修改？[中国某著名综合软件公司
2005 年面试题]

```

#include <iostream>
#include <vector>
using namespace std;

void print(vector<int>);

int main()
{
    vector<int> array;
    array.push back(1);
    array.push back(6);
    array.push back(6);
    array.push back(3);
    //删除 array 数组中所有的 6
    vector<int>::iterator itor;
    vector<int>::iterator itor2;
    itor=array.begin();

    for(itor=array.begin(); itor!=array.end(); )
    {
        if(6==*itor)
        {
            itor2=itor;
            array.erase(itor2);
        }
        itor++;
    }

    print(array);
    return 0;
}

void print(vector<int> v)
{
    cout << "\n vector size is: " << v.size() << endl;
    vector<int>::iterator p = v.begin();
}

```

解析：

这是迭代器问题，只能删除第一个 6，以后迭代器就失效了，不能删除之后的元素。

itor2=itor;这句说明两个迭代器是一样的。array.erase(itor2);等于array.erase(itor);，这时指针已经指向下一个元素 6 了。itor++;又自增了，指向了下一个元素 3，略过了第一个 6。

答案：

修改方法 1：使用 vector 模板里面的 remove 函数进行修改。代码如下：

```
#include <iostream>
#include <vector>
using namespace std;

void print(vector<int>);

int main()
{
    vector<int> array;
    array.push_back(1);
    array.push_back(6);
    array.push_back(6);
    array.push_back(3);
    //删除 array 数组中所有的 6
    vector<int>::iterator itor;
    vector<int>::iterator itor2;
    itor=array.begin();

    array.erase( remove( array.begin(), array.end(), 6 ) ,
                array.end() );

    print(array);
    return 0;
}
void print(vector<int> v)
{
    cout << "\n vector size is: " << v.size() << endl;
    vector<int>::iterator p = v.begin();
}
```

修改方法 2：为了让其不略过第二个“6”，可以使“itor--；”，再回到原来的位置上。具体代码如下：

```
#include <iostream>
#include <vector>
using namespace std;

void print(vector<int>);

int main()
{
    vector<int> array;
    array.push_back(1);

    array.push_back(6);
```

```

array.push_back(6);
array.push_back(3);
//删除array数组中所有的6
vector<int>::iterator itor;
vector<int>::iterator itor2;
itor=array.begin();

for(itor=array.begin(); itor!=array.end();itor++)
{
    if(6==*itor)
    {
        itor2=itor;
        array.erase(itor2);
        itor--;
    }
    //itor--;
}

print(array);
return 0;
}

void print(vector<int> v)
{
    cout << "\n vector size is: " << v.size() << endl;
    vector<int>::iterator p = v.begin();
}

```

9.2 泛型编程

面试例题 1：何谓泛型编程？[美国某著名 CPU 生产公司面试题]

答案：STL 代表用一致的方式编程是可能的。实际上，它完全不同于我们过去看到的 C++ 编程，也完全不同于大多数教科书里所描述的方式。STL 并不是试图用 C++ 编程，只是试图找到一种正确的方式来处理软件，寻找一种可以表达我的想法的语言而已。换句话说，我知道我想说什么。我能用 C++ 说，我能用 Ada 说，我能用 Scheme 说。我可以让自己去适应语言，但我说的东西从本质上和语言无关。

面试例题 2：解释一下什么是泛型编程，泛型编程和 C++ 及 STL 的关系是什么？并且，你是怎么在 C++ 环境里进行泛型编程的？[美国某著名 CPU 生产公司面试题]

答案：泛型编程是一种基于发现高效算法的最抽象表示的编程方法。也就是说，以算法为起点并寻找能使其工作且有效率工作的最一般的必

要条件集。令人惊讶的是，很多不同的算法都需要相同的必要条件集，并且这些必要条件有多种不同的实现方式。类似的事实在数学里也可以看到。大多数不同的定理都依赖于同一套公理，并且对于同样的公理存在多种不同的模型。抽象机制！泛型编程假定有某些基本的法则在支配软件组件的行为，并且基于这些法则有可能设计可互操作的模块，甚至还有可以使用此法则去指导我们的软件设计。STL 就是一个泛型编程的例子。C++是我可以实现令人信服的例子的语言。

面试例题 3：Below is usual way we find one element in an array:

In this case we have to bear the knowledge of value type "int", the size of array, even the existence of an array. Would you re-write it using template to eliminate all these dependencies? (我们通常求解一个数在一个数列里的方法是这样的：在这个例子中，我们得先知道 int 类型的知识，知道队列的大小，甚至要建立一个队列。你能否用泛型编程模拟这个队列，做到不知道上面的附加知识仍然能得出结果？) [德国某著名软件咨询企业 2004 年面试题]

```
const int *find1(const int* array, int n, int x)
{
    const int* p = array;
    for(int i = 0; i < n; i++)
    {
        if(*p == x)
        {
            return p;
        }
        ++p;
    }
    return 0;
}
```

解析：这是一个把普通函数改成泛型函数的问题，在这里公司考的是如何将普通函数转换成泛型函数。

答案：

修改代码如下：

```
template<typename T>
const T* My find(T *array, T n, T x)
{
    const T* p = array;
    int i;
    for(i=0;i<n;++i)
    {
        if(*p == x)
            return p;
        ++p;
    }
}
```

```

    }
    return 0;
};

或者:

```

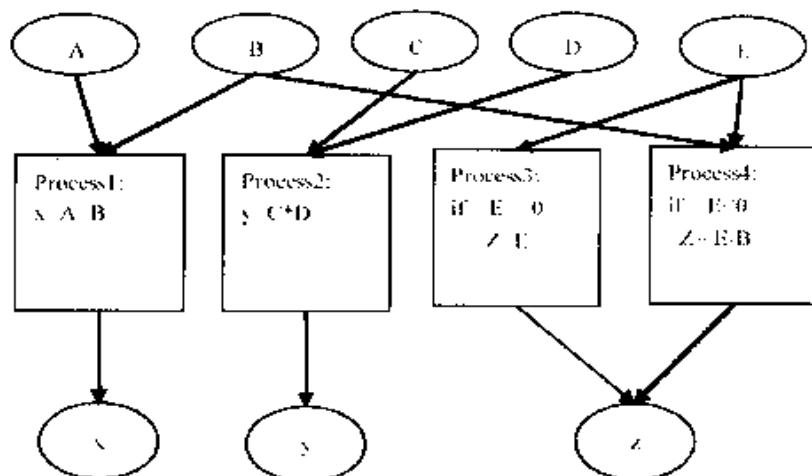
```

template<typename T>
const T* My find2(const T* s,const T* e,T x)
{
    const T* p=s;
    while(p!=e)
    {
        if(*p == x)
        {
            return p;
        }
        ++p;
    }
    return e;
};

```

9.3 模板

面试例题 1: Please write a program to realize the model described int the figure. You should design your program as generic as possible so that we can enhance the model in the future easily without making too much change in your program. (写一个程序来实现下图的模型中整型变量的问题。你必须尽可能地将程序设计为一类 (同类), 以便于我们今后在不做大量更改的情况下对它进行升级。) [德国某著名软件咨询企业 2005 年 11 月面试题]



解析: 这是一道考函数指针的面试例题, 当然也可以用模板来做。如题目所示: 要求求 a 与 b 之和、 c 与 d 之积, 以及由于 e 的情况不同而得出的结果。我们当然可以单独为 a 和 b 设置一个求和函数, 为 c

和 d 设置一个求积函数。但是一旦功能改变，比如说我们不得不求 a 与 b 之差或 c/d 的情况该怎么做？第三次也许是求最小值和两数之积……如果不使用函数指针，我们需要写多少个这样的 test() 函数？显然，函数指针为我们的编程提供了灵活性。

一个函数在编译时被分配给一个入口地址，这个入口地址就称为函数的指针，正如同指针是一个变量的地址一样。函数指针的用途很多，最常用的用途之一是把指针作为参数传递到其他函数。

答案：

程序源代码与解释如下：

```
#include<stdio.h>

//比较函数
int jug(int x,int y)
{
    if (x>=0)
    {
        return x;
    }
    else if (y==0)
    {
        return x;
    }
    else
        return x/y;
}

//求和函数
int sub(int x, int y)
{
    return(x+y);
}

//求差函数
int minus(int x,int y)
{
    return (x-y);
}

//函数指针
void test(int (*p)(int,int),int a,int b)
{
    int Int1;
```

```

    Int1=(*p)(a,b);

    printf("a=%d, a=%d %d\n",a,b,Int1);

}

int main()

{
    int a=1,b=2,c=3,d=4,e=-5;

    test(sub,a,b); //求和
    test(minus,c,d); //求差

    test(jug,e,b); //判断
    return 0;
}

```

另外，有些地方必须使用函数指针才能完成给定的任务，特别是异步操作的回调和其他需要匿名回调的结构。另外，像线程的执行和事件的处理，如果缺少了函数指针的支持也是很难完成的。当然，该程序也可以用静态模板类来实现，解决方法是一致的，代码如下：

```

#include <iostream>
using namespace std;

template<class T>
//建立一个静态模板类
class Operate{
public:
    static T Add(T a, T b) {
        return a+b;
    }
    static T Mul(T a, T b) {
        return a*b;
    }
    static T Judge(T a, T b=1) {
        if(a>=0) {
            return a;
        }
        else {
            return a/b;
        }
    }
};

int main() {
    int A, B, C, D, E, x, y, z;
    A=1, B=2, C=3, D=4, E=5;
    x=Operate<int>::Add(A,B);
    y=Operate<int>::Mul(C,D);
    z=Operate<int>::Judge(E,B);
    cout<<x<<'\n'<<y<<'\n'<<z<<endl;

    return 0;
}

```

扩展知识（模板与容器）

数据结构本身十分重要。当程序中存在着对时间要求很高的部分时，数据结构的选择就显得更加重要。

经典的数据结构数量有限，但是我们常常重复着一些为了实现向量、链表等结构而编写的代码。这些代码都十分相似，只是为了适应不同数据的变化而在细节上有所不同。STL 容器就为我们提供了这样的方便，它允许我们重复利用已有的实现构造自己的特定类型下的数据结构。通过设置一些模板类，STL 容器对最常用的数据结构提供了支持。这些模板的参数允许我们指定容器中元素的数据类型，可以将许多重复而乏味的工作简化。

容器部分主要由头文件<vector>、<list>、<deque>、<set>、<map>、<stack>和<queue>组成。对于常用的一些容器和容器适配器（可以看做由其他容器实现的容器），可以通过下表总结一下它们和相应头文件的对应关系。

数据结构	描述	实现头文件
向量 (vector)	连续存储的元素	<vector>
列表 (list)	由节点组成的双向链表	<list>
双队列 (deque)	连续存储的指向不同元素的指针所组成的数组	<deque>
集合 (set)	由节点组成的红黑树，每个节点都包含着一个元素，节点之间以某种作用于元素对的谓词排列，没有两个不同的元素能够拥有相同的次序	<set>
多重集合 (multiset)	允许存在两个次序相等的元素的集合	<set>
栈 (stack)	后进先出的值的排列	<stack>
队列 (queue)	先进先出的值的排列	<queue>
优先队列 (priority_queue)	元素的次序是由作用于所存储的值对上的某种谓词决定的一种队列	<queue>
映射 (map)	由{键，值}对组成的集合，以某种作用于键对上的谓词排列	<map>

第 10 章

面向对象

有这样一句话：“编程是在计算机中反映世界”，我觉得再贴切不过。面向对象（Object-Oriented）对这种说法的体现也是最优秀的。比如我们设计的数据结构是一个学生成绩的表现，而对数据结构的操作（函数）是分离的，虽然这些操作是针对这种数据结构而产生的。为了管理大量的数据，我们不得不小心翼翼地使用它们。

作为一名软件开发人员，我们可以深刻地体会到面向对象系统设计带来的种种便利。

- 良好的可复用性 开发同类项目的次数与开发新项目的时间成反比，减少重复劳动。
- 易维护 基本上不用花太大的精力跟维护人员讲解，他们可以自己读懂源程序并修改。否则开发的系统越多，负担就越重。
- 良好的可扩充性 以前，在向一个用结构化思想设计的庞大系统中加一个功能则必须考虑兼容前面的数据结构，理顺原来的设计思路。即使客户愿意花钱修改，作为开发人员多少都有点儿恐惧。在向一个用面向对象思想设计的系统中加入新功能，不外乎是加入一些新的类，基本上不用修改原来的东西。

在面试过程中，求职者是否对面向对象的基本概念、结构和类、多态性及构造函数有清晰的认识，是否能够有效地编程实现面向对象的各种功能，是 IT 企业考查的重点内容。

10.1 面向对象的基本概念

面试例题 1：面向对象技术的基本概念是什么？

解析：按照定义，OO（面向对象）编程语言必须提供对象、类和继承。

答案：对象、类和继承。

面试例题 2：C++中的空类默认产生哪些类成员函数？[中国某著名综合软件公司 2005 年面试题]

```
class Empty
{
public:
};
```

解析：类的概念问题。

答案：对于一个空类，编译器默认产生 4 个成员函数：默认构造函数、析构函数、拷贝构造函数和赋值函数。

面试例题 3：Which is incorrect about the class? (关于类，下面哪个叙述是错误的？) [中国某著名综合软件公司 2005 年面试题]

- A. A class is a blueprint to objects. (类是对象的设计蓝图。)
- B. We use the keyword class to create a class construct. (我们用关键字类来创建一个类的结构。)
- C. Once a class is declared, the class name becomes a type name and can be used to declare variables. (一旦一个类被声明，这个类名可以用做一个类型名来声明变量。)
- D. The class is same as the struct, and there are no different between class and struct. (类和结构一样，两者没有任何区别。)

解析：类的概念问题。

答案：D。

面试例题 4：Which is incorrect about the OOP? (下面关于面向对象技术的叙述，哪个是错误的？) [中国某著名综合软件公司 2005 年面试题]

- A. The central idea of OOP is to build programs using software objects.

(面向对象的关键特点是使用软件对象来编程。)

B. The OOP focuses mainly on the step-by-step procedure as procedure-oriented programming. (面向对象的特点和面向过程的特点一样，重点在于一步一步的过程。)

C. The OOP offers many advantages: simplicity, modularity, modifiability, extensibility, and so on. (面向对象程序提供了很多优势：简易化，模块化，易修改性，扩展性，等等。)

D. The key concept of object orientation is the attachment of procedure to data. (面向对象的关键特点是数据过程的附加。)

解析：OOP 的概念问题。面向对象和面向过程不能混为一谈。

答案：B 和 D。

10.2 类和结构

面试例题 1：structure 是否可以拥有 constructor / destructor 及成员函数？如果可以，那么 structure 和 class 还有区别么？[中国台湾某著名计算机硬件公司 2005 年 11 月面试题]

答案：区别是 class 中变量默认是 private，struct 中的变量默认是 public。struct 可以有构造函数、析构函数，之间也可以继承，等等。C++ 中的 struct 其实和 class 意义一样，唯一不同的就是 struct 里面默认的访问控制是 public，class 中默认的访问控制是 private。C++ 中存在 struct 关键字的唯一意义就是为了让 C 程序员有个归属感，是为了让 C++ 编译器兼容以前用 C 开发的项目。

扩展知识

我们可以写一段 struct 继承的例子，代码如下：

```
#include <iostream>
using namespace std;

enum BREED { GOLDEN, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };

struct Mammal
```

```

{
    public:
        Mammal(): itsAge(2), itsWeight(5){}
        ~Mammal(){}
        int GetAge() const {return itsAge;}
        void SetAge(int age) {itsAge = age;}
        int GetWeight() const {return itsWeight;}
        void SetWeight(int weight) {itsWeight = weight;}
        void Speak() const {cout << "\n Mammal sound !";}
        void Sleep() const {cout << "\n Shhh. I'm sleeping.";}
    protected:
        int itsAge;
        int itsWeight;
    };
    struct Dog : public Mammal
    {
        public:
            Dog():itsBreed(GOLDEN){}
            ~Dog(){}
            BREED GetBreed() const {return itsBreed;}
            void SetBreed(BREED breed) {itsBreed = breed;}
            void WagTail() const {cout << "Tail wagging ... \n";}
            void BegForFood() const{cout << "Begging for food... \n";}
        private:
            BREED itsBreed;
    };
    int main()
    {
        Dog fido;
        fido.Speak();
        fido.WagTail();
        cout << "Fido is " << fido.GetAge() << "years old \n";
        return 0;
    }
}

```

面试例题 2：现有以下代码，则编译时会产生错误的是_____。[中国某著名计算机金融软件公司 2005 年面试题]

```

struct Test
{
    Test(int){}
    Test(){}
    void fun(){}
};

int main()

```

```

{
    Test a(1); //语句1
    a.fun(); //语句2
    Test b(); //语句3
    b.fun(); //语句4
    return 0;
}

```

- A. 语句1 B. 语句2 C. 语句3 D. 语句4

解析：Test b()是不正确的，因为它不需要预先赋值。不像 Test a()需要预先赋值，所以 Test b()改为 Test b 即可。但在程序中这个错误在编译时是检测不出来的。出错的是语句 4 “b.fun();”，它是编译不过去的。

答案：D。

10.3 成员变量

面试例题 1：哪一种成员变量可以在同一个类的实例之间共享？[中国台湾某著名计算机硬件公司 2005 年 11 月面试题]

答案：必须使用静态成员变量在一个类的所有实例间共享数据。如果想限制对静态成员变量的访问，则必须把它们声明为保护型或私有型。不允许用静态成员变量去存放某一个对象的数据。静态成员数据是在这个类的所有对象间共享的。

面试例题 2：指出下面程序的错误。如果把静态成员数据设为私有，该如何访问？[中国台湾某著名计算机硬件公司 2005 年 11 月面试题]

```

#include <iostream>
using namespace std;

class Cat
{
public:
    Cat(int age):itsAge(age) {HowManyCats++; }
    virtual ~Cat() {HowManyCats--;}
    virtual int GetAge(){return itsAge;}
    virtual void SetAge(int age) {itsAge = age;}
    static int HowManyCats;
private:
    int itsAge;
};

int main()

```

```

    {
        const int MaxCats = 5;int i;
        Cat *CatHouse[MaxCats];
        for(i=0;i<MaxCats;i++)
            CatHouse[i]=new Cat(i);
        for(i=0;i<MaxCats;i++)
        {
            cout << "There are";
            cout << Cat::HowManyCats;
            cout <<"cats left!\n";
            cout <<"Deleting the one which is";
            cout << CatHouse[i]->GetAge();
            cout <<"years old\n";
            delete CatHouse[i];
            CatHouse[i]=0;
        }
        return 0;
    }

```

答案：该程序错在设定了静态成员变量，却没有给静态成员变量赋初值。如果把静态成员数据设为私有，可以通过公有静态成员函数访问。

```

#include <iostream>
using namespace std;

class Cat
{
public:
    Cat(int age):itsAge(age) {HowManyCats++;}
    virtual ~Cat() {HowManyCats--;}
    virtual int GetAge(){return itsAge;}
    virtual void SetAge(int age) {itsAge = age;}
    static int GetHowMany() {return HowManyCats;}
    //static int HowManyCats ;
private:
    int itsAge;
    static int HowManyCats ;
};

int Cat::HowManyCats = 0;

void tele();
int main()
{
    const int MaxCats = 5;int i;
    Cat *CatHouse[MaxCats];
    for(i=0;i<MaxCats;i++)
    {
        CatHouse[i]=new Cat(i);
        tele();
    }
    for(i=0;i<MaxCats;i++)
    {

```

```

        delete CatHouse[i];
        tele();
    }
    return 0;
}
void tele()
{
    cout << "There are" << Cat::GetHowMany() << "Cats alive!\n";
}

```

面试例题3：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```

#include <iostream>
using namespace std;

class Year {
    int y;
    const int InitY= 1970;

public:
    Year(): y=InitY; {
        int year() const { return y; }
        void add_year(int i) { y=year()+i; }
    };

    void main() {
        Year y1;
        Year* const pyl= new Year();

        y1.add_year(1);
        pyl->add_year(2);

        cout<<y1.year()<<', '<<pyl->year()<<endl;
        return;
    }
}

```

解析：这道程序体存在着成员变量问题。

答案：

正确的程序如下：

```

#include <iostream>
using namespace std;

class Year {
    int y;
    static const int InitY= 1970; // 错误1

public:
    Year(): y=InitY; // 错误2

```

```

        int year() const { return y; };
        void add_year(int i) { y=year()+i; };
    };

    int main() { // 在g++中不允许main()的返回值为void
        Year y1;
        Year* const py1= new Year();

        y1.add_year(1);
        py1->add_year(2);

        cout<<y1.year()<<', '<<py1->year()<<endl;

        return 0;
    }
}

```

面试例题 4：这个类声明正确吗？为什么？

```

class A
{
    const int Size = 0;
};

```

解析：这道程序体存在着成员变量问题。常量必须在构造函数的初始化列表里面初始化或者将其设置成 static。

答案：

正确的程序如下：

```

class A
{ A()
    const int Size=9;
};

```

或者：

```

class A
{ static const int Size=9;
};

```

面试例题 5：析构函数可以是内联函数么？[英国某著名计算机图形图像公司面试题]

解析：我们可以先构造一个类，让它的析构函数是内联函数，如下所示：

```

#include <iostream>
using namespace std;
class A
{
public :
    void foo()

```

```

        {cout<<"A";} ;
~A();

};

inline A::~A()
{ cout << "inline";}

int main()
{
    A *niu=new A();
    niu->foo();
    delete niu;
    return 0;
}

```

该程序可以正确编译并得出结果。

答案：析构函数可以是内联函数。

10.4 构造函数和析构函数

面试例题 1：MFC 类库中，CObject 类的重要性不言自明。在 CObject 的定义中，我们看到一个有趣的现象，即 CObject 的析构函数是虚拟的。为什么 MFC 的编写者认为 virtual destructors are necessary（虚拟的析构函数是必要的）？[美国某著名移动通信企业 2004 年面试题]

解析：

我们可以先构造一个类如下：

```

class CBase
{
public:
    ~CBase() { ... };
    ...
};

class CChild : public CBase
{
public:
    ~CChild() { ... };
    ...
};

main()
{
    Child c;
    ...
    return 0;
}

```

上段代码在运行时，由于在生成 CChild 对象 c 时，实际上在调用 CChild 类的构造函数之前必须首先调用其基类 CBase 的构造函数，所以当撤销 c 时，也会在调用 CChild 类析构函数之后，调用 CBase 类的析构函数（析构函数调用顺序与构造函数相反）。也就是说，无论析构函数是不是虚函数，派生类对象被撤销时，肯定会依次上调其基类的析构函数。

那么为什么 CObject 类要搞一个虚的析构函数呢？

因为多态的存在。

仍以上面的代码为例，如果 main() 中有如下代码：

```
CBase * pBase;
CChild c;
pBase = &c;
```

那么在 pBase 指针被撤销时，调用的是 CBase 的析构函数还是 CChild 的呢？显然是 CBase 的（静态联编）析构函数。但如果把 CBase 类的析构函数改成 virtual 型，当 pBase 指针被撤销时，就会先调用 CChild 类构造函数，再调用 CBase 类构造函数。

答案：在这个例子里，所有对象都存在于栈框中，当离开其所处的作用域时，该对象会被自动撤销，似乎看不出什么大问题。但是试想，如果 CChild 类的构造函数在堆中分配了内存，而其析构函数又不是 virtual 型的，那么撤销 pBase 时，将不会调用 CChild::~CChild(), 从而不会释放 CChild::CChild() 占据的内存，造成内存泄露。

将 CObject 的析构函数设为 virtual 型，则所有 CObject 类的派生类的析构函数都将自动变为 virtual 型，这保证了在任何情况下，不会出现由于析构函数未被调用而导致的内存泄露。这才是 MFC 将 CObject::~CObject() 设为 virtual 型的真正原因。

面试例题 2：析构函数可以为 virtual 型，构造函数则不能。那么为什么构造函数不能为虚呢？[美国某著名移动通信企业 2004 年面试题]

答案：虚函数采用一种虚调用的办法。虚调用是一种可以在只有部分信息的情况下工作的机制，特别允许我们调用一个只知道接口而不知道其准确对象类型的函数。但是如果要创建一个对象，你势必要知道对象的准确类型，因此构造函数不能为虚。

面试例题 3：如果虚函数是非常有效的，我们是否可以把每个函数都声明

为虚函数？

答案：不行，这是因为虚函数是有代价的：由于每个虚函数的对象都必须维护一个 v 表，因此在使用虚函数的时候都会产生一个系统开销。如果仅是一个很小的类，且不想派生其他类，那么根本没必要使用虚函数。

10.5 拷贝构造函数和赋值函数

面试例题 1：编写类 String 的构造函数、析构函数和赋值函数。
[中国某著名综合软件公司 2005 年面试题]

答案：

已知类 String 的原型为：

```
class String
{
public:
    String(const char *str = NULL);           // 普通构造函数
    String(const String &other);               // 拷贝构造函数
    ~String(void);                            // 析构函数
    String & operator =(const String &other); // 赋值函数
private:
    char *m_data;                            // 用于保存字符串
};
```

编写 String 的上述 4 个函数。

(1) String 的析构函数

为了防止内存泄漏，我们还需要定义一个析构函数。当一个 String 对象超出它的作用域时，这个析构函数将会释放它所占用的内存。代码如下：

```
String::~String(void)
{
    delete [] m_data;
    // 由于 m_data 是内部数据类型，也可以写成 delete m_data;
}
```

(2) String 的构造函数

这个构造函数可以帮助我们根据一个字符串常量创建一个 MyString 对象。这个构造函数首先分配了足量的内存，然后把这个字符串常量复制到这块内存，代码如下：

```
String::String(const char *str)
{
```

```

if(str==NULL)
{
    m_data = new char[1];           // 若能加 NULL 判断则更好
    *m_data = '\0';
}
else
{
    int length = strlen(str);
    m_data = new char[length+1];   // 若能加 NULL 判断则更好
    strcpy(m_data, str);
}
}

```

strlen 函数返回这个字符串常量的实际字符数（不包括 NULL 终止符），然后把这个字符串常量的所有字符赋值到我们在 String 对象创建过程中为 m_data 数据成员新分配的内存中。有了这个构造函数后，我们可以向下面这样根据一个字符串常量创建一个新的 String 对象：

```
string str("hello");
```

(3) String 的拷贝构造函数

所有需要分配系统资源的用户定义类型都需要一个拷贝构造函数，这样我们可以使用这样的声明：

```
MyString s1("hello");
MyString s2=s1;
```

拷贝构造函数还可以帮助我们在函数调用中以传值方式传递一个 MyString 参数，并且在当一个函数以值的形式返回 MyString 对象时实现“返回时复制”。

```

String::String(const String &other)      // 3 分
{
    int length = strlen(other.m_data);
    m_data = new char[length+1];           // 若能加 NULL 判断则更好
    strcpy(m_data, other.m_data);
}

```

(4) String 的赋值函数

赋值函数可以实现字符串的传值活动：

```
MyString s1(hello);
MyString s2;
s1=s2;
```

代码如下：

```

String & String::operator=(const String &other)
{
    // 检查自赋值
    if(this == &other)
        return *this;
}

```

```

// 释放原有的内存资源
delete [] m_data;
// 分配新的内存资源，并复制内容
int length = strlen(other.m_data);
m_data = new char[length+1];
strcpy(m_data, other.m_data);
// 返回本对象的引用
return *this;
}

```

扩展知识

这里还有一个问题：String & String::operator =(const String &other)的 const 是做什么用的？

Const 有两个作用。一个如果不加入 const 的话，比如：

```

MyString s3(pello);
Const MyString s4(qello);
s3=s4;

```

这样就会出现问题。因为一个 const 变量是不能随意转化成非 const 变量的。

其次是诸如：

```

MyString s7(pello);
MyString s8(pello);
MyString s9(qello);
S9=s7+s8;

```

不用 const 也会报错，因为用“+”赋值必须返回一个操作值已知的 MyString 对象，除非它是一个 const 对象。

面试例题 2： Which of the following is true about "Copy Constructor"? (下面关于拷贝构造函数的说法哪一个是正确的) [中国某著名综合软件公司 2005 年面试题]

- A. They copy constructor into each other. (给每一个对象拷贝一个构造函数。)
- B. A default is provided, but simply does a member-wise copy. (有一个默认的拷贝构造函数。)
- C. They can't copy arrays into each other. (不能拷贝队列。)
- D. All of the above. (以上结果都正确。)

解析：拷贝构造函数问题

答案：B。

面试例题 3：Which of the following class DOES NOT need a copy constructor? (下面所列举的类哪个不需要拷贝构造函数?) [中国台湾某著名杀毒软件公司 2004 年面试题]

- A. A matrix class in which the actual matrix is allocated dynamically within the constructor and is deleted within its destructor. (一个矩阵类：动态分配，对象的建立是利用构造函数，删除是利用析构函数。)
- B. A payroll class in which each object is provided with a unique ID. (一个花名册类：每一个对象对照着唯一的 ID。)
- C. A word class containing a string object and vector object of line and column location pairs. (一个 word 类，对象是字符串类和模板类。)
- D. A library class containing a list of book object. (一个图书馆类：由一系列书籍对象构成。)

解析：

按照题意，寻找一个不需要拷贝构造函数的类。

A 选项要定义拷贝构造函数。

B 选项中，不自定义拷贝构造函数的话，势必造成两个对象的 ID 不唯一。至于说自定义了拷贝构造函数之后，如何保证新对象的 ID 唯一，那是实现的问题。实现的方法有多种多样，比如可以使用当前的系统 tick 数作为新 ID。当然语义上有损失，不是完全意义上的拷贝，但在这儿只能在保持语义和实现目的之间来一个折中。

选 C 的原因是使用默认的拷贝构造，string 子对象和 vector 子对象的类都是成熟的类，都有合适的赋值操作，拷贝构造函数以避免“浅拷贝”问题。

D 选项显然是定义拷贝构造函数。

答案：C。

面试例题 4：Which virtual function re-declarations of the Derived class are correct? (哪个子类的虚函数重新声明是正确的?) [中国台湾某著名杀毒软件公司 2004 年面试题]

- A. Base* Base::copy(Base*);
Base* Derived::copy(Derived*);
- B. Base* Base::copy(Base*);
Derived* Derived::copy(Base*);
- C. ostream& Base::print(int,ostream&=cout);
ostream& Derived::print(int,ostream&);
- D. void Base::eval() const;
void Derived::eval();

解析：本题问的是哪个派生类的虚函数再声明是对的。

A 是重载；B 会导致编译错误；C 是真正的多态；D 是重载。

B 选项在 gcc 测试下也可以算是一种多态，覆盖的虚函数必须返回与父类的同名函数一致的类型。derived class 的虚函数的返回类型可以是 base class 中对应虚函数的返回类型的 public derived class。所谓多态指针的一致，是指“子类虚函数返回的多态指针的静态类型是父类虚函数所返回的多态指针的动态类型集合中的某个类型”。

答案：C。

10.6 多态的概念

面试例题 1：什么是多态？

答案：

开门，开窗户，开电视。在这里的“开”就是多态！

多态性可以简单地概括为“一个接口，多种方法”，在程序运行的过程中才决定调用的函数。多态性是面向对象编程领域的核心概念。

多态（Polymorphism），按字面的意思就是“多种形状”。多态性是允许你将父对象设置成为和它的一个或更多的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单地说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。多态性在 Object Pascal 和 C++ 中都是通过虚函数（Virtual Function）实现的。

扩展知识（多态的作用）

虚函数就是允许被其子类重新定义的成员函数。而子类重新定义父类虚函数的做法，称为“覆盖”（override），或者称为“重写”。这里有一个初学者经常混淆的概念。上面说了覆盖（override）和重载（overload）。覆盖是指子类重新定义父类的虚函数的做法。而重载，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。其实，重载的概念并不属于“面向对象编程”。重载的实现是：编译器根据函数不同的参数表，对同名函数的名称做修饰，然后这些同名函数就成了不同的函数（至少对于编译器来说是这样的）。如，有两个同名函数：function func(p:integer):integer; 和 function func(p:string):integer;。那么编译器做过修饰后的函数名称可能是这样的：int_func, str_func。对于这两个函数的调用，在编译器间就已经确定了，是静态的（记住：是静态）。也就是说，它们的地址在编译期就绑定了（早绑定），因此，重载和多态无关！真正与多态相关的是“覆盖”。当子类重新定义了父类的虚函数后，父类指针根据赋给它的不同的子类指针，动态（记住：是动态！）地调用属于子类的该函数，这样的函数调用在编译期间是无法确定的（调用的子类的虚函数的地址无法给出）。因此，这样的函数地址是在运行期绑定的（晚绑定）。结论就是：重载只是一种语言特性，与多态无关，与面向对象也无关！

引用一句 Bruce Eckel 的话：“不要犯傻，如果它不是晚绑定，它就不是多态。”

那么，多态的作用是什么呢？我们知道，封装可以隐藏实现细节，使得代码模块化；继承可以扩展已存在的代码模块（类）；它们的目的都是为了代码重用。而多态则是为了实现另一个目的——接口重用！而且现实往往是，要有效重用代码很难，而真正最具有价值的重用是接口重用，因为“接口是公司最有价值的资源。设计接口比用一堆类来实现这个接口更费时间。而且接口需要耗费更昂贵的人力和时间”。其实，继承为重用代码而存在的理

由已经越来越薄弱，因为“组合”可以很好地取代继承的扩展现有代码的功能，而且“组合”的表现更好（至少可以防止“类爆炸”）。因此笔者个人认为，继承的存在很大程度上是作为“多态”的基础而非扩展现有代码的方式。

那么什么是接口重用？我们举一个简单的例子。假设我们有一个描述飞机的基类如下（Object Pascal 语言描述）：

```
type
  plane = class
  public
    procedure fly(); virtual; abstract; //起飞纯虚函数
    procedure land(); virtual; abstract; //着陆纯虚函数
    function modal() : string; virtual; abstract;
    //查询型号纯虚函数
  end;
```

然后，我们从 plane 派生出两个子类，直升机（copter）和喷气式飞机（jet）：

```
copter = class(plane)
private
  fModal : String;
public
  constructor Create();
  destructor Destroy(); override;
  procedure fly(); override;
  procedure land(); override;
  function modal() : string; override;
end;

jet = class(plane)
private
  fModal : String;
public
  constructor Create();
  destructor Destroy(); override;
  procedure fly(); override;
  procedure land(); override;
  function modal() : string; override;
end;
```

现在，我们要完成一个飞机控制系统。有一个全局的函数 plane_fly，它负责让传递给它的飞机起飞，那么，只需要这样：

```
procedure plane_fly(const pplane : plane);
begin
  pplane.fly();
end;
```

就可以让所有传给它的飞机（plane 的子类对象）正常起飞！不管是直升机还是喷气机，甚至是现在还不存在的、以后会增加

的飞碟。因为，每个子类都已经定义了自己的起飞方式。

可以看到 `plane_fly` 函数接受的参数是 `plane` 类对象引用，而实际传递给它的都是 `plane` 的子类对象。现在回想一下开头所描述的“多态”：多态性是允许你将父对象设置成为和一个或更多的它的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。很显然，`parent = child;` 就是多态的实质！因为直升机“是一种”飞机，喷气机也“是一种”飞机，因此，所有对飞机的操作都可以对它们操作。此时，飞机类就是一种接口。多态的本质就是将子类型指针赋值给父类类型的指针（在 OP 中是引用），只要这样的赋值发生了，多态也就产生了，因为实行了“向上映射”。

应用多态的例子非常普遍。在 Delphi 的 VCL 类库中，最典型的就是：`TObject` 类有一个虚拟的 `Destroy` 虚构函数和一个非虚拟的 `Free` 函数。`Free` 函数中是调用 `Destroy` 的。因此，当我们对任何对象（都是 `TObject` 的子类对象）调用 `.Free();` 之后，都会执行 `TObject.Free();`，它会调用我们所使用的对象的析构函数 `Destroy();`。这就保证了任何类型的对象都可以正确地被析构。

多态性是面向对象最重要的特性！

面试例题 2：重载和覆盖有什么不同？

答案：虚函数总是在派生类中被改写，这种改写被称为“override”（覆盖）。

`override` 是指派生类重写基类的虚函数，就像我们前面在 B 类中重写了 A 类中的 `foo()` 函数。重写的函数必须有一致的参数表和返回值（C++ 标准允许返回值不同的情况，但是很少有编译器支持这个特性）。`Override` 这个单词好像一直没有合适的中文词汇来对应。有人译为“覆盖”，还贴切一些。

`overload` 约定俗成地被翻译为“重载”，是指编写一个与已有函数同名但是参数表不同的函数。例如一个函数既可以接受整型数作为参数，也可以接受浮点数作为参数。重载不是一种面向对象的编程，而只是一种语法规则，重载与多态没有什么直接关系。

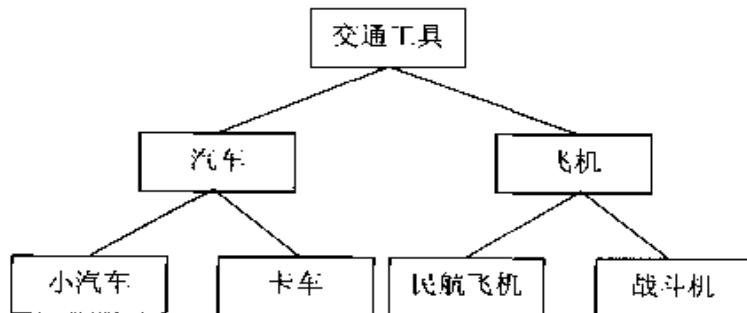
第 11 章

继承与接口

整个 C++ 程序设计全面围绕面向对象的方式进行。类的继承特性是 C++ 的一个非常重要的机制。继承特性可以使一个新类获得其父类的操作和数据结构，程序员只需在新类中增加原有类中没有的成分。

可以说这一章的内容是 C++ 面向对象程序设计的关键。

下面我们简单地说一下继承的概念，先看下图。



上图是一个抽象描述的特性继承表。

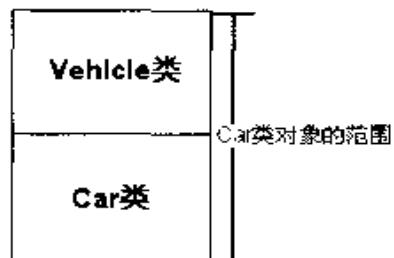
交通工具是一个基类（也称做父类）。在通常情况下所有交通工具所共同具备的特性是速度与额定载人的数量。但按照生活常规，我们继续对交通工具进行细分的时候，我们会分别想到汽车类和飞机类等。汽车类和飞机类同样具备速度和额定载人数量这样的特性，而这些特性是所有交通工具所共有的。那么当建立汽车类和飞机类的时候，我们无须再定义基类已经有的数据成员，而只需要描述汽车类和飞机类所特有的特性即可。飞机类和汽车类的特性是由其在交通工具类原有特性的基础上增加而来的，那么飞机类和汽车类就是交通工具类的派生类（也称做子类）。依次类推，层层递增，这种子类获得父类特性的概念就是继承。

一旦成功定义派生类，那么派生类就可以操作基类的所有数据成员，包括受保护型的。甚至我们可以在构造派生类对象的时候初始化它们，但我们不推荐这么做。因为类与类之间的操作是通过接口进行沟通的，为了不破坏类的这种封装特性，即使是父类与子类的操作也应遵循这个思想。这么做的好处也是显而易见的。当基类有错的时候，只要不涉及接口，那么基类的修改就不会影响到派生类的操作。

至于为什么派生类能够对基类成员进行操作，右图可以简单地说明基类与子类在内存中的排列状态。

我们知道，类对象操作的时候在内部构造时会有一个隐性的 this 指针。由于 Car 类是 Vehicle 的派生类，那么当 Car 对象创建的时候，这个 this 指针就会覆盖到 Vehicle 类的范围，所以派生类能够对基类成员进行操作。

在面试过程中，各大企业会考量你对虚函数、纯虚函数、私有继承、多重继承等知识点的掌握程度。因此，这是本书比较难掌握的一章。



11.1 覆盖

面试例题 1：以下代码的输出结果是什么？[中国某著名计算机金融软件公司 2005 年面试题]

```
# include <iostream>
using namespace std;
class Parent
{
public:
    virtual void foo()
    {
        cout << "foo from Parent";
    }
    void fool()
    {
        cout << "fool from Parent";
    }
};
class Son: public Parent
{
public:
    void foo()
    {
        cout << "foo from Son";
    }
};
```

```

void fool()
{
    cout << "fool from Son";
}
};

int main()
{
    Parent *p = new Son();
    p->foo();
    p->fool();
    return 0;
}

```

- A. foo from Parent fool from Son
- B. foo from Son fool from Parent
- C. foo from Son fool from Son
- D. Cannot compile

解析：这是一个典型的覆盖的问题。Parent 类里的 foo 函数是一个虚函数，虚函数是被子类同名函数所覆盖的。而 Parent 类里的 fool 函数是一个普通函数，不会被子类同名函数所覆盖。P 是一个指针，指向的是对象。由于 Parent 类里的 foo 函数被 Son 类里的 foo 函数所覆盖，所以结果是 foo from Son。

答案：B。

面试例题 2：以下代码的输出结果是什么？[德国某著名电子/通信/IT 企业 2005 年面试题]

```

#include <iostream>
using namespace std;

class A {
public:
    void virtual f() {
        cout<<"A"<<endl;
    }
};

class B :public A{
public:
    void virtual f() {
        cout<<"B"<<endl;
    }
};

int main() {
    A* pa=new A();
    pa->f();
    B* pb=(B*)pa;
    pb->f();

    delete pa,pb;
}

```

```

pa=new B();
pa->f();
pb=(B*)pa;
pb->f();
};


```

- A. AABA B. AABB C. AAAB D. ABBA

解析：这是一个虚函数覆盖虚函数的问题。A 类里的 f 函数是一个虚函数，虚函数是被子类同名函数所覆盖的。而 B 类里的 f 函数也是一个虚函数，它覆盖 A 类 f 函数的同时，也会被它的子类覆盖。但是在 B*pb=(B*)pa;里面，该语句的意思是转化 pa 为 B 类型并新建一个指针 pb，将 pa 复制到 pb。但这里有一点请注意，就是 pa 的指针始终没有发生变化，所以 pb 也指向 pa 的 f 函数。这里并不存在覆盖的问题。

delete pa,pb;删除了 pa 和 pb 所指向的地址，但 pa、pb 指针并没有删除，也就是我们通常说的悬浮指针。现在重新给 pa 指向新地址，所指向的位置是 B 类的，而 pa 指针类型是 A 类的，所以就产生了一个覆盖。pa->f();的值是 B。

pb=(B*)pa;转化 pa 为 B 类指针给 pb 赋值，但 pa 所指向的 f 函数是 B 类的 f 函数，所以 pb 所指向的 f 函数是 B 类的 f 函数。pb->f();的值是 B。

答案：B。

11.2 私有继承

面试例题 1：Tell me the difference in public inherit and private inherit. (公有继承和私有继承的区别是什么？) [中国某著名计算机金融软件公司 2005 年面试题]

- A. No difference. (没有区别。)
- B. Private inherit will make every member from parent class into private. (私有继承使父类中所有元素变成私有。)
- C. Private inherit will make functions from parent class into private. (私有继承使父类中的函数转化成私有。)
- D. Private inherit make every member from parent not-accessible to sub-class. (私有继承使父类中所有元素无法与子类联系。)

解析：

A 肯定错。

因为子类只能继承父类的 protected 和 public，所以 B 也是错误的。

C 的叙述不全面，而且父类可能有自己的私有方法成员，所以也是错误的。

答案：D。

扩展知识

一个私有的或保护的派生类不是子类，因为非公共的派生类不能做基类能做的所有的事。例如，下面的代码定义了一个私有继承基类的类：

```
#include
class Animal
{
public:
    Animal(){}
    void eat(){cout<<"eat\n";}
};

class Giraffe: private Animal
{
public:
    Giraffe(){}
    void StrecthNeck(double){cout<<"strecth neck \n";}
};

class Cat: public Animal
{
public:
    Cat(){}
    void Meaw(){cout<<"meaw\n";}
};

void Func(Animal& an)
{
    an.eat();
}

void main()
{
    Cat dao;
    Giraffe gir;
    Func(dao);
    Func(gir); //error
}
```

函数 Func()要用一个 Animal 类型的对象，但调用 Func(dao) 实际上传递的是 Cat 类的对象。因为 Cat 是公共继承 Animal 类，

所以 Cat 类中的对象可以使用 Animal 类的所有的公有成员变量或函数。Animal 对象可以做的事，Cat 对象也可以做。

但是，对于 gir 对象就不一样。Giraffe 类私有继承了 Animal 类，意味着对象 gir 不能直接访问 Animal 类的成员。其实，在 gir 对象空间中，包含 Animal 类的对象，只是无法让其公开访问。

公有继承就像是三口之家的小孩，饱受父母的温暖，享有父母的一切（public 和 protected 的成员）。其中保护的成员不能被外界所享有，但可以为小孩所拥有。只是父母还有其一点点隐私（private 成员）不能为小孩所知道。

私有继承就像是离家出走的小孩，一个人在外面漂泊。他（她）不能拥有父母的住房和财产（如 gir.eat() 是非法的），在外面自然也就不能代表其父母，甚至他（她）不算是其父母的小孩。但是在他（她）的身体中，流淌着父母的血液，所以，在小孩自己的行为中又有其与父母相似的成分。

例如下面的代码中，Giraffe 继承了 Animal 类，Giraffe 的成员函数可以像 Animal 对象那样访问其 Animal 成员：

```
#include

class Animal
{
public:
    Animal(){}
    void eat(){ cout << "eat.\n"; }
};

class Giraffe :private Animal
{
public:
    Giraffe(){}
    void StretchNeck(){ cout << "stretch neck.\n"; }
    void take(){ eat(); } //ok
};

void Func(Giraffe & an)
{
    an.take();
}

void main()
{
    Giraffe gir;
    gir.StretchNeck();
    Func(gir); //ok
}
```

}

运行结果为：

```
stretch neck.  
eat.
```

上例中，gir 对象就好比是小孩。eat()成员函数是其父母的行为，take()成员函数是小孩的行为，在该行为中，渗透着其父母的行为。但是小孩无法直接使用 eat()成员函数，因为，离家出走的他（她）无法拥有其父母的权力。保护继承与私有继承类似，继承之后的类相对于基类来说是独立的。保护继承的类对象，在公开场合同样不能使用基类的成员。代码如下：

```
#include  
class Animal  
{  
public:  
    Animal();  
    void eat() {cout<<"eat\n";}  
};  
class Giraffe: protected Animal  
{  
    Giraffe();  
    void StretchNeck(double) {cout<<"stretchneck\n";}  
    void take()  
    {  
        eat(); // ok  
    }  
};  
void main()  
{  
    Giraffe gir;  
    gir.eat(); // error  
    gir.take(); // ok  
    gir.StretchNeck();  
}
```

派生类的3种继承方式小结如下。

公有继承(public)、私有继承(private)和保护继承(protected)是常用的3种继承方式。

1. 公有继承方式

基类成员对其对象的可见性与一般类及其对象的可见性相同，公有成员可见，其他成员不可见。这里保护成员与私有成员相同。

基类成员对派生类的可见性对派生类来说，基类的公有成员和保护成员可见：基类的公有成员和保护成员作为派生类的成员

时，它们都保持原有的状态；基类的私有成员不可见：基类的私有成员仍然是私有的，派生类不可访问基类中的私有成员。

基类成员对派生类对象的可见性对派生类对象来说，基类的公有成员是可见的，其他成员是不可见的。

所以，在公有继承时，派生类的对象可以访问基类中的公有成员，派生类的成员函数可以访问基类中的公有成员和保护成员。

2. 私有继承方式

基类成员对其对象的可见性与一般类及其对象的可见性相同，公有成员可见，其他成员不可见。

基类成员对派生类的可见性对派生类来说，基类的公有成员和保护成员是可见的：基类的公有成员和保护成员都作为派生类的私有成员，并且不能被这个派生类的子类所访问；基类的私有成员是不可见的：派生类不可访问基类中的私有成员。

基类成员对派生类对象的可见性对派生类对象来说，基类的所有成员都是不可见的。

所以，在私有继承时，基类的成员只能由直接派生类访问，而无法再往下继承。

3. 保护继承方式

这种继承方式与私有继承方式的情况相同。两者区别仅在于对派生类的成员而言，基类成员对其对象的可见性与一般类及其对象的可见性相同，公有成员可见，其他成员不可见。

基类成员对派生类的可见性对派生类来说，基类的公有成员和保护成员是可见的：基类的公有成员和保护成员都作为派生类的保护成员，并且不能被这个派生类的子类所访问；基类的私有成员是不可见的：派生类不可访问基类中的私有成员。

基类成员对派生类对象的可见性对派生类对象来说，基类的所有成员都是不可见的。

所以，在保护继承时，基类的成员也只能由直接派生类访问，而无法再往下继承。

C++支持多重继承，从而大大增强了面向对象程序设计的能力。多重继承是一个类从多个基类派生而来的功能。派生类实际

上获取了所有基类的特性。当一个类是两个或多个基类的派生类时，必须在派生类名和冒号之后，列出所有基类的类名，基类间用逗号隔开。派生类的构造函数必须激活所有基类的构造函数，并把相应的参数传递给它们。派生类可以是另一个类的基类，这样，相当于形成了一个继承链。当派生类的构造函数被激活时，它的所有基类的构造函数也都会被激活。在面向对象的程序设计中，继承和多重继承一般指公共继承。在无继承的类中，protected 和 private 控制符是没有差别的。在继承中，基类的 private 对所有的外界都屏蔽（包括自己的派生类），基类的 protected 控制符对应用程序是屏蔽的，但对其派生类是可访问的。

保护继承和私有继承只是在技术上讨论时有其一席之地。

面试例题 2：请考虑标记为 A 到 J 的语句在编译时可能出现的情况。如果能够成功编译，请记为“RIGHT”，否则记为“ERROR”。[中国台湾某著名计算机硬件公司 2005 年 12 月面试题]

```
#include <iostream>
#include <stdio.h>
class Parent
{
public:
    Parent(int var = -1)
    {
        m_nPub = var;
        m_nPtd = var;
        m_nPrt = var;
    }
public:
    int m_nPub;
protected:
    int m_nPtd;
private:
    int m_nPrt;
};

class Child1:public Parent
{
public:
    int GetPub(){return m_nPub;};
    int GetPtd(){return m_nPtd;};
    int GetPrt(){return m_nPrt;};           //A
};

class Child2:protected Parent
{
public:
    int GetPub(){return m_nPub;};
    int GetPtd(){return m_nPtd;};
```

```
        int GetPrt(){return m_nPrt;};           //B
};

class Child3:private Parent
{
public:
    int GetPub(){return m_nPub;};
    int GetPtd(){return m_nPtd;};
    int GetPrt(){return m_nPrt;};           //C
};

int main()
{
    Child1 cd1;
    Child2 cd2;
    Child3 cd3;

    int nVar = 0;

//public inherited
    cd1.m_nPub = nVar;                    //D
    cd1.m_nPtd = nVar;                  //E
    nVar = cd1.GetPtd();                //F
//protected inherited
    cd2.m_nPub = nVar;                  //G
    nVar = cd2.GetPtd();                //H
//private inherited
    cd3.m_nPub = nVar;                  //I
    nVar = cd3.GetPtd();                //J

    return 0;
}
```

解析：

A、B、C 都是错误的。因为 `m_nPrt` 是父类 `Parent` 的私有变量，所以不能被子类访问。

D 正确。`cd1` 是公有继承，可以访问并改变父类的公有变量。

E 错误。`m_nPtd` 是父类 `Parent` 的保护变量，可以被公有继承的 `cd1` 访问，但不可以修改。

F 正确。可以通过函数访问父类的保护变量。

G 错误。`cd2` 是保护继承的，不可以直接修改父类的公有变量。

H 正确。可以通过函数访问父类的保护变量。

I 错误。`cd3` 是私有继承的，不可以直接修改父类的公有变量。

J 正确。可以通过函数访问父类的保护变量。

答案：

A、B、C、E、G、I 是 “ERROR”。

D、F、H、J 是 “RIGHT”。

11.3 虚函数继承和虚继承

面试例题 1：下面程序的结果是什么？

```
#include <iostream>
#include <memory.h>
#include<assert.h>

using namespace std;
class A
{
    char k[3];
public:
    virtual void aa() {};
};

class B : public virtual A
{
    char j[3];
//加入一个变量是为了看清楚 class 中的 vfptr 放在什么位置
public:
    virtual void bb() {};
};

class C : public virtual B
{
    char i[3];
public:
    virtual void cc() {};
};

int main(int argc,char *argv[])
{
    cout << "sizeof(A): " << sizeof(A) << endl;
    cout << "sizeof(B): " << sizeof(B) << endl;
    cout << "sizeof(C): " << sizeof(C) << endl;

    return 0;
}
```

解析：C++ 2.0 以后全面支持虚函数继承。这个特性的引入为 C++ 增强了不少功能，也引入了不少烦恼。如果能够了解编译器是如何实现虚函数继承，它们在类的内存空间中又是如何布局的，就可以对 C++ 的了解深入不少。

(1) 对于 class A，由于有一个虚函数，那么必须得有一个对应的虚函数表来记录对应的函数入口地址。每个地址需标有一个虚指针，指针的大小为 4。类中还有一个 char k[3]，每一个 char 值所占位置是 1，所以

`char k[3]`所占大小是 3。做一次数据对齐后（编译器里一般以 4 的倍数为对齐单位），`char k[3]`所占大小变为 4。`sizeof (A)`的结果就是 `char k[3]` 所占大小 4 和虚指针所占大小 4，三者之和等于 8。

(2) 对于 class B，由于 class B 虚继承了 class A，同时还拥有自己的虚函数，那么 class B 中首先拥有一个 `vptr_B`，指向自己的虚函数表。还有 `char j[3]`，大小为 4。可虚继承该如何实现？首先要通过加入一个虚类指针（记 `vptr_B_A`）来指向其父类，然后还要包含父类的所有内容。有些复杂了，不过还不难想象。`sizeof (B)` 的结果就是 `char k[3]` 所占大小 4 和虚指针 `vptr_B` 所占大小 4 加 `sizeof (A)` 所占大小 8，三者之和等于 16。

(3) 下面是 class C 了。class C 首先也得有个 `vptr_C`，然后是 `char i[3]`，然后是 `sizeof (B)`，所以 `sizeof (C)` 的结果就是 `char i[3]` 所占大小 4 和虚指针 `vptr_C` 所占大小 4 加 `sizeof (B)` 所占大小 16，三者之和等于 24。

答案：在 gcc 中打印上面几个类的大小，结果为 8、16、24。

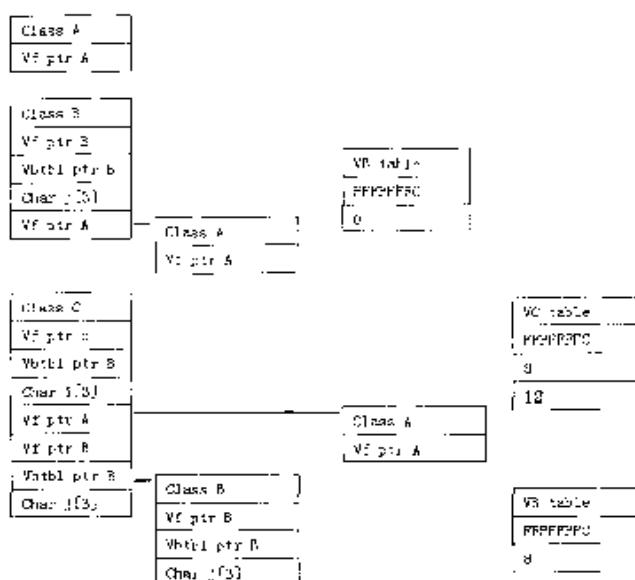
扩展知识

关键字 `virtual` 告诉编译器它不应当完成早绑定，相反，它应当自动安装实现晚绑定所有机制。这意味着，如果我们对 `brass` 对象通过基类 `instrument` 地址调用 `play()`，我们将得到恰当的函数。

为了完成这件事，编译器对每个包含虚函数的类创建一个表（称为 `vtable`）。在 `vtable` 中，编译器放置特定类的虚函数地址。在每个带有虚函数的类中，编译器秘密地置一指针，称为 `vpointer`（缩写为 `vptr`），指向这个对象的 `vtable`。通过基类指针做虚函数调用时（也就是做多态调用时），编译器静态地插入取得这个 `vptr`，并在 `vtable` 表中查找函数地址的代码，这样就能调用正确的函数使晚绑定发生。

为每个类设置 `vtable`、初始化 `vptr`、为虚函数调用插入代码，所有这些都是自动发生的，所以我们不必担心这些。利用虚函数，这个对象的合适的函数就能被调用，哪怕编译器还不知道这个对象的特定类型。

画了个图，简单表示一下虚表跟踪后的结果，如下图所示。



我们已经知道了有一个 vptr，不过 vptr 的位置也许在对象的开始，也许在对象的尾部。所以上面的操作的值应该是 8 或者 12（如果 vptr 在前面的话）。但实际上取回的值被加上了 1。原因是必须要区别一个不指向任何成员的指针和一个指向第一个成员的指针。这里又有点儿不好理解了，举个例子。

想象你和你的另外两个朋友合住一个有 3 个房间的别墅，你住在第三间。如果一个你们 3 人共同的朋友来找你玩，你就给他别墅的地址就行了，不用给出你们任意一个人的房间号（不指向任何成员）。但如果你的一个私人朋友来拜访你（这个私人朋友不认识你的那两位朋友），你会给出别墅的地址和的那个房间号。为了使这个地址有区别，你必须用第一个房间作为偏移量（offset）来表示你的房间位置。你可以对你的朋友说从进门后的第一间房子再往里面走两个房子就是第三间我的房子。如果房子间距是 4，那么第一间到第三间的距离是 8。

如果以上函数稍加变动，不采用虚继承的方式而是直接继承，结果就将不会产生偏移，结果为：8, 12, 16。

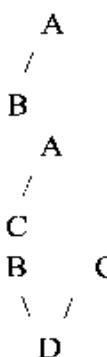
面试例题 2：什么是虚继承？它与一般的继承有什么不同？它有什么用？写出一段虚继承的 C++ 代码。[美国某著名计算机软件公司 2005 年面试题]

答案：

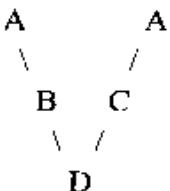
虚拟继承是多重继承中特有的概念。

虚拟基类是为解决多重继承而出现的。

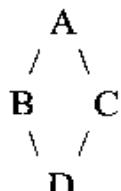
请看下图。



类 D 继承自类 B 和类 C，而类 B 和类 C 都继承自类 A，因此出现如下图所示这种情况：



在类 D 中会两次出现 A。为了节省内存空间，可以将 B、C 对 A 的继承定义为虚拟继承，而 A 就成了虚拟基类。最后形成如下图所示的情况：



代码如下：

```

class A;
class B : public virtual A; // virtual inheritance.
class C : public virtual A;
class D : public B, public C;

```

注意：虚函数继承和虚继承是完全不同的两个概念，请不要在面试中混淆。

面试例题 3：如果一个圆角矩形有直边和圆角，那么圆角矩形也就多重继承了圆形和矩形，而圆形和矩形又都是从 shape 类里继承的。请问：当你创建一个圆角矩形类时，共创建了多少个 shape？

答案：如果圆形类和矩形类都不是用关键字 virtual 继承 shape 类，那么生成两个 shape：一个为圆形类，一个为矩形类。如果圆形类和矩形

类都是用关键字 virtual 继承 shape 类，那么生成一个共享的 shape。

11.4 多重继承

面试例题 1：请评价多重继承的优点和缺陷。

答案：

多重继承在语言上并没有什么很严重的问题，但是标准本身只对语义做了规定，而对编译器的细节没有做规定。所以在使用时（即使是继承），最好不要对内存布局等有什么假设。此类的问题还有虚析构函数等。为了避免由此带来的复杂性，通常推荐使用复合。但是，在《C++设计新思维》(Andrei Alexandrescu)一书中对多重继承和模板有极为精彩的运用。

(1) 多重继承本身并没有问题，如果运用得当可以收到事半功倍的效果。不过大多数系统的类层次往往有一个公共的基类，就像 MFC 中的 Cobject，Java 中的 Object。而这样的结构如果使用多重继承，稍有不慎，将会出现一个严重现象——菱形继承，这样的继承方式会使得类的访问结构非常复杂。但并非不可处理，可以用 virtual 继承（并非唯一的方法）及 Loki 库中的多继承框架来掩盖这些复杂性。

(2) 从哲学上来说，C++多重继承必须要存在，这个世界本来就不是单根的。从实际用途上来说，多重继承不是必需的，但这个世界上有多少东西是必需的呢？对象不过是一组有意义的数据集合及其上的一组有意义的操作，虚函数（晚期绑定）也不过是一堆函数入口表，重载也不过是函数名扩展，这些东西都不是必需的，而且对它们的不当使用都会带来问题。但是没有这些东西行吗？很显然，不行。

(3) 多重继承在面向对象理论中并非是必要的——因为它不提供新的语意，可以通过单继承与复合结构来取代。而 Java 则放弃了多重继承，使用简单的 interface 取代。多重继承是把双刃剑，应该正确地对待。况且，它不像 goto，不破坏面向对象语义。跟其他任何威力强大的东西一样，用好了会带来代码的极大精简，用坏了那就不用说了。

C++是为实用而设计的，在语言里有很多东西存在着各种各样的“缺陷”。所以，对于这种有“缺陷”的东西，它的优劣就要看使用它的人。

C++不回避问题，它只是把问题留给使用者，从而给大家更多的自由。像 Ada、Pascal 这类定义严格的语言，从语法上回避了问题，但并不是真正解决了问题，而使人做很多事时束手束脚（当然，习惯了就好了）。

(4) 多重继承本身并不复杂，对象布局也不混乱，语言中都有明确的定义。真正复杂的是使用了运行时多态 (virtual) 的多重继承（因为语言对于多态的实现没有明确的定义）。为什么非要说多重继承不好呢？如果这样的话，指针不是更容易出错，运行时多态不是更不好理解吗？

因为 C++ 中没有 interface 这个关键字，所以不存在所谓的“接口”技术。但是 C++ 可以很轻松地做到这样的模拟，因为 C++ 中的不定义属性的抽象类就是接口。

(5) 要了解 C++，就要明白有很多概念是 C++ 试图考虑但是最终放弃的设计。你会发现很多 Java、C# 中的东西都是 C++ 考虑后放弃的。不是说这些东西不好，而是在 C++ 中它将破坏 C++ 作为一个整体的和谐性，或者 C++ 并不需要这样的东西。用一个例子来说，C# 中有一个关键字 base 用来表示该类的父类，C++ 却没有对应的关键字。为什么没有？其实 C++ 中曾经有人提议一个类似的关键字 inherited，用来表示被继承的类，即父类。这样一个好的建议为什么没有被采纳呢？这本书中就说得很明确，因为这样的关键字既不必须又不充分，不必须是因为 C++ 右一个 `typedef *inherited` 不充

解析：如果所有鸟都能飞，那鸵鸟就不是鸟！回答这种问题时，不要相信自己的直觉！将直觉和合适的继承联系起来还需要一段时间。

根据题干可以得知：鸟是可以飞的。也就是说，当鸟飞行时，它的高度是大于0的。鸵鸟是鸟类（生物学上）的一种。但它的飞行高度为0（鸵鸟不能飞）。

不要把可替代性和子集相混淆。即使鸵鸟集是鸟集的一个子集（每个鸵鸟集都在鸟集内），但并不意味着鸵鸟的行为能够代替鸟的行为。可替代性与行为有关，与子集没有关系。当评价一个潜在的继承关系时，重要的因素是可替代的行为，而不是子集。

答案：如果一定要让鸵鸟来继承鸟类，可以采取组合的办法，把鸟类中的可以被鸵鸟继承的函数挑选出来，这样鸵鸟就不是“a kind of”鸟了，而是“has some kind of”鸟的属性而已。代码如下：

```
#include<iostream>
#include<string>
using namespace std;

class bird
{
public:
    void eat();
    void sleep();
    void fly();
};

class ostrich
{
public:
    bird eat() {cout<<"ostrich eat";};
    bird sleep() {cout<<"ostrich sleep";};
};

int main()
{
    ostrich xiaoq;
    xiaoq.eat();
    xiaoq.sleep();
    return 0;
}
```

面试例题 2：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```
#include <iostream>
using namespace std;

class Base {
```

```

public:
    int val;
    Base() { val=1; };
};

class Derive: Base {
public:
    int val;
    Derive(int i) { val=Base::val+i; };
};

int main(int, char**, char**) {
    Derive d(10);
    cout<<d.Base::val<<endl<<d.val<<endl;
    return 0;
}

```

答案：把 class Derive: Base 改成 class Derive:public Base。

解析：这是个类继承问题。如果不指定 public，C++默认的是私有继承。私有继承是无法继承并使用父类函数中的公有变量的。

扩展知识（组合）

若在逻辑上 A 是 B 的“一部分”(a part of)，则不允许 B 从 A 派生，而是要用 A 和其他东西组合出 B。

例如眼(Eye)、鼻(Nose)、口(Mouth)、耳(Ear)是头(Head)的一部分，所以类 Head 应该由类 Eye、Nose、Mouth、Ear 组合而成，而不是派生而成。程序如下：

```

class Eye
{
public:
    void Look(void);
};

class Nose
{
public:
    void Smell(void);
};

class Mouth
{
public:
    void Eat(void);
};

class Ear
{
public:
    void Listen(void);
};

class Head
{

```

```

public:
    void Look(void) { m_eye.Look(); }
    void Smell(void) { m_nose.Smell(); }
    void Eat(void) { m_mouth.Eat(); }
    void Listen(void) { m_ear.Listen(); }

private:
    Eye m_eye;
    Nose m_nose;
    Mouth m_mouth;
    Ear m_ear;
);

```

Head 由 Eye、Nose、Mouth、Ear 组合而成。如果允许 Head 从 Eye、Nose、Mouth、Ear 派生而成，那么 Head 将自动具有 Look、Smell、Eat、Listen 这些功能。程序十分简短并且运行正确，但是下面这种设计方法却是不对的。

```

class Head : public Eye, public Nose, public Mouth, public Ear
{
};

```

面试例题 3：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [德国某著名软件咨询企业 2005 年面试题]

```

class base{
private: int i;
public: base(int x){i=x;}
};

class derived: public base{
private: int i;
public: derived(int x, int y) {i=x;}
        void printTotal() {int total = i+base::i;}
};

```

解析：要在子类中设定初始成员变量，把 derived(int x, int y) 改成 derived(int x, int y) : base(x)。

答案：

代码如下：

```

class base
{
protected: //这里的访问属性需要改变
    int i;
public:
    base(int x){i=x;}
};

class derived: public base

```

```

{
private:
    int i;
public:
    derived(int x, int y) : base(x) //以前没有初始化基类的成员变量
    {
        i=y;
    }
    void printTotal()
    {
        int total = i+base::i;
    }
}

```

11.6 纯虚函数

面试例题 1：下面的程序有何错误？[德国某著名软件咨询企业 2004 年面试题]

```

#include <iostream>
using namespace std;
class Shape
{
public:
    Shape(){}
    ~Shape(){}
    virtual void Draw()=0;
};

int main()
{
    Shape s1;
}

```

解析：因为 Shape 类中的 Draw 函数是一个纯虚函数，所以 Shape 类是不能实例化一个对象的。Shape s1; 是不可以的，解决方法是把 Draw 函数修改成一般的虚函数。

答案：

修改后的代码如下：

```

#include <iostream>
using namespace std;
class Shape
{
public:
    Shape(){}
    ~Shape(){}
    virtual void Draw();
};

```

```
int main()
{
    Shape s1;
}
```

面试例题 2：什么是虚指针？[美国某著名移动通信企业面试题]

答案：虚指针或虚函数指针是一个虚函数的实现细节。带有虚函数的类中的每一个对象都有一个虚指针指向该类的虚函数表。

面试例题 3：声明一个类 Vehicle，使其成为抽象数据类型。写出类 Car 和 Bus 的声明，其中每个类都从类 Vehicle 里派生。使 Vehicle 成为一个带有两个纯虚函数的 ADT，使 Car 和 Bus 不是 ADT。[美国某著名移动通信企业面试题]**答案：**

```
class Vehicle
{
public:
    virtual void Move()=0;
    virtual void Haul()=0;
};

class Car : public Vehicle
{
public:
    virtual void Move();
    virtual void Haul();
};

class Bus : public Vehicle
{
public:
    virtual void Move();
    virtual void Haul();
};
```

面试例题 4：虚函数的入口地址和普通函数有什么不同？[英国某著名计算机图形图像公司面试题]

答案：每个虚函数都在 vtable 中占了一个表项，保存着一条跳转到它的入口地址的指令（实际上就是保存了它的入口地址）。当一个包含虚函数的对象（注意，不是对象的指针）被创建的时候，它在头部附加一个指针，指向 vtable 中相应的位置。调用虚函数的时候，不管你是用什么指针调用的，它先根据 vtable 找到入口地址再执行，从而实现了“动态联编”。而不像普通函数那样简单地跳转到一个固定地址。

面试例题 5：

1. C++中如何阻止一个类被实例化？
2. 一般在什么时候构造函数被声明成 private 呢？
3. 什么时候编译器会生成默认的 copy constructor 呢？
4. 如果你已经写了一个构造函数，编译器还会生成 copy constructor 吗？ [英国某著名计算机图形图像公司面试题]

答案：

1. 使用抽象类，或者构造函数被声明成 private。
2. 比如要阻止编译器生成默认的 copy constructor 的时候。
3. 只要自己没写，而程序中需要，都会生成。
4. 会。

扩展知识

在 C# 中，有一个专门的 seal class（封闭类）来防止继承。

11.7 COM

面试例题 1：什么是 COM？你怎么理解 COM？[美国某著名计算机软件公司面试题]

解析：在讨论 COM 以前，我们得认识到一个事实，编写软件实际上是一项非常耗费时间和金钱的活动，所以人们不断寻找方法以减少这些花费。一个很重要的方法就是“软件重用”。在一个理想的环境下，我们应该能够编写一次代码，在任何地方都可以运行，即使编写者都没有想到过某个环境。当一个程序员修改了自己发布给别人使用的函数功能后，使用者应该不需要改变或者重新编译程序就可以使用这个功能。

早期的努力是使用类库，这在 C++ 中比较常见，但是这种做法有很大缺陷，因为要共享 C++ 的二进制代码是非常困难的。为了解决这个问题，程序员们试图建立一种标准去达到软件在二进制级别上的共用。

答案：Components Object Model (COM) 是软件组件互相通信的一种方式。它是一种二进制和网络标准，允许任意两个组件互相通信，而

不管它们是在什么计算机上运行（只要计算机是相连的），不管各计算机运行的是什么操作系统（只要该系统支持 COM），也不管该组件是什么语言编写的。COM 还提供了位置透明性：当你编写组件时，其他组件是进程内 DLL、本地 EXE，还是位于其他计算机上的组件，对用户而言都无所谓。COM 是基于对象的——但是这种对象概念与你熟悉的 C++ 或 Visual Basic 中的对象不太一样。

首先，COM 对象被很好地封装起来。你无法访问对象的内部实现细节，你无法知道对象使用了什么数据结构。实际上，对象的封装是如此严密，以至于 COM 对象通常被描绘为盒子。细节是不会告诉你的，但是我们可以通过接口来访问 COM 对象里面的方法。当然，如果 COM 组件提供商肯告诉你那些接口中的函数和属性起什么作用的话。

概括地说，COM 具有如下一些优越性。

- 编程技术难度和工作量降低，开发周期变短，开发成本降低。一般编程人员只须根据应用功能要求选用合适的组件，而不必事无巨细都自己动手去完成。组件模块将编程的技术难度和工作量在人员个体和时间上进行了分摊。我们使用 ESRI 的 COM 组件编写程序就属于这一级别。
- 实现分层次的编程，从而促进了软件的专业化生产。专业人员可以开发出具有很强专业性的软件组件，这样既保证了普通的编程应用人员能够完成所需要的应用开发，又不至于降低使用的性能。应用人员不便实现的组件模块可以让专业人员定做。ESRI 的程序员们使用 C 语言辨析了一个个 COM 组件给我们使用。
- 软件的复用率提高，使软件的使用效率得到提高并延长了使用寿命。组件编程体系使大量的编程问题局部化了，使软件的更新和维护变得快速和容易，软件的成本大大降低。新的函数功能在接口没有改变的情况下很容易使用。

面试例题 2：COM 是接口么？[美国某著名计算机软件公司面试题]

答案：

- (1) COM 不是接口，也不是对象，它是一种标准。
- (2) 符合 COM 标准的对象就是 COM 对象。其实 COM 对象无非

是实现了很多接口的对象而已。

(3) COM 对象必须实现 IUnknown 接口，这个接口是管理 COM 对象生命周期的。当 COM 对象不使用的时候，这个接口定义的方法负责释放内存。一个 COM 对象可以没有任何别的接口，但是必须要有这个接口，它是默认实现的接口。

(4) QI，即所谓查询接口。由于 COM 对象有很多个接口，不同的接口管理着 COM 的不同类型的方法，因此从一个接口可以使用的方法转到另一个接口可以使用的方法的过程称为 QI，这个过程是由 Idispatch 接口管理的。

(5) GUIDs 每个组件都有一个独一无二的标识，这就是所谓的广泛唯一标识符。这个标识符就是 COM 组件的身份，它是一个 128 位的数字，由系统自由分配。不用担心这个标识会有重复的一天。如果我们每秒产生 1 000 万个 UID，那么到 5770 年才可能遇到重复的情况。别告诉我那个时候我们还使用 Windows 这玩意。

(6) 一个 COM 对象可以有多个接口，一个接口也完全可以被多个 COM 对象实现。

(7) 接口分为两种：内置接口和外置接口。前一种定义的是 COM 对象的方法和属性，用 implements 实现，COM 对象必须实现所有的接口内容；后一种定义的是 COM 对象的事件，用 withEvents 实现，这种接口在实现的时候不必实现所有的内容。

(8) COM 组件必须被注册后才能使用，它得到注册表那里去登记“户口”。

面试例题 3：COM 有什么缺陷？[美国某著名计算机软件公司面试题]

答案：

COM 组件很不错，可是它也有致命的缺陷，这个缺陷就来自它本身。COM 是可以被重用的，COM 对象的实现过程也可以被修改升级（定义是不能修改的）。如果两个程序都使用一个 COM 对象，而这个 COM 组件升级了的话，很可能会出现某个程序无法使用新组件的情况，这被称为“DLL HELL”（DLL 灾难）。有时候我们安装了新软件后很多别的软件就无法使用了，很大程度上就是因为这个 DLL HELL。

第 12 章

位运算与嵌入式编程

C语言测试是招聘嵌入式系统程序员必须且有效的方法。我参加了许多这种测试，在此过程中我意识到这些测试能为面试者和被面试者提供许多有用的信息。此外，撇开面试的压力不谈，这种测试也相当有趣。

从被面试者的角度来讲，你能了解许多关于出题者或监考者的情况。这个测试只是出题者为显示其对 ANSI 标准细节的知识而不是技术技巧而设计的吗？如要你答出某个字符的 ASCII 值。这些面试例题着重考查你的系统调用和内存分配策略方面的能力吗？这标志着出题者也许花时间在微机上而不是在嵌入式系统上。如果上述任何问题的答案是“是”的话，那么我知道我得认真考虑是否应该去做这份工作。

从面试者的角度来讲，一个测试也许能从多方面揭示应试者的素质。最基本的，你能了解应试者 C 语言的水平。应试者是以好的直觉做出明智的选择，还是只是瞎蒙呢？当应试者在某个问题上卡住时是找借口，还是表现出对问题的真正的好奇心，把这看成学习的机会呢？我发现这些信息与面试者们的测试成绩一样有用。

有了这些想法，我们再结合一些真正针对嵌入式系统的考题，希望这些令人头痛的考题能给正在找工作的人一点儿帮助。其中有些题很难，但它们应该都能给你一点儿启迪。

12.1 位制转换

面试例题 1：求下列程序的输出结果。[美国某著名计算机硬件公司面试题]

```
#include<stdio.h>
int main()
{
    printf("%f", 5);
    printf("%d", 5.01);
}
```

解析：首先参数 5 为 int 型，32 位平台中为 4 字节，因此在 stack 中分配 4 字节的内存，用于存放参数 5。

然后 printf 根据说明符 “%f”，认为参数应该是个 double 型（在 printf 函数中，float 会自动转换成 double），因此从 stack 中读了 8 个字节。

很显然，内存访问越界，会发生什么情况不可预料。如果在 printf 或者 scanf 中指定了 “%f”，那么在后面的参数列表中也应该指定一个浮点数，或者一个指向浮点变量的指针，否则不应加载支持浮点数的函数。

于是("%f",5)有问题，而("%f",5.0)则可行。

答案：

第一个答案是 0.000000。

第二个答案是一个大数。

面试例题 2：Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```
//The function need set corresponding bit int 0
#define BIT_MASK(bit_pos) (0x01<<(bit_pos))

int Bit_Reset(unsigned int* val, unsigned char pos) {
    if(pos >= sizeof(unsigned int) * 8) {
        return 0;
    }

    *val=(*val && ~BIT_MASK(pos));
    return 1;
}
```

解析：这道程序体存在着位运算问题。

答案：*val=(*val && ~BIT_MASK(pos))这一语句中的“&&”应为“&”。

正确的程序如下所示：

```
#include <iostream>
#define BIT_MASK(bit_pos) (0x01<<(bit_pos))
```

```

int Bit_Reset(unsigned int* val, unsigned char pos) {
    if(pos >= sizeof(unsigned int) * 8) {
        return 0;
    }

    *val=(*val & ~BIT_MASK(pos));
    return 1;
}

int main() {
    unsigned int x=0xffffffff;
    unsigned char y=4;

    Bit_Reset(&x,y);
    std::cout<<std::hex<<x<<'\n';
    return 0;
}

```

面试例题 3: Write a program that convert an octet number to a decimal number. (写一个八进制数转化成十进制数的程序。) [中国台湾某著名杀毒软件公司 2005 年面试题]

解析: 这是典型的位制转化问题。十进制 = 八进制 / 10 × 8 + 八进制 % 10。

答案:

正确的程序如下所示：

```

#include <iostream>
using namespace std;

unsigned int oct2dec(unsigned int oct)
{
    //可以通过加入一个正则表达式确定是否是八进制数，即不包含非法字符
    return oct/10*8+oct%10;
}

int main() {
    unsigned int oct=0;
    unsigned int dec=0;
    cout<<"Please input a octet number (Besure the number you
input is begin with a '0'): "<<'\n';
    cin>>oct;
    dec=oct2dec(oct);
    cout<<dec<<endl;

    return 0;
}

```

面试例题 4: In C++, there are four types of Casting Operators. Please enumerate and explain them especially the difference. (在 C++ 中，有 4 种运算符转化，请列举它们并解释它们之间的不同。) [德国某著名软件咨询企业

2005 年面试题]

答案：

4 种运算符如下：

- static_cast 数制转换。
- dynamic_cast 用于执行向下转换和在继承之间的转换。
- const_cast 去掉 const。
- reinterpret_cast 用于执行并不安全的 implementation_dependent 类型转换。

扩展知识

C++同时提供了 4 种新的强制转型形式（通常称为新风格的或 C++风格的强制转型）：

- const_cast(expression)
- dynamic_cast(expression)
- reinterpret_cast(expression)
- static_cast(expression)

每一种适用于特定的目的。

dynamic_cast 主要用于执行“安全的向下转型（safe downcasting）”。也就是说，要确定一个对象是否是一个继承体系中的一个特定类型。它是唯一不能用旧风格语法执行的强制转型，也是唯一可能有重大运行时代价的强制转型。

static_cast 可以被用于强制隐型转换（例如，non-const 对象转型为 const 对象，int 转型为 double，等等），它还可以用于很多这样的转换的反向转换（例如，void* 指针转型为有类型指针，基类指针转型为派生类指针）。但是它不能将一个 const 对象转型为 non-const 对象（只有 const_cast 能做到）。它最接近于 C-style 的转换。

const_cast 一般用于强制消除对象的常量性。它是唯一能做到这一点的 C++风格的强制转型。

reinterpret_cast 是特意用于底层的强制转型，导致实现依赖（implementation-dependent）（就是说，不可移植）的结果。例如，将一个指针转型为一个整数。这样的强制转型在底层代码以外应

该极为罕见。

旧风格的强制转型依然合法，但是新的形式更可取。首先，在代码中它们更容易识别（无论是人还是像 grep 这样的工具都是如此），这样就简化了在代码中寻找类型系统被破坏的地方的过程。第二，更精确地指定每一个强制转型的目的，使得编译器诊断使用错误成为可能。例如，如果你试图使用一个 const_cas 以外的新风格强制转型来消除常量性，你的代码将无法编译。

面试例题 5：下面程序的运行结果是什么？[美国某著名计算机软硬件公司面试题]

```
#include <iostream>
using namespace std;
int main()
{
    unsigned short int i = 0;
    int j = 8, p;
    p = j << 1;
    i = i - 1;
    cout <<"\n i = " << i ;
    cout <<"\n p = " << p ;
    return 0;
}
```

解析：此题美国某著名计算机软硬件公司有两个考点：一是用最有效率的方法算出 2×8 等于几，二是无符号结果问题。

在这里，8 左移一位就是 8×2 的结果 16。移位运算是最有效率的计算乘/除算法的运算之一。在 unsigned short int 中无符号的 -1 的结果等于 65 535。

答案：16, 65535。

面试例题 6：建立一个联合体，由 char 类型和 int 类型组成。下面的程序运行结果是什么？

```
#include <iostream>
using namespace std;

union {
    unsigned char a;
    unsigned int i;
} u;

int main() {
    u.i=0xf0f1f2f3;
```

```

    cout<<hex<<u.i<<endl;
    cout<<hex<<int(u.a)<<endl;
    return 0;
}

```

解析：内存中数据的排列问题。

答案：

运行下面程序后，输出为：

```

f0f1f2f3
f3

```

这说明，内存中数据低位字节存入低地址，高位字节存入高地址，而数据的地址采用它的低地址来表示。

面试例题 7：嵌入式系统总是要用户对变量或寄存器进行位操作。给定一个整型变量 a，写两段代码，第一个设置 a 的 bit 3，第二个清除 a 的 bit 3。在以上两个操作中，要保持其他位不变。

解析：被面试者对这个问题有 3 种基本的反应。

一种是不知道如何下手。显然该被面试者从没做过任何嵌入式系统的工作。

还有一种是用 bit fields。bit fields 是被扔到 C 语言死角的东西，它保证你的代码在不同编译器之间是不可移植的，同时也保证了你的代码是不可重用的。

还有一种是用#define 和 bit masks 操作。这是一个有极高可移植性的方法，是应该被用到的方法。

一些人喜欢为设置和清除值而定义一个掩码，同时定义一些说明常数，这也是可以接受的。面试官希望看到几个要点：说明常数、“|=”和“&=~”操作。

答案：

最佳的解决方案如下：

```

#define BIT3 (0x1 << 3)
static int a;

void set_bit3(void) {
    a |= BIT3;
}

void clear_bit3(void) {

```

```
a &= ~BIT3;
}
```

面试例题 8：用<<、>>、|、&实现一个 WORD(ABCD)的高低位交换。[美国某著名计算机软件公司面试题]

答案：

最佳的解决方案如下：

```
# include<stdio.h>
int main()
{
    unsigned short a = 0xABCD;
    unsigned short b ;
    unsigned short c ,d;
    b = (a << 8)&0xff00;
    c = (a >> 8)&0x00ff;
    d = b | c;
    printf("\n%x",b);
    printf("\n%x",c);
    printf("\n%x",d);
    return 0;
}
```

结果是 CDAB

2 个字节是 16 位，前 8 位为高位，后 8 位为低位，然后结合。

12.2 嵌入式编程

面试例题 1：Interrupts are an important part of embedded systems. Consequently, many compiler vendors offer an extension to standard C to support interrupts. Typically, the keyword is _interrupt. The following routine(ISR). Point out the errors in the code. (中断是嵌入式系统中重要的组成部分，这导致了很多编译开发商提供一种扩展——让标准 C 支持中断。其代表事实是，产生了一个新的关键字_interrupt。请看下面的程序（一个中断服务子程序 ISR），请指出这段代码的错误。) [中国台湾某著名 CPU 生产公司 2005 年面试题]

```
interrupt double compute_area(double radius)
{
    double area= PI*radius*radius;
```

```

        printf("\nArea=%f", area);
        return area;
    }
}

```

解析：嵌入式编程问题。

答案：

- (1) ISR 不能返回一个值。如果你不懂这个，那么是会被雇用的。
- (2) ISR 不能传递参数。如果你没有看到这一点，被雇用的机会等同第一项。
- (3) 在许多处理器/编译器中，浮点一般都是不可重入的。有些处理器/编译器需要让额外的寄存器入栈，有些处理器/编译器就不允许在 ISR 中做浮点运算。此外，ISR 应该是短而有效率的，在 ISR 中做浮点运算是不明智的。
- (4) 与第三点一脉相承，printf()经常有重入和性能上的问题，所以一般不使用 printf()。

面试例题 2：In embedded system, we usually use the keyword "volatile", what does the keyword mean? (在嵌入式系统中，我们经常使用“volatile”这个关键字，它是什么意思？) [中国台湾某著名 CPU 生产公司 2005 年面试题]

解析：volatile 问题。

当一个对象的值可能会在编译器的控制或监测之外被改变时，例如一个被系统时钟更新的变量，那么该对象应该声明成 volatile。因此编译器执行的某些例行优化行为不能应用在已指定为 volatile 的对象上。

volatile 限定修饰符的用法与 const 非常相似——都是作为类型的附加修饰符。例如：

```

volatile int display_register;
volatile Task *curr_task;
volatile int ixa[ max_size ];
volatile Screen bitmap_buf;

```

display_register 是一个 int 型的 volatile 对象；curr_task 是一个指向 volatile 的 Task 类对象的指针；ixa 是一个 volatile 的整型数组，数组的每个元素都被认为是 volatile 的；bitmap_buf 是一个 volatile 的 Screen 类对象，它的每个数据成员都被视为 volatile 的。

volatile 修饰符的主要目的是提示编译器该对象的值可能在编译器未

监测到的情况下被改变，因此编译器不能武断地对引用这些对象的代码做优化处理。

答案：

`volatile` 的语法与 `const` 是一样的，但是 `volatile` 的意思是“在编译器认识的范围外，这个数据可以被改变”。不知何故，环境正在改变数据（可能通过多任务处理），所以，`volatile` 告诉编译器不要擅自做出有关数据的任何假定——在优化期间这是特别重要的。如果编译器说：“我已经把数据读进寄存器，而且再没有与寄存器接触。”在一般情况下，它不需要再读这个数据。但是，如果数据是 `volatile` 修饰的，编译器则不能做出这样的假定，因为数据可能被其他进程改变了，编译器必须重读这个数据而不是优化这个代码。

就像建立 `const` 对象一样，程序员也可以建立 `volatile` 对象，甚至还可以建立 `const volatile` 对象。这个对象不能被程序员改变，但可通过外面的工具改变。

面试例题 3：嵌入式系统中经常要用到无限循环，你怎样用 C 编写死循环呢？

解析：这个问题有几种解决方案。首选方案是：

```
while(1)
{
}
```

一些程序员更喜欢如下方案：

```
for(;;)
{
}
```

但这种实现方式让面试官为难，因为这个语法没有确切表达出到底是怎么回事。如果一个应试者给出这个作为方案，面试官将用这个作为一个机会去探究他们这样做的基本原因。如果他们的基本答案是：“我被教着这样做，但从没有想到过为什么。”这会给面试官留下一个坏印象。

第三个方案是用 `goto`：

```
Loop:
...
goto Loop;
```

应试者如给出上面的方案，这说明或者他是一个汇编语言程序员（这

也许是好事），或者他是一个想进入新领域的 Basic/ FORTRAN 程序员。

答案：

```
while(1)
{
}
```

面试例题 4：关键字 static 的作用是什么？

解析：这个简单的问题很少有人能回答完全。大多数应试者能正确回答第一部分，一部分人能正确回答第二部分，但是很少有人能懂得第三部分。这对于一个应试者是严重的缺点，因为他显然不懂得本地化数据和代码范围的好处和重要性。

答案：

在 C 语言中，关键字 static 有 3 个明显的作用。

在函数体内，一个被声明为静态的变量在这一函数被调用的过程中维持其值不变。

在模块内（但在函数体外），一个被声明为静态的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。在模块内，一个被声明为静态的函数只可被这一模块内的其他函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用。

面试例题 5：关键字 const 有什么含意？下面的声明都是什么意思？

```
const int a;
int const a;
const int *a;
int * const a;
int const * a const;
```

解析：只要一听到被面试者说“const 意味着常数”，面试官就知道自己正在和一个业余者打交道。因为 ESP (Embedded Systems Programming, 嵌入式系统编程) 的每一位求职者都应该非常熟悉 const 能做什么和不能做什么。正确的说法是能说出 const 意味着“只读”就可以了。尽管这个答案不是完全的答案，但面试官可以接受它为一个正确的答案。

关键字 const 的作用是为读你代码的人传达非常有用的信息。实际上，声明一个参数为常量是为了告诉用户这个参数的应用目的。如果你曾花很

多时间清理其他人留下的垃圾，你就会很快学会感谢这点儿多余的信息。当然，懂得用 `const` 的程序员很少会留下垃圾让别人来清理。通过给优化器一些附加的信息，使用关键字 `const` 也许能产生更紧凑的代码。

合理地使用关键字 `const` 可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。简而言之，这样可以减少 bug 的出现。

答案：前两个的作用是一样的。`a` 是一个常整型数（不可修改值的整型数）。第三个意味着 `a` 是一个指向常整型数的指针（也就是说，整型数是不可修改的，但指针可以修改）。第四个的意思是 `a` 是一个指向整型数的常指针（也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。最后一个意味着 `a` 是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时指针也是不可修改的）。

面试例题 6：关键字 `volatile` 有什么含意？并给出 3 个不同的例子。[中国台湾某著名计算机硬件公司面试题]

解析：回答不出这个问题的人是不会被雇用的。我认为这是区分 C 程序员和嵌入式系统程序员的最基本的问题。搞嵌入式的家伙们经常同硬件、中断、RTOS 等打交道，所有这些都要求用到 `volatile` 变量。不懂得 `volatile` 的内容将会带来灾难。

答案：一个定义为 `volatile` 的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是 `volatile` 变量的几个例子：

- 并行设备的硬件寄存器（如状态寄存器）
- 一个中断服务子程序中会访问到的非自动变量（Non-automatic variables）
- 多线程应用中被几个任务共享的变量

面试例题 7：一个参数可以既是 `const` 又是 `volatile` 吗？一个指针可以是 `volatile` 吗？解释为什么。

答案：

第一个问题：是的。一个例子就是只读的状态寄存器。它是 `volatile`，

因为它可能被意想不到地改变；它又是 const，因为程序不应该试图去修改它。

第二个问题：是的。尽管这并不很常见。一个例子是当一个中断服务子程序修改一个指向一个 buffer 的指针时。

面试例题 8：下面的函数有什么错误？

```
int square(volatile int *ptr)
{
    return *ptr * *ptr;
}
```

解析：这段代码的目的是用来返还指针*ptr 指向值的平方，但是，由于*ptr 指向一个 volatile 型参数，编译器将产生类似下面的代码：

```
int square(volatile int *ptr)
{
    int a,b;
    a = *ptr;
    b = *ptr;
    return a * b;
}
```

由于*ptr 的值可能被意想不到地改变，因此 a 和 b 可能是不同的。结果，这段代码可能无法返回你所期望的平方值！

答案：

正确的代码如下：

```
long square(volatile int *ptr)
{
    int a;
    a = *ptr;
    return a * a;
}
```

面试例题 9：嵌入式系统经常具有要求程序员去访问某特定位置的内存的特点。在某工程中，要求设置一绝对地址为 0x67a9 的整型变量的值为 0xaa66。编译器是一个纯粹的 ANSI 编译器。写代码去完成这一任务。

解析：

这一问题测试你是否知道为了访问一个绝对地址把一个整型数强制转换 (typecast) 为一个指针是合法的。这一问题的实现方式随着个人风格不同而不同。典型的代码如下：

```
int *ptr;
```

```
ptr = (int *)0x67a9;
*ptr = 0xaa55;
```

一个较晦涩的方法是：

```
* (int * const) (0x67a9) = 0xaa55;
```

建议你在面试时使用第一种方案。

答案：

```
int *ptr;
ptr = (int *)0x67a9;
*ptr = 0xaa55;
```

面试例题 10：评价下面的代码片断，找出其中的错误。

```
unsigned int zero = 0;
unsigned int compzero = 0xFFFF;
/*1's complement of zero */
```

解析：

这一问题真正能揭露出应试者是否懂得处理器字长的重要性。在我的经验里，好的嵌入式程序员非常准确地明白硬件的细节和它的局限，然而PC机程序往往把硬件作为一个无法避免的烦恼。对于一个int型且不是16位的处理器来说，上面的代码是不正确的。

答案：

应编写如下代码：

```
unsigned int compzero = ~0;
```

面试例题 11：下面的代码片段的输出是什么？为什么？

```
char *ptr;
if ((ptr = (char *)malloc(0)) == NULL)
    puts("Got a null pointer");
else
    puts("Got a valid pointer");
```

解析：

这是一道动态内存分配（Dynamic memory allocation）题。

尽管不像非嵌入式计算那么常见，嵌入式系统还是有从堆（heap）中动态分配内存的过程的。

面试官期望应试者能解决内存碎片、碎片收集、变量的执行时间等问题。

这是一个有趣的问题。故意把 0 值传给了函数 malloc，得到了一个合法的指针，这就是上面的代码，该代码的输出是“Got a valid pointer”。我用这个来讨论这样的一道面试例题，看看被面试者是否能想到库例程这样做是正确的。得到正确的答案固然重要，但解决问题的方法和你做决定的基本原理更重要。

将程序修改成：

```
char *ptr;
if (int pp = (strlen(ptr = (char *)malloc(0))) == 0)
    puts("Got a null pointer");
else
    puts("Got a valid pointer");
```

或者：

```
char *ptr;
if (int pp = (sizeof(ptr = (char *)malloc(0))) == 4)
    puts("Got a null pointer");
else
    puts("Got a valid pointer");
```

如果求 ptr 的 strlen 值和 sizeof 值，该代码的输出是“Got a null pointer”。

答案： Got a valid pointer。

面试例题 12： Typedef 在 C 语言中频繁用以声明一个已经存在的数据类型的同义字，也可以用预处理器做类似的事。思考一下下面的例子：

```
#define dPS struct s *
typedef struct s * tPS;
```

以上两种情况的意图都是要定义 dPS 和 tPS 作为一个指向结构 s 的指针。哪种方法更好呢（如果有多种方法的话）？为什么？

解析：这是一个非常微妙的问题，任何人答对这个问题（以正当的手段）都是应当被恭喜的。思考下面的例子：

```
dPS p1, p2;
tPS p3, p4;
```

第一个扩展为：

```
struct s * p1, p2;
```

上面的代码定义 p1 为一个指向结构的指针，p2 为一个实际的结构，这也许不是你想要的。第二个例子正确地定义了 p3 和 p4 两个指针。

答案： typedef 更好。

第3部分

数据结构和设计模式

Data structure and design pattern

本部分主要介绍求职面试过程中出现的第二个重要的板块——数据结构，包括字符串的使用、堆、栈、排序方法等。此外，随着外企研发机构大量内迁我国，在外企的面试中，软件工程的知识，包括设计模式、UML、敏捷软件开发，以及.NET技术和完全面向对象语言C#的面试题目将会有增无减。关于设计模式的题目在今后的面试中的比重会更加扩大。

第 13 章

数据结构基础

面试时间一般有 2 个小时，其中至少有 20~30 分钟左右是用来回答数据结构相关问题的。而由于链表是一种相对简单的数据结构，容易引起面试官多次反复发问。此外，数组的排序和逆置也是面试官必考的内容之一。事实上，单链表的复杂程度并不亚于树、图等复杂数据结构。面试官完全有可能构造出极富挑战性的试题。最后一个考点是堆栈，面试官会结合程序对你的思维能力进行考量。

13.1 单链表

面试例题 1：编程实现一个单链表的建立。[日本某著名家电/通信/IT 企业面试题]

答案：

完整代码如下：

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <conio.h>
using namespace std;

typedef struct student
{
    int data;
    struct student *next;
}node;
node *creat()
{
    node *head, *p, *s;
```

```
int x,cycle=1;
head=(node*)malloc(sizeof(node));
p=head;
while(cycle)
{
    printf("\nplease input the data: ");
    scanf("%d",&x);
    if(x!=0)
    {
        s=(node *)malloc(sizeof(node));
        s->data=x;
        printf("\n      %d",s->data);
        p->next=s;
        p=s;
    }
    else cycle=0;
}
head=head->next;
p->next=NULL;
printf("\n      YYY %d",head->data);
return(head);
}
```

面试例题 2：编程实现单链表的测长。

答案：

完整代码如下：

```
int length(node *head)
{
    int n=0;
    node *p;
    p=head;
    while(p!=NULL)
    {
        p=p->next;
        n++;
    }
    return(n);
}
```

面试例题 3：编程实现单链表的打印。

答案：

完整代码如下：

```
void print(node *head)

{ node *p;int n;
n=length(head);
printf("\nNow, These %d records are :\n",n);
```

```

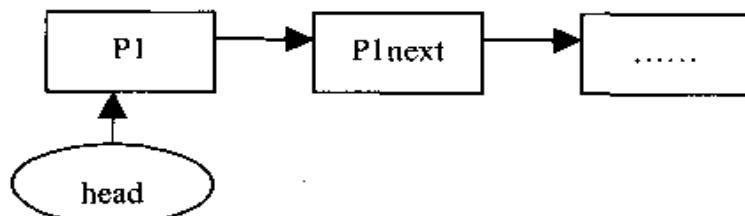
p=head;
if(head!=NULL)

    while(p!=NULL)
    { printf("\n    uuu %d      ",p->data);
      p=p->next;
    }
}

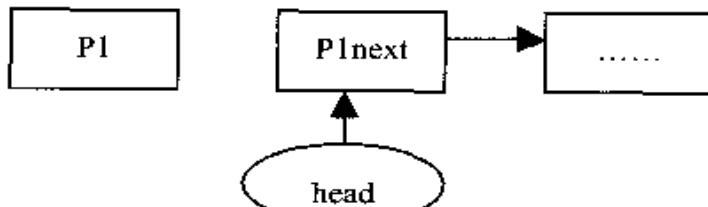
```

面试例题4：编程实现单链表删除节点。[美国某著名分析软件公司面试题]

解析：如果删除的是头节点，如下图所示。



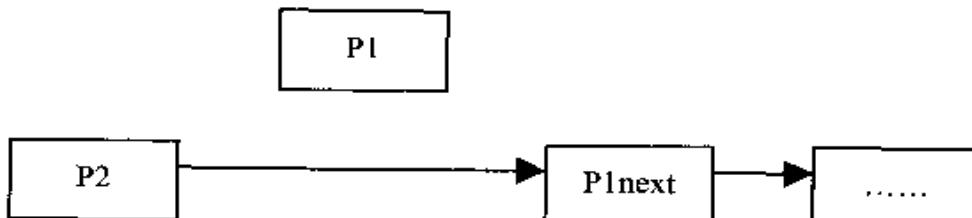
则把 head 指针指向头节点的下一个节点。同时 free p1，如下图所示。



如果删除的是中间节点，如下图所示。



则用 p2 的 next 指向 p1 的 next 同时，free p1，如下图所示。



答案：

完整代码如下：

```

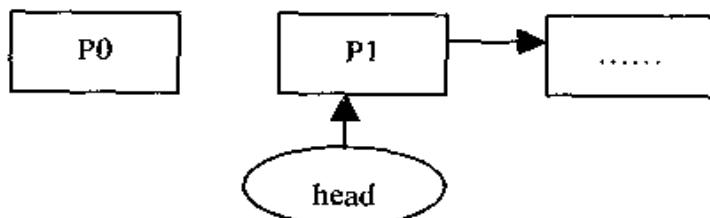
node *del(node *head, int num)
{
    node *p1,*p2;
    p1=head;
    while(num!=p1->data&&p1->next!=NULL)
        {p2=p1;p1=p1->next;}
}

```

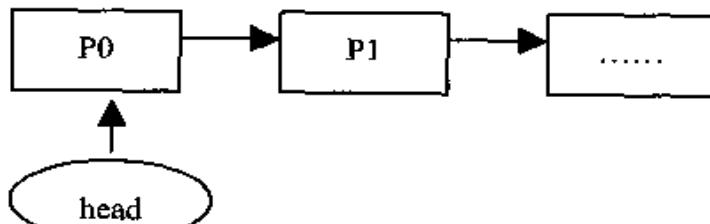
```
if (num==p1->data)
{
    if(p1==head)
        {head=p1->next;
         free(p1);}
    else
        p2->next=p1->next;
}
else
    printf("\n%d could not been found",num);
return(head);
}
```

面试例题 5：编写程序实现单链表的插入。[美国某著名计算机嵌入式公司 2005 年面试题]

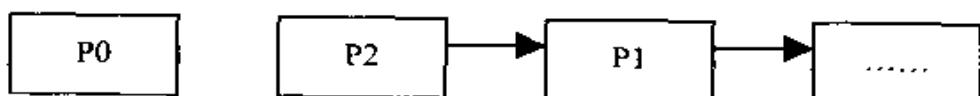
解析：单链表的插入，如下图所示。



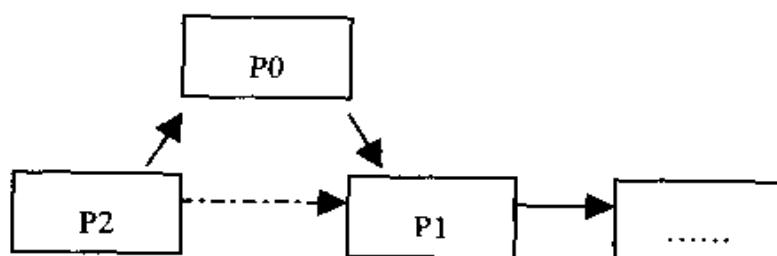
如果插人在头节点以前，则 p0 的 next 指向 p1，头节点指向 p0，如下图所示。



如果插人中间节点，如下图所示。



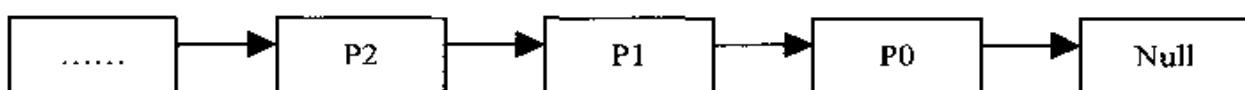
则先让 p2 的 next 指向 p0，再让 p0 指向 p1，如下图所示。



如果插入尾节点，如下图所示。



则先让 p1 的 next 指向 p0，再让 p0 指向空，如下图所示。



答案：

完整代码如下：

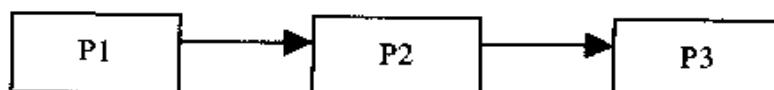
```

node *insert(node *head, int num)
{
    node *p0, *p1, *p2;
    p1 = head;
    p0 = (node *)malloc(sizeof(node));
    p0->data = num;
    while (p0->data > p1->data && p1->next != NULL)
        {p2 = p1; p1 = p1->next;}
    if (p0->data <= p1->data)
    {
        if (head == p1)
            {p0->next = p1;
            head = p0;}
    }
    else
    {
        p2->next = p0;
        p0->next = p1;
    }
}
else
{
    p1->next = p0; p0->next = NULL;
}
return (head);
}
  
```

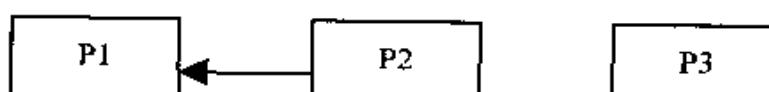
面试例题 6：编程实现单链表的逆置。[美国某著名分析软件公司 2005 年面试题]

解析：

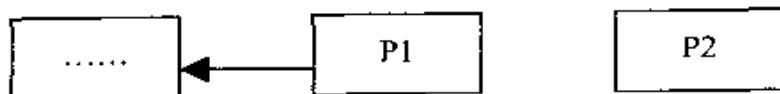
单链表模型如下图所示。



进行单链表逆置，首先要让 p2 的 next 指向 p1，如下图所示。



再由 p1 指向 p2，p2 指向 p3，如下图所示。



然后重复 p2 的 next 指向 p1, p1 指向 p2, p2 指向 p3。

答案：

完整代码如下：

```
node *reverse(node *head)
{
    node *p1, *p2, *p3;

    if (head == NULL || head->next == NULL)
        return head;

    p1 = head, p2 = p1->next;
    while (p2)
    {
        p3=p2->next;
        p2->next=p1;
        p1=p2;
        p2=p3;
    }

    head->next = NULL;
    head = p1;
    return head;
}

int main()
{
    node *head, stud;
    int n, del_num, insert_num;
    head=creat();
    print(head);
    cout << "\nInt : ";
    cin >> del_num;
    head=del(head, del_num);
    print(head);
    cout << "\nplease input the insert data: ";
    cin >> insert_num;
    head=insert(head, insert_num);
    print(head);

    return 0;
}
```

面试例题 7：有一个 C 语言用来删除单链表的头元素的函数，请找出其中的问题并加以纠正。[中国某著名综合软件公司 2005 年面试题]

```
void RemoveHead(node* head)
{
    free(head)
```

```

    head=head->next
}

```

解析：如果先做 free(head)之后，就找不到 head 了，下一句 head= head->next 就不能正确链接了。

答案：

正确程序如下：

```

void RemoveHead(node* head)
{
    node *p;
    p=head->next;
    head->next=p->next;
    free(p);
}

```

13.2 双链表

双链表的情况与单链表类似，只是增加了一个前置链而已。

面试例题 1：编程实现双链表的建立。

答案：

完整代码如下：

```

#include <iostream>
#include <stdio.h>
#include <string.h>
#include <conio.h>
using namespace std;

typedef struct student
{
    int data;
    struct student *next;
    struct student *pre;
}dnode;
//建立双链表
dnode *creat()
{
    dnode *head,*p,*s;
    int x,cycle=1;
    head=(dnode*)malloc(sizeof(dnode));
    p=head;
    while(cycle)
    {
        printf("\nplease input the data: ");
        scanf("%d",&x);
        if(x!=0)

```

```
    {
        s=(dnode *)malloc(sizeof(dnode));
        s->data=x;
        printf("\n      %d",s->data);
        p->next=s;
        s->pre=p;
        p=s;
    }
    else cycle=0;
}
head=head->next;
head->pre=NULL;
p->next=NULL;

printf("\n yyy  %d",head->data);
return(head);
}
```

面试例题 2：编程实现双链表的测长。

答案：

完整代码如下：

```
int length(dnode *head)
{
    int n=0;
    dnode *p;
    p=head;
    while(p!=NULL)
    {
        p=p->next;
        n++;
    }
    return(n);
}
```

面试例题 3：编程实现双链表的打印。

答案：

完整代码如下：

```
void print(dnode *head)

{ dnode *p;int n;
n=length(head);
printf("\nNow, These %d records are :\n",n);
p=head;
if(head!=NULL)
    while(p!=NULL)
        { printf("\n      %d      ",p->data);
```

```

        p=p->next;
    }
}

```

面试例题4：编程实现双链表删除节点。

答案：

完整代码如下：

```

dnode *del(dnode *head, int num)
{
    dnode *p1, *p2;
    p1=head;
    while(num!=p1->data&&p1->next!=NULL)
        {p1=p1->next; }

    if(num==p1->data)
    {
        if(p1==head)
            {head=head->next;
             head->pre=NULL;
             free(p1); }
        else if(p1->next==NULL)
        {
            p1->pre->next=NULL;
            free(p1);
        }
        else
        {
            p1->next->pre=p1->pre;
            p1->pre->next=p1->next;
        }
    }
    else
        printf("\n%d could not been found",num);
    return(head);
}

```

面试例题5：编程实现双链表的插入。

答案：

完整代码如下：

```

dnode *insert(dnode *head,int num)

{
    dnode *p0,*p1;
    p1=head;
    p0=(dnode *)malloc(sizeof(dnode));
    p0->data=num;
    while(p0->data>p1->data&&p1->next!=NULL)

```

```

        {pl=pl->next;}
        if(p0->data<=pl->data)
        {
            if(head==pl)
            {
                p0->next=pl;
                pl->pre=p0;
                head=p0;
            }

            else
            {
                pl->pre->next=p0;
                p0->next=pl;
                p0->pre=pl->pre;
                pl->pre=p0;
            }
        }
        else //比哪个都大的情况
        {
            pl->next=p0; p0->pre=pl; p0->next=NULL;
        }
    return(head);
}

int main()
{
    dnode *head,stud;
    int n,del_num,insert_num;
    head=creat();
    print(head);
    cout << "\nInt : ";
    cin >> del_num;
    head=del(head,del_num);
    print(head);
    cout << "\nplease input the insert data: ";
    cin >> insert_num;
    head=insert(head,insert_num);
    print(head);
    return 0;
}

```

13.3 循环链表

面试例题 1：

设单链表中节点的结构为：

```

typedef struct node
{
    ElemenType data;           // 数据
    struct node* Link;         // 节点后继指针
}

```

```
}; ListNode;
```

(1) 已知指针 p 所指节点不是尾节点, 若在*p 之后插入节点*s, 则应执行下列哪一个操作?

- A. s->link=p; p->link=s;
- B. s->link=p->link; p->link=s;
- C. s->link=p->link; p=s;
- D. p->link=s; s->link=p;

(2) 非空的循环单链表 first 的尾节点 (由 p 所指向) 满足:

- A. p->link==NULL;
- B. p==NULL;
- C. p->link==first;
- D. p==first;

[中国某著名综合软件公司 2005 年面试题]

解析: 循环链表问题。

答案: B, C。

面试例题 2: 如何证明一个表是循环链表? [美国某著名 CPU 生产公司面试题]

答案: 可以设置两个步长不等的游标, 一个步长为 1, 一个步长为 2。两个游标同时出发若能再次相遇则证明该表是循环链表。

13.4 队列

面试例题 1: 编程实现队列的入队。[美国某著名计算机嵌入式公司 2005 年面试题]

答案:

完整代码如下:

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <conio.h>
using namespace std;

typedef struct student
{
    int data;
    struct student *next;
```

```

}node;

typedef struct linkqueue
{
    node *first,*rear;
}queue;

//队列的入队
queue *insert(queue *HQ,int x)
{
    node *s;
    s=(node *)malloc(sizeof(node));
    s->data=x;
    s->next=NULL;

    if(HQ->rear ==NULL)
    {
        HQ->first = s;
        HQ->rear=s;
    }
    else
    {
        HQ->rear->next=s;
        HQ->rear=s;
    }
    return(HQ);
}

```

面试例题 2：编程实现队列的出队。[美国某著名计算机嵌入式公司 2005 年面试题]

答案：

完整代码如下：

```

queue *del(queue *HQ)
{
    node *p,int x;

    if(HQ->first ==NULL)
    {
        printf("\n yichu");
    }

    else
    {
        x=HQ->first->data;
        p=HQ->first;
        if(HQ->first==HQ->rear)
        {
            HQ->first=NULL;
            HQ->rear=NULL;
        }
        else
        {
            HQ->first=HQ->first->next;
        }
    }
}

```

```

    }
    else
    {
        HQ->first=HQ->first->next;
        free(p);
    }
    return(HQ);
}

}

```

面试例题3：编程实现队列的测长。

答案：

完整代码如下：

```

int length(node *HS)
{
    int n=0;
    node *p;
    p=HS;
    while(p!=NULL)
    {
        p=p->next;
        n++;
    }
    return(n);
}

```

面试例题4：编程实现打印队列。

答案：

完整代码如下：

```

void print(queue *HQ)

{ node *p;int n;
p=HQ->first;

while(p!=NULL)
{ printf("\n      %d      ",p->data);
  p=p->next;
}

int main()
{ node stud;
queue *HQ;
int n,del,num,insert num;
HQ=(queue *)malloc(sizeof(queue));
HQ=insert(HQ,1);

```

```

//print(HQ);
HQ=insert(HQ,2);
print(HQ);
HQ=del(HQ);
print(HQ);
return 0;
}

```

13.5 堆栈

面试例题 1:全部变量放在()；函数内部变量 static int ncount 放在()；函数内部变量 char *p="AAA", p 的位置在()；指向空间的位置()；函数内变量 char *p=new char;, p 的位置()；指向空间的位置()。

[中国某著名综合软件公司 2005 年面试题]

- A. 数据段
- B. 代码段
- C. 堆栈
- D. 堆
- E. 不一定，看情况

解析：

堆栈是一种简单数据结构，是一种只允许在其一端进行插入或删除的线性表。允许插入或删除操作的一端称为栈顶，另一端称为栈底。对堆栈的插入和删除操作称为入栈和出栈。有一组 CPU 指令可以实现对进程的内存实现堆栈访问。其中，POP 指令实现出栈操作，PUSH 指令实现入栈操作。CPU 的 ESP 寄存器存放当前线程的栈顶指针，EBP 寄存器中保存当前线程的栈底指针。CPU 的 EIP 寄存器存放下一个 CPU 指令存放的内存地址。当 CPU 执行完当前的指令后，从 EIP 寄存器中读取下一条指令的内存地址，然后继续执行。

答案：A, A, C, A, D, A。

扩展知识（变量的内存分配情况）

接触过编程的人都知道，高级语言都能通过变量名访问内存中的数据。那么这些变量在内存中是如何存放的呢？程序又是如何使用这些变量的呢？

首先，来了解一下 C 语言的变量是如何在内存分布的。C 语言有全局变量（Global）、本地变量（Local）、静态变量（Static）

和寄存器变量（Register）。每种变量都有不同的分配方式。先来看下面这段代码：

```
#include <stdio.h>
int g1=0, g2=0, g3=0;
int main()
{
    static int s1=0, s2=0, s3=0;
    int v1=0, v2=0, v3=0;
    //打印出各个变量的内存地址

    printf("0x%08x ", &v1); //打印各本地变量的内存地址
    printf("0x%08x ", &v2);
    printf("0x%08x ", &v3);
    printf("0x%08x ", &g1); //打印各全局变量的内存地址
    printf("0x%08x ", &g2);
    printf("0x%08x ", &g3);
    printf("0x%08x ", &s1); //打印各静态变量的内存地址
    printf("0x%08x ", &s2);
    printf("0x%08x ", &s3);
    return 0;
}
```

编译后的执行结果是：

```
0x0012ff78
0x0012ff7c
0x0012ff80

0x004068d0
0x004068d4
0x004068d8

0x004068dc
0x004068e0
0x004068e4
```

输出的结果就是变量的内存地址。其中 v1、v2、v3 是本地变量，g1、g2、g3 是全局变量，s1、s2、s3 是静态变量。你可以看到这些变量在内存中是连续分布的，但是本地变量和全局变量分配的内存地址差了十万八千里，而全局变量和静态变量分配的内存是连续的。这是因为本地变量和全局/静态变量是分配在不同类型的内存区域中的结果。对于一个进程的内存空间而言，可以在逻辑上分成 3 个部分：代码区、静态数据区和动态数据区。动态数据区一般就是“堆栈”。“栈（stack）”和“堆（heap）”是两种不同的动态数据区。栈是一种线性结构，堆是一种链式结构。进程的每个线程都有私有的“栈”，所以每个线程虽然代码一样，但本地变量的数据都是互不干扰的。

一个堆栈可以通过“基地址”和“栈顶”地址来描述。全局变量和静态变量分配在静态数据区，本地变量分配在动态数据区，即堆栈中。程序通过堆栈的基地址和偏移量来访问本地变量，如右图所示。

堆栈是一个先进后出的数据结构，栈顶地址总是小于等于栈的基地址。我们可以先了解一下函数调用的过程，以便对堆栈在程序中的作用有更深入的了解。不同的语言有不同的函数调用规定，这些因素有参数的压入规则和堆栈的平衡。Windows API 的调用规则和ANSI C 的函数调用规则是不一样的，前者由被调函数调整堆栈，后者由调用者调整堆栈。两者通过“`_stdcall`”和“`_cdecl`”前缀区分。先看下面这段代码：

```
#include <stdio.h>

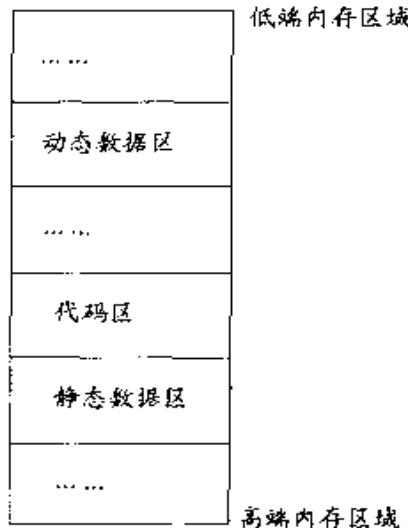
void stdcall func(int param1,int param2,int param3)
{
    int var1=param1;
    int var2=param2;
    int var3=param3;
    printf("0x%08x ",?m1); //打印出各个变量的内存地址
    printf("0x%08x ",?m2);
    printf("0x%08x ",?m3);
    printf("0x%08x ",&var1);
    printf("0x%08x ",&var2);
    printf("0x%08x ",&var3);
    return;
}

int main()
{
    func(1,2,3);
    return 0;
}
```

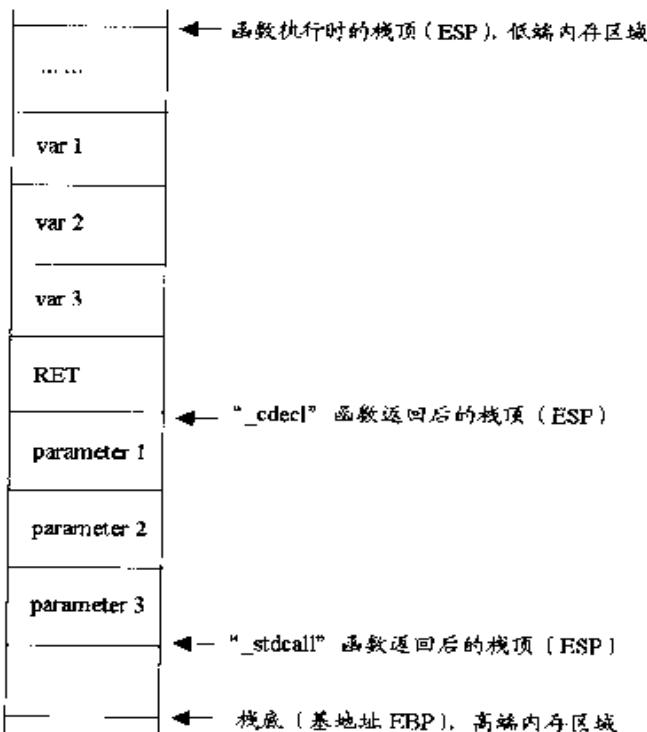
编译后的执行结果是：

```
0x0012ff78
0x0012ff7c
0x0012ff80

0x0012ff68
0x0012ff6c
0x0012ff70
```



下图就是函数调用过程中堆栈的样子。首先，3个参数以从右到左的次序压入堆栈，先压“param3”，再压“param2”，最后压入“param1”。然后压入函数的返回地址（RET），接着跳转到函数地址接着执行（这里要补充一点，介绍 UNIX 下的缓冲溢出原理的文章中都提到在压入 RET 后，继续压入当前 EBP，然后用当前 ESP 代替 EBP。然而，有一篇介绍 Windows 下函数调用的文章中说，在 Windows 下的函数调用也有这一步骤。但根据我的实际调试，并未发现这一步，这还可以从 param3 和 var1 之间只有 4 字节的间隙这点看出来）。第三步，将栈顶（ESP）减去一个数，为本地变量分配内存空间，上例中是减去 12 字节（ $ESP=ESP - 3*4$ ，每个 int 变量占用 4 个字节）。接着就初始化本地变量的内存空间。由于“_stdcall”调用由被调函数调整堆栈，所以在函数返回前要恢复堆栈，先回收本地变量占用的内存（ $ESP=ESP+3 \times 4$ ），然后取出返回地址，填入 EIP 寄存器，回收先前压入参数占用的内存（ $ESP=ESP+3 \times 4$ ），继续执行调用者的代码。



Windows 下的动态数据除了可存放在栈中，还可以存放在堆中。了解 C++的朋友都知道，C++可以使用 new 关键字来动态分配内存。来看下面的 C++ 代码：

```
#include <stdio.h>
#include <iostream.h>
#include <windows.h>

void func()
{
    char *buffer=new char[128];
    char bufflocal[128];
    static char buffstatic[128];
    printf("0x%08x ",buffer);           //打印堆中变量的内存地址
    printf("0x%08x ",bufflocal);        //打印本地变量的内存地址
    printf("0x%08x ",buffstatic);       //打印静态变量的内存地址
}

void main()
{
    func();
    return;
}
```

程序执行结果为：

```
0x004107d0
0x0012ff04
0x004068c0
```

可以发现用 new 关键字分配的内存既不在栈中，也不在静态数据区。

面试例题 2：以下哪一个不是栈的基本运算？[中国某著名综合软件公司 2005 年面试题]

- A. 删除栈顶元素
- B. 删除栈底元素
- C. 判断栈是否为空
- D. 将栈置为空栈

解析：删除栈底元素不是栈的基本运算

答案：B。

面试例题 3：heap 与 stack 的差别是什么？[美国某著名数据库公司 2006 年面试题]

答案：

heap 是堆，stack 是栈。

stack 的空间由操作系统自动分配/释放，heap 上的空间手动分配/释放。

stack 空间有限，heap 是很大的自由存储区。

C 中的 malloc 函数分配的内存空间即在堆上，C++ 中对应的是 new 操作符。

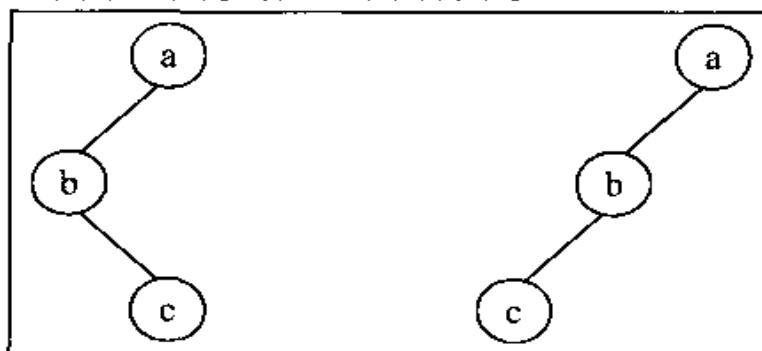
程序在编译期对变量和函数分配内存都在栈上进行，且程序运行过程中函数调用时参数的传递也在栈上进行。

13.6 树

面试例题1：如果一棵二叉树节点的前序序列是A、B、C，后序序列是C、B、A，则该二叉树节点的中序序列是什么？[中国某著名计算机金融软件公司2005年面试题]

- A. 必为A、B、C
- B. 必为A、C、B
- C. 必为B、C、A
- D. 不能确定

解析：举两个例子来说明，如下图所示。



上图左边的二叉树前序序列为a、b、c，后序序列为c、b、a，中序序列为b、a、c；右边的二叉树前序序列为a、b、c，后序序列为c、b、a，中序序列为c、b、a。也就是说，知道一个二叉树的前序序列和后序序列，不一定能够确定它的中序序列。

答案：D。

面试例题2：什么是平衡二叉树？[英国某著名计算机图形图像公司2005年面试题]

答案：平衡二叉树是一棵空树或具有下列特点的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。

13.7 排序

排序问题是各大IT公司必考的题目。

所谓排序，就是整理文件中的记录，使之按关键字递增（或递减）的顺序排列起来。其确切定义如下：

输入： n 个记录 R_1, R_2, \dots, R_n , 其相应的关键字分别为 K_1, K_2, \dots, K_n 。

输出： $R_{i1}, R_{i2}, \dots, R_{in}$, 使得 $K_{i1} \leq K_{i2} \leq \dots \leq K_{in}$ (或 $K_{i1} \geq K_{i2} \geq \dots \geq K_{in}$)。

1. 被排序对象——文件

被排序的对象——文件由一组记录组成。

记录则由若干个数据项（或域）组成。其中有一项可用来标识一个记录，称为关键字项。该数据项的值称为关键字（Key）。

2. 排序运算的依据——关键字

用来做排序运算依据的关键字，可以是数字类型，也可以是字符类型。关键字的选取应根据问题的要求而定。

在高考成绩统计中将每个考生作为一个记录。每条记录包含准考证号、姓名、各科的分数和总分数等项内容。若要唯一地标识一个考生的记录，则必须用“准考证号”作为关键字。若要按照考生的总分数排名次，则需用“总分数”作为关键字。

3. 排序的稳定性

当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。

在待排序的文件中，若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称这种排序方法是不稳定的。

注意：排序算法的稳定性是针对所有输入实例而言的。即在所有可能的输入实例中，只要有一个实例使得算法不满足稳定性要求，则该排序算法就是不稳定的。

4. 排序方法的分类

1) 按是否涉及数据的内、外存交换分

在排序过程中，若整个文件都是放在内存中处理，排序时不涉及数据的内、外存交换，则称为内部排序（简称内排序）。反之，若排序过程中要进行数据的内、外存交换，则称为外部排序。

注意：

- 内排序适用于记录个数不很多的小文件。
- 外排序则适用于记录个数太多，不能一次将其全部记录放入内存的大文件。

2) 按策略划分内部排序方法

可以分为5类：插入排序、选择排序、交换排序、归并排序和分配排序。

5. 排序算法的基本操作

大多数排序算法都有两个基本的操作：

- (1) 比较两个关键字的大小。
- (2) 改变指向记录的指针或移动记录本身。

注意：第二种基本操作的实现依赖于待排序记录的存储方式。

6. 待排文件的常用存储方式

1) 以顺序表（或直接用向量）作为存储结构

排序过程：对记录本身进行物理重排，即通过关键字之间的比较判定，将记录移到合适的位置。

2) 以链表作为存储结构

排序过程：无须移动记录，仅需修改指针。通常将这类排序称为链表（或链式）排序。

3) 用顺序的方式存储待排序的记录，但同时建立一个辅助表（如包括关键字和指向记录位置的指针组成的索引表）

排序过程：只需对辅助表的表目进行物理重排（即只移动辅助表的表目，而不移动记录本身）。适用于难于在链表上实现，且仍需避免排序过程中移动记录的排序方法。

7. 排序算法性能评价

1) 评价排序算法好坏的标准

评价排序算法好壞的标准主要有两条：

- 执行时间和所需的辅助空间
- 算法本身的复杂程度

2) 排序算法的空间复杂度

若排序算法所需的辅助空间并不依赖于问题的规模 n , 即辅助空间是 $O(1)$, 则称为就地排序 (In-PlaceSort)。

非就地排序一般要求的辅助空间为 $O(n)$ 。

3) 排序算法的时间开销

大多数排序算法的时间开销主要是关键字之间的比较和记录的移动。有的排序算法其执行时间不仅依赖于问题的规模, 还取决于输入实例中数据的状态。

面试例题 1: The following is an improved quick sort algorithm. Please fill in the blank. (下面的程序是一个快速排序问题, 请填空。) [美国某著名计算机嵌入式公司 2005 年面试题]

```
# include<iostream>
# include<stdio.h>
void improveqsort(int *list,int m,int n)
{
    int k,t,i,j;           /*
    for(i=0;i<10;i++)
        printf("%3d",list[i]);*/
    if(m<n)
    {
        i=m; j=n+1;k=list[m];
        while(i<j)
        {
            for(i=i+1;i<n;i++)
                if(list[i]>=k)
                    break;
            for(j=j-1;j>m;j--)
                if(list[j]<=k)
                    break;
            if(i<j)
                {t=list[i];list[i]=list[j];list[j]=t;}
        }
        t=list[m];list[m]=list[j];list[j]=t;
        improveqsort(      ,      ,      );
        improveqsort(      ,      ,      );
    }
}
main()
{
    int list[10];
    int n=9,m=0,i;
    printf(" input 10 number: ");
    for(i=0;i<10;i++)
        scanf("%d",&list[i]);
    printf("\n ");
    improveqsort(list,m,n);
    for(i=0;i<10;i++)
        printf("%5d",list[i]);
}
```

```

    printf("\n");
}

```

解析：数据结构的快速排序问题。

答案：improveqsort(list,m,j-1);
improveqsort(list,i,n);

扩展知识（快速排序的算法思想）

快速排序是 C.R.A.Hoare 于 1962 年提出的一种划分交换排序。它采用了一种分治的策略，通常称为分治法（Divide-and-Conquer Method）。

1. 分治法的基本思想

分治法的基本思想是：将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

2. 快速排序的基本思想

设当前待排序的无序区为 $R[\text{low}..\text{high}]$ ，利用分治法可将快速排序的基本思想描述为：

(1) 分解。

在 $R[\text{low}..\text{high}]$ 中任选一个记录作为基准 (Pivot)，以此基准将当前无序区划分为左、右两个较小的子区间 $R[\text{low}..\text{pivotpos}-1]$ 和 $R[\text{pivotpos}+1..\text{high}]$ ，并使左边子区间中所有记录的关键字均小于等于基准记录（不妨记为 pivot）的关键字 pivot.key，右边的子区间中所有记录的关键字均大于等于 pivot.key，而基准记录 pivot 则位于正确的位置 (pivotpos) 上，它无须参加后续的排序。

注意：

划分的关键是要求出基准记录所在的位置 pivotpos。划分的结果可以简单地表示为（注意 $\text{pivot}=R[\text{pivotpos}]$ ）：

```

 $R[\text{low}..\text{pivotpos}-1].\text{keys} \leq R[\text{pivotpos}].\text{key} \leq R[\text{pivotpos}+1..$ 
 $\text{high}].\text{keys}$ 
其中  $\text{low} \leq \text{pivotpos} \leq \text{high}$ 。

```

(2) 求解。

通过递归调用快速排序对左、右子区间 $R[low..pivotpos-1]$ 和 $R[pivotpos+1..high]$ 快速排序。

(3) 组合。

因为当“求解”步骤中的两个递归调用结束时，其左、右两个子区间已有序。对快速排序而言，“组合”步骤无须做什么，可看做是空操作。

3. 快速排序算法 QuickSort

代码如下：

```
void QuickSort(SeqList R, int low, int high)
{ // 对 R[low..high] 快速排序
    int pivotpos; // 划分后的基准记录的位置
    if (low < high) { // 仅当区间长度大于 1 时才须排序
        pivotpos = Partition(R, low, high);
        // 对 R[low..high] 做划分
        QuickSort(R, low, pivotpos - 1); // 对左区间递归排序
        QuickSort(R, pivotpos + 1, high); // 对右区间递归排序
    }
} // QuickSort
```

注意：

为排序整个文件，只须调用 $QuickSort(R, 1, n)$ 即可完成对 $R[1..n]$ 的排序。

4. 划分算法 Partition

1) 简单的划分方法

① 具体做法

第一步：(初始化) 设置两个指针 i 和 j ，它们的初值分别为区间的下界和上界，即 $i=low$, $j=high$ ；选取无序区的第一个记录 $R[i]$ (即 $R[low]$) 作为基准记录，并将它保存在变量 $pivot$ 中。

第二步：令 j 自 $high$ 起向左扫描，直到找到第 1 个关键字小于 $pivot.key$ 的记录 $R[j]$ ，将 $R[j]$ 移至 i 所指的位置上，这相当于 $R[j]$ 和基准 $R[i]$ (即 $pivot$) 进行了交换，使关键字小于基准关键字 $pivot.key$ 的记录移到了基准的左边，交换后 $R[j]$ 中是 $pivot$ ；然后，令 i 指针自 $i+1$ 位置开始向右扫描，直至找到第 1 个关键字大于 $pivot.key$ 的记录 $R[i]$ ，将 $R[i]$ 移到 i 所指的位置上，这相当于交换了 $R[i]$ 和基准 $R[j]$ ，使关键字大于基准关键字的记录移到了基准的右边，交换后 $R[i]$ 中又相当于存放了 $pivot$ ；接着令指针 j 自位置 $j-1$ 开始向左扫描，如此交

替改变扫描方向,从两端各自往中间靠拢,直至 $i=j$ 时,i便是基准 pivot 最终的位置,将 pivot 放在此位置上就完成了一次划分。

② 划分算法

代码如下:

```

int Partition(SeqList R, int I, int j)
    // 调用 Partition(R, low, high) 时, 对 R[low..high]
    // 做划分并返回基准记录的位置
    RecType pivot=R[i]; // 用区间的第 1 个记录作为基准
    while(i<j){ // 从区间两端交替向中间扫描, 直至 i=j 为止
        while(i<j&&R[j].key>=pivot.key)
            //pivot 相当于在位置 i 上
            j--; // 从右向左扫描, 查找第 1 个关键字小于 pivot.key
                  //的记录 R[j]
        if(i<j) // 表示找到的 R[j] 的关键字<pivot.key
            R[i++]=R[j]; // 相当于交换 R[i] 和 R[j],
                  //交换后 i 指针加 1
        while(i<j&&R[i].key<=pivot.key)
            //pivot 相当于在位置 j 上
            i++; // 从左向右扫描, 查找第 1 个关键字大于 pivot.
                  //key 的记录 R[i]
        if(i<j) // 表示找到了 R[i], 使 R[i].key>pivot.key
            R[j--]=R[i]; // 相当于交换 R[i] 和 R[j],
                  //交换后 j 指针减 1
    } //endwhile
    R[i]=pivot; // 基准记录已被最后定位
    return i;
} //partition

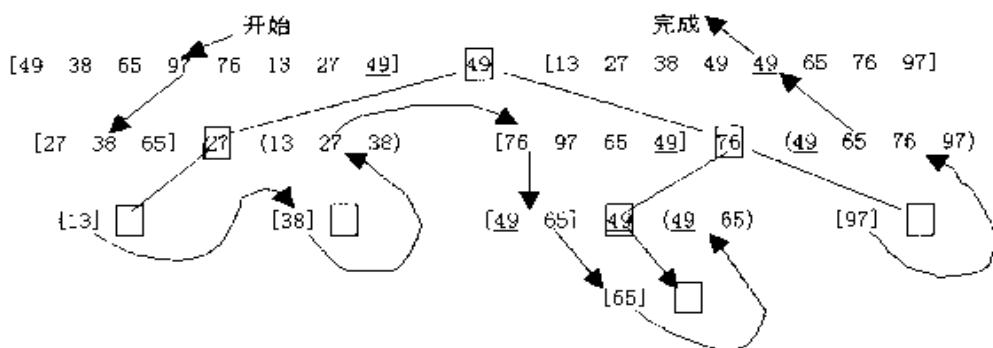
```

5. 快速排序执行过程

快速排序执行的全过程可用递归树来描述,如下图所示。

初始关键字:	[49] 38 65 97 76 13 27 49]
j 向左扫描	[49] 38 65 97 76 13 27 49]
第一次交换后	[27] 38 65 97 76 13 [] 49]
i 向右扫描	[27] 38 65 97 76 13 [] 49]
第二次交换后	[27] 38 [] 97 76 13 65 49]
j 向左扫描, 位置不变, 第三次交换后	[27] 38 13 97 76 [] 65 49]
i 向左扫描, 位置不变, 第四次交换后	[27] 38 13 [] 76 97 65 49]
j 向左扫描后划分过程结束	[27] 38 13 [49] 76 97 65 49]

一次划分过程



QuickSort 执行时的递归树

```

[49 38 65 97 76 13 27 49]           //初始关键字
(27 38 13) 49 [76 97 65 49]         //第一次划分完成后，对应递归树第 2 层
[13] 27 [38] 49 [49 65] 76 [97]      //对上一层个无序区划分完成后，对应递归树第 3 层
13 27 38 49 49 [65] 76 97          //对上一层个无序区划分完成后，对应递归树第 4 层
13 27 38 49 49 65 76 97          //最后的排序结果

```

对递归树的每层上个无序区划分之后的状态

6. 时间复杂度

快速排序法是一种不稳定的排序方法，平均时间复杂度 $O(n \times \lg n / \lg 2)$ ，最差情况时间复杂度为 $O(n^2)$ 。

面试例题 2：请用 C 或 C++写出一个冒泡排序程序，要求输入 10 个整数，输出排序结果。[中国某著名通信企业 H 面试题]

解析：交换排序中的冒泡排序问题。

答案：

程序如下：

```

# include<iostream>
# include<stdio.h>
void maopao(int *list)
{
    int i,j,temp;
    for (i=0;i<9;i++)
        for(j=0;j<9-i;j++)
        {
            if (list[j]>list[j+1])
                { temp=list[j];list[j]=list[j+1];list[j+1]=temp; }
        }
}
main()
{
    int list[10];
    int n=9,m=0,i;
    printf(" input 10 number: ");
    for(i=0;i<10;i++)

```

```

        scanf("%d", &list[i]);
        printf("\n ");
        maopao(list);
        for(i=0; i<10; i++)
            printf("%5d", list[i]);
        printf("\n");
    }
}

```

扩展知识（交换排序的算法思想）

交换排序的基本思想是：两两比较待排序记录的关键字，发现两个记录的次序相反时即进行交换，直到没有反序的记录为止。

应用交换排序基本思想的主要排序方法有：冒泡排序和快速排序。

1. 冒泡排序方法

将被排序的记录数组 $R[1..n]$ 垂直排列，每个记录 $R[i]$ 看做是重量为 $R[i].key$ 的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组 R 。凡扫描到违反本原则的轻气泡，就使其向上“飘浮”。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

(1) 初始。

$R[1..n]$ 为无序区。

(2) 第一趟扫描。

从无序区底部向上依次比较相邻的两个气泡的重量，若发现轻者在下、重者在上，则交换二者的位置。即依次比较 $(R[n], R[n-1])$ 、 $(R[n-1], R[n-2])$ 、 \dots 、 $(R[2], R[1])$ ；对于每对气泡 $(R[j+1], R[j])$ ，若 $R[j+1].key < R[j].key$ ，则交换 $R[j+1]$ 和 $R[j]$ 的内容。

第一趟扫描完毕时，“最轻”的气泡就飘浮到该区间的顶部，即关键字最小的记录被放在最高位置 $R[1]$ 上。

(3) 第二趟扫描。

扫描 $R[2..n]$ 。扫描完毕时，“次轻”的气泡飘浮到 $R[2]$ 的位置上……

最后，经过 $n-1$ 趟扫描可得到有序区 $R[1..n]$ 。

注意：

第 i 趟扫描时， $R[1..i-1]$ 和 $R[i..n]$ 分别为当前的有序区和无序区。

扫描仍是从无序区底部向上直至该区顶部。扫描完毕时，该区中最轻气泡飘浮到顶部位置 $R[i]$ 上，结果是 $R[1..i]$ 变为新的有序区。

2. 排序算法

1) 分析

因为每一趟排序都使有序区增加了一个气泡，在经过 $n - 1$ 趟排序之后，有序区中就有 $n - 1$ 个气泡，而无序区中气泡的重量总是大于等于有序区中气泡的重量，所以整个冒泡排序过程至多需要进行 $n - 1$ 趟排序。

若在某一趟排序中未发现气泡位置的交换，则说明待排序的无序区中所有气泡均满足轻者在上、重者在下的原则，因此，冒泡排序过程可在此趟排序后终止。为此，在下面给出的算法中，引入一个布尔量 `exchange`，在每趟排序开始前，先将其置为 FALSE。若排序过程中发生了交换，则将其置为 TRUE。各趟排序结束时检查 `exchange`，若未曾发生过交换则终止算法，不再进行下一趟排序。

2) 具体算法

代码如下：

```
void BubbleSort(SeqList R)
{
    //R(1..n)是待排序的文件，采用自下向上扫描，对R做冒泡排序
    int i, j;
    Boolean exchange;           //交换标志
    for(i=1; i<n; i++) {       //最多做n-1趟排序
        exchange=FALSE;         //本趟排序开始前，交换标志应为假
        for(j=n-1; j>=i; j--) //对当前无序区R[i..n]自下向上扫描
            if(R[j+1].key<R[j].key) { //交换记录
                R[0]=R[j+1];          //R[0]不是哨兵，仅做暂存单元
                R[j+1]=R[j];
                R[j]=R[0];
                exchange=TRUE;        //发生了交换，故将交换标志置为真
            }
        if(!exchange)             //本趟排序未发生交换，提前终止算法
            return;
    } //endfor(外循环)
} //BubbleSort
```

3. 算法分析

1) 算法的最好时间复杂度

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数 C 和记录移动次数 M 均达到最小值：

$$C_{\min}=n-1$$

$M_{\min} = 0$

冒泡排序最好的时间复杂度为 $O(n)$ 。

(2) 算法的最坏时间复杂度

若初始文件是反序的，需要进行 $n - 1$ 趟排序。每趟排序要进行 $n - i$ 次关键字的比较 ($1 \leq i \leq n - 1$)，且每次比较都必须移动记录 3 次来交换记录位置。在这种情况下，比较和移动次数均达到最大值：

$$C_{\max} = n(n - 1)/2 = O(n^2)$$

$$M_{\max} = 3n(n - 1)/2 = O(n^2)$$

冒泡排序的最坏时间复杂度为 $O(n^2)$ 。

3) 算法的平均时间复杂度为 $O(n^2)$

虽然冒泡排序不一定要进行 $n - 1$ 趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。

4) 算法稳定性

冒泡排序是就地排序，且它是稳定的。

面试例题 3：请用 C 或 C++写出一个 Shell 排序程序，要求输入 10 个整数，输出排序结果。[中国某著名通信企业 H 面试题]

解析：希尔排序 (Shell Sort) 是插入排序的一种，因 D.L.Shell 于 1959 年提出而得名。

答案：

完整代码如下：

```
# include<iostream>
# include<stdio.h>
void ShellSort(int* data, int left, int right)
{
    int len = right - left + 1;
    int d = len;
    while (d > 1)
    {
        d = (d + 1) / 2;
        for (int i = left; i < right + 1 - d; i++)
        {
            if (data[i + d] < data[i])
            {
                int tmp = data[i + d];
                data[i + d] = data[i];
                data[i] = tmp;
            }
        }
    }
}
```

```

        }
    }

void ShellSort2(int* data,int len)

{
    int d = len;
    while (d > 1)
    {
        d = (d + 1)/2;
        for (int i = 0; i < len - d;i++)
        {
            if (data[i + d] < data[i])
            {
                int tmp = data[i + d];
                data[i + d] = data[i];
                data[i] = tmp;
            }
        }

        for(int i=0;i<10;i++)
            printf("%5d",data[i]);
            printf("\n");
    }
}

main()
{
    int list[10];
    int n=9,m=0,i;
    printf(" input 10 number: ");
    for(i=0;i<10;i++)
        scanf("%d",&list[i]);
    printf("\n ");
    ShellSort2(list,10);
    //ShellSort(list,0,9);
    printf("\n");
    for(i=0;i<10;i++)
        printf("%5d",list[i]);
    printf("\n");
}

```

扩展知识（希尔（Shell）排序基本思想）

先取一个小于 n 的整数 d_1 作为第一个增量，把文件的全部记录分成 d_1 个组。所有距离为 d_1 的倍数的记录放在同一个组中。先在各组内进行直接插入排序，然后，取第二个增量 $d_2 < d_1$ 重复上述的分组和排序，直至所取的增量 $d_t=1$ ($d_t < d_{t-1} < \dots < d_2 < d_1$)，即

所有记录放在同一组中进行直接插入排序为止。

该方法实质上是一种分组插入方法。

给定实例的 Shell 排序的排序过程如下。

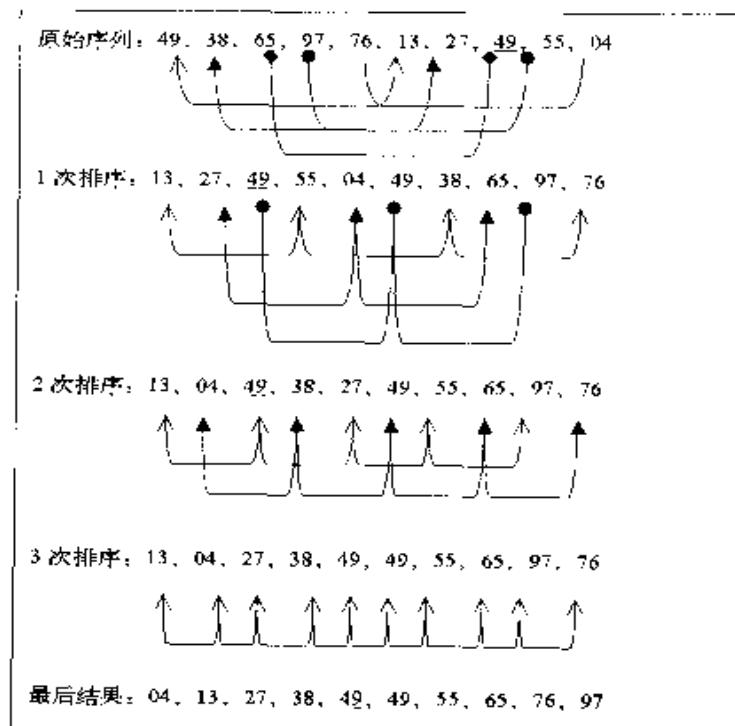
假设待排序文件有 10 个记录，其关键字分别是：

49, 38, 65, 97, 76, 13, 27, 49, 55, 04

增量序列的取值依次为：

5, 3, 2, 1

排序过程如下图所示。



Shell 排序的算法实现如下：

```

void ShellPass(SeqList R, int d)
    //希尔排序中的一趟排序, d 为当前增量
    for(i=d+1; i<=n; i++)    //将 R[d+1..n] 分别插入各组当前的
        //有序区
        if(R[i].key<R[i-d].key) {
            R[0]=R[i]; j=i-d;      //R[0]只是暂存单元, 不是哨兵
            do { //查找 R[i] 的插入位置
                R[j+d]=R[j];      //后移记录
                j=j-d;             //查找前一记录
            } while(j>0&&R[0].key<R[j].key);
            R[j+d]=R[0];          //插入 R[i] 到正确的位置上
        } //endif
    } //ShellPass

void ShellSort(SeqList R)

```

```

{
    int increment=n;           //增量初值，不妨设 n>0
    do {
        increment=increment/3+1; //求下一增量
        ShellPass(R, increment);
        //一趟增量为 increment 的 Shell 插入排序
    }while(increment>1)
} //ShellSort

```

注意：当增量 $d=1$ 时，ShellPass 和 InsertSort 基本一致，只是由于没有哨兵而在内循环中增加了一个循环判定条件 “ $j>0$ ”，以防下标越界。

算法分析：

1. 增量序列的选择

Shell 排序的执行时间依赖于增量序列。

好的增量序列的共同特征如下：

- 最后一个增量必须为 1。
- 应该尽量避免序列中的值（尤其是相邻的值）互为倍数的情况。

有人通过大量的实验，给出了目前较好的结果：当 n 较大时，比较和移动的次数约在 $n^{1.25} \sim 1.6n^{1.25}$ 之间。

2. Shell 排序的时间性能优于直接插入排序

希尔排序的时间性能优于直接插入排序的原因如下：

- 当文件初态基本有序时，直接插入排序所需的比较和移动次数均较少。
- 当 n 值较小时， n 和 n^2 的差别也较小，即直接插入排序的最好时间复杂度 $O(n)$ 和最坏时间复杂度 $O(n^2)$ 差别不大。

在希尔排序开始时增量较大，分组较多，每组的记录数目少，故各组内直接插入较快。后来增量 d_i 逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多。但由于已经按 d_i-1 作为距离排过序，使文件较接近于有序状态，所以新的一趟排序过程也较快。

因此，希尔排序在效率上较直接插入排序有较大的改进。

3. 稳定性

希尔排序是不稳定的。参见上述实例，该例中两个相同关键字 49 在排序前后的相对次序发生了变化。

面试例题4：以下哪种排序属于稳定排序？[美国某著名分析软件公司2005年面试题]

- A. 归并排序 B. 快速排序 C. 希尔排序 D. 堆排序

解析：只有归并排序是稳定排序，其他3个都是不稳定的。

答案：D。

扩展知识（各种排序方法比较）

按平均时间将排序分为以下4类。

- 平方阶($O(n^2)$)排序 一般称为简单排序，例如直接插入、直接选择和冒泡排序。
- 线性对数阶($O(nlgn)$)排序 如快速、堆和归并排序。
- $O(n^{1+\varepsilon})$ 阶排序 ε 是介于0和1之间的常数，即 $0 < \varepsilon < 1$ ，如希尔排序。
- 线性阶($O(n)$)排序 如桶、箱和基数排序。

简单排序中直接插入排序最好，快速排序最快。当文件为正序时，直接插入排序和冒泡排序均最佳。

1. 影响排序效果的因素

因为不同的排序方法适应不同的应用环境和要求，所以选择合适的排序方法应综合考虑下列因素：

- 待排序的记录数目 n 。
- 记录的大小（规模）。
- 关键字的结构及其初始状态。
- 对稳定性的要求。
- 语言工具的条件。
- 存储结构。
- 时间和辅助空间复杂度等。

2. 不同条件下排序方法的选择

(1) 若 n 较小(如 $n \leq 50$)，可采用直接插入或直接选择排序。

当记录规模较小时，直接插入排序较好。否则因为直接选择移动的记录数少于直接插入，应选直接选择排序为宜。

(2) 若文件初始状态基本有序(指正序), 则应选用直接插入排序、冒泡排序或随机的快速排序为宜。

(3) 若 n 较大, 则应采用时间复杂度为 $O(nlgn)$ 的排序方法: 快速排序、堆排序或归并排序。

快速排序被认为是目前基于比较的内部排序中最好的方法。当待排序的关键字随机分布时, 快速排序的平均时间最短。

堆排序所需的辅助空间少于快速排序, 并且不会出现快速排序可能出现的最坏情况。这两种排序都是不稳定的。

若要求排序稳定, 则可选用归并排序。但本章介绍的从单个记录起进行两两归并的排序算法并不值得提倡, 通常可以将它和直接插入排序结合在一起使用。先利用直接插入排序求得较长的有序子文件, 然后再两两归并之。因为直接插入排序是稳定的, 所以改进后的归并排序仍是稳定的。

面试例题 5: 用二分法查找一个长度为 10 的、排好序的线性表, 查找不成功时, 最多需要比较多少次? [美国某著名 CPU 生产公司面试题]

- A. 5 B. 2 C. 4 D1

解析: 二分法查找问题。折半查找为 10 的线性表, 4 次可以得出结果。

答案: C。

面试例题 6: 下面哪种排序法对 1234576 最快? [美国某著名 CPU 生产公司面试题]

- A. quick sort B. bubble sort C. merge sort

解析: 排序比较问题。对于已经接近排好序的情况, 冒泡法是比较快的。

答案: B。

面试例题 7: 何谓哈希表? [英国某著名计算机图形图像公司面试题]

答案: 在各种结构(线性表、树等)中, 记录在结构中的相对位置是随机的, 和记录的关键字之间不存在确定的关系, 因此, 在结构中查找记录时需进行一系列和关键字的比较。这一类查找方法建立在“比较”

的基础上，查找的效率依赖于查找过程中所进行的比较次数。

理想的情况是希望不经过任何比较，一次存取便能得到所查记录，那就必须在记录的存储位置和它的关键字之间建立一个确定的对应关系 f 。在查找时，只要根据这个对应关系 f 找到给定值 K 的像 $f(K)$ 。若结构中存在关键字和 K 相等的记录，则必定在 $f(K)$ 的存储位置上，由此，不需要进行比较便可直接取得所查记录。在此，我们称这个对应关系 f 为哈希（Hash）函数，按这个思想建立的表为哈希表。

假如有 100 个数组，有 2 005 个值要放在这 100 个数组里面。

我们可以假设一种规则，比如 $\text{mod}(x, 100)$ ，这样：

0：就放在编号为 0 的数组里面

1：就放在编号为 1 的数组里面

.....

这样，当我们知道这个值的时候就知道这个值存储在哪个数组里面。我们可以认为 $\text{mod}(x, 100)$ 就是一个 Hash 算法。也就是根据值能以某种规则（算法）而知道存储它的地方。

面试例题8：Map 中的数据存储方式是什么？Map 和 HashMap 有什么区别？[英国某著名计算机图形图像公司面试题]

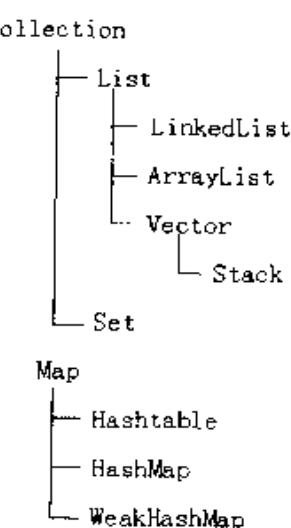
解析：HashMap 是一个类，它的构造方式是：

```
HashMap hm = new HashMap();
```

HashMap 类隶属于 Map 接口，如同 List 类隶属于 Collection 接口，如右图所示。

List 类继承于 Collection 接口，使用此接口能够精确地控制每个元素插入的位置。用户能够使用索引（元素在 List 中的位置，类似于数组下标）来访问 List 中的元素，这类似于数组。

除了具有 Collection 接口必备的 iterator() 方法外，List 还提供一个 listIterator() 方法，返回一个 ListIterator 接口。和标准的 Iterator 接口相比，ListIterator 多了一些 add() 之类的方法，允许添加、删除、设定元素，还能向前或向后遍历。



实现 List 接口的常用类有 LinkedList、ArrayList、Vector 和 Stack。

Map 提供 key 到 value 的映射。一个 Map 中不能包含相同的 key，每个 key 只能映射一个 value。Map 接口提供 3 种集合的视图，Map 的内容可以被当做一组 key 集合、一组 value 集合，或者一组 key-value 映射。

HashMap 和 Hashtable 类似，不同之处在于 HashMap 是非同步的，并且允许 null，即 null value 和 null key。但是将 HashMap 视为 Collection 时 (values()方法可返回 Collection)，其迭代子操作时间开销和 HashMap 的容量成比例。因此，如果迭代操作的性能相当重要的话，不要将 Hashmap 的初始化容量设得过高，或者 load factor 过低。

答案：Map 中的数据存储方式是 Hashtable。Map 和 HashMap 的区别是接口和类的区别，List 和 Map 是接口，ArrayList 和 HashMap 分别是它们的实现类。

HashMap 是 Hashtable 的改进版，主要是以 (key,value) 来存储对象。

面试例题 9：解释一下什么是哈夫曼编码问题？[英国某著名计算机图形图像公司面试题]

答案：Huffman 于 1952 年提出了这种方法，开始主要用于电报报文的编码，常用的英文字母应该如何编码，不常用的应该如何编码，这样编下来使报文最短。我们下面举一个例子。

可以想象几个字母 A、B、C、D，它们在某一文章中的使用频率为 7 次、5 次、1 次，等等。选择使用率小的两个点 1 和 3 构成新点 4。选择 5 和 4（也是两个最小的，注意不是 1 和 3，因为 1 和 3 现在已经归在 4 里面了）进行合并。这时最小两个点已经变为 7 和 6 了，这时合并它们两个生成新点 13。只剩两个点了，不管它们的值是多少，它们也是最小的了，则合并。

哈夫曼编码总是先把小的合并，因为先合并的会在最下面，所以编码长度最短。

第 14 章

字 符 串

基本上求职者进行笔试时没有不考字符串的。字符串也是一种相对简单的数据结构，容易多次引起面试官反复发问。我曾不止一次在面试时被考官要求当场写出 strcpy 函数的表达方式。事实上，字符串也是一个考验程序员编程规范和编程习惯的重要考点。不要忽视这些细节，因为这些细节会体现你在操作系统、软件工程、边界内存处理等方面的知识掌控能力，也会成为企业是否录用你的参考因素。

14.1 整数字符串转化

面试例题 1：怎样将整数转化成字符串数，并且不用函数 itoa。

解析：整数转化成字符串，可以采用加'0'，再逆序的办法，整数加'0'就会隐性转化成 char 类型的数。

答案：

程序代码如下：

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

int main(void)
{
    int num = 12345, j=0, i=0;
    char temp[7], str[7];

    //itoa(number, string, 10);
    while(num)
    {
```

```
    temp[i]=num%10+'0';
    i++;
    num=num/10;
}
temp[i]=0;
printf(" temp = %s\n", temp);
i=i-1;
printf(" temp = %d\n", i);
//刚刚转化的字符串是逆序的 必须把它反过来
while(i>=0)
{
    str[j]=temp[i];
    j++;
    i--;
}
str[j]=0;
printf(" string = %s\n", str);
return 0;
}
```

扩展知识

如果可以使用 itoa 函数的话，则十分简单，答案如下：

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

int main(void)
{
    int number = 12345;
    char string[7];

    itoa(number, string, 10);
    printf("integer = %d string = %c\n", number, string[1]);
    return 0;
}
```

面试例题 2：编程实现字符串数转化成整数的办法。[中国某著名 IT 培训企业公司 2005 年面试题]

解析：字符串转化成整数，可以采用减'0'再乘 10 累加的办法，字符串减'0'就会隐性转化成 int 类型的数。

答案：

程序代码如下：

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

int main(void)
{
    int num = 12345, j=0, i=0, sum=0;
    char temp[7]={'1','2','3','4','5','\0'}, str[7];
```

```

while(temp[i])
{
    sum= sum*10+ (temp[i]-'0');
    i++;
}
printf(" sum = %d\n", sum);
return 0;
}

```

14.2 字符数组和 strcpy

面试例题 1: Write a function about string copy, the strcpy prototype is "char* strcpy(char* strDest, const char* strSrc);". Here strDest is destination string, strSrc is source string. (已知 strcpy 函数的原型是 char *strcpy(char *strDest, const char *strSrc);, 其中 strDest 是目的字符串, strSrc 是源字符串。)

(1) Write the function strcpy, don't call C/C++ string library. (不调用 C++/C 的字符串库函数, 请编写函数 strcpy。)

(2) Here strcpy can copy strSrc to strDest, but why we use char* as the return value of strcpy? (strcpy 能把 strSrc 的内容复制到 strDest, 为什么还要 char * 类型的返回值?) [中国台湾某著名 CPU 生产公司 2005 年面试题]

解析: 字符串拷贝函数问题。

答案:

(1) 代码如下:

```

char *strcpy(char *strDest, const char *strSrc);
{
    assert((strDest!=NULL) && (strSrc !=NULL));
    char *address = strDest;
    while( (*strDest++ = * strSrc++) != '\0' )
        NULL ;
    return address ;
}

```

(2) 为了实现链式表达式, 返回具体值。

例如:

```
int length = strlen( strcpy( strDest, "hello world" ) );
```

面试例题 2: 下面的程序会出现何种问题? [美国某著名计算机软件公司面试题]

```

#include <iostream>
#include <stdio.h>
int main(void)
{

```

```

char s[]="123456789";
char d[]="123";
strcpy(d,s);
printf("%s,\n%s",d,s);
return 0;
}

```

解析：以上程序输出结果是：123456789,56789。

没经验的程序员一定会在此大跌眼镜的，源字串竟然被截掉了一部分（截掉的长度恰是目标字串原来的长度。至于原因，应该是当初分配的内存地址是连续内存的问题，原来是1234\0123456789\0，strcpy后变成了123456789\06789\0）！所以在分配空间的时候要给源字符串和目标字符串留足够的空间。

把目标字串定义在前，源字串定义在后，虽然可以看到正确的输出结果：123456789,123456789。但会产生一个运行期错误，原因估计是越过了目标字串的实际空间，访问到了不可预知的地址了。

微软在这里是写得非常简单的，代码如下：

```

char * cdecl strcpy(char * dst, const char * src)
{
    char * cp = dst;
    while( *cp++ = *src++ )
        ; /* Copy src over dst */
    return( dst );
}

```

微软为什么这么写？它这样安全漏洞太多了，所以必须预先为目标字串分配足够的空间，并且使用这个函数的时候得小心翼翼才行。

为了提高性能，减去那些罗嗦的安全检查是必要的。况且程序员在使用时应该知道哪些条件下会发生访问违例，这种做法就是把责任推给了程序员，让他来决定安全与性能的取舍。

答案：123456789,56789。

拷贝函数的一个完整的标准写法如下：

```

#include<stdio.h>
#include<malloc.h>
#include<assert.h>
#include<string.h>
void stringcopy(char *to,const char *form)
{
    assert(to!=NULL && form!=NULL);
    while(*form]!='\0')
    {
        *to++=*form++;
    }
    *to='\0';
}

```

```

int main(void)
{
    char *f;
    char *t;
    f=(char *)malloc(15);
    t=(char *)malloc(15);
    strcpy(f,"asdfghjkl");
    strcpy(t,f);
    printf("%s\n",f);
    printf("%s\n",t);
    return 0;
}

```

扩展知识（数组大小分配）

在使用数组的时候，总有一个问题困扰着我们：数组应该有多大？

在很多的情况下，你不能确定要使用多大的数组。你可能并不知道该班级的学生的人数，那么你就要把数组定义得足够大。这样，你的程序在运行时就申请了固定大小的、你认为足够大的内存空间。即使你知道该班级的学生数，但是如果因为某种特殊原因人数有增加或者减少，你又必须重新修改程序，扩大数组的存储范围。这种分配固定大小的内存分配方法称为静态内存分配。但是这种内存分配的方法存在比较严重的缺陷，特别是处理某些问题时：在大多数情况下会浪费大量的内存空间；在少数情况下，当你定义的数组不够大时，可能引起下标越界错误，甚至导致严重后果。

那么有没有其他的方法来解决这样的问题呢？有，那就是动态内存分配。

所谓动态内存分配就是指在程序执行的过程中动态地分配或者回收存储空间的内存分配方法。动态内存分配不像数组等静态内存分配方法那样需要预先分配存储空间，而是由系统根据程序的需要即时分配，且分配的大小就是程序要求的大小。从以上动、静态内存分配比较可以知道动态内存分配相对于静态内存分配的特点：

- 不需要预先分配存储空间。
- 分配的空间可以根据程序的需要扩大或缩小。

1. 如何实现动态内存分配及其管理

要实现根据程序的需要动态分配存储空间，就必须用到以下

几个函数

1) malloc 函数

malloc 函数的原型为：

```
void *malloc (unsigned int size)
```

其作用是在内存的动态存储区中分配一个长度为 size 的连续空间。其参数是一个无符号整型数，返回值是一个指向所分配的连续存储域的起始地址的指针。还有一点必须注意的是，若函数未能成功分配存储空间（如内存不足）就会返回一个 NULL 指针，所以在调用该函数时应该检测返回值是否为 NULL 并执行相应的操作。

下例是一个动态分配的程序：

```
main()
{
    int count, *array;

    /*count 是一个计数器，array 是一个整型指针，也可以理解为
    指向一个整型数组的首地址*/
    if((array(int *) malloc(10*sizeof(int)))==NULL)
    {
        printf("不能成功分配存储空间。");
        exit(1);
    }
    for (count=0;count < 10;count++) /*给数组赋值*/
        array[count]=count;
    for(count=0;count < 10;count++) /*打印数组元素*/
        printf("%2d",array[count]);
}
```

上例中动态分配了 10 个整型存储区域，然后进行赋值并打印。例中 if((array(int *) malloc(10*sizeof(int)))==NULL)语句可以分为以下几步：

- (1) 分配 10 个整型的连续存储空间，并返回一个指向其起始地址的整型指针。
- (2) 把此整型指针地址赋给 array。
- (3) 检测返回值是否为 NULL。

2) free 函数

由于内存区域总是有限的，不能无限制地分配下去，而且一个程序要尽量节省资源，所以当所分配的内存区域不用时，就要释放它，以便其他的变量或者程序使用。这时我们就要用到 free

函数。其函数原型是：

```
void free(void *p)
```

作用是释放指针 p 所指向的内存区域。

其参数 p 必须是先前调用 malloc 函数或 calloc 函数（另一个动态分配存储区域的函数）时返回的指针。给 free 函数传递其他的值很可能造成死机或其他灾难性的后果。

注意：这里重要的是指针的值，而不是用来申请动态内存的指针本身。例如：

```
int *p1,*p2;
p1=malloc(10*sizeof(int));
p2=p1;
.....
free(p2) /*或者 free(p1) */
```

malloc 返回值赋给 p1，又把 p1 的值赋给 p2，所以此时 p1、p2 都可作为 free 函数的参数。malloc 函数对存储区域进行分配。free 函数释放已经不用的内存区域。所以有这两个函数就可以实现对内存区域进行动态分配并进行简单的管理了。

面试例题 3：串拷贝 (strcpy) 和内存拷贝 (memcpy) 有什么不同？它们适合于在哪些情况下使用？

答案： strcpy() 函数只能拷贝字符串。strcpy() 函数将源字符串的每个字节拷贝到目的字符串中。当遇到字符串末尾的 NULL 字符 (\0) 时，它会删去该字符，并结束拷贝。

memcpy() 函数可以拷贝任意类型的数据。因为并不是所有的数据都以 NULL 字符结束，所以要为 memcpy() 函数指定要拷贝的字节数。

在拷贝字符串时，通常使用 strcpy() 函数；在拷贝其他数据（如结构）时，通常使用 memcpy() 函数。

面试例题 4：请写出一个函数来模拟 memcpy() 函数。

答案：

标准代码如下：

```
#include <iostream>
#include <memory.h>
#include<assert.h>
```

```

using namespace std;
//自己编写的 memcpy 算法
void *memcpy2( char *pvTo, char *pvFrom, size_t size)
{
    assert((pvTo != NULL) && (pvFrom != NULL)); // 使用断言
    char *pbTo = pvTo; // 防止改变 pvTo 的地址
    // cout << "ppp " << pbTo << endl;
    char *pbFrom = pvFrom; // 防止改变 pvFrom 的地址
    while(size -- > 0 )
        *pbTo ++ = *pbFrom ++ ;
    return pvTo;
}
int main(int argc,char *argv[])
{
    char Str1[] = "One Two Three";
    char Str2[81];
    char ptr1[] = "One Two Three";
    char ptr2[81];
    memcpy(Str2,Str1,7);
    memcpy2(ptr2,ptr1,7);

    // Str2[9] = 0;
    cout << "Source string:" << endl;
    cout << " " << Str1 << endl;
    cout << "Destination string:" << endl;
    cout << " " << Str2 << endl;

    cout << "Source string:" << endl;
    cout << " " << ptr1 << endl;
    cout << "Destination string:" << endl;
    cout << " " << ptr2 << endl;

    return 0;
}

```

14.3 数组越界

面试例题 1：下面的程序有何缺点？[中国台湾某著名 CPU 生产公司面试题]

```

#define MAX 255
int main()
{
    unsigned char A[MAX],i;
    for(i=0;i<=MAX;i++) {
        A[i]=i;
    }
    return 0;
}

```

解析：典型的数组越界问题。for 循环执行了 256 次，而数组 A 只有 255 个元素。

答案：

将循环控制条件 $i \leq MAX$ 改为 $i < MAX$ 即可。

面试例题 2: Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```

void test1() {
    char string[10];
    char* str1="0123456789";
    strcpy(string,str1);
    std::cout<<string<<'\n';
}

void test2() {
    char string[10],str1[10];
    for(int i=0;i<10;i++) {
        str1[i]='a';
    }
    strcpy(string,str1);
    std::cout<<string<<'\n';
}

void test3(char* str1) {
    char string[10];
    if(strlen(str1)<=10) {
        strcpy(string,str1);
    }
    std::cout<<string<<'\n';
}

```

解析：字符数组和 strcpy 问题。

字符数组并不要求最后一个字符为'0'。是否需要加入'0'，完全由系统需要决定。但是字符数组的初始化要求最后一个字符必须为'0'，所以 test2 虽然能够编译通过，但是会出现运行时错误。类似于 char c[5]={'C','h','i','n','a'} 这样的定义是错误的。

答案：

可以编译通过的程序如下所示：

```

#include <iostream>

void test1() {
    char string[10];
    char* str1="0123456789";
    strcpy(string,str1);
    std::cout<<string<<'\n';
}

void test2() {
    char string[10],str1[10];
    for(int i=0;i<9;i++) { // 错误 1
        str1[i]='a';
    }
    str1[9]='\0';
    strcpy(string,str1);
    std::cout<<string<<'\n';
}

```

```

    }

    void test3(char* str1) {
        char string[10];
        if(strlen(str1)<=10) {
            strcpy(string,str1);
        }
        std::cout<<string<<'\n';
    }

    int main()
    {
        test1();
        test2();
        char* str="0123456789";
        test3(str);
        return 0;
    }
}

```

面试例题 3: What is the output of these statements? (下面语句的输出是什么?) [中国台湾某著名杀毒软件公司 2005 年面试题]

```

char a=256;
int d=a;
printf("%d",d+1);

```

- A. -1 B. 1 C. 257 D. 0

解析: char 数值溢出问题。char 类型的变量赋值范围是 0~255。当把 256 赋值给 a 后, 超出了 a 的有效取值范围, 此时 a 的实际值为 0。

答案: B。

14.4 数字流和数组声明

面试例题 1: Which is not the standard I/O channel? (下面哪一个不是标准输入输出通道?) [中国某著名综合软件公司 2005 年面试题]

- A. std::cin B. std::cout C. std::cerr D. stream

解析: I/O stream 问题。头文件 iostream 中含有 cin、cout、cerr 几个对象, 对应于标准输入流、标准输出流和标准错误流。

答案: D。

面试例题 2: Which definition is correct? (下面哪个数组的声明是正确的?) [中国台湾某著名杀毒软件公司 2005 年 9 月面试题]

- A. int a[];
 C. int a[10+1]={0};
 B. int n=10,a[n];
 D. int a[3]={1,2,3,4};

解析：数组定义问题。

int a[]是错误的，不允许建立空数组。

int n=10,a[n];是可以的。

int a[10+1]={0};也是允许的。

int a[3]={1,2,3,4};会造成越界问题，因此不允许。

答案：B, C。

14.5 字符串其他问题

面试例题 1：输入一个数，测试其是否是回文数。[中国某著名 IT 培训企业
公司 2005 年 10 月面试题]

解析：建立数组，按位存储。比较首位和末位是否相同。如不同，则不是回文数。相同则继续比较，首位递增，末位递减，直到首位不再小于末位。

答案：

完整代码如下：

```
#include <iostream>
using namespace std;

int main()
{
    int j=10, k=12321,p,a[10],ss,i=0,begin,end;
    cout << "please input" << endl;
    cin >> k;
    p=k;
    while(p)
    {
        ss=p%10;
        a[i]=ss;
        p=p/10;
        i++;
    }
    begin = 0;
    end = i-1;

    while(begin < end)
    {
        if(a[begin]!=a[end])
            break;
        else
```

```
    { begin++; end--; }
}
if(begin < end)
    (cout << "jiade" << endl; )
else
    (cout << "zhende " << endl; )
cout << "i " << i << endl;
cout << k;
return 0;
}
```

面试例题 2：将字符串“askdaskaskdaskg”删除指定字符 ask，删除后的结果是“ddg”。

解析：删除的最好办法就是把不删除的东西提取出来。

答案：

完整代码如下：

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main()
{
    char uu[20], *p, *sub = "ask", *str = "askdaskaskdaskg", *str2;
    int n=0, i=0, v;

    p=sub;
    str2=str;
    while(*str2)
    {
        while(*p)
        {
            if(*p==*str2)
                break;
            p++;
        }
        if(*p=='\0')
        {
            uu[i]=*str2;
            i++;
        }
        str2++;
        p=sub;
    }
    uu[i]=0;
    str=uu;
    printf("\n %s", str);
    return 0;
}
```

面试例题 3：Please implement the function strstr() (Find a substring, returns a

pointer to the first occurrence of strCharSet in string), DO NOT use any C run-time functions. const char* strstr(const char* string, const char* strCharSet); (请写一个函数来模拟C++中的strstr()函数：该函数的返回值是主串中字符串的位置以后的所有字符。请不要使用任何C程序已有的函数来完成。) [中国台湾某著名杀毒软件公司2005年面试题]

解析：string字符串问题。做一个程序模拟C++中的strstr()函数。strstr()函数是把主串中子串及以后的字符全部返回。比如主串是“12345678”，子串是“234”，那么函数的返回值就是“2345678”。

答案：

正确程序如下：

```
#include <iostream>
using namespace std;

const char* strstr1(const char* string, const char* strCharSet)
{
    for(int i=0;string[i]!='\0';i++)
    {
        int j=0;
        int temp=i;
        if(string[i]==strCharSet[j])
        {
            while(string[i]==strCharSet[j])
            {
                if((strCharSet[j]=='\0'))
                    return &string[i-j];
                }
                i=temp;
            }
        }
        return NULL;
    }

int main()
{
    char* string="12345554555123";
    cout<<string<<endl;
    char strCharSet[10]={};
    cin>>strCharSet;
    cout<<strstr1(string,strCharSet)<<endl;
    // char* string2=strstr("12345554555123","234");
    // cout<<string2<<endl;
    return 0;
}
```

面试例题4：将一句话里的单词进行倒置，标点符号不倒换。比如一句话“i come from tianjin.”倒换后变成“tianjin. from come i”。

解析：

解决该问题可以分为两步：第一步全盘置换将该句变成“.nijnait morf emoc i”，第二步进行部分翻转，如果不是空格，则开始翻转单词。

答案：

具体代码如下：

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

int main(void)
{
    int num = -12345, j=0, i=0, flag=0, begin, end;
    char str[]="i come from tianjin.", temp;
    j=strlen(str)-1;

    printf(" string = %s\n", str);
    //第一步是进行全盘翻转 将单词变成“.nijnait morf emoc i”
    while(j>i)
    {
        //str[j]=temp[i];
        temp=str[i];
        str[i]=str[j];
        str[j]=temp;
        j--;
        i++;
    }
    printf(" string = %s\n", str);
    i=0;
    //第二步进行部分翻转 如果不是空格 则开始翻转单词
    while(str[i])
    {
        //str[j]=temp[i];
        if(str[i]!=' ')
        {
            begin = i;
            while(str[i]&&str[i]!=' ')
                i++;
            i=i-1;
            end=i;
            end++;
        }

        while(end>begin)
        {
            //str[j]=temp[i];
            temp=str[begin];
            str[begin]=str[end];
            str[end]=temp;
            end--;
            begin++;
        }

        i++;
    }
    printf(" string = %s\n", str);
    return 0;
}
```

面试例题5：有100个整数，其中有负数，找出连续3个数之和最大的部分。[中国某著名计算机金融软件公司2005年面试题]

答案：

完整代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int a[10]={-3, 4, 6, 8, -9, 7, 10, -6, 20, -9};
    int sum = a[0]+a[1]+a[2]-1, index, i;
    for (i=0; i<=7; i++)
    {
        if (sum<a[i]+a[i+1]+a[i+2])
        {
            sum=a[i]+a[i+1]+a[i+2];
            index=i+1;
        }
    }
    cout << "\n this is at the " << index << " 3 number the
    sum is largest";
    return 0;
}
```

面试例题6：有一个数组a[n]，里面的数只有两种：-1或1。i、j是两个整数，假设 $0 \leq i \leq j \leq n-1$ ，找出a[i]到a[j]中连续数之和最大的部分（如果大部分存在相等的，则优先找最短的）。[英国某著名计算机图形图像公司2005年面试题]

解析：从a[i]一直要累加到a[j]。有3个地方要注意：

(1) 既然数组a[n]里面的数只有两种：-1或1，则绝不可以以-1为起始点。因为如果 $a[i] == -1$ ，以a[i]为起始点的累积结果肯定要小于以a[i+1]为起始点的累积结果。

(2) 在不是-1的基础上，寻找起始点。寻找的基础是数组从前往后累加，以前数的累积之和一定要小于等于0，则设为起始点，否则起始点不改变。

(3) 在不是-1的基础上，寻找终止点。寻找的基础是数组从后往前累加，以前数的累积之和一定要小于等于0，则设为终止点，否则终止点不改变。

答案：

完整代码如下：

```

#include <iostream>
using namespace std;
int main()
{
    int a[10]={1,-1,-1,1,1,-1,1,1,1,-1}; //总数是从0到9
    //寻找起始点

    int begin,i=0, sum =0;
    while(i<=9)
    {
        sum = sum + a[i];
        if (a[i]==-1)
            i=i+1;
        else
        {
            if(sum<=0)
                {begin=i;
                i=i+1;}
            else
                i=i+1;
        }
        cout << " this sum is " << sum << endl;
    }
    cout << " this begin is " << begin << endl;
    //寻找终止点

    int end;
    i=9; int sum2 =0;
    while(i>=begin)
    {
        sum2 = sum2 + a[i];
        if (a[i]==-1)
            i=i-1;
        else
        {
            if(sum2<=0)
                {end=i;
                i=i-1;}
            else
                i=i-1;
        }
        cout << " this sum2 is " << sum2 << endl;
    }

    cout << " this end is " << end << endl;
    return 0;
}

```

第 15 章

设计模式

鲁迅先生曾经说过：“地上本没有路，走的人多了也就成了路。”设计模式如同此理，它是经验的传承，并非体系。它是被前人发现，经过总结形成的一套某一类问题的一般性解决方案，而不是被设计出来的定性规则。它不像算法那样可以照搬照用。

设计模式关注的重点在于通过经验提取的“准则或指导方案”在设计中的应用，因此在不同层面考虑问题的时候就形成了不同问题领域的模式。模式的目标是，把共通问题中的不变部分和变化部分分离出来。不变的部分，就构成了模式。因此，模式是一个经验提取的“准则”，并且在一次一次的实践中得到验证。在不同的层次有不同的模式，小到语言实现，大到架构。在不同的层面上，模式提供不同层面的指导。

和建筑结构一样，软件中亦有诸多的“内力”。和建筑设计一样，软件设计也应该努力疏解系统中的内力，使系统趋于稳定、有生气。一切软件设计都应该由此出发。任何系统都需要有变化，任何系统都会走向死亡。作为设计者，应该拥抱变化，利用变化，而不是逃避变化。

经典设计模式一共有 23 种，面试时重要的不是你熟记了多少个模式的名称，关键还在于付诸实践的运用。为了有效地设计而去熟悉某种模式所花费的代价是值得的，因为很快你会在设计中发现这种模式真的很好，很多时候它令你的设计更加简单了。

其实公司需要一个真正出色的设计师，懂得判断运用模式的时机。很多才踏入软件设计领域的人员，往往对设计模式很困惑。对于他们来说，由于没有项目的实际经验，OO 的思想也还未曾建立，设计模式未免

过于高深了。其实，即使是非常有经验的程序员，也不敢夸口对各种模式都能合理应用。

面试题目中关于设计模式的题目主要以 C# 或 Java 的形式出现，本书将结合各大公司（尤其是外企）C#、ASP.NET 等设计模式题目的详细分析，帮助读者应对关于设计模式方面的面试考量。

15.1 设计模式

面试例题 1：Which came first, chicken or the egg?（请用设计模式观点描述先有鸡还是先有蛋？）[德国某著名软件咨询企业 2005 年 10 月面试题]

解析：

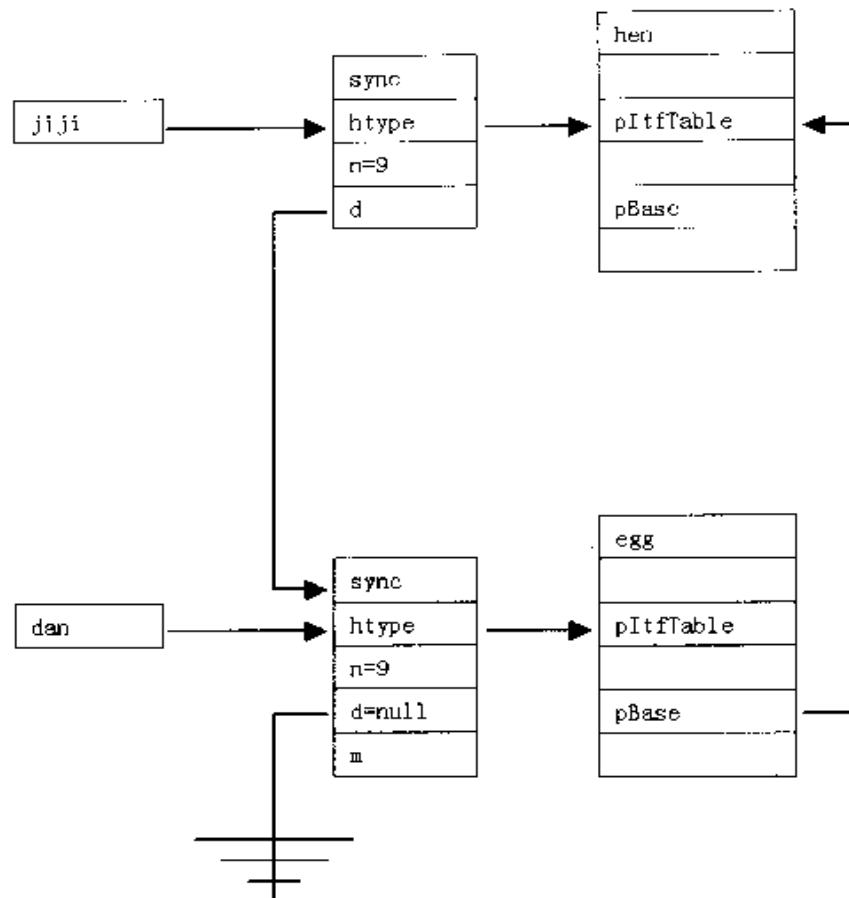
请先看下面的程序：

```
using System;
class Client
{
    public static void Main ()
    {
        hen jiji = new hen();
        egg dan = new egg();
        jiji.d = dan;
        Console.WriteLine(jiji.d.m);
    }
}
class hen
{
    public int n = 9;
    public egg d;
}
class egg : hen
{
    public int m = 10;
}
```

egg 继承自 hen，可以说没有 hen 就没有 egg，可 hen 里面有一个成员是 egg 类型。到底是先有鸡还是先有蛋？这个程序可以正常编译执行并打印结果 10。

答案：“先有鸡还是先有蛋”问题只是对面向对象本质的一个理解，将人类的自然语言放在此处来理解并不合适。由下图可知，根本不存在鸡与蛋的问题，而是型与值的问题，以及指针引用的问题，因为鸡和蛋两个

对象间是“引用”关系而不是“包含”关系。



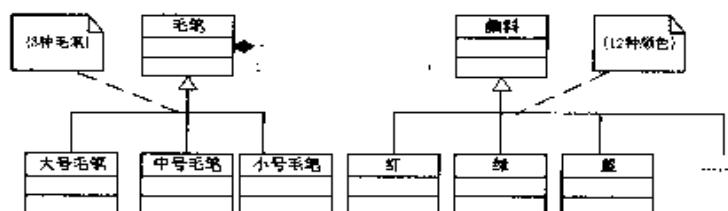
面试例题 2：请用设计模式的思想描述蜡笔和毛笔有什么不同，并用 C# 程序实现。

解析：大家小时候都有用蜡笔画画的经历吧。红红绿绿的蜡笔一大盒，根据想象描绘出各式图样。而毛笔下的国画更是工笔写意各展风采。而今天我们的故事从蜡笔与毛笔说起。

设想要绘制一幅图画，蓝天、白云、绿树、小鸟。如果画面尺寸很大，那么用蜡笔绘制就会遇到点儿麻烦。毕竟细细的蜡笔要涂出一片蓝天是有些麻烦的。如果有可能，最好有套大号蜡笔，粗粗的蜡笔很快能涂抹完成。至于色彩嘛，最好每种颜色来支粗的，除了蓝天还有绿地呢。这样，如果一套 12 种颜色的蜡笔，我们需要两套 24 支，同种颜色的一粗一细。要是再有一套中号蜡笔就更好了，于是，不多不少总共 36 支蜡笔（见下图）。



再看看毛笔这一边，居然如此简陋：一套水彩 12 色，外加大、中、小 3 支毛笔（见下图）。你可别小瞧这“简陋”的组合，画蓝天用大毛笔，画小鸟用小毛笔，各有专长。



这就是 Bridge 模式。为了一幅画，我们需要准备 36 支型号不同的蜡笔，而改用毛笔的话 3 支就够了，当然还要搭配上 12 种颜料。通过 Bridge 模式，我们把乘法运算 $3 \times 12 = 36$ 改为了加法运算 $3 + 12 = 15$ ，这一改进可不小。那么这里蜡笔和毛笔到底有什么区别呢？

实际上，蜡笔和毛笔的一个关键区别就在于笔和颜色是否能够分离。桥梁模式的用意是“将抽象化（Abstraction）与实现化（Implementation）脱耦，使得二者可以独立地变化”。关键就在于能否脱耦。蜡笔的颜色和蜡笔本身是分不开的，所以必须使用 36 支色彩、大小各异的蜡笔来绘制图画。而毛笔与颜料能够很好地脱耦，各自独立变化，简化了操作。在这里，抽象层面的概念是：“毛笔用颜料作画”，而在实现时，毛笔有大、中、小 3 号，颜料有红、绿、蓝等 12 种，于是便可出现 3×12 种组合。每个参与者（毛笔与颜料）都可以在自己的自由度内随意转换。

蜡笔由于无法将笔与颜色分离，造成笔与颜色两个自由度无法单独变化，使得只有创建 36 种对象才能完成任务。Bridge 模式将继承关系转换为组合关系，从而降低了系统间的耦合，减少了代码编写量。

答案：

代码如下：

```
// 桥接模式类型举例
using System;

// "Abstraction"类
class Abstraction
{
    // 字段
```

```

protected Implementor implementor;
// 属性
public Implementor Implementor
{
    set{ implementor = value; }
}
// 函数
virtual public void Operation()
{
    implementor.Operation();
}
}

// "Implementor"类
abstract class Implementor
{
    // 函数
    abstract public void Operation();
}

// "RefinedAbstraction"类
class RefinedAbstraction : Abstraction
{
    // 函数
    override public void Operation()
    {
        implementor.Operation();
    }
}

// "ConcreteImplementorA"类
class ConcreteImplementorA : Implementor
{
    // 函数
    override public void Operation()
    {
        Console.WriteLine("ConcreteImplementorA Operation");
    }
}

// "ConcreteImplementorB"
class ConcreteImplementorB : Implementor
{
    // 函数
    override public void Operation()
    {
        Console.WriteLine("ConcreteImplementorB Operation");
    }
}

***** <summary>
/// Client test
/// </summary>
public class Client
{
    public static void Main( string[] args )
    {
        Abstraction abstraction = new RefinedAbstraction();
        // 设置 implementation 委托并且调用
    }
}

```

```

abstraction.Implementor = new ConcreteImplementorA();
abstraction.Operation();

// 修改 implementation 委托并且调用

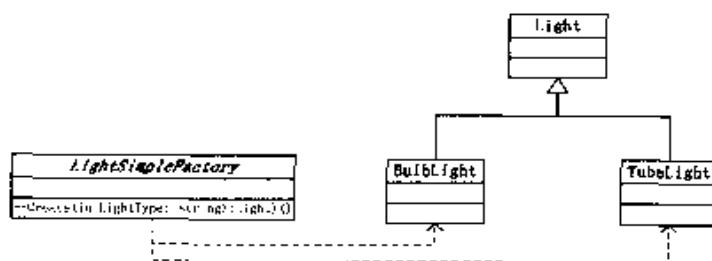
abstraction.Implementor = new ConcreteImplementorB();
abstraction.Operation();
}
}

```

面试例题 3：请用设计模式的思想描述简单工厂和工厂有什么不同，并用 C# 程序举例实现。

解析：

简单工厂如下图所示。



工厂类角色 Creator (LightSimpleFactory)：工厂类在客户端的直接控制下 (Create 方法) 创建产品对象。

抽象产品角色 Product (Light)：定义简单工厂创建的对象的父类或它们共同拥有的接口。可以是一个类、抽象类或接口。

具体产品角色 ConcreteProduct (BulbLight, TubeLight)：定义工厂具体加工出的对象。

简单工厂举例如下：

```

using System;

public abstract class Light
{
    public abstract void TurnOn();
    public abstract void TurnOff();
}

public class BulbLight : Light
{
    public override void TurnOn()
    {
        Console.WriteLine("Bulb Light is Turned on");
    }

    public override void TurnOff()
    {
        Console.WriteLine("Bulb Light is Turned off");
    }
}

```

```

public class TubeLight : Light
{
    public override void TurnOn()
    {
        Console.WriteLine("Tube Light is Turned on");
    }

    public override void TurnOff()
    {
        Console.WriteLine("Tube Light is Turned off");
    }
}

public class LightSimpleFactory
{
    public Light Create(string LightType)
    {
        if(LightType == "Bulb")
            return new BulbLight();
        else if(LightType == "Tube")
            return new TubeLight();
        else
            return null;
    }
}

public class Client
{
    public static void Main()
    {
        LightSimpleFactory lsf = new LightSimpleFactory();

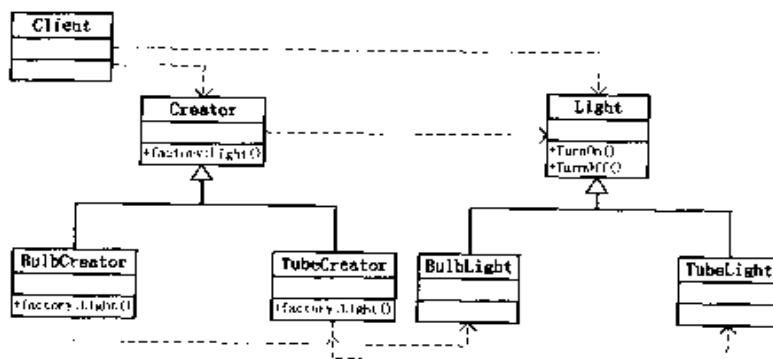
        Light l = lsf.Create("Bulb");
        l.TurnOn();
        l.TurnOff();

        Console.WriteLine("-----");

        l = lsf.Create("Tube");
        l.TurnOn();
        l.TurnOff();
    }
}

```

工厂模式如下图所示。



抽象工厂角色 Creator：是工厂方法模式的核心，与应用程序无关。

任何在模式中创建的对象的工厂类必须实现这个接口。

具体工厂角色 Concrete Creator: 这是实现抽象工厂接口的具体工厂类，包含与应用程序密切相关的逻辑，并且受到应用程序调用以创建产品对象。在上图中有两个这样的角色：BulbCreator 与 TubeCreator。

抽象产品角色 Product: 工厂方法模式所创建的对象的超类型，也就是产品对象的共同父类或共同拥有的接口。在上图中，这个角色是 Light。

具体产品角色 Concrete Product: 这个角色实现了抽象产品角色所定义的接口。某具体产品由专门的具体工厂创建，它们之间往往一一对应。

工厂举例如下：

```
using System;

public abstract class Light
{
    public abstract void TurnOn();
    public abstract void TurnOff();
}

public class BulbLight : Light
{
    public override void TurnOn()
    { Console.WriteLine("Bulb Light is Turned on"); }

    public override void TurnOff()
    { Console.WriteLine("Bulb Light is Turned off"); }
}

public class TubeLight : Light
{
    public override void TurnOn()
    { Console.WriteLine("Tube Light is Turned on"); }

    public override void TurnOff()
    { Console.WriteLine("Tube Light is Turned off"); }
}

public abstract class Creator
{
    public abstract Light factory();
}

public class BulbCreator : Creator
{
    public override Light factory()
    { return new BulbLight(); }
}

public class TubeCreator : Creator
{
    public override Light factory()
    { return new TubeLight(); }
}

public class Client
```

```

    {
        public static void Main()
        {
            Creator c1 = new BulbCreator();
            Creator c2 = new TubeCreator();

            Light l1 = c1.factory();
            Light l2 = c2.factory();

            l1.TurnOn();
            l1.TurnOff();

            Console.WriteLine("-----");
            l2.TurnOn();
            l2.TurnOff();
        }
    }
}

```

答案：

工厂方法模式与简单工厂模式在结构上的不同不是很明显。工厂方法类的核心是一个抽象工厂类，而简单工厂模式把核心放在一个具体类上。

工厂方法模式之所以有一个别名叫多态性工厂模式，是因为具体工厂类都有共同的接口，或者有共同的抽象父类。当系统扩展需要添加新的产品对象时，仅仅需要添加一个具体对象及一个具体工厂对象，原有工厂对象不需要进行任何修改，也不需要修改客户端，很好地符合了“开放-封闭”原则。而简单工厂模式在添加新产品对象后不得不修改工厂方法，扩展性不好。

工厂方法模式退化后可以演变成简单工厂模式。

假设一个生产灯管和灯泡的例子。简单工厂把工厂和车间混同，如果生产灯泡就整个车间统一学习灯泡的生产技术，如果生产灯管就整个车间统一学习灯管的生产技术。工厂模式把工厂和车间分离开，分别建立灯泡车间和灯管车间，接到生产任务后分别进行生产即可。如果以后要进行灯笼的生产，简单工厂就得继续统一学习灯笼的生产技术，工厂模式只需添加一个灯笼车间就够了。当然这意味着更好的扩展性，也意味着更大的开销。

面试例题4：请用设计模式的思想描述什么是原型模式？在原型模式中浅拷贝和深拷贝有什么不同？并用C#程序举例实现。

解析：

原型模式是：通过给出一个原型对象来指明所要创建的对象类型，

然后用复制这个原型对象的办法创建出更多的同类型对象。

孙悟空在与黄风怪的战斗中，“使一个身外身的手段：把毫毛揪下一撮，用口嚼得粉碎，望上一喷，叫声‘变’，变有百十个行者，都是同样的打扮，各执一根铁棒，把那怪围在空中。”换而言之，孙悟空可以根据自己的形象，复制出很多“身外身”来。

老孙这种身外身的手段在面向对象设计领域里就叫原型（Prototype）模式。

在 C#里面，我们可以很容易地通过 Clone()方法实现原型模式。任何类，只要想支持克隆，必须实现 C#中的 ICloneable 接口。ICloneable 接口中有一个 Clone 方法，可以在类中复写实现自定义的克隆方法。克隆的实现方法有两种：浅拷贝（shallow copy）与深拷贝（deep copy）。

浅拷贝是指当对象的字段值被拷贝时，字段引用的对象不会被拷贝。例如，如果一个对象有一个指向字符串的字段，并且我们对该对象做了一个浅拷贝，那么两个对象将引用同一个字符串。而深拷贝是对对象实例中字段引用的对象也进行拷贝的一种方式，所以如果一个对象有一个指向字符串的字段，并且我们对该对象做了一个深拷贝的话，我们将创建一个新的对象和一个新的字符串——新对象将引用新字符串。需要注意的是，执行深拷贝后，原来的对象和新创建的对象不会共享任何东西，改变一个对象对另外一个对象没有任何影响。

答案：

下面给出浅拷贝与深拷贝的两个例子，例子使用了 ICloneable 接口。C#中的数组是引用型的变量，我们通过数组来进行演示。

浅拷贝：

```
using System;
class ShallowCopy : ICloneable
{
    public int[] v = {1,2,3};
    public Object Clone()
    {
        return this.MemberwiseClone();
    }
    public void Display()
    {
        foreach(int i in v)
            Console.Write(i + " ");
        Console.WriteLine();
    }
}
```

```

        }
    }

class Client
{
    public static void Main()
    {
        ShallowCopy sc1 = new ShallowCopy();
        ShallowCopy sc2 = (ShallowCopy)sc1.Clone();
        sc1.v[0] = 9;
        sc1.Display();
        sc2.Display();
    }
}

```

ShallowCopy 对象实现了一个浅拷贝，因此当对 sc1 进行克隆时，其字段 v 并没有克隆，这导致 sc1 与 sc2 的字段 v 都指向了同一个 v。因此，当修改了 sc1 的 v[0]后，sc2 的 v[0]也发生了变化。

深拷贝：

```

using System;

class DeepCopy : ICloneable
{
    public int[] v = {1, 2, 3};

    // 默认构造函数
    public DeepCopy()
    {
    }

    // 供 Clone 方法调用的私有构造函数
    private DeepCopy(int[] v)
    {
        this.v = (int[])v.Clone();
    }

    public Object Clone()
    {
        // 构造一个新的 DeepCopy 对象，构造参数为
        // 原有对象中使用的 v
        return new DeepCopy(this.v);
    }

    public void Display()
    {
        foreach(int i in v)
            Console.Write(i + ", ");
        Console.WriteLine();
    }
}

class Client
{
    public static void Main()
    {
        DeepCopy dc1 = new DeepCopy();
        DeepCopy dc2 = (DeepCopy)dc1.Clone();
    }
}

```

```

        dc1.v[0] = 9;
        dc1.Display();
        dc2.Display();
    }
}

```

这次在克隆的时候，不但克隆对象本身，连里面的数组字段一并克隆了。因此，最终打印出来的 dc1 与 dc2 不同。

面试例题 5：以下代码实现了设计模式中的哪种模式？[美国某著名搜索引擎公司 Y 2005 年面试题]

```

public sealed class SampleSingleton1
{
    private int m_Counter = 0;
    private SampleSingleton1()
    {
        Console.WriteLine("初始化 SampleSingleton1。");
    }
    public static readonly SampleSingleton1 Singleton = new
        SampleSingleton1();
    public void Counter()
    {
        m_Counter++;
    }
}

```

- A. 原型 B. 抽象工厂 C. 单键 D. 生成器

解析：这是一个典型的单键模式。

答案：C。

面试例题 6：Consider a context-sensitive help feature for graphical user interface. The user can obtain help information on any part of the interface just by clicking on it. The help that's provided depends on the part of interface that's selected and its context; for example, a button widget in a dialog box might have different help information than a similar button in the main window. If no specific help information exists for that part of the interface, then the help system should display a more general help message about the immediate context—the dialog box as a whole, for example.

(为图形用户界面设计一种特性——能领会语境并提供帮助，使得用户仅仅通过在界面的任何部分上单击鼠标就可以获得相应的帮助信息。它所提供的帮助主要依赖于所选择的界面及相应的语境。

举个例子，在具备这样特性的一个对话框中的一个按钮部件上单击鼠标可能就会比一个主窗口中类似的按钮有不同的帮助信息。如果没有该界面的这部分内容的具体帮助信息，那么帮助系统就会根据即时的具体语境提供较为通用的帮助信息，这就是对于上述问题的一个具体的例子。）

为了设计上面的东西该选用（ ）。[中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. 观察者模式
- B. 桥接模式
- C. 供应链模式
- D. 原型模式

解析：本题应该选择观察者模式。为了形象描述观察者模式，我举一个给教工发大米的例子。

每年到年根的时候，工会都要给每位教工发大米、油什么的。正好我家的米快吃完了，所以这两天就盼着发大米。可是等了好多天也没见什么动静，扳着手指头数数离年根还有多半个月呢，看我心急的。不过，盼大米的老师恐怕不只是我一个，都等着工会有什么动静就去领大米呢。

教工等着发大米是一个典型的观察者模式。当工会大米来了，教工就会做出响应。不过这观察也有两种说头：一种是拉（Pull）模式，要求教工时不时到工会绕一圈，看看大米来了没有，恐怕没有人认为这是一种好办法。当然，还有另外一种模式，就是推（Push）模式，大米到了工会，工会会给每家每户把大米送过去。第二种方法好不好呢？好？呵呵，谁说好了，谁说好我让谁到工会上班去。

其实，我们还有一种方法，就是大米到了工会后，工会不把大米给每人送去，而是给每人发个轻量级的“消息”，教工得到消息后，再把大米拉回各家。这要求每位教工有一个工会的引用，在得到消息后到指定的地点领取大米。这样工会不用给每家教工送大米，而教工也不用每天到工会门口巴望着等大米了。

而本题鼠标帮助的例子也与教工拉大米相似。观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题帮助。这个帮助对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

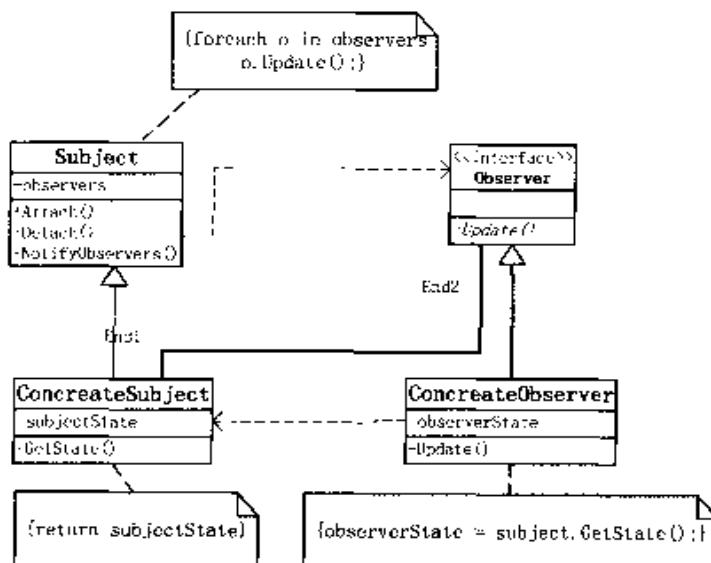
一个软件系统常常要求在某一个对象的状态发生变化的时候，某些其他的对象做出相应的改变。做到这一点的设计方案有很多，但是为了

使系统能够易于复用，应该选择低耦合度的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计师需要使这些低耦合度的对象能够维持行动的协调一致，保证高度的协作（collaboration）。观察者模式是满足这一要求的各种设计方案中最重要的- -种。

答案：A。

扩展知识（观察者模式的结构）

观察者模式的类图如下图所示。



可以看出，在这个观察者模式的实现里有下面这些角色。

抽象主题（Subject）角色：主题角色把所有对观察者对象的引用保存在一个聚集里，每个主题都可以有任何数量的观察者。
抽象主题提供一个接口，可以增加和删除观察者对象。抽象主题角色又叫做抽象被观察者（Observable）角色，一般用一个抽象类或者一个接口实现。

抽象观察者（Observer）角色：为所有的具体观察者定义一个接口，在得到主题的通知时更新自己。这个接口叫做更新接口。
抽象观察者角色一般用一个抽象类或者一个接口实现。在这个示意性的实现中，更新接口只包含一个方法（即 **Update()**方法），这个方法叫做更新方法。

具体主题（ConcreteSubject）角色：将有关状态存入具体观察

者对象；在具体主题的内部状态改变时，给所有登记过的观察者发出通知。具体主题角色又叫做具体被观察者角色（ConcreteObservable）。具体主题角色通常用一个具体子类实现。

具体观察者（ConcreteObserver）角色：存储与主题的状态自恰的状态。具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。如果需要，具体观察者角色可以保存一个指向具体主题对象的引用。具体观察者角色通常用一个具体子类实现。

从具体主题角色指向抽象观察者角色的合成关系，代表具体主题对象可以有任意多个对抽象观察者对象的引用。之所以使用抽象观察者而不是具体观察者，意味着主题对象不需要知道引用了哪些 ConcreteObserver 类型，而只知道抽象 Observer 类型。这就使得具体主题对象可以动态地维护一系列的对观察者对象的引用，并在需要的时候调用每一个观察者共有的 Update() 方法。这种做法叫做针对抽象编程。

面试例题 7：Implement the simplest singleton pattern(initialize if necessary).

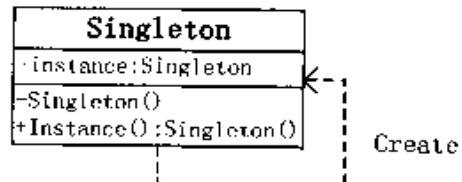
(写一个单例模式，如果有必要的话就初始化。) [德国某著名软件咨询企业 2005 年 10 月面试题]

解析：

单例模式的特点有三：

- 单例类只能有一个实例。
- 单例类必须自己创建自己的唯一实例。
- 单例类必须给所有其他对象提供这一实例。

Singleton 模式包含的角色只有一个，就是 Singleton。Singleton 拥有一个私有构造函数，确保用户无法通过 new 直接实例化它。除此之外，该模式中包含一个静态私有成员变量 instance 与静态公有方法 Instance()。Instance 方法负责检验并实例化自己，然后存储在静态成员变量中，以确保只有一个实例被创建。单例模式的结构图如右图所示。



答案：

完整代码如下：

```
// 单件模式类型举例
using System;

// "Singleton"
class Singleton
{
    // 字段
    private static Singleton instance;

    // 结构
    protected Singleton() {}

    // 函数
    public static Singleton Instance()
    {
        // 设定初值进行初始化
        if( instance == null )
            instance = new Singleton();

        return instance;
    }
}

/**/*<summary>
/// Client test
/// </summary>
public class Client
{
    public static void Main()
    {
        // 构造函数被保护——不能使用 new 字符
        Singleton s1 = Singleton.Instance();
        Singleton s2 = Singleton.Instance();

        if( s1 == s2 )
            Console.WriteLine( "The same instance" );
    }
}
```

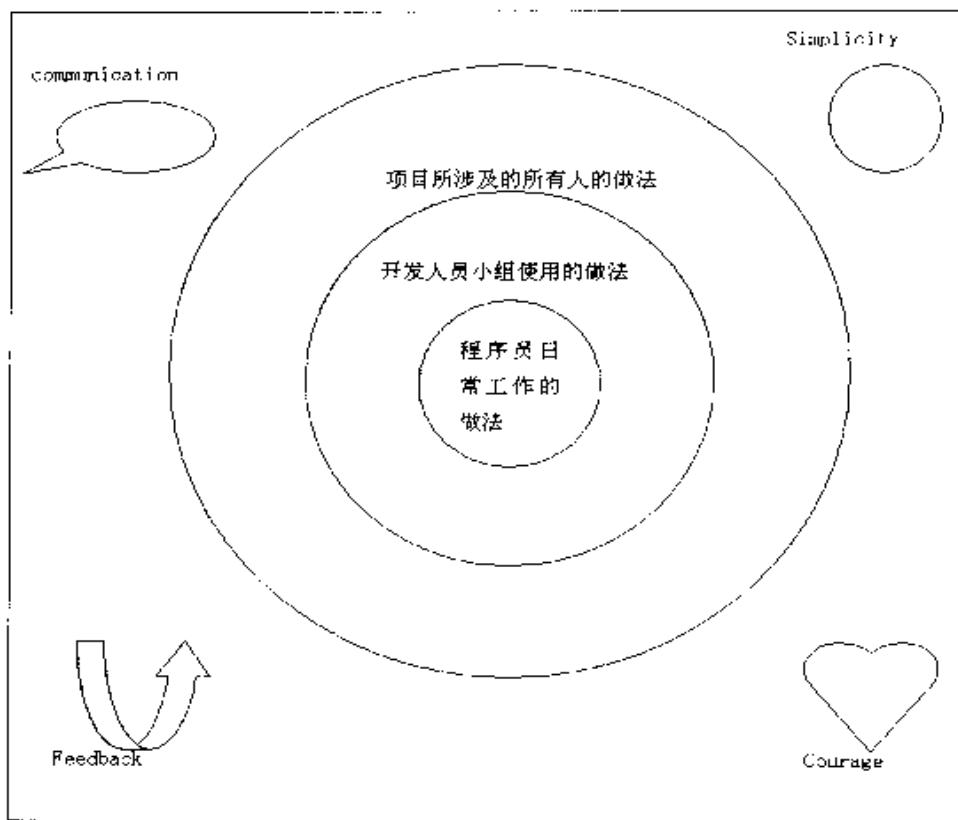
面试例题 8：什么是敏捷软件开发？[美国某著名软件咨询公司面试题]

答案：

敏捷软件开发不是一个具体的过程，而是一个涵盖性术语（umbrella term），用于概括具有类似基础的方式和方法。这些方法中包括极限编程（Extreme Programming）、动态系统开发方法（Dynamic System Development Method）、SCRUM（混合嵌入编程）、Crystal（结构编程）和Lean（精益编程）等，都着眼于快速交付高质量的工作软件，并做到客户满意。

尽管构成这个敏捷开发过程的每种方法都具有类似的目标，但是它们实现这个目标的做法（practice）却不尽相同。下面的图表基本勾画出了我

们提炼出来的这些敏捷开发最佳做法。最中间的圆环代表一对程序员日常工作 的做法。紧接着的中间一个圆环表示开发人员小组使用的做法。最外面的一个圆环是项目所涉及的所有人的做法——客户、开发人员、测试人 员、业务分析师，等等。



这些圆环里的所有做法都直接与 4 个角上显示的敏捷开发的核心价值 相关：沟通（Communication）、反馈（Feedback）、勇气（Courage）和简 单（Simplicity）。也就是说，每个做法都给予我们一条实现敏捷开发价值 并让它们成为该过程一部分的具体方法。

15.2 软件工程

面试例题 1：test.exe 是一个判断 1800 年至 2000 年之间的某个年份是否为 闰年的程序，请设计测试用例测试该程序。[加拿大某著名网络开发公司面试题]

答案：

我们可以对程序功能实施测试（正常测试或反常测试），也可以对边

界点进行测试，等等。

- (1) 输入一个负数看程序做出什么样的判断。
- (2) 输入一个 1~1799 的年份看程序的输出结果如何。
- (3) 输入一个 2001 以后的年份看程序的输出结果如何。
- (4) 输入年份 1800、2000 看程序是否能够做出正确的判断。
- (5) 输入一个介于 1800~2000 之间的年份看程序是否能够做出正确的判断。
- (6) 特殊字符的输入（我把负号放在这里面测试）。
- (7) 其他特殊功能的输入（如年代自动补位）。
- (8) 安全性能测试（启动多个 test.exe 程序）。
- (9) 部分黑盒测试（各个适用系统的测试）。
- (10) 画面测试（如果有的话）。

面试例题 2：有一个传真调度程序。这段代码将从一个指定的文件名发传真到一个电话号码。其中有校验的要求：带区号的电话必须是这样的格式 xnn-nnn-nnnn，其中 x 必须是在 2~9 之间，而 n 可以是从 0~9 的任何数字。[加拿大某著名网络开发公司面试题]

以下的区域是预留的，现在还不是有效的区号：x11、x9n、37n、96n。
函数的原型是：

```
/*
Send the named file as a fax to the given phone number.
*/
public boolean sendFax(String phone,
String filename)
```

给定了这些需求，请设计测试用例。

答案：

- (1) 输入的 x 的值不为 2~9，系统应该给出提示。
- (2) 当输入的 x 的值正确时，判断整个字符串不为 x11、x9n、37n、96n。
- (3) 当指定的文件名不存在时，程序能够给出相应的提示。
- (4) 当给定的号码不存在时，程序能够给出相应的提示。
- (5) 当指定的电话号码无应答时/忙时/不可到达时，程序能够给出相应的提示。

- (6) 当传真的文件非常大时，系统是如何处理的。
- (7) 判断输入位数（长度）。
- (8) 健康性判断（输入其他字符，字符位置不一致），如：2347-123-4567, 2-12345-679。
- (9) 错误字符（/*+·#¥%——……）。
- (10) 双字节字符（中、日字符里面有占2字节的数字字符）。

面试例题3：对程序进行功能性穷举测试可能吗？为什么？[加拿大某著名网络开发公司面试题]

解析：穷举不是无穷举例的意思，而是为测试准备尽可能多的数据进行测试。

黑盒是针对产品的功能进行测试。比如测试除法功能，要求提供2个参数，返回一个结果。那么，我们就可以把所有能想到的数都放在测试数据中。

白盒是针对程序逻辑进行测试。测试人员需要了解程序逻辑，针对逻辑中容易出现的问题进行测试。那么，测试的结果只能是基于逻辑的，只能验证逻辑是否被正常执行了，而不能验证这个逻辑是否正确，等等。

题目中的可能与不可能，是对测试目标的确定。从两个方向都能达到自己的目标，但是黑盒的测试结果可能更全面。黑盒测试在理论上可以反映所有bug。

答案：功能性穷举测试对程序是做不到的，符合条件的测试用例举不胜举，一一进行测试不仅耗费时间和精力，而且有些等价的测试用例重复进行测试就是多余的了。

15.3 C#基础

面试例题1：C#有固定的开发工具吗？

答案：没有。小到 NotePad，大到 VS.NET、C#Builder，都可以做C#开发。网上有很多免费C#编译器，如 Sharp-Develop，都可以用来从事C#的开发。

面试例题 2：C#与 C++有什么区别？

答案：区别很多。首先是托管与非托管的区别。托管代码不允许进行对内存的操作，而是由固定的垃圾回收机制来完成，而 C++则不然。其次 C#和 Java 类似，都是运行在虚拟机上的（分别是.NET 虚拟机和 Java 虚拟机），而 C++不需要这样一个平台。最后 C#是完全面向对象的。在 C#里，万物皆是类，绝对不存在一个超越类以上的函数或变量。C++也是面向对象的，但其仍然保留面向过程语言的特点（比如说 C++存在全局变量）。最后，C#摒弃了 C++中的多重继承等不易掌握的特点，代之以接口等，使编程变得更加轻松和简便。

面试例题 3：C#在哪些方面有优势？

答案：C#兼具 VB 的快速简练、Delphi 的可视化控件编程、Java 的完全面向对象和 C++的语法规则。C#最长于 Web 开发。微软宣称 ASP.NET 1.1 以后，所有的代码均由 C#写成。C#在 Web 开发领域的能力由此可见一斑。

面试例题 4：一列数的规则如下：1、1、2、3、5、8、13、21、34……求第 30 位数是多少，用递归算法实现，C#语言编写。

答案：

完整代码如下：

```
public class MainClass
{
    public static void Main()
    {
        Console.WriteLine(Foo(30));
    }
    public static int Foo(int i)
    {
        if (i < 0)
            return 0;
        else if(i == 0 || i == 1)
            return 1;
        else return Foo(i - 2) + Foo(i - 1);
    }
}
```

面试例题 5：面向对象的语言具有_____性、_____性、_____性。

答案：继承，封装，多态。

面试例题 6：下列关于 C#中索引器的理解正确的是（ ）。[中国某著名计算机金融软件公司 2005 年面试题]

- A. 索引器的参数必须是两个或两个以上
- B. 索引器的参数类型必须是整型数
- C. 索引器没有名字
- D. 以上皆非

答案：C 索引器是没有名字的。索引器使用的是关键字 this，而不是一个名称。

面试例题 7：简述 private、protected、public、internal 修饰符的访问权限。

答案：

private：私有成员，在类的内部才可以访问。

protected：保护成员，该类内部和继承类中可以访问。

public：公共成员，完全公开，没有访问限制。

internal：在同一命名空间内可以访问。

面试例题 8：int[][] myArray3=new int[3][] {new int[3]{5,6,2}, new int[5]{6,9,7,8,3}, new int[2]{3,2}}；，myArray3[2][2]的值是多少？[美国某著名搜索引擎公司 Y 2005 年面试题]

- A. 9
- B. 2
- C. 6
- D. 越界

解析：这个数组首先是一个二维数组，一共是 3 行。第 1 行是 int[3]{5,6,2}，其中 int[0][0]是 5，int[0][1]是 6，int[0][2]是 2；第 2 行是 int[5]{6,9,7,8,3}，其中 int[1][0]是 6，int[1][1]是 9，int[1][2]是 7，int[1][3]是 8，int[1][4]是 3；第 3 行是 int[2]{3,2}，其中 int[2][0]是 3，int[2][1]是 2。所以 myArray3[2][2]的值是越界。

答案：D。

面试例题 9：以下叙述正确的是（ ）。

- A. 接口中可以有虚方法
- B. 一个类可以实现多个接口
- C. 接口不能被实例化
- D. 接口中可以包含已实现的方法

解析：一个类可以实现多个接口，就像一架机器可以由不同零部件

组成一样。接口类似于纯虚类，所以不可以被实例化。

答案：C。

面试例题 10：写出下列程序的输出结果。[美国某著名搜索引擎公司 Y 2005 年面试题]

```
using System;
class Class1
{
    public static int Count = 0;
    static Class1()
    {
        Count++;
    }
    public Class1()
    {
        Count++;
    }
}
public class B
{
    public static void Main()
    {
        Class1 c1 = new Class1();
        Class1 c2 = new Class1();
        Console.WriteLine(Class1.Count);
    }
}
```

- A. 1 B. 2 C. 3 D. 4

解析：本题考的是静态构造函数的使用方法。只有在第一次实例化对象时才启动 static Class1() 函数，以后再实例化对象时该函数不起作用。

答案：C。

15.4 C#继承

面试例题 1：写出下列程序的输出结果。[美国某著名搜索引擎公司 Y 2005 年面试题]

```
public abstract class A
{
    public A()
    {
        Console.WriteLine('A');
    }
    public virtual void Fun()
    {
        Console.WriteLine("A.Fun()");
    }
}
```

```

}

public class B: A
{
    public B()
    {
        Console.WriteLine('B');
    }

    public override void Fun()
    {
        Console.WriteLine("B.Fun()");
    }

    public static void Main()
    {
        A a = new B();
        a.Fun();
    }
}

```

解析：大家看主函数 A a = new B();首先声明了一个 A 的对象 a，但被赋给不同于 A 的对象 B，在此期间分别调用了 A 类的构造函数和 B 类的构造函数。然后调用 a 的 fun()函数，因为 a 的 fun()函数是虚函数，被 b 的 fun()函数覆盖，所以实际执行的是 b 的 fun()函数。

答案：

- A
- B
- B.Fun()

扩展知识

如果我们把上面的程序稍做修改，把：

```

public override void Fun()
{
    Console.WriteLine("B.Fun()");
}

```

改做：

```

public override void Fun()
{
    Console.WriteLine("B.Fun()");
}

```

答案就会变成：

- A
- B

A.Fun()

因为没有覆盖，a.Fun()还是执行着 A 类的 Fun 函数。

面试例题 2：写出下列程序的输出结果。[中国某著名软件外包企业 2004 年面试题]

```

using System;
class myApp
{
    public class A
    {
        public virtual void Fun1(int i)
        {
            Console.WriteLine(i);
        }
        public void Fun2(A a)
        {
            a.Fun1(1);
            Fun1(5);
        }
    }
    public class B: A
    {
        public override void Fun1(int i)
        {
            base.Fun1(i+1);
        }
    }
    public static void Main()
    {
        B b=new B();
        A a=new A();
        a.Fun2(b);
        b.Fun2(a);
    }
}

```

解析：a.Fun2(b)是指先执行 A 类下的 Fun2 函数。Fun2 函数下第一步是 a.Fun1(1)，但实际参数是 b，也就是执行 b.Fun1(1)。b.Fun1(1)函数是在 B 类当中重载过的，其内容是 base.Fun1(i+1)，也就是执行 Fun1(1+1)，结果是 2。然后执行 Fun1(5)，结果是 5。

b.Fun2(a)是指先执行 B 类下的 Fun2 函数，可 B 类本身没有 Fun2 函数，只有执行它的基类的 Fun2 函数。但实际参数是 a，也就是执行 a.Fun1(1)，结果是 1。然后执行 Fun1(5)。注意，这里的 Fun1(5)实际上是 b.Fun1(5)，实际执行的结果是 base.Fun1(5+1)，结果是 6。

答案：

2 5 1 6

扩展知识

这是非常让人头晕的一个题目，要搞清楚：一个函数的执行必须在一个具体的对象中实现。如果函数明确告诉是那个对象，则在该对象下执行；如果没有，则在默认的对象下执行。

面试例题 3：写出下列程序的输出结果。[美国某著名搜索引擎公司 Y 2005 年面试题]

```

using System;
using System.Collections;
abstract class BaseClass
{
    public virtual void MethodA()
    {
        Console.WriteLine("BaseClass");
    }
    public virtual void MethodB()
    {
    }
}
class Class1: BaseClass
{
    public new void MethodA()
    {
        Console.WriteLine("Class1");
    }
    public override void MethodB()
    {
    }
}
class Class2: Class1
{
    new public void MethodB()
    {
    }
}
class MainClass
{
    public static void Main(string[] args)
    {
        Class2 o = new Class2();
        o.MethodA();
    }
}

```

请问，此程序输出结果是：

- | | |
|--------------|---------------------|
| A. BaseClass | B. BassClass Class1 |
| C. Class1 | D. Class1 BassClass |

解析：三重继承问题。孙类只是继承父类，而不直接继承祖父类。

答案：C。

面试例题 4：MARS ROVERS 问题。[美国某著名软件咨询公司面试题]

A squad of robotic rovers are to be landed by NASA on a plateau on Mars.

This plateau, which is curiously rectangular, must be navigated by the rovers so that their on-board cameras can get a complete view of the surrounding terrain to send back to Earth.

A rover's position and location is represented by a combination of x and y co-ordinates and a letter representing one of the four cardinal compass points. The plateau is divided up into a grid to simplify navigation. An example position might be 0, 0, N, which means the rover is in the bottom left corner and facing North.

In order to control a rover, NASA sends a simple string of letters. The possible letters are 'L', 'R' and 'M'. 'L' and 'R' makes the rover spin 90 degrees left or right respectively, without moving from its current spot.

'M' means move forward one grid point, and maintain the same heading. Assume that the square directly North from (x, y) is (x, y+1).

INPUT

The first line of input is the upper-right coordinates of the plateau, the lower-left coordinates are assumed to be 0,0.

The rest of the input is information pertaining to the rovers that have been deployed. Each rover has two lines of input. The first line gives the rover's position, and the second line is a series of instructions telling the rover how to explore the plateau.

The position is made up of two integers and a letter separated by spaces, corresponding to the x and y co-ordinates and the rover's orientation.

Each rover will be finished sequentially, which means that the second rover won't start to move until the first one has finished moving.

OUTPUT

The output for each rover should be its final co-ordinates and heading.

INPUT AND OUTPUT

Test Input:

5 5

```

1 2 N
LMLMLMLMM
3 3 E
MMRMMRMRRM
Test Output:
1 3 N
5 1 E

```

(火星探测器问题。

一小队机器人探测器将由 NASA 送上火星高原，探测器将在这个奇特的矩形高原上行驶。用它们携带的照相机将周围的全景地势图发回地球。

每个探测器的方向和位置将由一个 (x, y) 系坐标图和一个表示地理方向的字母表示出来。为了方便导航，平原将被划分为网格状。位置坐标示例：0, 0, N，表示探测器在坐标图的左下角，且面朝北方。为控制探测器，NASA 会传送一串简单的字母。可能传送的字母为：“L”、“R”和“M”。“L” 和 “R” 分别表示使探测器向左、向右旋转 90° ，但不离开它所在地点。“M” 表示向前开进一个网格的距离，且保持方向不变。假设以广场（高原）的正北方向为 Y 轴的指向。

输入：首先输入的 line 是坐标图的右上方。假定左下方顶点的坐标为 $(0, 0)$ 。剩下的要输入的是被分布好的探测器的信息。每个探测器需要输入两个 lines。第一条 line 提供探测器的位置，第二条是关于这个探测器怎样进行高原探测的一系列说明。位置是由两个整数和一个区分方向的字母组成的，对应了探测器的 (x, y) 坐标和方向。每个探测器的移动将按序完成，即后一个探测器不能在前一个探测器完成移动之前开始移动。

输出：每个探测器的输出应该是它行进到的最终位置坐标和方向。

输入和输出测试如下。

期待的输入：

```

5 5
1 2 N
LMLMLMLMM
3 3 E
MMRMMRMRRM

```

期待的输出：

1 3 N

5 1 E

答案：

程序完整代码如下：

```
using System;
using System.Collections;

namespace ConsoleApplication5
{
    enum Heading {E,S,W,N};
    enum X way {E = 1,S = 0,W = -1,N = 0};
    enum Y way {E = 0,S = -1,W = 0,N = 1};

    class Area
    {
        private static int area X;
        private static int area Y;
        private static ArrayList aryRovers = new ArrayList();
        public Area()
        {}
        protected void AddRover(Rover aRover)
        {
            aryRovers.Add(aRover);
        }
        public static bool SetArea(int x, int y)
        {
            if(area X == 0 && area Y == 0)
            {
                area X = x;
                area Y = y;
                return true;
            }
            else
                return false;
        }

        public static string GetState()
        {
            string strTemp = "";
            foreach(Rover aobj in aryRovers)
            {
                strTemp += String.Format("{0} {1} {2} ",
                    aobj.X,aobj.Y,aobj.heading);
            }
            return strTemp;
        }

        public static bool CheckArea(int x,int y)
        {
            if(x >= 0 && y >= 0 && x <= area X && y <= area Y)
                return true;
            else
                return false;
        }
    }
    class Rover:Area
    {
        protected int m_X;
```

```

private int m X;
private Heading m heading;
public int X
{
    get
    {
        return m X;
    }
}
public int Y
{
    get
    {
        return m Y;
    }
}
public Heading heading
{
    get
    {
        return m heading;
    }
}

public Rover(int X in, int Y in, Heading heading in)
{
    m X = X in;
    m Y = Y in;
    m heading = heading in;
    AddRover(this);
}

public bool Turnning(bool isLeftTurnning)
{
    try
    {
        m heading = (Heading)((int)heading +
        (isLeftTurnning?3:1))%4;
        return true;
    }
    catch
    {
        return false;
    }
}

public bool Move()
{
    try
    {
        int x move = (int)(X way)(Enum.Parse
        (typeof(X way),m heading.ToString()));
        int y move = (int)(Y way)(Enum.Parse
        (typeof(Y way),m heading.ToString()));
        if(CheckArea(m X + x move,m Y + y move))
        {
            m X += x move;
            m Y += y move;
        }
        return true;
    }
    catch
    {
        return false;
    }
}

```

```

        }
    }

    class Class1
    {
        /// <summary>
        /// 应用程序的主进入点
        /// </summary>

        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: 在此加入激活应用程序的代码
            //

            string strTemp = Console.ReadLine();
            try
            {
                string[] cTemp = strTemp.Trim().Split(new char[] {' '});
                Area.SetArea(Convert.ToInt32(cTemp[0]), Convert.ToInt32(cTemp[1]));
            }
            catch
            {
                Console.WriteLine("Error...");
            }

            while ((strTemp = Console.ReadLine()).ToUpper() != "State".ToUpper())
            {
                try
                {
                    string[] cTemp = strTemp.Trim().ToUpper().Split(new char[] {' '});
                    Rover aRover = new Rover(Convert.ToInt32(cTemp[0]), Convert.ToInt32(cTemp[1]),
                                              (Heading)Enum.Parse(typeof(Heading),
                                              cTemp[2]));
                    strTemp = Console.ReadLine().ToUpper();
                    foreach (char active in strTemp.
                        ToCharArray())
                    {
                        switch (active)
                        {
                            case 'L':
                                aRover.Turnning(true);
                                break;
                            case 'R':
                                aRover.Turnning(false);
                                break;
                            case 'M':
                                aRover.Move();
                                break;
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Error...");
                }
            }
        }
    }
}

```

```
        Console.WriteLine(Area.GetState());
        Console.Read();
    }
}
```

15.5 C#委托

面试例题 1：程序设计：猫大叫一声，所有的老鼠都开始逃跑，主人被惊醒。[中国某著名 IT 培训企业公司 2005 年 8 月面试题]

要求：一要有联动性，老鼠和主人的行为是被动的。二考虑可扩展性，猫的叫声可能引起其他联动效应。

解析：首先是联动效果，运行代码只要执行 Cat.Cryed()方法。然后对老鼠和主人进行抽象；

- (1) 构造出 Cat、Mouse、Master 这 3 个类，并能使程序运行。
 - (2) 从 Mouse 和 Master 中提取抽象。
 - (3) 联动效应，只要执行 Cat.Cryed()就可以使老鼠逃跑，主人惊醒。

答卷二

具体程序代码如下：

```
public interface Observer
{
    void Response();      //观察者的响应，如是老鼠见到猫的反应
}
public interface Subject
{
    void AimAt(Observer obs);
    //针对哪些观察者，这里指猫要捕捉的对象——老鼠
}
public class Mouse : Observer
{
    private string name;
    public Mouse(string name, Subject subj)
    {
        this.name = name;
        subj.AimAt(this);
    }
    public void Response()
    {
        Console.WriteLine(name + " attempt to escape!");
    }
}
public class Master : Observer
{
    public Master(Subject subj)
    {
        subj.AimAt(this);
    }
}
```

```

        }

        public void Response()
        {
            Console.WriteLine("Host waken!");
        }
    }
    public class Cat : Subject
    {
        private ArrayList observers;
        public Cat()
        {
            this.observers = new ArrayList();
        }
        public void AimAt(Observer obs)
        {
            this.observers.Add(obs);
        }
        public void Cry()
        {
            Console.WriteLine("Cat cryed!");
            foreach (Observer obs in this.observers)
            {
                obs.Response();
            }
        }
    }
    class MainClass
    {
        static void Main(string[] args)
        {
            Cat cat = new Cat();
            Mouse mouse1 = new Mouse("mouse1", cat);
            Mouse mouse2 = new Mouse("mouse2", cat);
            Master master = new Master(cat);
            cat.Cry();
        }
    }
}

```

设计方法二：使用 event-delegate 设计。代码如下。

```

public delegate void SubEventHandler();
public abstract class Subject
{
    public event SubEventHandler SubEvent;
    protected void FireAway()
    {
        if (this.SubEvent != null)
            this.SubEvent();
    }
}
public class Cat : Subject
{
    public void Cry()
    {
        Console.WriteLine("cat cryed.");
        this.FireAway();
    }
}
public abstract class Observer
{
    public Observer(Subject sub)
    {

```

```

        sub.SubEvent += new SubEventHandler(Response);
    }
    public abstract void Response();
}
public class Mouse : Observer
{
    private string name;
    public Mouse(string name, Subject sub) : base(sub)
    {
        this.name = name;
    }
    public override void Response()
    {
        Console.WriteLine(name + " attempt to escape!");
    }
}
public class Master : Observer
{
    public Master(Subject sub) : base(sub) {}
    public override void Response()
    {
        Console.WriteLine("host waken");
    }
}
class Class1
{
    static void Main(string[] args)
    {
        Cat cat = new Cat();
        Mouse mouse1 = new Mouse("mouse1", cat);
        Mouse mouse2 = new Mouse("mouse2", cat);
        Master master = new Master(cat);
        cat.Cry();
    }
}

```

15.6 ASP.NET

面试例题 1：在 ASP.NET 中，对于 Command 对象的 ExecuteNonQuery() 方法和 ExecuteReader() 方法，下面叙述错误的是（ ）。[中国某著名 IT 培训企业 2005 年 8 月面试题]

- A. insert、update、delete 等操作的 SQL 语句主要由 ExecuteNonQuery() 方法执行
- B. ExecuteNonQuery()方法返回执行 SQL 语句所影响的行数
- C. Select 操作的 SQL 语句只能由 ExecuteReader()方法执行
- D. ExecuteReader()方法返回一个 DataReader 对象

解析：Select 操作的 SQL 语句可以选择很多方法执行。

答案：C。

面试例题 2：下列 ASP.NET 语句（ ）正确地创建了一个与 SQL Server 2000 数据库的连接。[中国某著名 IT 培训企业 2005 年 8 月面试题]

- A. SqlConnection con1 = new Connection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");
- B. SqlConnection con1 = new SqlConnection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");
- C. SqlConnection con1 = new SqlConnection(Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB);
- D. SqlConnection con1 = new OleDbConnection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");

答案： B。

面试例题 3：下列 ASP.NET 语句（ ）正确地创建了一个与 Access 2000 数据库的连接。[中国某著名 IT 培训企业 2005 年 8 月面试题]

- A. SqlConnection con1 = new Connection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");
- B. SqlConnection con1 = new SqlConnection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");
- C. SqlConnection con1 = new SqlConnection(Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB);
- D. SqlConnection con1 = new OleDbConnection("Data Source = localhost; Integrated Security = SSPI; Initial Catalog = myDB");

解析： Access 数据库连接要使用 OleDbConnection 参数。

答案： D。

面试例题 4：Which of the following operations can you NOT perform on an ASP.NET DataSet? (以下选项中哪个操作在 ADO.NET 的 dataset 上是无法运用的？) [美国某著名搜索引擎公司 Y 2005 年 12 月面试题]

- A. A DataSet can be synchronised with a RecordSet. (DataSet 与 RecordSet 同步。)
- B. A DataSet can be synchronised with the database. (DataSet 与 Database

同步。)

- C. A DataSet can be converted to XML. (DataSet 可以转换成 XML。)
- D. You can infer the schema from a DataSet. (从 DataSet 中可以提取架构。)

解析：DataSet 完全独立于提供程序，换言之，它们没有任何一种功能需要依赖用于连接数据源的底层提供程序。由于 DataSet 是“断开”的，所以在 DataSet 对象的整个生命周期，底层连接都不必开放。这样一来，就可高效率地使用当前可用的数据库连接。可采取几种方式在 DataSet 中填充数据：通过底层提供程序特有的一系列命令对象来填充，从一个 XML 文档或者文档片断中填充数据，或者手工提供数据。DataSet 非常灵活，并不一定要用一个源数据库来提供数据。

答案：B。

面试例题 5：In Object Oriented Programming, how would you describe encapsulation? (在面向对象的程序中，你如何描述封装？) [美国某著名搜索引擎公司 Y 2005 年 12 月面试题]

- A. The conversion of one type of object to another. (一个对象类型的转化。)
- B. The runtime resolution of method calls. (方法调用的动态解决方式。)
- C. The exposition of data. (数据的展示。)
- D. The separation of interface and implementation. (接口和执行的分离。)

解析：面向对象和基于对象的区别。

很多人不能区分“面向对象”和“基于对象”这两个概念。面向对象的三大特点（封装，继承，多态）缺一不可。通常“基于对象”是使用对象，但是无法利用现有的对象模板产生新的对象类型，继而产生新的对象。也就是说“基于对象”没有继承的特点。而“多态”表示为父类类型的子类对象实例，没有了继承的概念也就无从谈论“多态”。现在的很多流行技术都是基于对象的，它们使用封装好的对象，调用对象的方法，设置对象的属性，但是无法让程序员派生新对象类型，而只能使用现有对象的方法和属性。所以当你判断一个新的技术是否是面向对象的时候，通常可以使用后两个特性。“面向对象”和“基于对象”都实现了“封装”的概念，但是面向对象实现了“继承和多态”，而“基于对象”没有实现这些。

从事面向对象编程的人按照分工可以分为“类库的创建者”和“类库

的使用者”。使用类库的人并不都是具备了面向对象思想的人。一般知道如何继承和派生新对象就可以使用类库了，然而我们的思维并没有真正地转过来。使用类库在形式上是面向对象，而实质上只是库函数的一种扩展。

面向对象是一种思想，是我们考虑事情的方法，通常表现为：是将问题按照过程方式来解决呢，还是将问题抽象为一个对象。在很多情况下，我们会不知不觉地按照过程方式来解决它，而不是考虑将要问题抽象为对象去解决它。有些人打着面向对象的幌子，干着过程编程的勾当。

答案：D。

面试例题 6：How does assembly versioning in .NET prevent DLL Hell? (.NET 是如何预防 DLL 陷阱的) [美国某著名搜索引擎公司 Y 2005 年 12 月面试题]

- A. The runtime checks to see that only one version of an assembly is on the machine at any one time. (运行时检测是否只有一个版本在机器里运行。)
- B. .NET allows assemblies to specify the name AND the version of any assemblies they need to run. (.NET 允许配置版本详细的名称和记录需要在执行时用到的版本号。)
- C. The compiler offers compile time checking for backward compatibility. (编译器允许版本向后兼容。)
- D. It doesn't. (无法预防。)

解析：

DLL 是微软提出的动态链接库概念。其原理是每个程序所需的代码可以从同一个地址获取，这样可以节约系统空间。但这也就成了 Windows 最大的弱点：应用程序在安装时为了让自己运行得更好，会将这些数据升级成自己的版本，这样就导致其他程序甚至连 Windows 自己都无法启动，DLL 成了“陷阱”。.NET 改进了系统的配置、伸缩性、安全性和可靠性。对于简单的 ASP 应用程序，配置其实并不算什么问题，但是当你将其移植到一个利用组件的 N 层结构中时就会遇到问题。当你对这些应用程序进行配置和维护时，DLL 陷阱问题（组件注册、版本、锁定的 DLL 等）就会出现。ASP.NET 中则取消了组件注册及 DLL 锁定，全面使用了 XML 配置文件，从而解决了这个问题。

答案：B。

第 4 部分

操作系统、数据库和 网络

Operating system, database, network

本部分主要介绍求职面试过程中出现的第三个重要的板块——操作系统、数据库和网络知识。这些内容虽不是面试题目中的主流，但仍然具有重要的意义。

第 16 章

操作 系 统

操作系统面试例题主要包括进程、线程、内存管理、垃圾回收，以及缓存等诸方面。

16.1 进程

面试例题 1：请描述进程和线程的差别。[美国某著名软件公司 2005 年面试题]

答案：

进程是程序的一次执行。而什么是线程（Thread）呢？线程可以理解为进程中执行的一段程序片段。在一个多任务环境中下面的概念可以帮助我们理解两者间的差别。

进程间是独立的，这表现在内存空间、上下文环境上；线程运行在进程空间内。

一般来讲（不使用特殊技术），进程无法突破进程边界存取其他进程内的存储空间；而线程由于处于进程空间内，所以同一进程所产生的线程共享同一内存空间。

同一进程中的两段代码不能够同时执行，除非引入线程。

线程是属于进程的，当进程退出时该进程所产生的线程都会被强制退出并清除。线程占用的资源要少于进程所占用的资源。进程和线程都可以有优先级。

面试例题 2：进程间的通信如何实现？[日本某著名家电/通信/IT 企业面试题]

答案：现在最常用的进程间通信的方式有：信号，信号量，消息队列，共享内存。

所谓进程通信，就是不同进程之间进行一些“接触”。这种接触有简单，也有复杂。机制不同，复杂度也不一样。通信是一个广义上的意义，不仅仅指传递一些 message。它们的使用方法是基本相同的，所以只要掌握了一种使用方法，然后记住其他的使用方法就可以了。信号在我学习的内容中，主要接触了信号来实现同步的机制，据说信号也可以用来做其他的事情，但是我还真不知道做什么。

信号和信号量是不同的，它们虽然都可用来实现同步和互斥，但前者是使用信号处理器来进行的，后者是使用 P、V 操作来实现的。消息队列是比较高级的一种进程间通信方法，因为它真的可以在进程间传送 message，你传送一个“*I seek you*”都可以。

一个消息队列可以被多个进程所共享（IPC 就是在这个基础上进行的）；如果一个进程的消息太多，一个消息队列放不下，也可以用多于一个的消息队列（不过可能管理会比较复杂）。共享消息队列的进程所发送的消息中除了 message 本身外还有一个标志，这个标志可以指明该消息将由哪个进程或者是哪类进程接受。每一个共享消息队列的进程针对这个队列也有自己的标志，可以用来声明自己的身份。

面试例题 3：在网络编程中设计并发服务器，使用多进程与多线程有什么区别？[中国台湾某著名杀毒软件公司 2005 年面试题]

解析：

进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。

线程：相对于进程而言，线程是一个更加接近于执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。

两者都可以提高程序的并发度，提高程序运行效率和响应时间。

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在 SMP 机器上运行，而进程则可以跨机器迁移。

答案：用多进程时每个进程有自己的地址空间（address space），线程则共享地址空间。所有其他区别都是由此而来的：

- **速度：**线程产生的速度快，线程间的通信快、切换快等，因为它们在同一个地址空间内。
- **资源利用率：**线程的资源利用率比较好也是因为它们在同一个地址空间内。
- **同步问题：**线程使用公共变量/内存时需要使用同步机制，还是因为它们在同一个地址空间内。

面试例题4：在Windows编程中互斥器(mutex)的作用和临界区(critical section)类似，请说一下二者间的主要区别。[中国台湾某著名杀毒软件公司2005年面试题]

解析：多线程编程问题。

答案：两者的区别是 mutex 可以用于进程之间互斥，critical section 是线程之间的互斥。

面试例题5：What are these ways in which a thread can enter the waiting state?
(进程进入等待状态有哪几种方式？)[德国某著名软件咨询企业2005年面试题]

解析：操作系统问题。

答案：

CPU 调度给优先级更高的 Thread(线程)，原先 Thread 进入 Waiting(等待) 状态。

阻塞的 Thread 获得资源或者信号，进入 Waiting 状态。

在时间片轮转的情况下，如果时间片到了，也将进入等待状态。

16.2 图形学

面试例题1：下列说法错误的是：()。

- A. Direct3DCreat9(D3D_SDK_VERSION); 此函数返回 D3D 实例
- B. IDirect3D9::CreateDevice(); 此函数创建设备实例
- C. IDirect3D9::BeginScene()与 IDirect3D9::EndScene()要求成对使用，

且可以嵌套使用

- D. 在绘制图形之后必须调用函数 IDirect3D9::Present()把后台缓存里的数据处理到前台，才能绘制出图形

解析：图形学问题。

答案：C 错误，因为不能嵌套使用，每对 BeginScene 和 EndScene 之间不能再有其他相似调用。

面试例题 2：有关纹理的说法错误的是：（ ）。

- A. 纹理坐标一般用 (V, U) 来表示，分别对应纹理的宽和高
- B. bmp、tga、jpg 类型的文件都可以加载到纹理变量中
- C. 网格若想贴上纹理需要有纹理坐标
- D. 在自定义格式中可以提供多组纹理坐标

解析：纹理坐标一般用 V,U 来表示，分别对应纹理的宽和高这句不正确。

纹理坐标的范围一般是[0,1]，纹理的宽和高需要规格化才能表示成纹理坐标。

答案：A。

面试例题 3：下列说法不正确的有：（ ）。

- A. Alpha 测试不能实现半透明
- B. Alpha 测试在速度上优于 Alpha 混合
- C. SetRenderState(D3DRS_ALPHAFUNC, D3DCMP_GREATER)是设置 Alpha 混合的混合函数
- D. Alpha 混合可以生成半透明的效果
- E. Alpha 测试中通过测试则绘制像素，否则不绘制

解析：使用 Alpha 融合技术可以实现半透明的效果。在进行 Alpha 融合的时候，要进行 Alpha 测试，我们可以指定不同的测试策略。

- A. Alpha 是透明或者不透明的，不能是半透明的。
- B. 对。
- C. 错误，这是设置 Alpha 测试的混合函数。
- D. 对。

E. 对。

答案：C。

16.3 内存管理

面试例题 1：垃圾回收的优点和原理是什么？并考虑 2 种回收机制。

答案：Java 语言中一个显著的特点就是引入了垃圾回收机制，使 C++ 程序员最头疼的内存管理的问题迎刃而解，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有垃圾回收机制，Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。垃圾回收可以有效地防止内存泄露，有效地使用可以使用的内存。垃圾回收器通常作为一个单独的低级别的线程运行，在不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时地调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收、标记垃圾回收和增量垃圾回收。

扩展知识（垃圾回收技术简介）

和生活中环卫工人们清运垃圾的工作相似，软件开发里的垃圾回收机制其实是一种自动打扫和清除内存垃圾的技术，它可以有效防范动态内存分配中可能发生的两个危险：

- 因内存垃圾过多而引发的内存耗尽（这和生活垃圾堵塞排污管道的危险没有什么本质的不同）。
- 不恰当的内存释放所造成的内存非法引用（这类似于我们在生活中买到了一瓶已经过期 3 年的牛奶）。

据历史学家们介绍，4000 多年前的古埃及人已经在城市里建设了完善的排污和垃圾清运设施，1000 多年前的中国人更是修筑了当时世界上保洁能力最强的都市——长安。今天，当我们在软件开发中体验自动垃圾收集的便捷与舒适时，我们至少应当知道，这种拒绝杂乱、追求整洁的“垃圾收集”精神其实是人类自古以

来就已经具备了的。

1. 拓荒时代

国内的程序员大多是在 Java 语言中第一次感受到垃圾收集技术的巨大魅力的，许多人也因此把 Java 和垃圾收集看成了密不可分的整体。但事实上，垃圾收集技术早在 Java 语言问世前 30 多年就已经发展和成熟起来了，Java 语言所做的不过是把这项神奇的技术带到了广大程序员们身边而已。

如果一定要为垃圾收集技术找一个孪生兄弟，那么，LISP 语言才是当之无愧的人选。1960 年前后诞生于 MIT 的 LISP 语言是第一种高度依赖于动态内存分配技术的语言：LISP 中几乎所有的数据都以“表”的形式出现，而“表”所占用的空间则是在堆中动态分配得到的。LISP 语言先天就具有的动态内存管理特性要求 LISP 语言的设计者必须解决堆中每一个内存块的自动释放问题（否则，LISP 程序员就必然被程序中不计其数的 free 或 delete 语句淹没），这直接导致了垃圾收集技术的诞生和发展。在了解垃圾收集算法的起源之前，有必要先回顾一下内存分配的主要方式。我们知道，大多数主流的语言或运行环境都支持 3 种最基本的内存分配方式，它们分别是：

- 静态分配 (Static Allocation) 静态变量和全局变量的分配形式。我们可以把静态分配的内存看成是家里的耐用家具。通常，它们无须释放和回收，因为没人会天天把大衣柜当做垃圾扔到窗外。
- 自动分配 (Automatic Allocation) 在栈中为局部变量分配内存的方法，栈中的内存可以随着代码块退出时的出栈操作被自动释放。这类似于到家中串门的访客，天色一晚就要各回家，除了个别不识时务者以外，我们一般没必要把客人捆在垃圾袋里扫地出门。
- 动态分配 (Dynamic Allocation) 在堆中动态分配内存空间以存储数据的方式。堆中的内存块好像我们日常使用的餐巾纸，用过了就得扔到垃圾箱里，否则屋内就会满地狼藉。像我这样的懒人做梦都想有一台家用机器人跟

在身边打扫卫生。在软件开发中，如果你懒得释放内存，那么你也需要一台类似的机器人——这其实就是一个由特定算法实现的垃圾收集器。也就是说，下面提到的所有垃圾收集算法都是在程序运行过程中收集并清理废旧“餐巾纸”的算法，它们的操作对象既不是静态变量，也不是局部变量，而是堆中所有已分配的内存块。

1) 引用计数 (Reference Counting) 算法

1960 年以前，人们为胚胎中的 LISP 语言设计垃圾收集机制时，第一个想到的算法是引用计数算法。拿餐巾纸的例子来说，这种算法的原理大致可以描述为：

午餐时，为了把脑子里突然跳出来的设计灵感记下来，我从餐巾纸袋中抽出一张餐巾纸，打算在上面画出系统架构的蓝图。按照“餐巾纸使用规约之引用计数版”的要求，画图之前，我必须先在餐巾纸的一角写上计数值 1，以表示我在使用这张餐巾纸。这时，如果你也想看看我画的蓝图，那你就要把餐巾纸上的计数值加 1，将它改为 2，这表明目前有 2 个人在同时使用这张餐巾纸（当然，我是不会允许你用这张餐巾纸来擦鼻涕的）。你看完后，必须把计数值减 1，表明你对该餐巾纸的使用已经结束。同样，当我将餐巾纸上的内容全部誊写到笔记本上之后，我也会自觉地把餐巾纸上的计数值减 1。此时，不出意外的话，这张餐巾纸上的计数值应当是 0，它会被垃圾收集器（假设那是一个专门负责打扫卫生的机器人）捡起来扔到垃圾箱里，因为垃圾收集器的唯一使命就是找到所有计数值为 0 的餐巾纸并清理它们。

引用计数算法的优点和缺陷同样明显。这一算法在执行垃圾收集任务时速度较快，但算法对程序中每一次内存分配和指针操作提出了额外的要求（增加或减少内存块的引用计数）。更重要的是，引用计数算法无法正确释放循环引用的内存块。对此，D. Hillis 有一段风趣而精辟的论述：

一天，一个学生走到 Moon 面前说：“我知道如何设计一个更好的垃圾收集器了。我们必须记录指向每个节点的指针数目。” Moon 耐心地给这位学生讲了下面这个故事：“一天，一个学生走

到 Moon 面前说：‘我知道如何设计一个更好的垃圾收集器了。’……”

D. Hillis 的故事和我们小时候常说的“从前有座山，山上有座庙，庙里有个老和尚”的故事有异曲同工之妙。

这说明，单是使用引用计数算法还不足以解决垃圾收集中的所有问题。正因为如此，引用计数算法也常常被研究者们排除在

高（标记和清除是两个相当耗时的过程）等诸多缺陷，但在后面的讨论中，我们可以看到，几乎所有现代垃圾收集算法都是标记—清除思想的延续，仅此一点，J. McCarthy 等人在垃圾收集技术方面的贡献就丝毫不亚于他们在 LISP 语言上的成就了。

3) 复制 (Copying) 算法

为了解决标记—清除算法在垃圾收集效率方面的缺陷，M. L. Minsky 于 1963 年发表了著名的论文《一种使用双存储区的 LISP 语言垃圾收集器 (A LISP Garbage Collector Algorithm Using Serial Secondary Storage)》。M. L. Minsky 在该论文中描述的算法被人们称为复制算法，它也被 M. L. Minsky 本人成功地引入到了 LISP 语言的一个实现版本中。

复制算法别出心裁地将堆空间一分为二，并使用简单的复制操作来完成垃圾收集工作。这个思路相当有趣。借用餐巾纸的比喻，我们可以这样理解 M. L. Minsky 的复制算法：

餐厅被垃圾收集机器人分成南区和北区两个大小完全相同的部分。午餐时，所有人都先在南区用餐（因为空间有限，用餐人数自然也将减少一半），用餐时可以随意使用餐巾纸。当垃圾收集机器人认为有必要回收废旧餐巾纸时，它会要求所有用餐者以最快的速度从南区转移到北区，同时随身携带自己正在使用的餐巾纸。等所有人都转移到北区之后，垃圾收集机器人只要简单地把南区中所有散落的餐巾纸扔进垃圾箱就算完成任务了。

下一次垃圾收集的工作过程也大致类似，唯一的不同只是人们的转移方向变成了从北区到南区。如此循环往复，每次垃圾收集都只需简单地转移（也就是复制）一次，垃圾收集速度无与伦比——当然，对于用餐者往返奔波于南北两区之间的辛劳，垃圾收集机器人是决不会流露出丝毫怜悯的。

M. L. Minsky 的发明绝对算得上一种奇思妙想。分区、复制的思路不仅大幅度提高了垃圾收集的效率，而且也将原本繁纷复杂的内存分配算法变得前所未有地简明和扼要（既然每次内存回收都是对整个半区的回收，内存分配时也就不用考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存就可以了），这简直是个

奇迹！不过，任何奇迹的出现都有一定的代价。在垃圾收集技术中，复制算法提高效率的代价是人为地将可用内存缩小了一半。实话实说，这个代价未免也太高了些。无论优缺点如何，复制算法在实践中都获得了可以与标记—清除算法相比拟的成功。

至此，垃圾收集技术的三大传统算法（引用计数算法、标记—清除算法和复制算法）都在 1960 年前后相继问世，3 种算法各有所长，也都存在致命的缺陷。从 20 世纪 60 年代后期开始，研究者的主要精力逐渐转向对这 3 种传统算法的改进或整合，以扬长避短，适应程序设计语言和运行环境对垃圾收集的效率和实时性所提出的更高要求。

2. 走向成熟

从 20 世纪 70 年代开始，随着科学的研究和应用实践的不断深入，人们逐渐意识到，一个理想的垃圾收集器不应当在运行时导致应用程序的暂停，不应额外占用大量的内存空间和 CPU 资源，而 3 种传统的垃圾收集算法都无法满足这些要求。人们必须提出更新的算法或思路，以解决实践中碰到的诸多难题。当时，研究者的努力目标包括：

第一，提高垃圾收集的效率。使用标记—清除算法的垃圾收集器在工作时要消耗相当多的 CPU 资源。早期的 LISP 运行环境收集内存垃圾的时间竟占了系统总运行时间的 40%！——垃圾收集效率的低下直接造就了 LISP 语言在执行速度方面的坏名声。直到今天，许多人还条件反射似地误以为所有 LISP 程序都奇慢无比。

第二，减少垃圾收集时的内存占用。这一问题主要出现在复制算法中。尽管复制算法在效率上获得了质的突破，但牺牲一半内存空间的代价仍然是巨大的。在计算机发展的早期，在内存价格以“KB”计算的日子里，浪费客户的一半内存空间简直就是在变相敲诈或拦路抢劫。

第三，寻找实时的垃圾收集算法。无论执行效率如何，3 种传统的垃圾收集算法在执行垃圾收集任务时都必须打断程序的当前工作。这种因垃圾收集而造成的延时是许多程序，特别是执行关键任务的程序没有办法容忍的。如何对传统算法进行改进，以便

实现一种在后台悄悄执行，不影响或至少看上去不影响当前进程的实时垃圾收集器，显然是一件更具挑战性的工作。

研究者们探寻未知领域的决心和研究工作的进展速度同样令人惊奇：在20世纪70年代到20世纪80年代的短短十几年中，一大批在实用系统中表现优异的新算法和新思路脱颖而出。正是因为有了这些日趋成熟的垃圾收集算法，今天的我们才能在Java或.NET提供的运行环境中随心所欲地分配内存块，而不必担心空间释放时的风险。

1) 标记—整理 (Mark-Compact) 算法

标记—整理算法是标记—清除算法和复制算法的有机结合。把标记—清除算法在内存占用上的优点和复制算法在执行效率上的特长综合起来，这是所有人都希望看到的结果。不过，两种垃圾收集算法的整合并不像1加1等于2那样简单，我们必须引入一些全新的思路。1970年前后，G. L. Steele、C. J. Cheney 和 D. S. Wise等研究者陆续找到了正确的方向，标记—整理算法的轮廓也逐渐清晰了起来：在我们熟悉的餐厅里，这一次，垃圾收集机器人不再把餐厅分成南北两个区域了。需要执行垃圾收集任务时，机器人先执行标记—清除算法的第一个步骤，为所有使用中的餐巾纸画好标记，然后，机器人命令所有就餐者带上有标记的餐巾纸向餐厅的南面集中，同时把没有标记的废旧餐巾纸扔向餐厅北面。这样一来，机器人只需站在餐厅北面，怀抱垃圾箱，迎接扑面而来的废旧餐巾纸就行了。

实验表明，标记—整理算法的总体执行效率高于标记—清除算法，又不像复制算法那样需要牺牲一半的存储空间，这显然是一种非常理想的结果。在许多现代的垃圾收集器中，人们都使用了标记—整理算法或其改进版本。

2) 增量收集 (Incremental Collecting) 算法

对实时垃圾收集算法的研究直接导致了增量收集算法的诞生。最初，人们关于实时垃圾收集的想法是这样的：为了进行实时的垃圾收集，可以设计一个多进程的运行环境，比如用一个进程执行垃圾收集工作，另一个进程执行程序代码。这样一来，垃

圾收集工作看上去就仿佛是在后台悄悄完成的，不会打断程序代码的运行。在收集餐巾纸的例子中，这一思路可以被理解为：垃圾收集机器人在人们用餐的同时寻找废弃的餐巾纸并将它们扔到垃圾箱里。这个看似简单的思路会在设计和实现时碰上进程间冲突的难题。比如说，如果垃圾收集进程包括标记和清除两个工作阶段，那么，垃圾收集器在第一阶段中辛苦标记出的结果很可能被另一个进程中的内存操作代码修改得面目全非，以至于第二阶段的工作没有办法开展。

增量收集算法的基础仍是传统的标记—清除和复制算法。增量收集算法通过对进程间冲突的妥善处理，允许垃圾收集进程以分阶段的方式完成标记、清理或复制工作。详细分析各种增量收集算法的内部机理是一件相当烦琐的事情，在这里，读者们需要了解的仅仅是：H. G. Baker 等人的努力已经将实时垃圾收集的梦想变成了现实，我们再也不用为垃圾收集打断程序的运行而烦恼了。

3) 分代收集(Generational Collecting)算法

分代收集算法和大多数软件开发技术一样，统计学原理总能在技术发展的过程中起到强力催化剂的作用。1980 年前后，善于在研究中使用统计分析知识的技术人员发现，大多数内存块的生存周期都比较短，垃圾收集器应当把更多的精力放在检查和清理新分配的内存块上。这个发现对于垃圾收集技术的价值可以用餐巾纸的例子概括如下：

如果垃圾收集机器人足够聪明，事先摸清了餐厅里每个人在用餐时使用餐巾纸的习惯——比如有些人喜欢在用餐前后各用掉一张餐巾纸，有的人喜欢自始至终攥着一张餐巾纸不放，有的人则每打一个喷嚏就用去一张餐巾纸——机器人就可以制定出更完善的餐巾纸回收计划，并总是在人们刚扔掉餐巾纸没多久时就把垃圾捡走。

分代收集算法通常将堆中的内存块按寿命分为两类：年老的和年轻的。垃圾收集器使用不同的收集算法或收集策略，分别处理这两类内存块，并特别地把主要工作时间花在处理年轻的内存块上。分代收集算法使垃圾收集器在有限的资源条件下，可以更

为有效地工作——这种效率上的提高在今天的 Java 虚拟机中得到了最好的证明。

3. 应用浪潮

LISP 是垃圾收集技术的第一个受益者，但显然不是最后一个。在 LISP 语言之后，许许多多传统的、现代的、后现代的语言已经把垃圾收集技术拉入了自己的怀抱。

随便举几个例子吧：诞生于 1964 年的 Simula 语言，1969 年的 Smalltalk 语言，1970 年的 Prolog 语言，1973 年的 ML 语言，1975 年的 Scheme 语言，1983 年的 Modula-3 语言，1986 年的 Eiffel 语言，1987 年的 Haskell 语言——它们都先后使用了自动垃圾收集技术。当然，每一种语言使用的垃圾收集算法可能不尽相同，大多数语言和运行环境甚至同时使用了多种垃圾收集算法。但无论怎样，这些实例都说明，垃圾收集技术从诞生的那一天起就不是一种曲高和寡的“学院派”技术。对于我们熟悉的 C 和 C++ 语言，垃圾收集技术一样可以发挥巨大的功效。正如我们在学校中就已经知道的那样，C 和 C++ 语言本身并没有提供垃圾收集机制，但这并不妨碍我们在程序中使用具有垃圾收集功能的函数库或类库。我们可以在 C 语言或 C++ 语言中使用该函数库完成自动垃圾收集功能，必要时，甚至还可以让传统的 C/C++ 代码与使用自动垃圾收集功能的 C/C++ 代码在一个程序里协同工作。

1995 年诞生的 Java 语言在一夜之间将垃圾收集技术变成了软件开发领域里最为流行的技术之一。从某种角度来说，我们很难分清究竟是 Java 从垃圾集中受益，还是垃圾收集技术本身借 Java 的普及而扬名。值得注意的是，不同版本的 Java 虚拟机使用的垃圾收集机制并不完全相同，Java 虚拟机其实也经过了一个从简单到复杂的发展过程。在 Java 虚拟机的 1.4.1 版中，人们可以体验到的垃圾收集算法就包括分代收集、复制收集、增量收集、标记—整理、并行复制（Parallel Copying）、并行清除（Parallel Scavenging）、并发（Concurrent）收集等多种。Java 程序运行速度的不断提升在很大程度上应该归功于垃圾收集技术的发展与完善。尽管历史中已经有许多由各种垃圾回收机制所引起的悲剧发生

系统出现，但 Microsoft .NET 却是第一种真正实用化的、包含了垃圾收集机制的通用语言运行环境。

事实上，.NET 平台上的所有语言，包括 C#、Visual Basic .NET、Visual C++ .NET、J# 等，都可以通过几乎完全相同的方式使用.NET 平台提供的垃圾收集机制。我们似乎可以断言，.NET 是垃圾收集技术在应用领域里的一次重大变革，它使垃圾收集技术从一种单纯的技术变成了应用环境乃至操作系统中的一种内在文化。这种变革对未来软件开发技术的影响力也许要远远超过.NET 平台本身的商业价值。

4. 大势所趋

今天，致力于垃圾收集技术研究的人们仍在不懈努力，他们的研究方向包括分布式系统的垃圾收集、复杂事务环境下的垃圾收集、数据库等特定系统的垃圾收集，等等。

面试例题 2：What is "cache" in CPU and "cache" in OS? (CPU 中的缓存和操作系统中的缓存分别是什么？) [美国某著名计算机嵌入式公司面试题]

答案：

1. 快表——Cache 在 OS 中运用的典型范例

在操作系统中，为提高系统的存取速度，在地址映射机制中增加一个小容量的联想寄存器（相联存储器），即快表，用来存放当前访问最频繁的少数活动页面的页号。当某用户需要存取数据时，根据数据所在的逻辑页号在快表中找到其对应的内存块号，再联系页内地址，形成物理地址。如果在快表中没有相应的逻辑页号，则地址映射仍可以通过内存中的页表进行，得到空闲块号后须将该块号填入快表的空闲块中。如果快表中没有空闲块，则根据淘汰算法淘汰某一行，再填入新的页号和块号。

快表查找内存块的物理地址消耗的时间大大降低了，使得系统效率得到了极大的提高。

例如，Linux 使用页面 Cache 的目的是加快对磁盘上文件的访问。内存映射文件以每次一页的方式读出并将这些页面存储在页面 Cache 中。

2. 高速缓冲存储器（Cache）——Cache 在 CPU 中运用的典型范例

CPU 的执行速度越来越快，系统架构越来越先进，而主存的结构和存取速度改进则较慢，因此，高速缓存技术将越来越重要。

高速缓冲存储器（Cache）是位于 CPU 与内存之间的临时存储器，它的容量比内存小但交换速度快。在 Cache 中的数据是内存中的一小部分，但这一小部分是短时间内 CPU 即将访问的。当 CPU 调用大量数据时，就可避开内存直接从 Cache 中调用，从而加快读取速度。由此可见，在 CPU 中加入 Cache 是一种高效的解决方案，这样整个内存储器（Cache+内存）就变成了既有 Cache 的高速度又有内存的大容量的存储系统了。Cache 对 CPU 性能的影响很大，这主要是由 CPU 的数据交换顺序和 CPU 与 Cache 间的带宽引起的。

16.4 DOS、Linux、UNIX

面试例题 1：DOS 与 Windows NT 的权限的区别是什么？[美国某著名移动通信企业面试题]

答案：

DOS 是个单任务、单用户的操作系统。当我们打开一台装有 DOS 操作系统的计算机的时候，我们就拥有了这个操作系统的管理员权限，而且，这个权限无处不在。所以，我们只能说 DOS 不支持权限的设置，不能说它没有权限。随着人们安全意识的提高，权限设置随着 NTFS 的发布诞生了。

在 Windows NT 里，用户被分成许多组，组和组之间有不同的权限。当然，一个组的用户和用户之间也可以有不同的权限。NT 中常见的用户组如下。

- Administrators，管理员组。在默认情况下，Administrators 中的用户对计算机/域有不受限制的完全访问权。分配给该组的默认权限允许对整个系统进行完全控制。所以，只有受信任的人员才可成为该组的成员。
- Power Users，高级用户组。Power Users 可以执行除了为 Administrators 组保留的任务外的其他任何操作系统任务。

- Users，普通用户组。这个组的用户无法进行有意或无意的改动。因此，用户可以运行经过验证的应用程序，但不可以运行大多数旧版应用程序。Users 组是最安全的组，因为分配给该组的默认权限不允许成员修改操作系统的设置或用户资料。Users 组提供了一个最安全的程序运行环境。在经过 NTFS 格式化的卷上，默认安全设置旨在禁止该组的成员危及操作系统和已安装程序的完整性。用户不能修改系统注册表设置、操作系统文件或程序文件。Users 可以关闭工作站，但不能关闭服务器。Users 可以创建本地组，但只能修改自己创建的本地组。
- Guests，来宾组。按默认值，来宾与普通用户组的成员有同等访问权，但来宾账户的限制更多。
- Everyone，顾名思义，所有的用户，这个计算机上的所有用户都属于这个组。

第 17 章

数据库与 SQL 语言

数 据库面试题主要包括范式、事物、存储过程、SQL 语言，以及索引等诸方面。

17.1 数据库理论

面试例题 1：设有关系 $R(S,D,M)$ ，其函数依赖集 $F=\{S \rightarrow D, D \rightarrow M\}$ 。则关系 R 至多满足_____。[美国某著名搜索引擎公司 Y 面试题]

- A. 1NF B. 2NF C. 3NF D. BCNF

解析：数据库模式的 4 个范式问题。

1NF：第一范式。如果关系模式 R 的所有属性的值域中每一个值都是不可再分解的值，则称 R 属于第一范式模式。如果某个数据库模式都是第一范式的，则称该数据库模式属于第一范式的数据库模式。

第一范式的模式要求属性值不可再分裂成更小部分，即属性项不能是属性组合或由组属性组成。

2NF：第二范式。如果关系模式 R 为第一范式，并且 R 中每一个非主属性完全函数依赖于 R 的某个候选键，则称 R 为第二范式模式。如果某个数据库模式中每个关系模式都是第二范式的，则称该数据库模式属于第二范式的数据库模式。（注：如果 A 是关系模式 R 的候选键的一个属性，则称 A 是 R 的主属性，否则称 A 是 R 的非主属性。）

3NF：第三范式。如果关系模式 R 是第二范式，且每个非主属性都

不传递依赖于 R 的候选键，则称 R 是第三范式的模式。如果某个数据库模式中的每个关系模式都是第三范式，则称 R 为 3NF 的数据库模式。

BCNF：BC 范式。如果关系模式 R 是第一范式，且每个属性都不传递依赖于 R 的候选键，那么称 R 为 BCNF 的模式。

4NF：第四范式。设 R 是一个关系模式，D 是 R 上的多值依赖集合。如果 D 中成立非平凡多值依赖 $X \rightarrow\!\!\! \rightarrow Y$ 时，X 必是 R 的超键，那么称 R 是第四范式的模式。

上题属于传递依赖，所以至多满足第二范式。

答案：B。

面试例题 2：存储过程和函数的区别是什么？[美国某著名搜索引擎公司 Y 面试题]

答案：存储过程是用户定义的一系列 SQL 语句的集合，涉及特定表或其他对象的任务，用户可以调用存储过程。而函数通常是数据库已定义的方法，它接收参数并返回某种类型的值，并且不涉及特定用户表。

面试例题 3：事务是什么？[美国某著名搜索引擎公司 Y 面试题]

答案：事务是作为一个逻辑单元执行的一系列操作。一个逻辑工作单元必须有 4 个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能称为一个事务。

1) 原子性

事务必须是原子工作单元。对于其数据修改，要么全都执行，要么全都不执行。

2) 一致性

事务在完成时，必须使所有的数据都保持一致。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。

3) 隔离性

由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的

数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

4) 持久性

事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

面试例题 4：游标的作用是什么？如何知道游标已经到了最后？[中国某著名计算机金融软件公司面试题]

答案：游标用于定位结果集的行。通过判断全局变量@@FETCH_STATUS 可以判断其是否到了最后。通常此变量不等于 0 表示出错或到了最后。

面试例题 5：触发器分为事前触发和事后触发，这两种触发有何区别？语句级触发和行级触发有何区别？[美国某著名计算机软件公司面试题]

答案：事前触发器运行于触发事件发生之前，而事后触发器运行于触发事件发生之后。语句级触发器可以在语句执行前或后执行，而行级触发在触发器所影响的每一行触发一次。

面试例题 6：什么叫做 SQL 注入式攻击？如何防范？[中国台湾某著名杀毒软件公司面试题]

答案：所谓 SQL 注入式攻击，就是攻击者把 SQL 命令插入到 Web 表单的输入域或页面请求的查询字符串中，欺骗服务器执行恶意的 SQL 命令。在某些表单中，用户输入的内容直接用来构造（或者影响）动态 SQL 命令，或作为存储过程的输入参数，这类表单特别容易受到 SQL 注入式攻击。

防范 SQL 注入式攻击闯入并不是一件特别困难的事情，只要在利用表单输入的内容构造 SQL 命令之前，把所有输入内容过滤一番就可以了。过滤输入内容可以按多种方式进行。

第一，替换单引号，即把所有单独出现的单引号改成两个单引号，防止攻击者修改 SQL 命令的命令。

问权限。

第三，对于用来执行查询的数据库账户，限制其权限。用不同的用户账户执行查询、插入、更新、删除操作。由于隔离了不同账户可执行的操作，因而也就防止了原本用于执行 SELECT 命令的地方却被用于执行 INSERT、UPDATE 或 DELETE 命令。

第四，用存储过程来执行所有的查询。SQL 参数的传递方式将防止攻击者利用单引号和连字符实施攻击。此外，它还使得数据库权限可以被限制到只允许特定的存储过程执行，所有的用户输入必须遵从被调用的存储过程的安全上下文，这样就很难再发生注入式攻击了。

第五，检查用户输入的合法性，确信输入的内容只包含合法的数据。数据检查应当在客户端和服务器端都执行。之所以要执行服务器端验证，是为了弥补客户端验证机制脆弱的安全性。在客户端，攻击者完全有可能获得网页的源代码，修改验证合法性的脚本（或者直接删除脚本），然后将非法内容通过修改后的表单提交给服务器。因此，要保证验证操作确实已经执行，唯一的办法就是在服务器端也执行验证。

第六，将用户登录名称、密码等数据加密保存。加密用户输入的数据，然后再将它与数据库中保存的数据比较，这相当于对用户输入的数据进行了“消毒”处理。用户输入的数据不再对数据库有任何特殊的含义，从而也就防止了攻击者注入 SQL 命令。

第七，检查提取数据的查询所返回的记录数量。如果程序只要求返回一个记录，但实际返回的记录却超过一行，那就当做出错处理。

17.2 SQL 语言

面试例题 1：找出表 ppp 里面 num 最小的数，不能使用 min 函数。[中国某著名软件外包企业 2004 年面试题]

答案：

```
select * from ppp where num <= all(select num from ppp)
```

或者：

```
select top 1 num from ppp order by num
```

面试例题2：找出表 ppp 里面最小的数，可以使用 min 函数。[中国某著名软件外包企业 2004 年面试题]

答案：

```
select * from ppp where num = (select Min(num) from ppp)
```

面试例题3：选择表 ppp2 中 num 重复的记录。[中国某著名软件外包企业 2004 年面试题]

答案：

```
select * from ppp2
where num in(select num from ppp2 group by num having(count(num)>1))
```

面试例题4：写出复制表、拷贝表和四表联查的 SQL 语句。[中国某著名软件外包企业 2004 年面试题]

答案：

复制表（只复制结构，源表名：A，新表名：B）：

```
select * into B from A where 1=0
```

拷贝表（拷贝数据，源表名：A，新表名：B）：

```
select * into B from A
```

四表联查：

```
select * from A,B,C,D where 关联条件
```

面试例题5：在 SQL Server 中如何用 SQL 语句建立一张临时表？[中国某著名软件外包企业 2004 年面试题]

答案：

```
create table #Temp(字段1 类型,字段2 类型.....)
```

注意，临时表要在表名前面加“#”。

面试例题6：为了防止在查询记录的时候被其他用户更改记录，应该采用什么方法？如何用查询语句实现该方法？[中国某著名软件外包企业 2004 年面试题]

答案：添加一个“时间戳”类型的字段就可以了。timestamp 这种数据类型会根据当前时间自动产生一个时间字符串，确保这些数在数据库

中是唯一的。timestamp 一般用做给表行加版本戳的机制，存储大小为 8 字节。一个表只能有一个 timestamp 列。每次插入或更新包含 timestamp 列的行时，timestamp 列中的值均会更新。这一属性使 timestamp 列不适合作为键使用，尤其是不能作为主键使用。对行的任何更新都会更改 timestamp 值，从而更改键值。

面试例题 7: Let's say we have a database with 1 one-column table. It contains 1000 same records. Could you please give at least 1 solution to help get records between line 5 and 7. No line number, row id or index etc. (有一个数据库，只有一个表，包含着 1000 个记录，你能想出一种解决方案来把第五到第七行的记录取出来么？不要使用航标和索引。) [德国某著名软件咨询企业 2005 年面试题]

答案：

第一步 建立数据库：

```
declare @i int
set @i=1
create table #T(userid int)
while (@i<=10)
begin
insert into #T
select @i
set @i=@i+1
end

select userid from
(
    select top 3 userid from (select top 7 userid from #T order by
        userid)Ta order by userid desc
) TB order by userid
```

删除数据库：

```
drop table #T
```

提取数据：

```
select top 3 userid from T where userid not in (select top 4 userid from
    T order by userid) order by userid
```

或者：

```
select top 7 userid from T where userid > ANY (select top 4 userid from
    T order by userid) order by userid
```

或者：

```
select top 7 userid from T where userid > ALL (select top 4 userid from
```

```
T order by userid) order by userid
```

17.3 SQL 语言客观题

面试例题1: Which statement shows the maximum salary paid in each job category of each department? (下面哪个SQL语句描述了每一个部门的每个工种的工资最大值?) [中国某著名计算机金融软件公司2005年面试题]

- A. select dept_id, job_cat,max(salary) from employees where salary > max (salary);
- B. select dept_id, job_cat,max(salary) from employees group by dept_id, job_cat;
- C. select dept_id, job_cat,max(salary) from employees;
- D. select dept_id, job_cat,max(salary) from employees group by dept_id;
- E. select dept_id, job_cat,max(salary) from employees group by dept_id, job_cat,salary;

答案: B。

面试例题2: Description of the students table (以下是学生表的字段描述):

sid_id	number
start_date	date
end_date	date

which two function are valid on the start_date column? (关于对start_date字段的使用,以下哪两个函数是合法的?) [中国某著名计算机金融软件公司2005年面试题]

- | | |
|----------------------|-----------------------------|
| A. sum(start_date) | B. avg(start_date) |
| C. count(start_date) | D. avg(start_date,end_date) |
| E. min(start_date) | F. maximum(start_date) |

答案: C, E。

面试例题3: For which two constraints does the Oracle server implicitly create a unique index? (以下哪两种约束的情况下,Oracle数据库会隐性创建一个唯一索引?) [中国某著名计算机金融软件公司2005年面试题]

- A. not null
- B. primary
- C. foreign key
- D. check
- E. unique

答案：B, E。

面试例题 4：In a select statement that includes a where clause, where is the group by clause placed in the select statement? (在 select 语句中包括一个 where 关键词, 请问 group by 关键词一般在 select 语句中什么位置?) [中国某著名计算机金融软件公司 2005 年面试题]

- A. immediately after the select clause (紧跟 select 关键词之后)
- B. before the where clause (在 where 关键词之前)
- C. before the from clause (在 from 关键词之前)
- D. after the order by clause (在 order by 关键词之后)
- E. after the where clause (在 where 关键词之后)

答案：E。

面试例题 5：In a select statement that includes a where clause, where is the order by clause placed in the select statement? (在 select 语句中包括一个 where 关键词, 请问 order by 关键词一般在 select 语句中什么位置?) [中国某著名计算机金融软件公司 2005 年面试题]

- A. immediately after the select clause (紧跟 select 关键词之后)
- B. before the where clause (在 where 关键词之前)
- C. after all clause (在所有关键词之后)
- D. after the where clause (在 where 关键词之后)
- E. before the from clause (在 from 关键词之前)

答案：C。

面试例题 6：Evaluate there two SQL statements.(对比下面两个 SQL 语句。)
[中国某著名计算机金融软件公司 2005 年面试题]

Select last_name,salary from employees order by salary;

Select last_name,salary from employees order by 2 asc;

- A. the same result (相同的结果)
- B. different result (不同的结果)

C. the second statement returns a syntax error (第二个结果会显示错误)

答案：A。

面试例题 7：You would like to display the system date in the format “20051110 14: 44: 17”。Which select statement should you use? (如果你想把时间显示成像“20051110 14: 44: 17”这样的格式，下面哪个 select 语句应该被使用？) [中国某著名计算机金融软件公司 2005 年面试题]

- A. select to_date(sydate,'yearmmdd hh:mm:ss')from dual;
- B. select to_char(sydate,'yearmonthday hh:mi:ss')from dual;
- C. select to_date(sydate,'yyyymmdd hh24:mi:ss')from dual;
- D. select to_char(sydate,'yyyymmdd hh24:mi:ss')from dual;
- E. select to_char(sydate,'yy-mm-dd hh24:mi:ss')from dual;

答案：D。

面试例题 8：Which select statement will the result ‘ello world’ from the string‘Hello world’? (如果要从字符串“Hello world”中提取出“ello world”这样的结果，下面的哪条 SQL 语句适合？) [中国某著名计算机金融软件公司 2005 年面试题]

- A. select substr('Hello World',1)from dual;
- B. select substr(trim('Hello World',1,1))from dual;
- C. select lower(substr('Hello World', 1,6))from dual;

面试例题 10：Select 语句中用来连接字符串的符号是_____。[中国某著名计算机金融软件公司 2005 年面试题]

A. +

B. &

C. ||

D. |

答案：A。

17.4 SQL 语言主观题

面试例题 1：什么是聚集索引？什么是非聚集索引？什么是主键？[中国某著名综合软件公司 2005 年面试题]

答案：

表中经常有一个列或列的组合，其值能唯一地标识表中的每一行。这样的一列或多列称为表的主键。聚集索引确定表中数据的物理顺序。聚集索引类似于电话簿，后者按姓氏排列数据。由于聚集索引规定数据在表中的物理存储顺序，因此一个表只能包含一个聚集索引。但该索引可以包含多个列（组合索引），就像电话簿按姓氏和名字进行组织一样。

非聚集索引与课本中的索引类似。数据存储在一个地方，索引存储在另一个地方，索引带有指针指向数据的存储位置。索引中的项目按索引键值的顺序存储，而表中的信息按另一种顺序存储（这可以由聚集索引规定）。

如果在表中未创建聚集索引，则无法保证这些行具有任何特定的顺序。

面试例题 2：现有一张表，有两个字段：ID, NAME。ID 为主键。如果希望查询出所有拥有 2 个或更多 ID 的 NAME，查询语句应该如何写？[中国某著名综合软件公司 2005 年面试题]

答案：

```
SELECT [Name] FROM [table]
GROUP BY [Name]
HAVING (COUNT([ID]) >= 2)
```

面试例题 3：设供应商供应零件的关系模式为 SP(Sno, Pno, Qty)，其中 Sno 表示供应商号，Pno 表示零件号，Qty 表示零件数量。整个数据库如下表：

Sno	Pno	Qty
-----	-----	-----

168	r1	3
-----	----	---

168	r2	4
168	r3	7
169	r2	1
169	r3	5
170	r4	8
171	r7	5
172	r2	1
172	r7	3

请问下面的 SQL 语句返回值是什么？[中国某著名综合软件公司 2005 年面试题]

```
SELECT *
FROM SP SPY where EXISTS
(SELECT *
FROM SP SPZ
WHERE Sno='169' );
```

解析：与你所想的不一样，在 EXISTS 中的子查询在这个例子中只返回一个值。因为从子查询中返回的行数至少有一行 EXIST 返回为 TRUE，这就使得表中的所有记录都被显示了出来。

答案：

Sno	Pno	Qty
168	r1	3
168	r2	4
168	r3	7
169	r2	1
169	r3	5
170	r4	8
171	r7	5
172	r2	1
172	r7	3

面试例题 4：数据库与上题相同，请问下面的 SQL 语句返回值是什么？[中国某著名综合软件公司 2005 年面试题]

```
SELECT *
FROM SP SPY where EXISTS
(SELECT *
FROM SP SPZ
WHERE Sno='169' and SPZ.Pno=SPY.Pno );
```

解析：你可以使用 EXIST 来检查查询是否确实存在输出，从而实现对查询确实有结果才输出。如果你在相关查询中使用 EXISTS 关键字，

它将会检查你所指出的每一种情况。此题是查找与 168 号供应商所供零件相同编号的情况。

答案：

Sno	Pno	Qty
168	r1	3
168	r2	4
168	r3	7
169	r2	1
169	r3	5
172	r2	1

面试例题 5：数据库与上题相同，请问若想得到与 168 号供应商所供零件相同的全部供应商的全部产品情况，即要返回形如下表的值，SQL 语句应该如何写？[中国某著名综合软件公司 2005 年面试题]

Sno	Pno	Qty
168	r1	3
168	r2	4
168	r3	7
169	r2	1
169	r3	5
172	r2	1
172	r7	3

解析：现在是要返回与 168 供应商所供零件相同的全部供应商的全部产品的情况（产品号可以与其不同），而不是返回与 168 供应商所供零件相同的全部供应商的部分产品情况（与 168 号的产品）。

答案：

```
select * from SP where Sno
in (select Sno from SP
where Pno in (select Pno from SP where Sno='168'));
```

或者：

```
SELECT *
FROM SP SPX
WHERE EXISTS
(SELECT *
FROM SP SPY
WHERE SPY.Sno='168' AND EXISTS
(SELECT *
FROM SP SPZ
WHERE SPZ.Sno=SPX.Sno AND SPZ.Pno=SPY.Pno));
```

第 18 章

计算机网络及分布式系统



络面试例题主要包括局域网、广域网、IP 管理等诸方面。

18.1 网络结构

面试例题 1：在 OSI 参考模型中，物理层的作用是 (1)。对等实体在一次交互作用中传送的信息单位称为 (2)，它包括 (3) 两部分。上下层实体之间的接口称为服务访问点（SAP），网络层的服务访问点也称为 (4)，通常分为 (5) 两部分。[中国某著名综合软件公司 2005 年面试题]

- | | |
|------------------|--------------|
| (1) A. 建立和释放连接 | B. 透明的传输比特流 |
| C. 在物理实体间传送数据帧 | D. 发送和接收用户数据 |
| (2) A. 接口数据单元 | B. 服务数据单元 |
| C. 协议数据单元 | D. 交互数据单元 |
| (3) A. 控制信息和用户数据 | B. 接口信息和用户数据 |
| C. 接口信息和控制信息 | D. 控制信息和校验信息 |
| (4) A. 用户地址 | B. 网络地址 |
| C. 端口地址 | D. 网卡地址 |
| (5) A. 网络号和端口号 | B. 网络号和主机地址 |
| C. 超网号和子网号 | D. 超网号和端口地址 |

解析：网络问题。

OSI 参考模型有 7 层，其分层原则如下：

- ①根据不同层次的抽象分层。
- ②每层应当有一个定义明确的功能。
- ③每层功能的选择应该有助于制定网络协议的国际标准。
- ④各层边界的选择应尽量节省跨过接口的通信量。
- ⑤层数应足够多，以避免不同的功能混杂在同一层中，但也不能太多，否则体系结构会过于庞大。

根据以上标准，OSI 参考模型分为：物理层，数据链路层，网络层，传输层，会话层，表示层，应用层。

物理层涉及到在信道上传输的原始比特流。

数据链路层的主要任务是加强物理层传输原始比特流的功能，使之对应的网络层显现为一条无错线路。发送包把输入数据封装在数据帧，按顺序传出去并处理接收方回送的确认帧。

网络层关系到子网的运行控制，其中一个关键问题是确认从源端到目的端如何选择路由。

传输层的基本功能是从会话层接收数据而且把其分成较小的单元传递给网络层。

会话层允许不同机器上的用户建立会话关系。

表示层用来完成某些特定的功能。

应用层包含着大量人们普遍需要的协议。

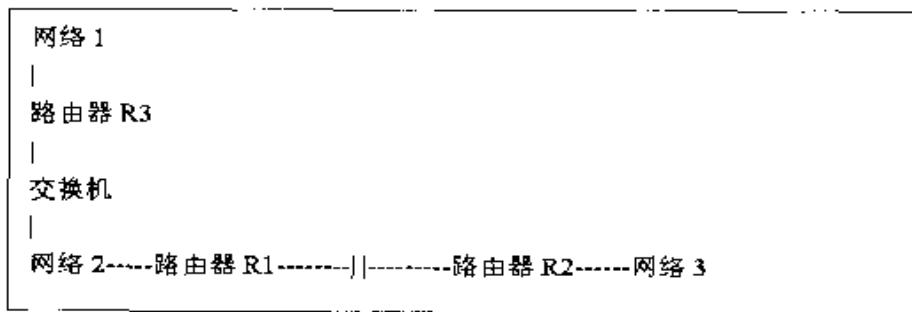
答案：B, C, A, B, B。

面试例题 2：交换和路由的区别是什么？VLAN 有什么特点？[中美合资某著名通信企业面试题]

解析：路由器和交换机是网络上最常见的两个硬件。路由器和交换机有时使我们有点无法区别，为什么呢？从表面看来，我们似乎很容易区别路由器和交换机，因为路由器是网络层硬件设备，主要任务是对 IP 包的处理并用 IP 地址进行路由选路；而交换机是链路层硬件设备，它主要处理帧，通过帧的物理地址来交换信息，并选择把信息发给哪个网络。其实从概念上来讲，它们中的一个通过 IP 选路，一个是通过物理地址来选路的。人们通常用 HUB（集线器）和交换机相比较，而交换机和 HUB 的最大不同

就是能隔离冲突域，这个过程必然要对物理地址进行选路。

区别交换机和路由器是不难的。路由器虽然是IP选路，但是最终要在物理网络上传输，还得把IP地址映射或者转换为物理地址来进行传输，那么就是说必须知道对方的物理地址才能进行通信。我们看看下面这个结构图。



我们看到这里面有3个网络、两个路由器和一个交换机。在这里通过示图来说明交换机和路由器的区别。如果网络2有一台主机要和网络1的主机通信，通过路由器R1选路，把信息传给交换机，交换机用物理目标地址决定发向哪个端口。在开始交换时没有这样的物理地址信息，怎么办呢？交换机只有发一个广播了，很遗憾的是路由器R3会过滤这个广播，这个广播并不能到达网络1。好像我们这个通信并不能实现。在这里我们把交换机当路由器，在互联路由上我们对交换机希望过高。工作在链路层的设备只能以物理地址作为依据，我们来看看网络是怎样处理这个问题的。其实对于路由器来说，它并不知道交换机的存在，即使在物理网络上传输的帧也没有交换机的物理地址，也就是路由器认为到R3路由器是一条通路，中间没有任何东西，而事实上却有一个交换机存在。我们先来看看路由器R1是怎样和R3通信的。当路由器R1没有R3路由器的路由信息时，路由器广播一个信息，这样R2和R3同时会得到这个报文，处理以后，看目的主机是否在网络3。如果路由器R2得不到反应，或是在路由表中找不到这个IP，报文就被丢弃，并回应R1路由器，报文不可到达。而路由器R3通过交换机却找到了主机，并回应路由器，报文可达，并将自己的物理地址交给路由器R1。这样路由器R1有了到R2和R3的路由表了，下次传输时就可以用这个路由表进行选路了。同时路由器R1通过路由器R2和R3知道了整个网络的拓扑结构，这样路由器就能智能地处理路由了。

我们再来看看交换机是一个什么角色。在这里交换机连接了 3 个路由器，正因为路由器在物理网络上传输帧时只有对方的路由器的物理地址而没有交换机的物理地址，这样，当这个信息传到交换机时就对帧头的目的地址分析，然后看对应表中有没有这个物理地址的信息。如果没有，就发一个广播，要求有这个地址的设备给出回应。交换机从回应的信息中得到那个设备的物理地址并和交换机端口建立对应关系，然后把数据发向对方。整个过程就算完成了。

这是关键的一点。我们想想如果路由器是以交换机的地址作为下一站地址，那么情况是怎么样呢？因为交换机没有处理 IP 包的能力，它只能通过帧中的物理地址来选择交换，此时交换机发现没有任何目的地的信息，那么这个包就到此为止了，交换机最终不得不丢弃它。所以从 IP 层去看交换机的话，显然是不可见的。对于路由器来说，它完全把交换机看成了线路中的一部分，与一条直通的线路没有什么区别。也就是说交换机所做的工作，路由器是根本无法察觉的。

答案：交换是指转发和过滤帧，是交换机的工作，它在 OSI 参考模型的第二层。而路由是指网络线路当中非直连的链路，它是路由器的工作，在 OSI 参考模型的第三层。交换和路由的区别很多。首先，交换是不需要 IP 的，而路由需要，因为 IP 就是第三层的协议，第二层需要的是 MAC 地址；再有，第二层的技术和第三层不一样，第二层可以做 VLAN、端口捆绑等，第三层可以做 NAT、ACL、QOS 等。

VLAN 是虚拟局域网的英文缩写，它是一个纯二层的技术，它的特点有三：控制广播，安全，灵活性和可扩展性。

18.2 TCP/IP

面试例题 1：If we divide the network 40.15.0.0 into two subnets, and the first one is 40.15.0.0/17, then the second subnet will be _____. (如果把一个网络 40.15.0.0 分为两个子网，第一个子网是 40.15.0.0/17，那么第二个子网将会是_____) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. 40.15.1.0/17 B. 40.15.2.0/16

- C. 40.15.100.0/17 D. 40.15.128.0/17

解析：让主网分成两个网段，子网掩码分别是 0xff 0xff 0x80 0x00 和 0xff 0xff 0x00 0x00。

答案：D。

面试例题 2：If a worm scans the hosts in Class A IP address space on a home PC, it is quite probably that the host will receive a lot of _____. (如果一个蠕虫病毒攻击了一个家用 PC 机的 A 类地址主机的话，这个地址最有可能接收很多 _____.) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. HTTP response packet (HTTP 回应包)
- B. DNS response packet (DNS 回应包)
- C. ICMP destination unreachable packet (ICMP 目的无法抵达包)
- D. ARP response (ARP 回应)

解析：大量发出 IP 请求，肯定很多不可达，返回不可达错误。

答案：C。

面试例题 3：Before an IP datagram arrived at the destination, it _____. (在一个 IP 数据包到达目的地址之前，它 _____.) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. may be fragmented but never reassembled (可能成为碎片，而且不会重组)
- B. may be fragmented or reassembled (可能成为碎片，或者重组)
- C. can't be fragmented or reassembled (不能成为碎片，或者重组)
- D. can't be fragmented but may be reassembled (不能成为碎片，但是可能会重组)

解析：网络问题，包未达到终点不可能重组，但可以分散成碎片。

答案：A。

面试例题 4：In TCP/IP protocol stack, which of following is taken as an indication of congestion? (在 TCP/IP 协议栈里，如果出现阻塞情况，下面哪种情况最有可能发生？) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. Link failure (连接错误)
- B. Free buffer (释放缓存)
- C. Packet loss (丢包)
- D. Packet error (包错误)

解析：网络阻塞问题。拥塞导致丢包。

答案：C。

面试例题 5：Which protocol is FTP based on for file transfer? (文件传输是基于哪种协议?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. TCP
- B. UDP
- C. Both a and b
- D. None of above

解析：FTP 是有连接的服务，所以必须基于 TCP 协议。

答案：A。

面试例题 6：What is the maximum number of host can exists in a Class C network? (一个 C 类网络最多能容纳多少台主机?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. 256
- B. 254
- C. 255
- D. 128

解析：子网中 IP 为 0~255，其中 0 和 255 不能用，所以是 254 个。

答案：B。

面试例题 7：Which of the following is the E-mail transfer protocol of the Internet? (下面哪一个缩写是电子邮件传输协议?) [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. HTTP
- B. POP
- C. SSL
- D. SMTP

解析：SMTP 的全称是“Simple Mail Transfer Protocol”，即简单邮件传输协议。它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。SMTP 协议属于 TCP/IP 协议族，它帮助每台计算机在发送或中转信件时找到下一个目的地。SMTP 服务器就是遵循 SMTP 协议的邮件发送服务器。

答案：D。

面试例题 8：Ethernet Switch forwarding packet based on _____. (以太网转换控制包是基于_____)。 [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. Destination MAC address (目的 MAC 地址)

- B. Source MAC address (源 MAC 地址)
- C. Destination IP address (目的 IP 地址)
- D. Source and destination IP address (源和目的 IP 地址)

解析: 网络面试例题。

答案: C。

面试例题 9: If the TCP based server program crashed before the client data arrived on the connection that established earlier, the TCP/IP stack may return a _____. (如果 TCP 服务器在客户端发出数据报之前已经崩溃了, TCP/IP 栈可能返回一个_____)。 [中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. RST
- B. FIN
- C. SYN
- D. ACK

解析: SYN 包是 TCP 连接的第一个包, 是非常小的一种数据包。SYN 攻击包括大量此类的包。由于这些包看上去来自实际不存在的站点, 因此无法有效地进行处理。SYN 攻击就是利用 TCP 连接的 3 次握手机制, 但发起攻击端只来一两次握手, 而被攻击端一直在试图完成 TCP 连接, 因此造成资源不足。

答案: C。

面试例题 10: 在 Windows 2000 操作系统中, 配置 IP 地址的命令是(1)。若用 ping 命令来测试本机是否安装了 TCP/IP 协议, 则正确的命令是(2)。如果要列出本机当前建立的连接, 可以使用的命令是(3)。 [中国某著名综合软件公司 2005 年 11 月面试题]

- | | |
|-----------------------|-------------------|
| (1) A. winipcfg | B. ipconfig |
| C. ipcfg | D. winipconfig |
| (2) A. ping 127.0.0.0 | B. ping 127.0.0.1 |
| C. ping 127.0.1.1 | D. ping 127.1.1.1 |
| (3) A. netstat -s | B. netstat -o |
| C. netstat -a | D. netstat -r |

解析: 网络问题。

答案: B, B, C。

18.3 SNMP

面试例题 1：什么是 SNMP 协议？它有什么特点？

答案：SNMP（Simple Network Management Protocol）即简单网络管理协议，它为网络管理系统提供了底层网络管理的框架。SNMP 协议的应用范围非常广泛，诸多种类的网络设备、软件和系统中都有所采用，主要是因为 SNMP 协议有如下几个特点。

首先，相对于其他种类的网络管理体系或管理协议而言，SNMP 易于实现。SNMP 的管理协议、MIB 及其他相关的体系框架能够在各种不同类型的设备上运行，包括低档的个人电脑到高档的大型主机、服务器及路由器、交换器等网络设备。一个 SNMP 管理代理组件在运行时不需要很大的内存空间，因此也就不需要太强的计算能力。SNMP 协议一般可以在目标系统中快速开发出来，所以它很容易在面市的新产品或升级的老产品中出现。尽管 SNMP 协议缺少其他网络管理协议的某些优点，但它设计简单、扩展灵活、易于使用，这些特点大大弥补了 SNMP 协议应用中的其他不足。

其次，SNMP 协议是开放的免费产品。只有经过 IETF 的标准议程批准（IETF 是 IAB 下设的一个组织），才可以改动 SNMP 协议。厂商们也可以私下改动 SNMP 协议，但这样做的结果很可能得不偿失，因为他们必须说服其他厂商和用户支持他们对 SNMP 协议的非标准改进，而这样做却有悖于他们的初衷。

第三，SNMP 协议有很多详细的文档资料（例如 RFC，以及其他的一些文章、说明书等），网络业界对这个协议也有着较深入的了解，这些都是 SNMP 协议进一步发展和改进的基础。

最后，SNMP 协议可用于控制各种设备，比如电话系统、环境控制设备，以及其他可接入网络且需要控制的设备等，这些非传统装备都可以使用 SNMP 协议。

正是由于有了上述这些特点，SNMP 协议已经被认为是网络设备厂商、应用软件开发者及终端用户的首选管理协议。

面试例题 2：SNMP 协议需要专门的连接么？

答案：SNMP 是一种无连接协议。无连接的意思是它不支持像 TELNET 或 FTP 这种专门的连接。通过使用请求报文和返回响应的方式，SNMP 在管理代理和管理员之间传送信息。这种机制减轻了管理代理的负担，它不需要非得支持其他协议及基于连接模式的处理过程。因此，SNMP 协议提供了一种独有的机制来处理可靠性和故障检测方面的问题。

另外，网络管理系统通常安装在一个比较大的网络环境中，其中包括大量的不同种类的网络和网络设备。因此，为划分管理职责，应该把整个网络分成若干个用户分区，可以把满足以下条件的网络设备归为同一个 SNMP 分区：它们可以提供用于实现分区所需要的安全性方面的分界线。SNMP 协议支持这种基于分区名（community string）信息的安全模型，可以通过物理方式把它添加到选定的分区内的每个网络设备上。目前 SNMP 协议中基于分区的身份验证模型被认为是很不牢靠的，它存在一个严重安全问题。主要原因是 SNMP 协议并不提供加密功能，也不保证在 SNMP 数据包交换过程中不能从网络中直接复制分区信息。只需使用一个数据包捕获工具就可把整个 SNMP 数据包解密，这样分区名就暴露无遗。因为这个原因，大多数站点禁止管理代理设备的设置操作。但这样做有一个副作用，就是这样一来只能监控数据对象的值而不能改动它们，限制了 SNMP 协议的可用性。

面试例题 3：SNMP 的命令和报文格式是什么？

答案：SNMP 协议定义了数据包的格式，以及网络管理员和管理代理之间的信息交换，它还控制着管理代理的 MIB 数据对象。因此，可用于处理管理代理定义的各种任务。SNMP 协议之所以易于使用，这是因为它对外提供了 3 种用于控制 MIB 对象的基本操作命令。它们是：Set、Get 和 Trap。

- Set 是一个特权命令，因为可以通过它来改动设备的配置或控制设备的运转状态。
- Get 是 SNMP 协议中使用率最高的一个命令，因为该命令是从网络设备中获得管理信息的基本方式。
- Trap 它的功能就是在网络管理系统没有明确要求的前提下，由管理代理通知网络管理系统有一些特别的情况或问题发生了。

SNMP 协议也定义了执行以上 3 个命令时的报文流，但它没有定义其他的设备管理代理命令，可应用于 MIB 数据对象的操作只有 Set 和 Get 命令，这两个命令的目标是数据对象的值。比如说，SNMP 协议中没有定义 reboot（重启）命令。然而，管理代理软件把 MIB 数据对象和设备的内部命令联系起来，这样就可以实现某些特殊的命令操作。如果现在想要重启某个设备，管理系统就把某个与重启有关的 MIB 数据对象的值设为 1（我们的假定）。这样就会触发管理代理执行重新启动设备的命令，同时还把这个 MIB 数据对象重新设置为原来的状态。

一条 SNMP 报文由 3 个部分组成：版本域（version field）、分区域（community field）和 SNMP 协议数据单元域（SNMP protocol data unit field），数据包的长度是不固定的。

版本域：这个域用于说明现在使用的是哪个版本的 SNMP 协议。目前，version 1 是使用最广泛的 SNMP 协议。

分区域：分区（community）是基本的安全机制，用于实现 SNMP 网络管理员访问 SNMP 管理代理时的身份验证。分区名（Community String）是管理代理的口令，管理员被允许访问数据对象的前提就是网络管理员知道网络代理的口令。如果把管理代理配置成可以执行 Trap 命令，当网络管理员用一个错误的分区名查询管理代理时，系统就发送一个 authenticationFailure trap 报文。

协议数据单元域：SNMP v1 的 PDU 有 5 种类型，有些是报文请求（Request），有些则是响应（Response）。它们包括：GetRequest、GetNextRequest、SetRequest、GetResponse 和 Trap。SNMP v2 又增加了两种 PDU：GetBulkRequest 和 InformRequest。

SNMP 管理员使用 GetRequest 从拥有 SNMP 代理的网络设备中检索信息，SNMP 代理以 GetResponse 消息响应 GetRequest。可以交换的信息很多，如系统的名称、系统自启动后正常运行的时间、系统中的网络接口数，等等。GetRequest 和 GetNextRequest 结合起来使用可以获得一个表中的对象。GetRequest 取回一个特定对象，而使用 GetNextRequest 则是请求表中的下一个对象。使用 SetRequest 可以对一个设备中的参数进行远程配置。Set-Request 可以设置设备的名字，关掉一个端口或清除一个地址解析表中的项。Trap 即 SNMP 陷阱，是 SNMP 代理发送给管理站

的非请求消息。这些消息告知管理站本设备发生了一个特定事件，如端口失败、掉电重启等，管理站可相应地做出处理。

18.4 网络其他问题

面试例题 1：在子网 210.27.48.21/30 中有多少个可用地址？分别是什么？

[中美合资某著名通信企业面试题]

答案：210.27.48.21/30 代表的子网的网络号是 30 位，即网络号是 $210.27.48.21 \& 255.255.255.251 = 210.27.48.20$ ，此子网的地址空间是 2 位，即可以有 4 个地址：210.27.48.20、210.27.48.21、210.27.48.22 和 210.27.48.23。第一个地址的主机号（host number/id）是 0，而主机号 0 代表的是 multicast（多播）地址。最后一个地址的最后两位是 11，主机号每一位都为 1 代表的是广播（broadcast）地址。所以只有中间两个地址可以给 host 使用。

面试例题 2：TTL 是什么？有什么用处？通常哪些工具会用到它？（ping? traceroute? ifconfig? netstat?）[中美合资某著名通信企业面试题]

答案：TTL 的全称是“生存时间（Time To Live）”。简单地说，它表示 DNS 记录在 DNS 服务器上缓存的时间。目前 TTL 采用消息计数的形式，当包每经过一个路由器它就会被减去 1。如果它变成 0，路由器就会把包丢掉。IP 网络往往带有环（loop），比如子网 A 和子网 B 有两个路由器相连，它就是一个 loop。TTL 的主要目的是防止包在有回路的网络上死转，因为包的 TTL 最终会变成 0 而使得此包从网上消失（此时往往路由器会送一个 ICMP 包回来，traceroute 就是根据这个做的）。ping 会送包出去，所以里面有 TTL，但是 ping 不一定非要它不可。traceroute 则是完全因为有 TTL 才能工作的。ifconfig 是用来配置网卡的，netstat -rn 是用来列路由表的，所以都用不着 TTL。

面试例题 3：路由表是做什么用的？在 Linux 环境中怎么配置一条默认路由？[中美合资某著名通信企业面试题]

答案：路由表是用来决定如何将包从一个子网传送到另一个子网的，

换句话说就是用来决定从一个网卡接收到的包应该送到哪一个网卡上去。路由表的每一行至少有目标网络号、netmask、到这个子网应该使用的网卡这 3 条信息。当路由器从一个网卡接收到一个包时，它扫描路由表的每一行，用里面的 netmask 和包里的目标 IP 地址做并逻辑运算 (&) 找出目标网络号。如果此网络号和这一行里的网络号相同，就将这条路由保留下作为备用路由。如果有备用路由了，就在这两条路由里将网络号最长的留下来，另一条丢掉。如此接着扫描下一行直到结束。如果扫描结束仍没有找到任何路由，就用默认路由。确定路由后，直接将包送到对应的网卡上去。在具体的实现中，路由表可能包含更多的信息为选路由算法的细节所用。题外话：路由算法其实效率很差，而且升级困难，解决办法是使用 IP 交换机，比如 MPLS。

在 Linux 上可以用“route add default gw <默认路由器 IP>”命令配置一条默认路由。

面试例题 4：在网络中有两台主机 A 和 B，通过路由器和其他交换设备连接起来，已经确认物理连接正确无误，怎么来测试这两台机器是否连通？如果不通，怎么判断故障点？怎么排除故障？[中美合资某著名通信企业面试题]

答案：测试这两台机器是否连通：从一台机器 ping 另一台机器。

如果 ping 不通，用 traceroute 命令可以确定是哪个路由器不能连通，然后再找问题是否在交换设备或 HUB 或网线等。

面试例题 5：每个路由器在寻找路由时需要知道哪 5 部分信息？[中美合资某著名通信企业面试题]

答案：所有的路由器需要如下信息为报文寻找路由：

- 目的地址 报文发送的目的主机。
- 邻站的确定 指明谁直接连接到路由器的接口上。
- 路由的发现 发现邻站知道哪些网络。
- 选择路由 通过从邻站学习到的信息，提供最优的（与度量值有关）到达目的地的路径。
- 保持路由信息 路由器保存一张路由表，它存储所知道的所有路由信息。

面试例题 6：什么是 BGP？[中美合资某著名通信企业面试题]

答案：BGP (Border Gateway Protocol, 边界网关协议) 是一种在自治系统之间动态交换路由信息的路由协议。一个自治系统的经典定义是在一个管理机构控制之下的一组路由器，它使用 IGP 和普通度量值向其他自治系统转发报文。

在 BGP 中使用自治系统这个术语是为了强调这样一个事实：一个自治系统的管理对于其他自治系统而言是提供一个统一的内部选路计划，它为那些通过它可以到达的网络提供了一个一致的描述。

面试例题 7：自适应网卡只有红灯闪烁，绿灯不亮，这种情况正常吗？[中美合资某著名通信企业面试题]

答案：自适应网卡红灯代表 Link/Act (连通/工作)，即连通时红灯长亮，传输数据时闪烁；绿灯代表 FDX (全双工)，即全双工状态时亮，半双工状态时灭。如果一个半双工的网络设备（如 HUB）和自适应网卡相连，由于这张网卡是自适应网卡，它就会工作在半双工状态，所以绿灯不亮也属于正常情况。

面试例题 8：两台笔记本电脑连起来后 ping 不通，你觉得可能存在哪些问题？[中美合资某著名通信企业面试题]**答案：**

(1) 首先想到的就是你的网线问题。确认网线是否正确，电脑之间连的线和电脑与 HUB 之间连的线分正线、反线，是不同的。但是对于使用千兆位网卡的除外，很多笔记本电脑使用千兆位网卡。千兆位网卡有自动识别的功能，既可以是正线也可以是反线。可以在电脑间通过网线直连，或者通过 HUB 连接多个电脑。

(2) 局域网设置问题。电脑互连是要设置的。看看是否安装了必要的网络协议，最重要的是，IP 地址是否设置正确。互连的时候，最好一台为主，一台为副，主的设为网关。

- (3) 网卡驱动未正确安装。
- (4) 防火墙设置有问题。
- (5) 是否有什么软件阻止 ping 包。

面试例题 9：解释什么叫“透明”？什么叫“网格”？[中美合资某著名通信企业面试题]

答案：透明即向高层隐蔽其具体实现。比如，邮政递送服务对客户是透明的，因为不知道也不需要知道参加递送的邮递员是谁、走的是哪一条路线等具体的实现方法。

“网格”就是有规律的方格集，是虚拟的。网格是把整个因特网整合成一台巨大的超级计算机，实现各种资源的全面共享。当然，网格并不一定非要这么大，也可以构造地区性的网格，如中关村科技园区网格、企事业内部网格、局域网网格，甚至家庭网格和个人网格，等等。网格的根本特征不是它的规模，而是资源共享、消除资源孤岛。在不同地区的计算机各自分析某一项计算的一部分，综合起来计算出同一项东西。

面试例题 10：我们在南京，与深圳的网络是通的，但和北京的网络不通，你以怎样的顺序查找问题所在？[中美合资某著名通信企业面试题]

答案：查找路由器是否可以测试到目的地、所经过的路由器及路由延迟状态。通过这个命令看最后的一个数据包是在哪儿被丢弃或中断的。

面试例题 11：香农定理是什么？[英国某著名计算机图形图像公司面试题]

答案：

香农定理：香农定理描述了有限带宽、有随机热噪声信道的最大传输速率与信道带宽、信号噪声功率比之间的关系。在有随机热噪声的信道上传输数据信号时，数据传输率 R_{\max} 与信道带宽 B 、信噪比 S/N 的关系为： $R_{\max}=B \times \log_2 (1+S/N)$ 。

在信号处理和信息理论的相关领域中，通过研究信号在经过一段距离后如何衰减及一个给定信号能加载多少数据后得到了一个著名的公式，叫做香农（Shannon）定理。它以比特每秒（bps）的形式给出一个链路速度的上限，表示为链路信噪比的一个函数，链路信噪比用分贝（dB）衡量。因此我们可以用香农定理来检测电话线的数据速率。

香农定理由如下的公式给出： $C=B\log_2 (1+S/N)$ ，其中 C 是可得到的链路速度， B 是链路的带宽， S 是平均信号功率， N 是平均噪声功率，信噪比（ S/N ）通常用分贝（dB）表示，分贝数= $10 \times \lg (S/N)$ 。

第 5 部分

综合面试题

Compositive interview questions

本部分主要介绍求职面试过程中出现的第四个重要的板块——英语面试、电话面试和智力测试。这里的英语面试不同于普通的英语面试。就一个程序员而言，最好能够用英文流利地介绍自己的求职经历，这是进外企非常重要的一步。此外还必须对几个常用的问题有相关的解答，比如你最大的缺点是什么。有些问题即便是中文你都很难回答，更何况是用英文去回答。但是求职过程本身就是一个准备的过程，精心地准备等待机会——机会总是垂青于那些精心准备的人。

第 19 章

英 语 面 试

如果你是一个具有战略眼光，期待进入国际性跨国大企业的求职者，本章值得你仔细研读。

英语面试主要考察两个部分：英语口语能力和你做人的特质。外企会有专门的人力资源经理和你聊天，会问你关于人生、经历、团队合作、成功收获、失败教训等一系列问题。这里和一般的技术类面试不同，不是考验你的技术能力，而是考验你的语言能力及情商。所以事先要预测

一下面试官可能提出的问题。本章取材于实际英语面试中关于工作问题、个人特质、未来企划等知识点，并给出了详细的参考答案。请读者结合个人经历修改这些答案以应对可能出现的英语面试。

19.1 面试过程和技巧

现在，不管是国企还是外企，在招聘时都非常看重应聘者的英语交际能力，公司往往通过英语面试，对应聘者的英语交际能力进行考查。我们对参加英语面试的应聘者提出 4 个建议。

建议一：精心设计一个自然的开场白。例如：

C（应聘者）：May I come in? （我能进来吗？）

I（考官）：Yes, please. Oh, you are Jin Li, aren't you? （请进。哦，你是李劲吧？）

C：Yes, I am. （对，我是。）

I：Please sit here, on the sofa. （请坐在沙发上。）

C: Thank you. (谢谢。)

采用“Excuse me. Is this personnel department?”（请问，这里是人力资源部吗？）或“Excuse me for interrupting you. I'm here for an interview as requested.”（不好意思打搅了，我是依约来应聘的。）类似的问法都比较合适。在确定了面试场所和面试官之后，简洁地用“Good morning / afternoon”向面试官打招呼，也能令在场的人提升对你的印象。

建议二：不要害怕外表冷冰冰的考官。

一些用人单位与面试者的最初交流是比较冷冰冰的。例如：

I: Your number and name, please. (请告知你的号码和名字。)

C: My number is sixteen and my name is Zhixin Zhang. (我是 16 号，我叫张志新。)

这时，面试者不要觉得有压力，面试官的态度冷漠并不是针对你个人，你只需要照实简洁地回答即可。在某些情况下，面试官会问一些看起来比较普通和随意的问题，但实际上却是暗藏深意的，例如：

How did you come? A very heavy traffic? (你怎么来的？路上很堵吧？)

你的回答可以是这样：“Yes, it was heavy but since I came here yesterday as a rehearsal, I figured out a direct bus line from my school to your company, and of course, I left my school very early so it doesn't matter to me.”（是的，很堵。但我昨天已经预先来过一次并且找到了一条从我们学校直达贵公司的公交路线，并且在今天提早从学校出发。因此，路上的拥堵并没有影响到我。）这样的回答就点出了你的计划性及细心程度。

建议三：抓住考官问题的关键点回答。

如果应聘者突然听不懂面试官的话，或者问题太复杂，该如何回答？下面是一名学生参加一知名化妆品公司英文面试中的一段：

I: You are talking shop. How will you carry out marketing campaign if we hire you? (你很内行。如果我们雇用你的话，你打算如何开始你的市场工作？)

C: Sorry, sir. I beg your pardon. (对不起，能重复一下吗？)

I: I mean that how will you design your work if we hire you? (我的意思是如果我们决定录用你，你打算如何开展市场工作？)

C: Thank you, sir. I'll organize trade fair and symposium, prepare all marketing materials, and arrange appointments for our company with its business partners. (谢谢。我将组织贸易展销会和研讨会，预备所有营销材料，为公司及其商业伙伴安排见面会。)

I: Do you know anything about this company? (你对本公司的情况了解吗?)

C: Yes, a little. As you mentioned just now, yours is an America-invested company. As far as I know, ×× Company is a world-famous company which produces cosmetics and skincare products. Your cosmetics and skincare products are very popular with women in all parts of the world. (是的，了解一点点。正如你刚才所提到的那样，贵公司是一家美资公司。据我所知，贵公司是一家世界闻名的生产美容护肤品的公司。你们的美容护肤品深受世界各地妇女的欢迎。)

在没听清对方问题的情况下，可以要求对方重复，除了“*I beg your pardon.*”还可以用“*Would you please rephrase your sentence?*”的表达方式。

建议四：巧用过渡语，表明自己用心听问题。

面试者在面试时可以用一些类似“*As you mentioned*”（正如您所说的）或者“*As far as I know*”（据我所知）之类的句子，表示你一直在认真听对方的谈话。此外，你还可以选择“*As it is shown in my resume*”（正如我的简历所提到的）或“*As my previous experience shows*”（如我之前的工作经验所示）之类的表达法。

另外，在面试之前，面试者对应聘公司应有所了解。比如公司的规模、业务、未来发展等，这些往往被学生忽略了。对公司文化理解是否深刻，是你超出其他应聘者的一个亮点。但如果实在不了解，就应根据所知诚实回答。

19.2 关于工作 (About Job)

面试例题 1：Can you sell yourself in two minutes? Go for it. (你能在两分钟内自我推荐吗？大胆试试吧！)

A: With my qualifications and experience, I feel I am hardworking, responsible and diligent in any project I undertake. Your organization could benefit from my analytical and interpersonal skills. (依我的资格和经验，我觉得我对所从事的每一个项目都很努力、负责、勤勉。我的分析能力和与人相处的技巧，对贵单位必有价值。)

面试例题 2：Tell us about your project experiences? (告诉我你的项目经验是什么？)

A: Northwest University Personnel Managing System is a system of auto-manage the persons' information. it is designed by PowerDesign. The project is a C/S architecture system. It is based on a Microsoft SQL database, and the UI is developed by Delphi 7. In this project, I designed the schema of database, programmed database connectivity using Delphi 7 and ADO.

(西北大学人事管理系统是一个自动的人事信息管理系统，它是用 PowerDesign 设计的。整个项目是 C/S 系统架构。它的后台基于微软的 SQL 数据库，前台设计由 Borland 公司的软件 Delphi 7 完成。在这个项目中我负责系统的架构及数据库的连接。)

A: VC# Casus Design was published by Publishing House Of ×××× in 2005.10. As the author I finished most of leading articles. Including GDI+, Multi-Media, Database, Web System, and so on. It's composed of 8 chapters, 50 programs. I concentrate my energy on the book. In addition, I attained much pleasure in process of writing. Now I'm going to write another book named White-collar obtain employment guiding-----flying with C#.

(《VC#案例××××》是××××出版社于 2005 年 10 月出版的书籍。作为第一作者的我完成了绝大多数重要的章节，包括 GDI+、多媒体、数据库、网络系统，等等。本书由 8 章、50 个程序组成。在此书的写作中，我投入了很大的精力，也获得了极大的愉悦。现在我正在准备写另一本书，名字叫《××××指南——与 C#共飞翔》。)

A: Base on ASP.NET+SQL 2000 We finished Northwest University Network Course-selected System. Everybody in this school can select, cancel, query course in network. The project is a B/S architecture system; the code is

developed by Visual C#, and run on the .NET plat. In this project, I used the ADO interface which is provided by the Database program and after that, I joined the testing of whole system.

(基于 ASP.NET+SQL 2000 的平台，我们实现了西北大学网络选课系统。学校内的任何人都能够在网上选择、取消、查询课程。整个项目是 B/S 的系统架构。项目基于.NET 平台，前段代码是用 C#完成的。在项目中我们使用 ADO 接口实现数据库的支持。整个系统架设结束后，我参加了系统的测试工作。)

面试例题 3：Give me a summary of your current job description. (对你目前的工作，能否做个概括的说明。)

A: I have been working as a computer programmer for five years. To be specific, I do system analysis, trouble shooting and provide software support.

(我干了 5 年的电脑程序员。具体地说，我做系统分析、解决问题及软件供应方面的支持。)

面试例题 4：Why did you leave your last job? (你为什么离职呢？)

A: Well, I am hoping to get an offer of a better position. If opportunity knocks, I will take it. (我希望能获得一份更好的工作。如果机会来临，我会抓住。)

A: I feel I have reached the "glass ceiling" in my current job. I feel there is no opportunity for advancement. (我觉得目前的工作已经达到顶峰，即没有升迁的机会。)

面试例题 5：How do you rate yourself as a professional? (作为一位专业人员，你如何评估自己呢？)

A: With my strong academic background, I am capable and competent. (凭借我良好的学术背景，我可以胜任自己的工作，而且我认为自己很有竞争力。)

A: With my teaching experience, I am confident that I can relate to students very well. (依我的教学经验，我相信能与学生相处得很好。)

A: My background has been focused on preparing me for the IT field, so I can exhibit my ability right away. I already have obtained the educational technology and skills I am confident of my ability to learn quickly in any assignment which I'm not familiar for the moment. (我的背景使我非常适合 IT 领域，我会在此展现我的能力。我已经获得了教育方面的能力和技巧，我确信我的能力能够迅速地学习那些我暂时不了解的任务。)

A: I realize that there are many other college students who have the ability to do this job. I also have that ability. But I also bring an additional quality that makes me the very best person for the job -- my attitude for excellence. I am a multi-tasked individual who work well under pressure. My ability to be trained in any area would definitely be a good reason to hire me to work for this firm. (我知道这有很多大学生有能力去做这个工作。我也有这个能力。但是相对于这些人而言，我的积极态度决定了我是做这件工作的最佳人选。我是一个可以承担多种任务压力的人。我可以胜任公司的各个领域并为公司努力工作。)

面试例题 6: What contribution did you make to your current (previous) organization? (你对目前/从前的工作单位有何贡献？)

A: I have finished three new projects, and I am sure I can apply my experience to this position. (我已经完成了 3 个新项目，我相信我能将我的经验用在这份工作上。)

面试例题 7: What do you think you are worth to us? (你如何知道你对我们有价值呢？)

A: I feel I can make some positive contributions to your company in the future. (我觉得我对贵公司能做些积极的贡献。)

面试例题 8: What make you think you would be a success in this position? (你如何知道你能胜任这份工作？)

A: My graduate school training combined with my internship should qualify me for this particular job. I am sure I will be successful. (我在研究所

的训练，加上实习工作，使我适合这份工作。我相信我能成功。)

面试例题 9：Are you a multi-tasked individual? (你是一位可以同时承担数项工作的人吗？) Do you work well under stress or pressure? (你能承受工作上的压力吗？)

A: Yes, I think so. The trait is needed in my current (or previous) position and I know I can handle it well. (这种特点就是我目前（先前）工作所需要的，我知道我能应付自如。)

面试例题 10：What will it take to attain your goals, and what steps have you taken toward attaining them? (你将通过什么手段达到你的成功？你将采取哪些步骤？)

A: I have finished writing a book recently. And currently I am learning a lot of network certification. I obtained my first certification through self-study, so I am learning and keeping up with the latest technology trends. To be successful in this career, one should have to be a excellent problem solver, critical thinker, and team oriented. (我刚刚写完一本书。日前我在做一个网络认证，是完全通过自学完成的第一个认证，所以我一直学习，与最新的技术同步。为了在职场取得成功，我们必须成为一个优秀的解决问题专家，一个具有批判性的思考者，并以团队利益为主导。)

面试例题 11：What steps do you follow to study a problem before making a decision? (在对一项问题进行研究并给出答案之前，你会遵循什么样的步骤？)

A: Following standard models for problem-solving and decision-making can be very helpful. Here are the steps and how I solve a problem with a group project:

- (1) Define the problem to be solved and decision to be made
- (2) Have a plan. To solve a problem, you must establish a plan of attack which leads to a specific goal.
- (3) Solve the problem.

(遵循标准的解决问题和做决策的模式会有很大帮助。以下就是我解决问题的步骤：

- (1) 给要解决的问题和要做的决定做一个限定。
- (2) 为要解决的问题拟一个计划，你应该为具体的目标制定一个进攻计划。
- (3) 解决问题。)

19.3 关于个人 (About Person)

面试例题 1：What is your strongest trait(s)? (你个性上最大的特点是什么？)

A: Helpfulness and caring. (乐于助人和关心他人。)

A: Adaptability and sense of humor. (适应能力和幽默感。)

A: Cheerfulness and friendliness. (乐观和友爱。)

面试例题 2：How would your friends or colleagues describe you? (你的朋友或同事怎样形容你？)

A: (Pause a few seconds) (稍等几秒钟再答，表示慎重考虑。)

They say Mr. Chen is an honest, hardworking and responsible man who deeply cares for his family and friends. (他们说陈先生是位诚实、工作努力、负责任的人，他对家庭和朋友都很关心。)

A: They say Mr. Chen is a friendly, sensitive, caring and determined person. (他们说陈先生是位很友好、敏感、关心他人和有决心的人。)

A: They say I am an active, innovative man, a good team-worker, with rich IT knowledge and developing experience. (他们说我是一个积极、革新的人，是一个很好的同事，并且具备丰富的 IT 知识和研发经验。)

面试例题 3：What personality traits do you admire? (你欣赏哪种性格的人？)

A: (I admire a person who is) honest, flexible and easy-going. (诚实、不死板而且容易相处的人。)

A: (I like) people who possess the "can do" spirit. (有“实际行动”的人。)

面试例题4: What leadership qualities did you develop as an administrative personnel? (作为行政人员, 你有什么样的领导才能?)

A: I feel that learning how to motivate people and to work together as a team will be the major goal of my leadership. (我觉得学习如何把人们的积极性调动起来, 以及如何配合协同的团队精神, 是我行政工作的主要目标。)

A: I have refined my management style by using an open-door policy. (我以开放式的政策改进我的行政管理方式。)

面试例题5: How do you normally handle criticism? (你通常如何处理别人的批评?)

A: Silence is golden. Just don't say anything; otherwise the situation could become worse. I do, however, accept constructive criticism. (沉默是金。不必说什么, 否则情况更糟。不过我会接受建设性的批评。)

A: When we cool off, we will discuss it later. (我会等大家冷静下来再讨论。)

面试例题6: What do you find frustrating in a work situation? (在工作中, 什么事令你不高兴?)

A: Sometimes, the narrow-minded people make me frustrated. (胸襟狭窄的人有时使我泄气。)

A: Minds that are not receptive to new ideas. (不能接受新思想的那些人。)

面试例题7: How do you handle your conflict with your colleagues in your work? (你如何处理与同事在工作中的意见不和?)

A: I will try to present my ideas in a more clear and civilized manner in order to get my points across. (我要以更清楚和文明的方式提出我的看法, 使对方了解我的观点。)

面试例题 8: How do you handle your failure? (你怎样对待自己的失败?)

A: None of us was born "perfect". I am sure I will be given a second chance to correct my mistake. (我们大家生来都不是十全十美的，我相信我有机会改正我的错误。)

面试例题 9: Are you more energized by working with data or by collaborating with other individuals? (你能使组里气氛活跃，并且易于沟通么?)

A: The best thing about working in a group or a team is combining the great minds from different facets. Compared with when you're working alone, communication can generate vitality in the project you're working on. No matter how much wisdom you've got together, without communication, you can't go very far. The perfect situation would be a combination of communication and people, and I'm confident of my abilities in both areas.

(在一个团队或一个组里工作，你最重要的件事就是集思广益，化腐朽为伟大，云集大家的智慧，而不要只是一个人闷头单干。与此同时，沟通是很重要的，可以产生活力。无论你汇集了多高的智慧，如果没有沟通，你将不会成功。完美的境地是把人和沟通有机地组合。我确信我能在两方面都做得很好。)

面试例题 10: What provide you with a sense of accomplishment? (什么会让你有成就感?)

A: Doing my best job for your company. (为贵公司竭力效劳。)

A: Finishing a project to the best of my ability. (尽我所能完成一个项目。)

面试例题 11: If you had a lot of money to donate, where would you donate it to? Why? (假如你有很多钱可以捐赠，你会捐给什么单位？为什么？)

A: I would donate it to the medical research because I want to do something to help others. (我会捐给医药研究方面，因为我要为他人做点事。)

A: I prefer to donate it to educational institutions. (我乐意捐给教育机构。)

19.4 关于未来 (About Future)

面试例题 1: What is most important in your life right now? (眼下你生活中最重要的是什么?)

A: To get a job in my field is most important to me. (对我来说, 能在这个领域找到工作是最重要的。)

A: To secure employment hopefully with your company. (能在贵公司任职对我来说最重要。)

面试例题 2: What current issues concern you the most? (目前什么事是你最关心的?)

A: The general state of our economy and the impact of China's entry to WTO on our industry. (目前中国经济的总体情况及中国入世对我们行业的影响。)

面试例题 3: How long would you like to stay with this company? (你会在本公司服务多久呢?)

A: I will stay as long as I can continue to learn and to grow in my field. (只要我能在我的行业里继续学习和成长, 我就会留在这里。)

面试例题 4: Could you project what you would like to be doing five years from now? (你能预料 5 年后你会做什么吗?)

A: As I have some administrative experience in my last job, I may use my organizational and planning skills in the future. (我在上一个工作中积累了一些行政经验, 我将来也许要运用我组织和计划上的经验和技巧。)

A: I hope to demonstrate my ability and talents in my field adequately. (我希望能充分展示我在这个行业的能力和智慧。)

A: Perhaps, an opportunity at a management position would be exciting. (也许有机会, 我将会从事管理工作。)

如果不正面回答, 也可以说:

It would be premature for me to predict this. (现在对此问题的预测，尚嫌过早。)

面试例题 5：What range of pay-scale are you interested in? (你喜欢哪一种薪水标准?)

A: Money is important, but the responsibility that goes along with this job is what interests me the most. (薪水固然重要，但伴随工作而来的责任更吸引我。)

假如你有家眷，可以说：

To be frank and open with you, I like this job, but I have a family to support. (坦白地说，我喜欢这份工作，不过我必须要负担我的家庭。)

面试例题 6：What specific goals, including those related to your occupation, have you established for your life (career)? (您为您的职业生涯制定了什么样的具体目标?)

A: My specific goals related to my occupation is to work for a company where I can apply my technical and business skills I obtained from college and my past experience. To take advantage of the continuous learning process that goes along with the many technological advances. (关于我的职业的具体目标是：为一个可以让我施展我在大学及过往经验中积累的技术与商业技巧的公司工作。通过技术上的不断学习推进技术创新。)

19.5 其他建议 (Other Tips)

Know something about the organization you are applying to. (了解一些你申请工作单位的情况。)

Dress properly. Don't shake hand with the interviewer until he/she extends his/her hand. (穿着要得体，人家伸手时才握手。)

Don't sit down until invited to do so by the interviewer. (人家未请，先别坐下。)

Make eye-contact with the interviewer during the interview. (面试时，

眼睛要看着对方。)

Listen actively and stay calm. (注意听，保持冷静。)

If invited to a meal, be especially careful about your table manners. (被邀吃饭时，要特别注意餐桌礼节。)

Don't talk with your mouth full. (嘴里有食物，不可开口说话。)

Don't make much noise while you eat. (吃东西不要出声音。)

Don't blow your nose or use the toothpick at table. (不要擤鼻涕或用牙签剔牙。)

Don't appear to be pushy or overly anxious to get a job. (不必过分表现急着要工作。)

Be honest but not too modest. (要诚实，但不必太谦虚。)

Don't put yourself down or cut yourself up. (不可妄自菲薄或自贬。)

19.6 英文面试常用词汇

19.6.1 工作经历相关词汇

work experience 工作经历

work history 工作经历

occupational history 工作经历

professional history 职业经历

employment 经历

employment history 工作经历

experience 经历

business experience 工作经历

specific experience 具体经历

employment record 工作经历

employment experience 工作经历

business background 工作经历

position 职位

job title 职位

responsibilities 职责

adapted to 适应于

accomplish 完成 (任务等)

appointed 被任命的

adept in 善于

analyze 分析

assist 辅助

authorized 委任的，核准的

behave 表现

break the record 打破记录

conduct 经营，处理

nominated 被提名的，被任命的

promote 推销 (商品)，创立 (企业) 等

be promoted to 被提升为

be proposed as 被提名为，被推荐为

advanced worker 先进工作者

duties 职责
second job 第二职业
achievements 工作成就, 业绩
administer 管理

working model 劳动模范
excellent Party member 优秀党员
excellent League member 优秀团员

19.6.2 个人资料相关词汇

name 姓名
alias 别名
pen name 笔名
date of birth 出生日期
birth date 出生日期
born 出生于
birth place 出生地点
age 年龄
native place 籍贯
province 省
city 市
autonomous region 自治区
prefecture 专区
county 县
nationality 民族, 国籍
citizenship 国籍
duel citizenship 双重国籍
address 地址
current address 目前地址
present address 目前地址
permanent address 永久地址
postal code 邮政编码
home phone 住宅电话
office phone 办公电话
business phone 办公电话
Tel. 电话
Sex/gender 性别
weight 体重

male 男
female 女
height 身高
divorced 离异
separated 分居
number of children 子女人数
none 无
street 街
lane 胡同, 巷
road 路
district 区
house number 门牌
health 健康状况
health condition 健康状况
blood type 血型
short-sighted 近视
far-sighted 远视
color-blind 色盲
ID card No. 身份证号码
date of availability 可到职时间
available 可到职
membership 会员, 资格
president 会长
vice-president 副会长
director 理事
standing director 常务理事
secretary general 秘书长
society 学会

marital status 婚姻状况
family status 家庭状况
married 已婚
single/unmarried 未婚

association 协会
research society 研究会

19.6.3 个人品质相关词汇

able 有才干的，能干的
active 主动的，活跃的
adaptable 适应性强的
adroit 灵巧的，机敏的
aggressive 有进取心的
alert 机灵的
ambitious 有雄心壮志的
amiable 和蔼可亲的
amicable 友好的
analytical 善于分析的
apprehensive 有理解力的
aspiring 有志气的，有抱负的
audacious 大胆的，有冒险精神的
capable 有能力的，有才能的
careful 办事仔细的
candid 正直的
charitable 宽厚的
competent 能胜任的
confident 有信心的
conscientious 认真的，自觉的
considerate 体贴的
constructive 建设性的
contemplative 好沉思的
cooperative 有合作精神的
creative 富创造力的
dashing 有一股子冲劲的，有拼搏精神的
dedicated 有奉献精神的
devoted 有献身精神的

humble 恭顺的
humorous 幽默的
impartial 公正的
independent 有主见的
industrious 勤奋的
ingenious 有独创性的
initiative 首创精神
have an inquiring mind 爱动脑筋
intellective 有智力的
intelligent 理解力强的、聪明的
inventive 有发明才能的，有创造力的
just 正直的
kind-hearted 好心的
knowledgeable 有见识的
learned 精通某门学问的
liberal 心胸宽大的
logical 条理分明的
loyal 忠心耿耿的
methodical 有方法的
modest 谦虚的
motivated 目的明确的
objective 客观的
open-minded 虚心的，开明的
orderly 守纪律的
original 有独创性的
painstaking 辛勤的，苦干的，刻苦的
practical 实际的
precise 一丝不苟的

dependable 可靠的	persevering 不屈不挠的
diplomatic 老练的，有策略的	punctual 严守时刻的
disciplined 守纪律的	purposeful 意志坚强的
discreet (在行动，说话等方面) 谨慎的	qualified 合格的
dutiful 尽职的	rational 有理性的
dynamic 精悍的	realistic 实事求是的
earnest 认真的	reasonable 讲道理的
well-educated 受过良好教育的	reliable 可信赖的
efficient 有效率的	responsible 负责的
energetic 精力充沛的	self-conscious 自觉的
enthusiastic 充满热情的	selfless 无私的
expressive 善于表达	sensible 明白事理的
faithful 守信的，忠诚的	sincere 真诚的
forceful (性格) 坚强的	smart 精明的
frank 直率的，真诚的	spirited 生气勃勃的
friendly 友好的	sporting 光明正大的
frugal 俭朴的	steady 稳实的
generous 宽宏大量的	straightforward 老实的
genteel 有教养的	strict 严格的
gentle 有礼貌的	systematic 有系统的
hard-working 勤劳的	strong-willed 意志坚强的
hearty 精神饱满的	sweet-tempered 性情温和的
honest 诚实的	temperate 稳健的
hospitable 殷勤的	tireless 孜孜不倦的

19.6.4 学历相关词汇

educational history 学历	intelligence quotient 智商
curriculum 课程	pass 及格
major 主修	fail 不及格
minor 副修	marks 分数
educational highlights 课程重点部分	grades 分数
curriculum included 课程包括	scores 分数
specialized courses 专门课程	examination 考试
courses taken 所学课程	grade 年级

courses completed 所学课程	class 班级
special training 特别训练	monitor 班长
social practice 社会实践	vice-monitor 副班长
part-time jobs 业余工作	commissary in charge of studies 学习委员
summer jobs 暑期工作	commissary in charge of entertainment 文娱委员
vacation jobs 假期工作	commissary in charge of sports 体育委员
refresher course 进修课程	commissary in charge of physical labor 劳动委员
extracurricular activities 课外活动	Party branch secretary 党支部书记
physical activities 体育活动	League branch secretary 团支部书记
recreational activities 娱乐活动	commissary in charge of organization 组织委员
academic activities 学术活动	commissary in charge of publicity 宣传委员
social activities 社会活动	degree 学位
rewards 奖励	post doctorate 博士后
scholarship 奖学金	doctor (Ph.D) 博士
"Three Goods" student 三好学生	master 硕士
excellent League member 优秀团员	bachelor 学士
excellent leader 优秀干部	student 学生
student council 学生会	graduate student 研究生
off-job training 脱产培训	abroad student 留学生
in-job training 在职培训	returned student 回国留学生
educational system 学制	foreign student 外国学生
academic year 学年	undergraduate 大学肄业生, (尚未取得学位的) 大学生
semester 学期 (美)	senior 大学四年级学生; 高中三年级学生
term 学期 (英)	junior 大学三年级学生; 高中二年级学生
president 校长	sophomore 大学二年级学生; 高中一年级学生
vice-president 副校长	freshman 大学一年级学生
dean 院长	guest student 旁听生 (英)
assistant dean 副院长	auditor 旁听生 (美)
academic dean 教务长	government-supported student 公费生
department chairman 系主任	
professor 教授	
associate professor 副教授	
guest professor 客座教授	
lecturer 讲师	
teaching assistant 助教	

research fellow 研究员
research assistant 助理研究员
supervisor 论文导师
principal 中学校长（美）
headmaster 中学校长（英）
master 小学校长（美）
dean of studies 教务长
dean of students 教导主任
teacher 教师
probation teacher 代课教师
tutor 家庭教师
governess 女家庭教师

commoner 自费生
extern 走读生
day-student 走读生
intern 实习生
prize fellow 奖学金生
boarder 寄宿生
classmate 同班同学
schoolmate 同校同学
graduate 毕业生
excellent Party member 优秀党员
excellent League member 优秀团员

19.6.5 离职原因相关词汇

objective 目标
career objective 职业目标
employment objective 工作目标
position wanted 希望职位
job objective 工作目标
position applied for 申请职位
position sought 谋求职位
position desired 希望职位
for more specialized work 为更专门的工作
for prospects of promotion 为晋升的前途
for higher responsibility 为更高层次的工作责任

for wider experience 为扩大工作经验
due to close-down of company 由于公司倒闭
due to expiry of employment 由于雇佣期满
offered a more challenging opportunity 获得更有挑战性的工作机会
sought a better job 找到了更好的工作
to look for a more challenging opportunity 寻找一个更有挑战性的工作机会
to seek a better job 找一份更好的工作

第 20 章

电 话 面 试

求职时，经常会遭遇电话面试，戏称“触电”。我曾经在开会、洗澡、吃饭、坐车时都接到过电话。问的问题也是五花八门，千奇百怪。

不要轻视电话面试，通常打电话的人也是具有否决权的。俗话说“阎王好见，小鬼难缠”，这里的“阎王”是最终录用你的 HR，而“小鬼”可能是企业授权进行电话面试的外包公司。“小鬼”虽不具备最终录用你的权力，但仍然可以否决你（我就在通过笔试后，倒在 SAP 公司的电话面试上，当时电面我的是一家外包公司）。

电面前，要调整好电话，保证通话质量清晰，不要关机、欠费或超出服务区。如果是越洋电面（我曾经参加过 Motorola 公司和 Sybase 公司的越洋电面），要选择座机，以保证通话质量。电面通常用英文，所以平时一定要注意英语口语的练习。

20.1 电话面试之前的准备工作

一般来说，正规外企在电话面试之前会发邮件来确认你是否有空，并且确认你的电话号码是多少。通常的格式如下：

To help us schedule your interview, please respond with the following information:

When are you available to speak with us?

We generally schedule phone appointments at least three days in advance.

Please suggest several one-hour time slots when you will be available, keeping in mind that we need at least three days of lead time to schedule your interview. Please provide your time zone for the appointment as well.

What phone number should we call?

Confirm the phone number where you can be reached for the phone interview.

(为了帮助我们确认你的面试时间, 请回答下面的问题:

你什么时候有时间接受我们的电话面试?

通常在 3 天内我们可以与你做一个电话的交流。

请提出一个你的空闲时间段, 大概一个小时左右。我们需要 3 天的时间来为你的面试做一个时间表。

我们应拨打哪个电话号码?

确认电话号码以助于你能准确地接到此电话。)

20.2 电话面试交流常见问题

电话面试往往是非常突然的。投递简历后就会接到这样的电话, 它可能在你洗澡时、开会时、吃饭时打来。电话面试的组织方有可能是第三方中介, 因此, 面试问题多半是格式化的、单调的问题。电话开始时, 考官会问你有没有空, 如果你想准备一下, 不妨告诉他没有空, 让他 10 分钟后再打过来, 以便做一些相应的准备。

面试例题 1: Introduce yourself, please. (介绍一下你自己。)

对于上面这个问题你可以选择重点来回答。对于你什么时候上的小学, 什么时候初中毕业就不用讲了。你可以这么说:

A: I was admitted by Northwest University with excellent student record, which was quite satisfied by the teaching staff. (我是从西北大学毕业的, 我的成绩非常优异。来自西北大学不要说 I come from Northwest University, 最好说 I was admitted by Northwest University。)

My field of study is Software and Theory, which is a famous one in China. (我的研究方向是计算机软件与理论, 这个专业是在中国很著名的。)

你说专业的时候可以不说 major, 你可以说 field of study, 研究方向。)

My undergraduate study gave me a wide range of vision. I fulfilled the courses like English, Network and Programming. I have a deep understanding in Programming. (我的本科教育给了我宽广的视野, 我涉猎的课程有英语、网络和程序设计。我对编程方面有很深的见解。)

I developed several professional interests, like History and Micro Economics at my spare time. (我有很多个性爱好, 空闲的时间我研究历史和微观经济学。)

The several years working experience give me full play to my creativity, diligence and intelligence. I believe I can do my job well. (这几年的工作经历使我充满创造力, 勤勉, 有智慧。我相信我能把工作做好。)

面试例题 2：What is your greatest weakness? (你最大的弱点是什么?)

你不应该说你没有任何弱点, 以此来回避这个问题。每个人都有弱点, 最佳策略是承认你的弱点, 但同时表明你在予以改进, 并有克服弱点的计划。可能的话, 你可说出一项可能会给公司带来好处的弱点, 如可说: "I'm such a perfectionist that I won't stop until a job is well done." ("我是一个完美主义者。工作做得不漂亮, 我是不会撒手的。") 或者对于一个学生而言, 缺乏工作经验不是很大的缺点, 你可以直言不讳。

A: I'm lacking of working experience. But I'm taking a course. (我缺乏工作经验, 但我正在学习。)

A: I'm lacking of supervision. But I'm reading a book. (我缺乏远见。但我会用阅读来弥补。)

A: I'm just graduation. (我只是刚毕业。)

面试例题 3：Do you know anything about this company? (你对本公司的情况了解吗?)

A: Yes, a little. As you mentioned just now, yours is an America-invested company. As far as I know, ×× Company is a world-famous company which produces database and applications software products. Your products like PowerBuilder and 美国某著名数据库公司

database products are very popular with company in all parts of the world. (是的，了解一点点。正如你刚才所提到的那样，贵公司是一家美资公司。据我所知，××公司是一家世界闻名的生产数据库产品和应用软件的公司。你们的产品 PowerBuilder 和美国某著名数据库公司数据库深受世界各地公司的欢迎。)

面试例题 4：What kind of programming do you study in your project? What have you learned in this process? (在你的项目中你用到了哪种程序？在此过程中你学到了什么？)

A: Base on ASP.NET+SQL 2000 we finished Northwest Network Course-selected System. Everybody in this school can select, cancel, query course in network. The project is a B/S architecture system; the code is developed by Visual C#, and run on the .NET plat. In this project, I used the ADO interface which is provided by the Database program. And after that, I joined the testing of whole system. (基于 ASP.NET 和 SQL 2000 平台基础，我们完成了西北大学网络选课系统。学校中的每一个人都可以在网上选择、取消、查询课程。这个项目是一个 B/S 结构系统；代码是用 C# 编写的，在.NET 平台上运行。在项目中，我使用 ADO 接口来支持数据库程序。在此之后，我参加了对整个系统的测试。)

In order to achieve the function of background database, I have designed the database including: primary key, foreign key, database connection, data view and so on. I use the SQL language to search delete update data. In this process I encountered a lot of difficulties. But I conquered them as best I could and finally I promoted and enriched myself.

(为了实现后台数据库，我进行了数据库的设计，包括主键、外键的设计、连接、视图及其他，并运用 SQL 语句实现数据的查找、删除和修改。其中我遇到了大量的困难，但我尽我所能去克服它们，最终获得了提升和自我的丰富。)

面试例题 5：In this progress what are your most challenge? (在这个过程中你遇到的最大挑战是什么？你是怎么解决的？)

A: In the network course select system, I once met a intractable problem that the server response the client too slowly, especially when lots and lots of people upload their message at the same time. I have to find out what cause that situation? It was very urgent for me at that time, because there were about 20 thousands students in the school waiting for selecting courses, so I was undertaking great pressure. But after I checked the web log very carefully I finally found that the sticking point was on the submission button .the button has appended too many futile functions and received too much data which caused data redundancy. I suggested it is challenge for me to overcome.

(在网络选课系统中，我遇到了一个非常棘手的问题。即：服务器响应客户端的数据非常缓慢，特别是人比较多的时候。我不得不去寻找是什么造成了这样的结果。而那时对我来说是非常紧急的，大约 2 万多学生正等着选课呢。我承担着很大的压力。我检查了网络日志，最后发现症结出现在提交按钮上。一个提交按钮附加了太多的功能，读取数据量过大，导致了数据冗余。去解决这样一个问题对我来讲是一个挑战。)

A: First I try to disassemble a complicate function into several simple functions. Second, some operations are handle in database instead of web, for example set up a trigger or store procedure. We also can read web log to see how many thread success. If too many thread can not carry out, that imply reading data need to be optimized. (我的解决办法是：把功能分解，放到不同的按钮上面。我们可以设置触发器或者存储过程来解决这些问题。通过日志主要看在同一时刻有多少并发线程，执行成功的线程有多少。如果很大一部分并发线程没有成功，这说明数据的读取操作没有优化。)

面试例题 6：What your largest interest in the project? (你所做的项目中哪个是你最感兴趣的) ?

A: The book VC# **casus design** was full of a lot of instances.

My friends and assistants in lab or group provide lots of material for me. A friend in a bank ask me how to convert Arabic numerals to Chinese numerals: So we set up to handle these question: how to handle decimal, how to handle integer, zero, radix point.

My associate in another group study distribute system plot, he used .NET remoting technique and overcome a lot of difficulties such as how to get long distance object, how to login a channel and so on. I write down this in my book.

I concentrate my energy on the book. In addition, I attained much pleasure in process of writing. I'm interesting in this.

(我所编写的《VC# 案例××××》有很多案例。)

我的朋友、助手给我提供了很多素材。一个在银行的朋友问我如何将阿拉伯数字转化成中文。于是我们下决心解决这个问题：如何处理小数，如何处理整数、零、小数点等问题。

我的一个隔壁教研室的朋友正在研究分布式系统绘图问题。他使用.NET remoting 技术克服了很多的困难，比如说解决了如何进行远距离对象传输的问题，如何注册一个通道，等等。最后我都将其写入了我的书中。

我为这本书付出了很大的精力，与此同时，我在写作过程中获得了很大的快乐。我对这个非常感兴趣。)

面试例题 7：How do you realize the job you apply, what is your blue print in future? (你如何理解你所应聘的职位？你未来几年的规划是什么？)

A: I apply for a QA job ,(QA means query assurance.)I think I'll communicate with R&D, respect their thoughts. And listen to their idea, I'll get to know the schema of the project deeply, Collaborate with my colleagues well and: I hope I'll reveal my ability and talents in my field adequately.

QA is a profession requires much patience, sometimes you cannot get the conclusion you want although you have tested it time and time again, and it will waste your time and you maybe feel frustrated. I think if you wish to be a automatic thorough repeatable independent professional QA, these is a long road from you to go, a lot of troubles for you to overcome, but I 'm sure I can success.

(我应聘的是 QA 职位，QA 意味着质量保证。我们应该很好地与研发部门交流。我计划对项目的结构做深刻的理解，与我的同事们很好地

合作。我希望能充分展示我在这个行业的能力和智慧。

QA 是一个需要耐心的职位，有时即便测试千回也不能得到你想要的结果，有时候使用不当会浪费很多的时间，自己也会觉得很无聊。我想，做一个自动化的、彻底的、独立的、专业的测试人员肯定会有些坎坷，但我坚信我能成功。)

A: Perhaps, an opportunity at a management position would be exciting. As I have some administrative experience in my job, I may use my organizational and planning skills in the future. (也许有机会，我将会从事管理工作。我在上一个工作中积累了一些行政经验，我将来也许要运用我组织和计划上的经验和技巧。)

面试例题 8: What is the largest-scale company you have worked for, and the smallest one? what have you done for them? (你所工作的最大的组织是什么？最小的是什么？都做些什么工作？)

A: In this three years I worked for my tutor, To be specific, I do system analysis, trouble shooting and provide software support. (在 3 年研究生的阶段中我为我的导师工作。具体地说，我做系统分析、解决问题及软件供应方面的支持。)

面试例题 9: What is a Project Manager in your eyes? (在你眼中的项目管理者是什么样的？)

A: The person or firm responsible for the planning, coordination and controlling of a project from inception to completion, meeting the project's requirements and ensuring completion on time, within cost and to required quality standards. A systems analyst with a diverse set of skills—management, leadership, technical, conflict management, and customer relationship—who is responsible for initiating, planning, executing, and closing down a project. (项目管理者是那种负责项目从概念到实施整个过程的计划、协调、控制的人或公司，他（们）能够满足项目的需求，并确保项目在有限成本内按时完成，达到要求的质量标准。要启动、计划、执行并完成一个项目，系统分析师需要具备一些不同的技能，如管理能力、领导能力、技术、

冲突管理及客户关系。他必须为项目的起始、计划、执行，以及项目的完成或中断负责。)

A: The Project Manager defines, plans, schedules, and controls the project. The project plan must include tasks, deliverables and resources – the people who will perform the tasks. The manager will monitor and coordinate the activities of the team, and will review their deliverables. (项目管理者定义、计划、制定、控制整个项目。一个项目计划必须包含目标、可行性、资源及可以执行此项目的人选。管理者应该可以监控和调整整个项目的行动，并时时回顾项目的可行性。)

A: PM is coordinator, assistant and best friend of team members, planner, monitor, reporter of the project or progress. (项目管理者是一个协调者，一个很好的助理，是研发人员最好的朋友，是一个策划者、领导者，是项目流程的报告者。)

面试例题 10: What is your favorite working atmosphere? (你理想的工作环境是什么?)

A: I'd like to work in a harmonious surrounding. Everybody pay great effort to do his job for the company. Accomplish every project with the best of my ability. What provide me with a sense of accomplishment, your effort would won approbation. Working as a team member allows me to gain from others within the group. It also gives the individual to focus their strengths.

(我喜欢工作在一个和谐的环境里。每个人能尽其最大的努力为公司做事，尽其最大的努力完成每一个项目。完成一份工作，我们会得到满足和成就感，你的努力能够得到公司的嘉许。在这个组里作为一员我能从他人身上受益，我也会令他人受益。)

A: My ideal job is one that allows me to combine my technical attributes, business skills, and critical thinking skills in an attempt to help solve problems and create solutions. (我的理想的工作是允许我把技术品质和事业技巧结合在一起，尝试帮助解决问题和寻求解决方案。)

A: I will try my best to arrange my time, I think I will find balance point between work and relax. (我将尽我所能去安排时间，我相信我会在工作

和休闲之间寻找平衡。)

面试例题 11: Do you have the qualifications and personal characteristics necessary for success in your chosen career? (你拥有在你所选择的职业里获得成功的资质和必备的个性特点么?)

A: I believe I have a combination of qualities to be successful in this career. First, I have a strong interest, backed by a solid, well-rounded, state-of-the-art education, especially in a career that is technically oriented. This basic ingredient, backed by love of learning, problem-solving skills, well-rounded interests, determination to succeed and excel, strong communication skills, and the ability to work hard, are the most important qualities that will help me succeed in this career. To succeed, you also need a natural curiosity about how systems work -- the kind of curiosity I demonstrated when I upgraded my two computers recently. Technology is constantly changing, so you must a fast learner just to keep up or you will be overwhelmed. All of these traits combine to create a solid team member in the ever-changing field of information systems. I am convinced that I possess these characteristics and am ready to be a successful team member for your firm. (我相信我具备在这一职业中取得成功的一系列资质。首先，我有极强的爱好，以扎实、广泛、艺术性的教育为背景，特别是在一个以技术为主导的职业领域中。这些基本条件，加上对学习的热爱、解决问题的能力、广泛的兴趣、成功与求胜的决心、很强的沟通技巧、努力工作的能力，都是能够帮助我在职业生涯中取得成功的重要品质。为了成功，我们需要有一种对系统如何运行的出自本能的好奇心，这种好奇心是最近我在升级我的两个计算机的时候发现的。新技术不断取代旧的，所以我们必须成为能跟上步伐的快速学习者，否则你将被他人打倒。所有的这些优点联合起来，塑造成一个能在不断变换的信息社会中生存的坚实的团队中的一员。我确信我拥有这些品质，我有信心成为贵公司成功团队中的一员。)

面试例题 12: Any question? (你有什么要问我的么?)

A: No sir. (没有, 先生。)

A: Since I can go to work in May. I'd like to know when the HR will send me the offer. (既然我可以在五月份入职, 我想知道什么时候人力资源部会给我发录用通知。)

面试例题 13: How do you estimate our interview? (你如何评价我们的面试过程?)

A: I've already passed interview for the second turn and also the calling interview from the states. but the HR notified me to go there for work in July

第 21 章

智力测试

智力测试是企业招聘时有可能出现的一个环节，但是这个环节并不是十分重要。以往的面试书籍把智力测试环节过度地渲染了。而事实上，IT 企业求职招聘还是主要以基本的程序设计及数据结构为主。智力测试是考验人的综合智商、逻辑能力的过程，本身是很难复习和准备的。这些年来，智力测试的一个新的趋势是和编程及算法结合起来。我在分析 Activework 公司、IBM 公司的智力测试考题时，发现它们都与算法中的迭代问题、递归问题、循环问题有紧密的联系，这也是读者值得注意的地方。

21.1 关于数字的智力测试

面试例题 1：小明和小强都是张老师的学生。张老师的生日是 M 月 N 日。2 人都知道张老师的生日是下列 10 组中的一天，张老师把 M 值告诉了小明，把 N 值告诉了小强。张老师问他们知道他的生日是哪一天吗？

3 月 4 日 3 月 5 日 3 月 8 日

6 月 4 日 6 月 7 日

9 月 1 日 9 月 5 日

12 月 1 日 12 月 2 日 12 月 8 日

小明说：“如果我不知道的话，小强肯定也不知道。”

小强说：“本来我也不知道，但是现在我知道了。”

小明说：“哦，那我也知道了。”

请根据以上对话推断出张老师的生日是哪一天？

解析：

由小明说“如果我不知道的话，小强肯定也不知道。”推理出不会是6月和12月，因为如果小强知道的是2或7，那么小强可以确定是哪一天。于是就剩下：

3月4日 3月5日 3月8日

9月1日 9月5日

由小强说“本来我也不知道，但是现在我知道了。”推理出小强知道的数肯定是只存在一个的。如果是5日的话，就有3月5日和9月5日之分，那小强还是不知道，所以只剩下：

3月4日 3月8日

9月1日

由小明说“哦，那我也知道了。”推理出小明要知道的话，那这个月当中就只能有一天，那就只能是9月1日了。

答案：9月1日。

面试例题2：村子中有50个人，每人有一条狗。在这50条狗中有病狗（这种病不会传染）。于是人们就要找出病狗。每个人可以观察其他的49条狗，以判断它们是否生病，只有自己的狗不能看。观察后得到的结果不得交流，也不能通知病狗的主人。主人一旦推算出自己家的是病狗就要枪毙自己的狗，而且每个人只有权利枪毙自己的狗，没有权利打死其他人的狗。第一天、第二天都没有枪响。到了第三天传来一阵枪声，问有几条病狗？如何推算得出？

解析：

第一种推论：

(1) 假设有1条病狗，病狗的主人会看到其他狗都没有病，那么就知道自己的狗有病，所以第一天晚上就会有枪响。因为没有枪响，说明病狗数大于1。

(2) 假设有2条病狗，病狗的主人会看到有1条病狗，因为第一天没有听到枪响，是病狗数大于1，所以病狗的主人会知道自己的狗是病狗，因而第二天会有枪响。既然第二天也没有枪响，说明病狗数大于2。

(3) 由此推理，如果第三天枪响，则有3条病狗。

第二种推论：

(1) 如果为1，第一天那条狗必死，因为狗主人没看到病狗，但病狗存在。

(2) 若为2，令病狗主人为a、b。a看到一条病狗，b也看到一条病狗，但a看到b的病狗没死，故知狗数不为1，而其他人没病狗，所以自己的狗必为病狗，故开枪；而b的想法与a一样，故也开枪。

由此，为2时，第一天看后2条狗必死。

(3) 若为3条，令狗主人为a、b、c。a第一天看到2条病狗，若a设自己的狗不是病狗，由推理2，第二天看时，那2条狗没死，故狗数肯定不是2，而其他人没病狗，所以自己的狗必为病狗，故开枪；而b和c的想法与a一样，故也开枪。

由此，为3时，第二天看后3条狗必死。

(4) 若为4条，令狗主人为a、b、c、d。a第一天看到3条病狗，若a设自己的狗不是病狗，由推理3，第三天看时，那3条狗没死，故狗数肯定不是3，而其他人没病狗，所以自己的狗必为病狗，故开枪；而b、c和d的想法与a一样，故也开枪。

由此，为4时，第三天看后4条狗必死。

(5) 余下即为递推了，由 $n-1$ 推出 n 。

答案： n 为4。第四天看时，狗已死了，但是在第三天死的，故答案是3条。

面试例题3：3升的杯子一个，5升的杯子一个，杯子形状不规则，问怎么才能得到4升的水，水无限多。

答案：

(1) 5升杯子装满，倒入3升空杯子，则5升还剩2升。

(2) 3升杯子清空，把5升中还剩的2升倒入3升杯子中。

(3) 5升杯子装满，把5升倒1升给还剩2升的3升杯子中，3升杯子满，5升倒了1升还剩4升。

面试例题4：1元钱一瓶汽水，喝完后两个空瓶换一瓶汽水，问：你有20元钱，最多可以喝到几瓶汽水？

答案： $20+10+5+2+1+1=39$ 。

面试例题 5：你有 4 个装药丸的罐子，每个药丸都有一定的重量，被污染的药丸是没被污染的重量加 1。只称量一次，如何判断哪个罐子的药被污染了？

答案：从 1 号瓶取 1 个药丸，从 2 号瓶取 2 个药丸，从 3 号瓶取 3 个药丸，从 4 号瓶取 4 个药丸，称一下，重量多几就是几号瓶。

21.2 关于推理的智力测试

面试例题 1：有 3 箱水果，一箱全是苹果，一箱全是橘子，还有一箱是两种水果的混装。3 个箱子上都贴了标签，不过所有的标签都贴错了。现在要求你只拿出一个水果，来判断 3 个箱子里的情况。

答案：很明显，从标签是橘子的箱子中拿水果和从标签是苹果的箱子中拿水果性质是一样的，所以必须拿标签是混装的。如果拿出来的是苹果，说明这箱都是苹果，则标签为橘子的是混装的；如果拿出来的是橘子，说明这箱都是橘子，则标签为苹果的是混装的。

面试例题 2：想象你在镜子前，请问，为什么镜子中的影像可以颠倒左右，却不能颠倒上下？

答案：人眼是左右对称的，如果人眼是上下对称的，则颠倒上下。

面试例题 3：

1
1 1
2 1
1 2 1 1
1 1 1 2 2 1

What is the next line? (下一行将会是多少?) [美国某著名搜索引擎公司 G 面试题]

解析：

1-----1 个 1

1 1 ----- 2 个 1
 2 1 ----- 1 个 2 1 个 1
 1 2 1 1 ----- 1 个 1 1 个 2 2 个 1
 1 1 1 2 2 1 ----- 3 个 1 2 个 2 1 个 1

答案：3 个 1，2 个 2，1 个 1，所以答案是 3 1 2 2 1 1。

面试例题 4：一个长方形，里面随机挖去另一个长方形，如何一刀使它面积平分。[英国某著名计算机图形图像公司面试题]

答案：把两个长方形中心点相连接，然后一刀切开，可以保证长方形是平分的。

21.3 关于时间的智力测试

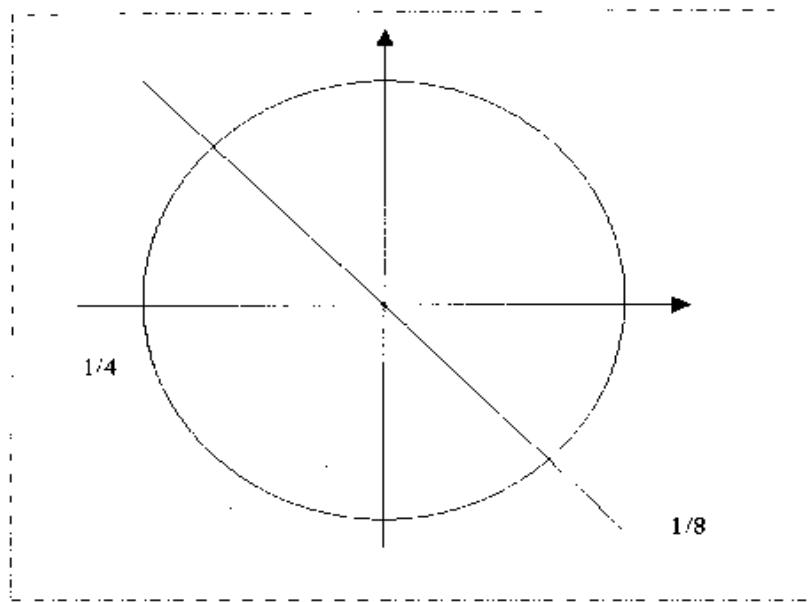
面试例题 1：有两根不均匀分布的香，香烧完的时间是一个小时，你能用什么方法来确定一段 15 分钟的时间？[美国某著名计算机软件公司面试题]

答案：第一根香两头一起烧，同时烧第二根香的一头，第一根香烧完同时再烧第二根香的两头，烧完第二根香时间为 15 分钟， $1/4$ 小时。

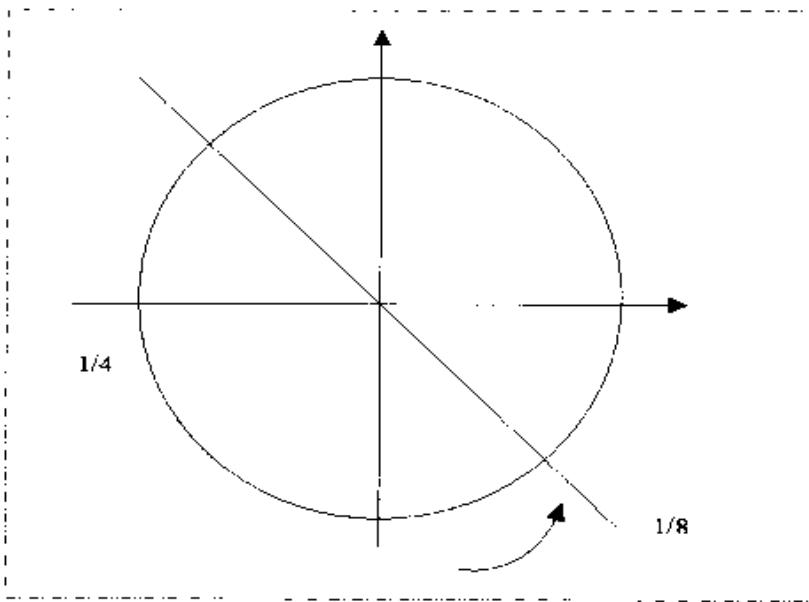
面试例题 2：飞机加油问题。已知：每个飞机只有一个油箱，飞机之间可以相互加油（注意是相互，没有加油机），一箱油可供一架飞机绕地球飞半圈。为使至少一架飞机绕地球一圈回到起飞时的飞机场，至少需要使用几架飞机？飞行几架次？（所有飞机从同一机场起飞，而且必须安全返回机场，不允许中途降落，中间没有飞机场，一架飞机一起一落算一架次，假设加油可以瞬间完成。）[中国某著名通信企业 H 面试题]

解析：设地球周长为 1，则一架飞机的持航距离为 $1/2$ ，飞机起始点在 0 点处。

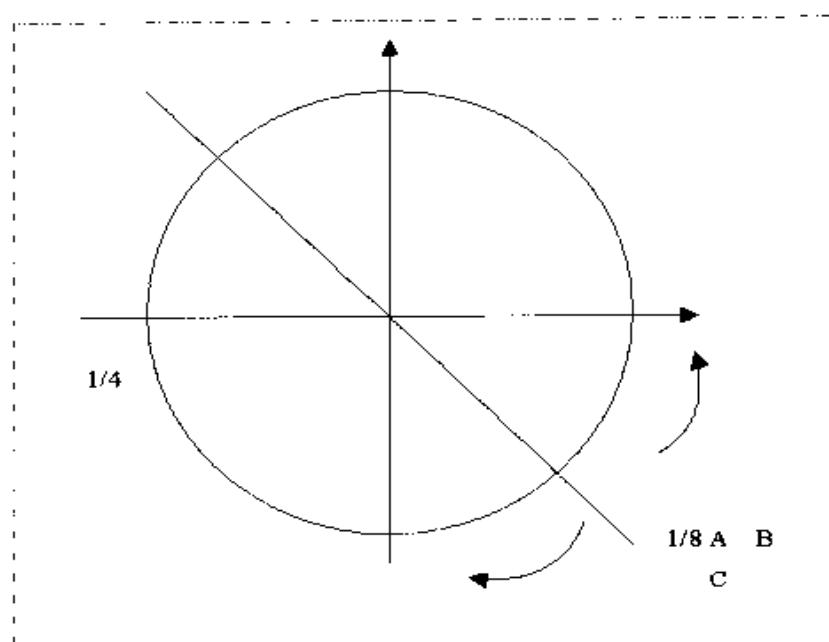
如下图所示，设有 3 架飞机 A、B、C。其中 A 是主机，担负着环航地球的使命，B、C 是副机，起到空中加油的目的。



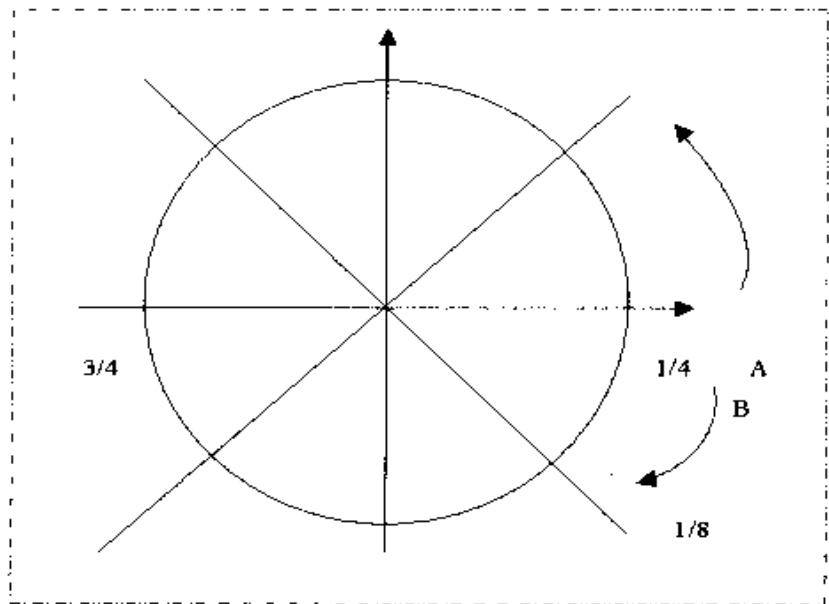
如下图所示，A、B、C 3 架飞机，在 0 点处起飞，航行到 $1/8$ 处，这时 A、B、C 的持航距离是 $3/8$ 。



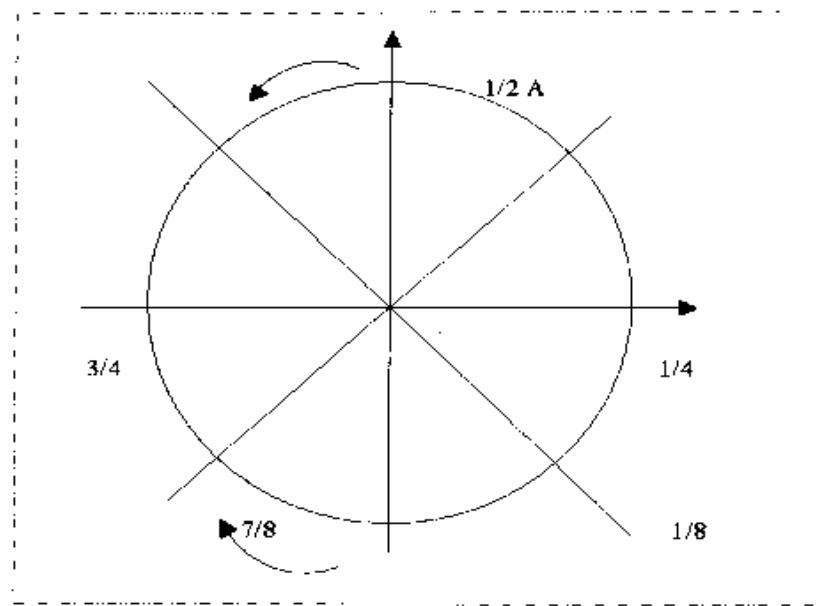
如下图所示，在 $1/8$ 处，C 机的持航距离为 $3/8$ ，它把能航行 $1/8$ 距离的油分给 A，再把能航行 $1/8$ 距离的油分给 B，此时自己只剩持航距离 $1/8$ 的油。此时 C 必须返航，否则就会坠毁。而 A、B 加油后持航距离重新变为 $1/2$ 。



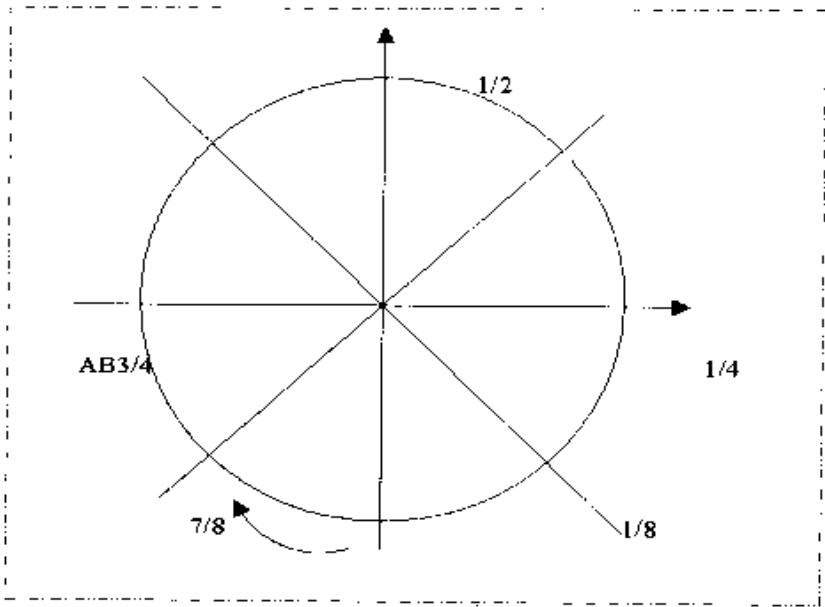
如下图所示，在 $1/8$ 处，B机的持航距离为 $3/8$ ，它把能航行 $1/8$ 距离的油分给A，此时自己只剩持航距离 $1/4$ 的油。此时B必须返航，否则就会坠毁。而A加油后持航距离重新变为 $1/2$ 。



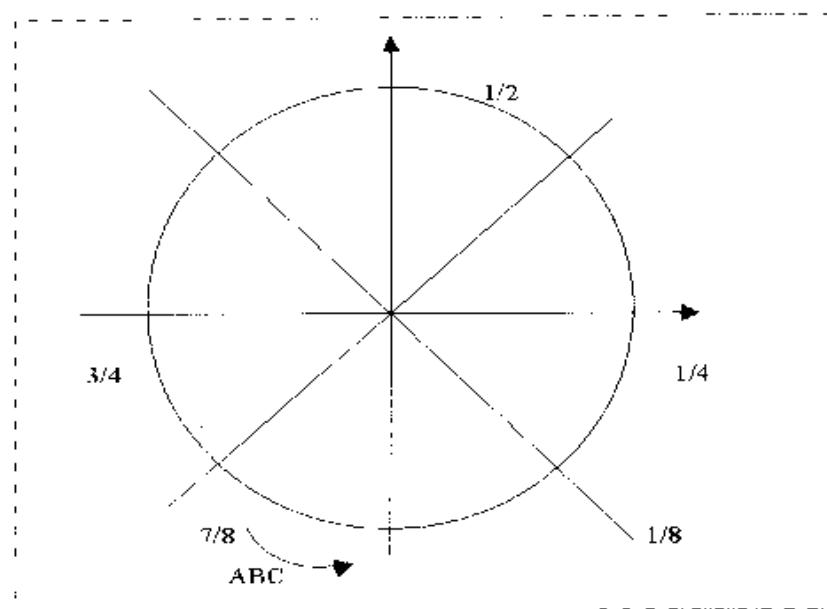
如下图所示，在 $1/4$ 处，A机的持航距离为 $1/2$ 。当它飞到 $1/2$ 处时，B机必须顺时针飞行以接应A机，否则A会在 $3/4$ 处坠毁。



如下图所示，A、B 在 $3/4$ 处会师。此时 A 持航距离为 0，B 持航距离为 $1/4$ 。B 将能持航 $1/8$ 的燃油输入 A，这时 A、B 的持航距离都是 $1/8$ ，换句话说，它们都可以飞到 $7/8$ 处。此时 C 必须从 0 点出发接应 A、B，否则 A、B 都会坠毁。



如下图所示，A、B、C 在 $7/8$ 处会师。此时 A 持航距离为 0，B 持航距离为 0，C 持航距离为 $3/8$ 。C 将能持航 $1/8$ 的燃油输入 A，再将能持航 $1/8$ 的燃油输入 B，这时 A、B、C 的持航距离都是 $1/8$ 。A、B、C 一起返航，顺利回到 0 点，回来时 3 机皆没油。



答案：3架飞机，5架次。

扩展知识

本题属于极限编程的一个范例，一般的解法可以分为如下两个部分。

1. 直线飞行

一架飞机载满油飞行距离为1， n 架飞机最远能飞多远？在不是兜圈又没有迎头接应的情况下，这个问题就是 n 架飞机能飞多远？存在的极值问题是不要重复飞行，比如两架飞机同时给一架飞机加油且同时飞回来即可认为是重复。或者换句话说，离出发点越远，在飞的飞机就越少，这个极值条件是显然的。因为 n 架飞机带的油是一定的，如重复，则浪费的油就越多。比如最后肯定是只有一架飞机全程飞行。注意“全程”这两个字，也就是不要重复的极值条件。如果是两架飞机的话，肯定是一架给另一架加满油，并使剩下的油刚好能回去，就是说第二架飞机带的油耗在3倍于从出发到加油的路程上，有3架飞机，第三架带的油耗在5倍于从出发到其加油的路程上，所以 n 架飞机最远能飞行的距离为 $1+1/3+\dots+1/(2n+1)$ 。这个级数是发散的，所以理论上只要飞机足够多，最终可以使一架飞机飞到无穷远。当然，实际上

不可能一架飞机在飞行 $1/(2n+1)$ 时间内同时给 n 架飞机加油。

2. 可以迎头接应加油的情况

一架飞机载满油飞行距离为 $1/2$, 最少几架飞机能飞行距离为 1 ? 也是根据不要重复飞行的极值条件, 得出最远处肯定是只有一架飞机飞行, 这样得出由 $1/2$ 处对称两边的 $1/4$ 肯定是一架飞机飞行。用上面的公式即可知道一边至少需要两架飞机支持, $(1/3+1/5)/2 > 1/4$ (左边除以 2 是一架飞机飞行距离为 $1/2$), 但是有一点点剩余, 所以想象为一个滑轮 (中间一架飞机是绳子, 两边两架飞机是棒) 的话, 可以滑动一点距离, 就是说加油地点可以在一定距离内变动(很容易算出来每架飞机的加油地点和加油数量, 等等), 所以至少需要 5 架飞机。

附 A 录

简历模板

随着求职热潮的来临，我们发现越来越多的求职者有同一个困扰，那就是写一份能体现自己实力的简历很困难，要不就是投了许多简历都石沉大海。究竟如何把一份求职简历写好呢？

简历内容要丰富

在网上发布个人资料时，一定要注意详细填写工作经历和教育经历，这是网上招聘单位最为看重的两项内容。许多网上求职者往往忽略了这两点，只简单填写一些自己最基本的情况，这是远远不够的。因为只有从你的工作经历和教育经历中，才最能看出你的优势所在。否则，用人单位仅凭你的一些简单基本情况，是很难决定让你前来面试的。

另外，现在网上求职的人越来越多，招聘单位很难一一通知每个求职者前来面试，只能挑选少数专业对口、优势明显的求职者前来面试。因此不妨将你的求职资料多投递几家公司，以增加面试机会。

IT企业通常将很多招聘工作外包给专业人才网站，因此在筛选简历、笔试和面试时都遵循着一个既定的程序和标准。一个优秀人才应聘系统包含以下几个程序：软件系统筛选简历→人工筛选简历→笔试→第一轮面试→第二轮面试。自动软件系统会通过考查几个方面来挑选简历：学校和专业、英语能力和项目经验都会是IT企业最看重的地方。IT企业青睐专业对口的大学生，而名校背景、突出的英语能力、项目开发的经历，都会成为应聘的加分亮点。

简历求职意向要有针对性

简历是你向企业递出的第一信息，通过这个简历企业才能初步了解求职者的情况。如果只是泛泛而谈，那就没办法从数百份简历中脱颖而出，吸引人事主管的日光。为了不使简历内容烦琐（大多数企业没有时间看太长的简历），简历的内容要注意抓重点，抓曾经从事的主要工作内容或工作技能。如若会一二门外语，有必要再写一份外语版，以增加简历的分值。

许多求职朋友可能多才多艺，既会这，也会那，就在简历上写着什么都会，然后遍地撒网。从职业指导的角度来看，这不是最好的办法。求职意图来自于你想做而且有能力做得很好的工作，而不是只是知道却从来没有涉及的工作。这样会浪费大量的时间和精力，反而不得企业的青睐。

简历的书写/排版要整洁

一份干净、整洁的简历给企业的感觉是最起码的尊重。而企业也可以通过简历了解你的行为、习惯、修养。但是过犹不及的是有些求职者不惜成本，拍写真贴在上面，做豪华的简历封面，这其实是犯了大忌。公司用人看重的是才能，而不是相貌；不是选美，而是选才。

不要因为省钱而去使用低廉质粗的纸张。检查一下是否有排版、语法错误，甚至水、咖啡渍。在使用文字处理软件时，使用拼写检查项并请你的朋友来检查你可能忽略的错误。

长短适中的简历最受欢迎

一份长短适中的简历是最受企业欢迎的，长度在2~3页纸比较合适。个人信息、工作经验的叙述与招聘职位的要求越接近越容易赢得入围机会。那些精美或者花里胡哨的简历并不见得受欢迎。简历的真实内容才是考核重点。

太多的人想把他们的经历压缩在一页纸上，因为他们曾经听说履历最好不要超过一页。当将履历格式化地缩到一页时，许多求职者就删除了他们给人深刻印象的成就。如果你需要用两页纸来完成简历，请清楚、完整地把你的经历和取得的成绩表现出来。不要压缩版面，不要把字体缩小到别人难以阅读的程度。反之亦然。就那些在履历上用几页纸漫谈不相干的事的候选人来说，看的人很容易觉得无聊。所以，当你写履历

时，试着问自己：“这些陈述会让我得到面试的机会吗？”然后，仅仅保留那些回答“是”的信息。

决定履历篇幅是否恰当的规则就是没有定则。决定其篇幅的因素包括职业、企业、工作经历、教育和造诣程度，等等。最重要的就是，履历中的每一个字都要能够推销该候选人。

细节考查职业诚信

求职的过程本身就是要将自己的优势展现给招聘单位，所以在某些情况下，为了引起对方的注意，适当地采取夸张的手法也不为过。但是，现在拥有本科以上学历的人越来越多，学校里的各种头衔也五花八门，简历上千篇一律的全是学业、能力出众的“十全十美”人才。招聘者不仅对这样的简历缺乏兴趣，甚至可能怀疑其真实程度。许多求职者往往觉得优点越多越好，因此在简历中罗列无数的长处，却没有突出重点，这样反而给人一种全面但平庸之感。

招聘单位，尤其是一些实力雄厚的大公司，对于人适其职极为重视。他们需要的不是什么都会的全才，而是在某一方面有过人之处、能为公司所用的实用之才。有些求职者在面试时表现得天衣无缝，但最终却未被录用，其原因就在于他们回答虽然巧妙，但却犯了回避缺点的错误。这样的结果通常是两个：要么是考官认为其言不属实，要么是认为聘用一位“十全十美”的天才很有风险。当然，我们也不需要刻意去抹黑自己，在真实展现自己的基础上突出所长，不要为掩盖自己的缺点而去塑造一个虚假的自我。

仔细措辞体现个人业绩

有时候，简历上的几个字就足以“激怒”HR，使其停止阅读你的简历。招聘经理与招聘人员心中都有一张讨厌措辞表。他们厌烦那些在简历中尽可能多地堆砌动词、形容词或副词的人。“非常出色”、“做出很大的贡献”这些用词都是不合适的。最好能够改成“我完成了多少销售业绩，联系了多少家公司”。如果数字过于敏感不适宜表达，可以用百分比或企业的表彰来表达，还可以写上获得的证书。

有些不像销售部门那么容易量化的部门，比如行政部门，可以通过办公设备的维护和采购、降低成本、客户满意度、如何及时维修等方面

做出说明；HR 部门可以通过客户满意度、招聘周期、人岗的匹配、离职率等来体现。

还有关于某类知识掌握情况的词语，简历可以把知识按个人熟练程度分为 4 类：精通、掌握、熟悉、理解，便于企业一目了然。以下是一些招聘经理认为会降低简历说服力的词。

援助、帮助 (assist, assisted)

应避免的原因：招聘经理想知道你做了什么，而不是你怎样提供帮助。如果你十分了解某项任务，把它写进简历中，你可以选择比“帮助”更好的词。

实例：帮助销售主管研究 PDA。

可行的改写：为销售部研究 PDA。

实验 (experiment)

应避免的原因：没有人想知道你试图做什么，他们只想知道你做过什么。

实例：实验使用新型局域网 (LAN) 管理软件。

可行的改写：测试并评估新型局域网 (LAN) 管理软件。

技术熟练地 (skillfully)

应避免的原因：招聘经理通常对你形容把一件事做得多棒的话感到反感。在很多情况下，这都是自夸的表现，而且这也是不必要的。“如果你不擅长于此，你凭什么递简历呢？”一名招聘人员如是说。

实例：在管理由 Windows NT 到 Windows Server 2003 过渡方面技术熟练。

可行的改写：在营业时间不停工的情况下，将所在组织的 Windows NT 移植到 Windows Server 2003。

负责 (responsible for)

应避免的原因：你是一名管理者，当然要负一些责任。详细说明你的责任及你在一些领域内的工作，陈述你的职责范围。

实例：负责管理存货；监督网络运作；购置新的设备；发现并修理工作站故障。

可行的改写：为 70 名 Windows XP 用户与二台运行 Windows Server

2003 的服务器提供支持管理；为存货设备执行资产管理计划；建立一个负责内部基础设施的网络运作团队。

简历模板（供应届毕业生参考）

个人概况：

求职意向：

姓 名：性 别：

出生年月：_____年____月____日 健康状况：

毕业院校：专 业：

电子邮件：传 呼：

联系电话：通信地址：

邮 编：

教育背景：

____年至今____年____大学____专业（请依个人情况酌情增减）

主修课程：（注：如需要详细成绩单，请联系我）

论文情况：（注：请注明是否已发表）

英语水平：

* 基本技能：听、说、读、写能力

* 标准测试：国家四、六级；TOEFL；GRE……

计算机水平：

编程、操作应用系统、网络、数据库……（请依个人情况酌情增减）

获奖情况：

_____、_____、_____（请依个人情况酌情增减）

实践与实习：

____年____月至____年____月____公司____工作

____年____月至____年____月____公司____工作（请依个人

情况酌情增减)

(请描述自己的个性、工作态度、自我评价等)

另：(如果你还有什么要写上去的，请填写在这里！)

* 附言：(请写出你的希望或总结此简历的一句精练的话！)

例如，相信您的信任与我的实力将为我们带来共同的成功！或：希望我能为贵公司贡献自己的力量！

简历模板举例（供应届毕业生参考）

个人简历

☆个人信息：

姓名：王大满

性别：男

民族：汉

籍贯：广东湛江

学历：大专

专业：计算机应用

出生年月：1982年8月 毕业学校：西北大学

联系电话：13571945048 E-mail：jin@263.net

☆自我评价/职业目标：

一年的外企实习经验，一年的专科院校代课经验，通过CET6。在我实习期间，我的客户这样评价我：工作努力，具有快速接受新事物的能力，良好的团队合作能力及优秀的英语沟通能力。我年轻有活力，具有扎实的专业知识和良好的学习能力，因此我有理由相信我能给您创造价值。

现阶段的目标是成为一名优秀的软件工程师。

☆教育简历：

2003年9月至2006年7月：西北大学计算机应用专业(硕士)

本人研究生期间在西北大学计算机系软件工程研究所深造。主要从事的研究方向为软件工程、软件度量、软件配置等。

1999年9月至2003年7月：西北大学计算机科学(本科)

☆专业技能：

熟练掌握 HTML、XML、CSS、JAVASCRIPT 等技术。

熟练掌握 JSP、Servlet、JDBC 等相关技术，Java 语言基础扎实。

熟悉 MySQL、SQL Server、Oracle 等数据库的开发。

熟悉 Tomcat 应用服务器的使用。

对 Struts、Spring、Hibernate 等开源框架有所了解。

了解 C 语言、VB 程序设计和汇编语言。

熟练使用开发工具 Eclipse。

☆实践经验：

2005 年 11 月

毕业设计：开发药品管理系统，包括管理药品的进库、修改、出库、出库历史、库存查看等功能，使用 Struts、Hibernate、MySQL 开发，应用前端为 Struts，用户通过 Web 浏览器进行访问，使用 Hibernate 把对象模型映射到数据库。

2005 年 6 月

独立开发本人的个人主页，具有文章分类显示、后台动态管理，在后台动态发布文章、修改文章、删除文章等功能，使用 JSP、Servlet、JavaBean、MySQL 开发。

模型开发，JSP 用于前台显示，Servlet 用于转发，是整个系统的中心，JavaBean 主要用于与后台数据库的交互。

2005 年 1 月

参加学校的网站设计策划大赛（主要是静态页面设计和图片动画设计），获得优秀奖。

☆外语能力：

通过英语六级，具有很强的英语读与写能力，能够阅读专业相关的英语文档。

☆奖励情况：

2004—2005 学年获得优秀学生一等奖学金。

☆发表论文：

在校发表论文一篇：《基于数据水印技术的软件度量保护》（《大众软件》，2005.10）

简历模板（供有工作经验者参考）

个人概况：

求职意向：（可以是一个与多个）

姓 名： 性 别：

出生年月：_____年____月____日 健康状况：

年 龄：____岁学 历：_____

毕业院校：_____专 业：_____

工作年限：____年

联系方式：

电子邮件：_____ 手机：_____

家庭电话：_____ 传呼：_____

通信地址：_____ 邮编：_____

教育背景：

_____年至_____年_____大学_____专业（请依个人情况酌情增减）

_____年至_____年_____大学_____专业（可将在学的业余课程写上）

工作经验：

_____年至_____年_____公司_____部门_____工作

_____年至_____年_____公司_____部门_____工作

（请依个人情况酌情增减）

此处应为整篇简历的核心内容，应聘者可以着重叙述此项，并根据个人工作情况不同而重点突出说明工作具体内容与经历，尤其是与求职目标相关的工作经历。一定要说出最主要、最有说服力的工作经历和最具证明性的为公司获取的利润和相关成绩。说明的语气要坚定、积极、有力。写工作经验时，一般是先写近期的，然后按时间顺序写。

英语水平：

- * 基本技能：听、说、读、写能力
- * 标准测试：国家四、六级；TOEFL；GRE……

计算机水平：

编程、操作应用系统、网络、数据库……（请依个人情况酌情增减）

业余爱好：

个性特点：

_____（请描述自己的个性、工作态度、自我评价等）

另：（如果你还有什么要写上去的，请填写在这里！）

*** 附言：**（请写出你的希望或总结此简历的一句精练的话！）

例如，相信您的信任与我的实力将为我们带来共同的成功！或：希望我能为贵公司贡献自己的力量！

简历模板举例（供有工作经验者参考）

个人简历

基本信息：

姓名：柯小平 性别：男

出生日期：1976/5/8 居住地：深圳市

工作年限：五年以上 户口：黑龙江省哈尔滨市

电子邮件：ppon@gmail.com

移动电话：13590161234

自我评价：

- 1) 有很强的创造能力、拓展能力、抽象思维能力与项目管理能力。
- 2) 有很强的判断、决策、计划与执行能力。
- 3) 有良好的沟通、协调、组织和团队建设能力。

4) 高度的工作热情，良好的职业道德。

求职意向：

工作性质：全职 目标地点：深圳市 期望工资：面议

工作经验：

2004/05 至 2005/10：陕西新科网络科技有限公司

所属行业：互联网/电子商务

技术部 技术总监/经理

1) 参与制定公司发展战略、年度经营计划和预算方案。

2) 组织研究行业最新产品的技术发展方向，主持制定技术发展战略规划。

3) 及时了解和监督技术发展战略规划的执行情况。

4) 领导分管部门制度并组织实施年度工作计划，完成年度任务目标。

5) 研究公司技术服务的决策流程和规范，并监督有关部门加以执行。

6) 指导、审核项目总体技术方案，对各项目进行最后的质量评估。

7) 与用户进行技术交流，了解用户在技术与业务上的发展要求，并解答用户提出的与产品技术相关的问题。

8) 对潜在或具体的项目、用户进行跟踪，管理所在区域内的用户拜访、技术交流、方案制作及合同谈判。

9) 制定技术人员的培训计划，并组织安排公司其他相关人员的技术培训。

10) 负责建设及维护公司内部网络。

2003/08 至 2004/04：深圳沃尔夫网络技术有限公司

所属行业：互联网/电子商务

培训部 专业顾问

负责帮助客户现场鉴定并解决有关网络技术及安全问题，保证客户网络畅通；整体安全服务解决方案项目的设计；网络培训课程的开发、设计；企业管理信息技术高级客户培训。

2000/08 至 2003/07：哈尔滨工程大学思科网络技术学院

所属行业：教育/培训

培训部 专业顾问

CISCO 认证课程讲授（CCNA、CCNP）；组织网络培训课程的开发、设计；组

组织、主讲大学课外网络技术讲座。

项目经验：

2005/04 至 2005/05：陕西建行 DCC 测试网

项目描述：中国建设银行陕西区分行辖内测试网项目的实施，范围覆盖了区分行（西安）和全区 10 个二级分行。数据交换主要是区分行和全区二级分行之间的交换。

责任描述：

- 1)与用户进行技术交流。
- 2)审核项目总体技术方案。
- 3)对参与该项目的工程师进行培训。
- 4)与厂家、客户进行沟通。

2004/10 至 2005/10：陕西中烟网络安全

项目描述：陕西中烟在经过机构调整之后，机构合并重组，原来的秦烟和陕烟分别作为中烟工业公司的制造一部和制造二部。本项目同时包括两个分部的安全建设。总投资约 250~300 万。

责任描述：

- 1) 与用户进行技术交流。
- 2) 指导、审核项目总体技术方案。
- 3) 对参与该项目的工程师进行培训。
- 4) 协调技术部工程师与市场部销售人员的关系。

教育经历：

1999/09 至 2003/07：哈尔滨工业大学 计算机科学与技术 本科

语言能力：

英语 通过六级，听说熟练。

IT 技能：

技能名称 熟练程度 使用时间

Security 精通 24 月

CISCO 精通 60 月

附 B 录

面试经历总结

经在论坛上看到一个北邮计算机硕士写的一篇名为《我的快乐求职》的面试经历总结，觉得百感交集。作为本书作者的我，并非出身名校，找工作的过程中也颇费周折。求职的过程是让你接触社会、了解社会的过程，在此期间你会彷徨，迷茫，经历感情的低潮，感受到社会的强大和自己的卑微。这段过程不会是很快乐的，甚至是痛苦的炼狱。

我想很多应届毕业生和正在工作的朋友们有的正在求职，有的想着跳槽，我也经历过这种情况，所以非常明白和理解一些朋友的心情。还是那句话，每个人的行业不同决定着经历不同，我只能写写我的主要是从求职过程总结出来的几点建议和面试的一些经验。我想，虽然只是个体案例，但是有些东西应该是共性的，如果能够帮到一些朋友的话，我也真的很开心。

求职的时间分配

如果早预料到 2005 年的最后 4 个月会在如此的挑战和压力下度过，我会好好利用暑假，养马拭剑。求职的生活，像在攀登一座底部平坦而顶峰陡峭的山，开始时还可以漫不经心，最后的时光却是摸爬滚打，十分紧张。

大四研三等毕业班学生，最后一学期就是在写论文和找工作的双重压力下勉强度过的。如果能够在农历春节前搞定一切，这样是最好不过的。春节前的 4 个月是各公司蜂拥招人、供需两旺的时刻，如果顺利，你将拥有一个无比轻松自在的春节。否则拖到年后，答辩和求职孰轻孰

重，首尾难顾。

如果要在 3 月份毕业的时候拿到学位证书，你得赶在 2 月 20 号之前完成论文答辩。这意味着你得在 12 月 20 号之前论文成稿，去抽盲审。同时意味着你得在 10 月底就完成论文的初稿，然后交给导师，导师提意见你修改，运气不好的话修改的地方还蛮多的呢。而 10 月底之前完成论文的初稿，意味着 9 月份一开学你就得动手写论文，暑假之前你就得做论文要用的东西……

我的最后一学期就是在写论文和找工作的双重压力下勉强度过的。9 月份开始动笔写论文，投出第一份简历，直到次年 2 月中旬完成论文答辩，最终签约。现在想来，这 4 个月中我的体验和成长，胜过之前的任何 4 个月。所以实在有必要写一点东西，让师弟师妹们有所借鉴和启发。

找工作越早越好？

由于一直以来对自己的实力没有把握，找工作开始得特别早，几乎是一开学就是找工作网站和各大高校 BBS 的常客了。9 月份招聘网站上的工作职位几乎都是针对有工作经验的人的。有些职位要求本科且有两年以上工作经验，我们也可以投投试试。我把凡是我比较感兴趣的大公司的职位都投了，后来有回音的并不多，即便有回音，能通过面试给我 offer 的也不多。9 月的时候有些紧张也有些兴奋。兴奋的是发现今年的工作形势还可以，招人的公司很多；紧张的是回音并不多，同时还要抓紧时间写论文。

对于这些面向有工作经验的人的招聘，建议试试那些你特别感兴趣的职位和公司，因为如果一个很一般的公司这么早给了你 offer，他多半会让你马上去上班（像对有经验的人的要求一样）。一来你没有时间马上去上班，二来这么早签了，一般的公司你会不甘心，所以也就没必要浪费时间在上面。

我就是因为太早开始找工作，到 12 月中旬已经对找工作有点厌烦了，最后在手头的 offer 中选了一个就签了。实在没有那么多精力再投新的简历，留意新的工作机会了。建议大家不用开始得太早，10 月份开始找就很好了。

招聘网站及 BBS 的选择

通常最常去的招聘网站有：www.chinahr.com，www.51job.com，www.zhaopin.com。

[chinahr](http://www.chinahr.com)（中华英才网）上的校园招聘比较多，但是每个职位都是全国各地的学生蜂拥而至，竞争之激烈可想而知，通过 [chinahr](http://www.chinahr.com) 求职成功的几率很小。在线登记简历的时候，注意突出用人单位要求的关键词，每一栏都别空着。[chinahr](http://www.chinahr.com) 通常机械地筛选简历，如果你不多写点就很容易被漏掉。[chinahr](http://www.chinahr.com) 筛选简历，做第一轮面试，然后才把选出的人交给公司，由公司做最后的面试。我参加了两次 [chinahr](http://www.chinahr.com) 的面试，分别是 IBM 和微软，都没能闯过 [chinahr](http://www.chinahr.com) 这一关。[chinahr](http://www.chinahr.com) 的面试官不懂技术，他们仅仅按照客户的要求机械地问一些技术问题，再问一些你简历上的东西和性格情商之类。因此想过这一关，首先要表现得很自信，回答技术问题时答对要害就行，没必要说太多。

[zhaopin](http://www.zhaopin.com) 上面针对有工作经验的人的职位很多，比 [chinahr](http://www.chinahr.com) 多。那些要求本科且有两年以上工作经验的，研究生们也可以尝试。面试之前根据相应职位的要求好好准备一下，因为是面向有经验的人的招聘，所以问的问题就不仅仅是 C、C++、数据结构这些内容了。

建议每投一份简历，就把职位要求记录下来，一旦几天之后给了你面试机会，可以参照职位的要求复习一下，更有把握些。

BBS 是一定要去的。有时候 BBS 上会贴出有用的招聘信息，随时留意总归是好的。华为、西门子的招聘我就是从复旦 BBS 上看到的。有的时候 BBS 会定期地删除一些 Job 版上有招聘信息的帖子，所以大家要随时关注。如果想进 IBM、微软这样的大公司，建议去清华 BBS 看看。他们有 IBM 和微软的专门板块。

宣讲会 vs 网上申请

从 10 月份起，各大公司在交大或科技大开宣讲会。虽然绝大多数公司只接受网上注册的简历，但也有例外。比如 Intel，他们在交大的宣讲会上留下了不同部门 Manager 的 E-mail，让同学们分别投简历，后来得

到面试机会的也是把简历寄到 Manager 信箱去的那些人。我们只在网上注册了简历，没参加宣讲，不知道还有 Manager E-mail 这回事儿，只能被 reject 了。

建议大家分工合作，大公司的宣讲会还是去听一下比较好。不用全去，每次派一个人作代表，记录一些有用信息。有的时候可以兵分多路，我们就有同时参加百度和招商银行宣讲会的体验。

网上申请也是很有学问的。我在 chinahr 上登记的简历，有回应的挺多，然而有的人在网上登记的简历却一个回音都没有。我怀疑他少写了什么关键字。在填写网上简历时切不可贪图一时省事，这里省事就是费事，必须把相关的经历详细填好。而且目前各大招聘网站都有简历模板，一次填好以后可以反复使用。

离理想大公司仅一步之遥

“面”了很多，“笔”了很多，有几次离心目中理想的大公司仅一步之遥，可惜本人那时功力尚浅，始终没能得到心目中理想的 offer。

通过校园招聘招人的大公司，一份有分量的简历只是第一步。有分量指的是成绩尚可，有让他们感兴趣的实习经历，有一定的获奖经历，担任过一定的职务，英语能力还行。这仅仅是第一步。它能让你从众多应聘者中被选出来参加初试，接下来就看你的真正功力和造化了。

求职是实力加运气的综合体现。也许你会发现各个方面都不如你的人签的公司却比你还好。这里的原因很多：比如说，就公司而言，如果某个公司正处在上升期，需要大量人才加盟，他这时的用人标准就是比较松的，大概看你的简历还行，就很轻松地要了。如果某个公司处在平稳期，不是很需要人，这时他的 HR 用人标准相对严格，你除非有特别出色的履历和极其显赫的业绩，否则很难打动他。这里就和战争时期选拔飞行员比较松，和平时期选拔飞行员比较严格是一样的道理。

以上说的是“时”，求职过程中还有“势”，只有时势皆顺利，才能产生 offer。这里的“势”指一些非常细节性的东西。我们可能一路过关斩将，但却有可能在非常细小的方面翻船，如同天意一样。很多事情是我们不能左右的，但却是可以预防的。

SAP 公司，我一路通过了笔试和面试，最后没有通过它的电话面试。

其实其他的电话面试是外包出去的，是由外包公司来替他们实现的，主要考查的是求职者的口语能力。这就是一个很有趣的现象。外包公司会问你一些很机械的问题，比如说，用英语介绍你自己，用英语介绍一下你的项目。这些是很简单的问题，但是让你用英语回答却很难回答——如果事先没有相关准备的话。我们如果因为实力不济或是技术方面的原因不能通过公司最后的考试倒也不算冤枉，技不如人当自强而已。但是仅是让你介绍项目经验，却由于自己英语没能过关，磕磕巴巴表达不出而命断外包公司之手，实在很可惜。所以我们平时要加强口语练习，做到临危不乱。

还有的情况是我们自己不明所以而且十分无奈的。一次是去投西安某某研究所。他们的 HR 说，对不起同学，你的学校（西北大学）不在我们招聘学校范围内，我们只要科技大和工大的学生。另一次是参加 SPSS 的笔试，顺利通过后参加了它的面试，感觉面试回答得非常好，但最终没能等来 offer。这其中会有一些未知的因素，我们不用抱怨。不能因为存在未知因素我们就不去努力。找工作像一场孤独的战争，艰难坎坷无限，得到 offer 是最后的结果。我们会用阴谋、阳谋、明争暗夺来实现顺利求职的目的。如果不想被社会淘汰就需要努力准备，这也是自己见容于社会也被社会见容的前提。求职也罢，做人也好，都是一样的。

几场面试经历

找工作不是一个简单的事情，如果决定了找工作就一定要专心地准备。首先是要清楚地认识自己，看自己喜欢什么样的职业。还有就是多了解一下自己专业的就业范围情况。然后就是开始准备简历，只有你对自己充分了解了才可能把简历做好，把你的优点都表现出来。这是第一步也是最关键的一步。只有你有一个好的简历，一个能突出你自己优势的简历，才能给你进入下一轮的机会。刚开始的时候要多投简历，不要舍不得自己的简历，投出去一份就是投出去一个希望，就说不定能给你一次笔试/面试的机会。

最主要的还是笔试和面试。大部分学生都是第一次找工作，所以多参加一下笔试和面试还是非常必要的。

由于笔试投的职位很多，有时候是硬件相关职位，有时候是软件相

关职位，考的范围也很广。这里建议找工作的同学最好把以前学过的专业知识，还有专业的基础课，都看一下，至少要有个大体的印象。后面的面试也经常会问到技术方面的问题，更有公司会直接给你出题让你当场给他做出来。

我从找工作到现在一共参加了几十个公司的面试，基本上跑遍了西安大大小小的豪华酒店和软件园所有知名的企业。下面讲讲我的面试经历，希望对大家有所启发。

北大青鸟面试

第一次是北大青鸟，这是全国最大的软件培训机构。由于是第一次面试，我什么都没准备，穿了一身破旧的牛仔服走进了奥罗酒店的总统套房。一进门就看到很多西装革履的帅哥。一开始我还以为都是招聘方的工作人员，后来才知道也是一起来面试的。先是发了两套卷子，我就靠在沙发上答题。题目很多，不深但是很广泛，包括 VB、C#、.net、C++、SQL 都有。我花了大概约 1 个小时答完了题。然后工作人员收走了卷子，问我对哪个方面比较擅长，我说 C#，于是他们又发了一套纯 C# 的卷子给我做。这一套就很有难度了，有关于委托、事件的一些问题，感觉答得不好。笔试后等了 30 多分钟立即面试（有几个西装小伙笔试后没有等待就直接被打发走了）。面试官看到穿得破破烂烂的我一笑，他的态度很和蔼，基本上就是谈谈家常，我家几口人，父母在哪，以前有没有工作经验，等等。感觉上他对我的着装虽觉怪异，但并不是不可接受。大概是我的题目答得还是不错的缘故，几天后通知我可以拿 offer 了。这是我拿到的第一个 offer。

深圳华为面试

第二次是深圳华为公司。说起来还有点搞笑。和大家一样，刚开始网投华为，第一志愿是华为西安研究所（研发类）。然后华为给我打电话，但当时我有事在北京回不来。10 月 13 号下火车，然后当天去郦苑酒店“强面”，但被告知无法参加华为西安研究所（研发类）的面试，随转行参加深圳 IT 管理的面试。13 号一面结束，14 号参加二面、三面、四面。二面结束后我和女朋友出去逛，也不敢走太远，因为怕随时给我来电话。天气很冷，我买了个烤白薯暖手，这时通知三面的电话响起，我忙不迭

地往楼上跑，白薯就放在裤子后面的口袋里。面试结束后发现白薯都被我坐扁了。下午参加四面，晚上打电话通知给 offer。这就是我两天搞定华为的经历。

下面说说我面试的具体细节：华为的一、二、三、四面分别在郦苑酒店的 2 楼、3 楼、4 楼和 5 楼。每面试成功一次就往上走一层。

一面的是一个很和蔼的哥哥，不过说话声音比我还小。他翻了简历，看了成绩单，问我在什么方面学得好，我就说英语。他说在专业方面呢，我就大致说了一下我们专业的情况，学习的课程什么的，等等。然后他给我出了两道题，就是本书递归一章的打靶问题，其他人有的出的是智力测试，比如说飞机环航问题。反正我基本上很快编出了程序。然后还有一些无关紧要的聊天。后来他又问我对信息管理是否了解，我就把自己了解的东西一股脑说了出来，不过也说不了多少。然后他就说不只这些，开始给我解释。接着问我对华为有多少了解，我就把自己知道的说了一下。最后他说让我回去等消息。大概晚上 9 点左右电话通知我明天早上面试。

二面是一个综合的面试。面试官先让我自我介绍，然后问了些兴趣爱好、性格之类的。他翻简历看到了我做过的教务管理系统和网络选课系统。他详细问了项目的组成原理和架构设计，以及你在项目中的贡献等。然后我就解释了一下。然后我补充说我曾经写过一本书，把内容又说了一下。在这里简历还是很重要的，他们面试的时候总是会拿着简历问你一些相关的问题，所以面试的时候一定要把这些都好好准备一下。还有成绩单，华为好像很看重这个，特别是本科生。最好把自己学得好的科目大致复习一下，有可能会问到相关的问题。二面完了，然后拿了表填好了等待三面。

三面在 4 楼，我和一个女生共同面对面试官。面试官坐在床上，问了很多问题。他先解释了一下工作地点的选择，然后就开始问都做过什么项目，说一下你自己认为做得最成功的事，都遇到过什么样的挫折，影响最深的是什么，等等。后又说“华为你们也知道，会很累的……让我们谈一下自己的看法。你们两个应聘信息管理，对这个职位怎么看，了解不？”他以一对二，不是每个人都得回答每一个问题。他会挑着让你去回答，而且在你回答的时候会对你说的继续提问，不断地抹杀你的

观点，有点像压力测试，所以在面试的时候一定不要让自己的话有漏洞。即使有也要想办法来说明，不过这个还是比压力测试会好一些，不会把你否认得一无是处。总之他会刁难你，应该是测你的抗压能力和临场反应能力。

最后是四面。四面就是华为的高层来随便聊聊，没有什么问题了，是一对一的，他填个表，然后就给口头 offer 了。过了一面、二面而且填了表的同学千万不能大意，也得好好准备一下。我同学就可怜地栽在三面了。四面后当晚正式打电话给 offer。

平时多面试一下，增加自己的经验也是很必要的。对于这些常见的问题，最好都准备一下，如：自我介绍，你自己最大的优点、缺点，自己认为最成功的事，最尴尬的事，等等。

神州数码面试

第三次是神州数码公司。收到面试通知的时候，实在感到十分意外。因为在笔试的时候，当我看到封面写着对于研发，Java（题）是必做而 C++（题）是选做的时候，立马眼前一黑，感觉自己已经被鄙视了，但还是硬着头皮写着参加研发。但是做的题目全是 C++。神码 C++ 不难，但是数据库的题全与 Oracle 相关，颇有一些难度。

神州数码的面试是考察计算机专业知识最多的一场面试，除了专业知识的交流外，还涉及到市场调研、人际关系、人生理想、职业道德、日常生活各个方面的话题。这次近一个小时的面试是我表达得最充分的一个面试，敲门进去，互相问候之后，面试就正式开始了。

HR：你几点到的？（典型问题）

Me：（努力回忆+猜测中……）大概一点五十多吧……

HR：那也等了一段时间了。

Me：呵呵，其实也没多久……

HR：我先自我介绍一下我自己，我叫 xx。（HR 脸上有阳光般的笑容，好有亲和力啊。）接下来，你可以用两三分钟的时间介绍一下你自己。

Me：Blah-blah…

HR：会用 Java 吗？（估计是因为发现我的笔试一道 Java 都没做）。

Me：（尴尬……）虽然没有相关的课程，也没有做过 Java 的项目，但课余的时间还是看了点儿书，学会一点儿的……而且我擅长.NET 和

C#。

HR：你为什么喜欢 C#呢？

Me：C#兼具 VB 的快速简练、Delphi 的可视化控件编程、Java 的完全面向对象和 C++的语法规则。C#最长于 Web 开发，微软宣称 ASP.NET 1.1 以后，所有的代码均由 C#写成，由此 C#在 Web 开发领域的实力可见一斑。

HR：你能给我解释一下 C#与 C++有什么区别吗？

Me：区别很多。首先是托管与非托管的区别。托管代码不允许进行对内存的操作，而是由固定的垃圾回收机制来完成的，而 C++则不然。其次 C#和 Java 类似，都是运行在虚拟机上的（分别是.NET 虚拟机和 Java 虚拟机），而 C++不需要这样一个平台。最后 C#是完全面向对象的。在 C#里，万物皆是类，绝对不存在一个超越类以上的函数或是变量。C++也是面向对象的，但其仍然保留面向过程语言的特点（比如说 C++存在全局变量）。最后，C#摒弃了 C++中的多重继承等不易掌握的特点，代之以接口等，使编程变得更加轻松和简便。

HR：可以说一下你的社团活动吗？

Me：极其诚实地讲，我不过是充当那种螺丝钉的角色。但我还是很喜欢社团组织的各种活动，很愿意参与团队合作过程。

HR：说一下你的项目经验吧。

Me：Blah-blah…（介绍自己做过的项目和写过的书。）

HR：你的论文情况如何？能介绍一下你的论文么？

Me：我论文做的是协同过滤技术在网络数据挖掘上的应用。

HR：能描述一下么？（递给我一支签字笔，让我在后面的白板上画出来。）

Me：Blah-blah…（我一边说一边画，估计他也听不懂我的论文情况。其实这一段就是看你的表达能力如何，而并不是一定要弄懂你做的东西。）

面试结束后两天给我打电话，说可以发 offer 了。

其他拿到的 offer 还有深圳平安、信息产业部 52 所等。

感觉国企发 offer 的速度、还是很快的，效率很高。基本上如果想要

你，就会很快地联系你并发 offer。如果迟迟未能收到结果，估计就是没戏了。

我经历过几场失败的面试：他们是中国移动、Thoughtworks、SPSS、Sybase、Trend、SAP、Siemens，等等。失败给我的教训远胜于成功的喜悦，也是最值得我们记录的。

思科面试

思科面试是在一个房间有 3 组人同时进行，每组 9 个人。这是群面……群面如果不能一起合作的话，就像是一群人拉一辆车，但是每个人朝不同的方向走去……

面试的第一部分是，大家在编有号码的 A4 大白纸上，用 5 分钟时间画一个名片，怎么画都行；然后小组成员互相帮助，将名片别在自己的背上；再然后小组成员互相认识，组成一个团队。

接下来的未来城建造，大家都想做决策者，于是此起彼伏的声音说“我是这样想的……我觉得还是这样好……”真是一件比较郁闷的事情。最后终于讨论出一张图使大家大体同意，但是又就组成的部件是要圆柱、圆锥还是方体争论一番。最后尽量在细节上照顾了所有人的意见做了修改。建造时间也过去了一半……还好大家的行动比较快，搞定设计之后一致行动，很快就有了大致的规模。

接下来是 Presentation，也就是介绍自己组的设计。一个交大男孩说他要做 Presentation，一个理工大的女孩说我也想做，剩下的人也都在抢。因为整个陈述时间只有 2 分钟，剩下来就是每个组的提问时间。所以每个人 40 秒，为了好控制和监督，请了一个学姐计算时间。

Assessment Center 是很多个人表现优秀的人容易失误的地方。因为大家都很优秀，也想争取更多的机会表现自己，然而只顾自己的最终结果就是整个团队的失败。那么这个团队一定需要一个人做类似 Leader 的工作，也就是协调和分配组内的事务，大家合力完成目标。可是怎样才可以让你的队友在短时间内认同你的 Leader 位置？强迫认同的人会在评估中成为 too aggressive，一样会被淘汰；缺乏控制力的 Leader 会把整个团队的活动搞成松散的个人 show。

怎么做比较好？这个问题我还没有搞懂。但是做到以下几点的人我才可以认同他/她成为我的 Leader。一是有自知，作 leader 要有好的交流

和逻辑性，有自信和经验做好协调。最重要的是，leader 要给予同组的人展示自己的机会，创造平等的竞争和友好的氛围，绝对不能只顾自己的展示而忽略了团队。但是真正做起来每个人都会多多少少出现一些问题。

我们那场 27 人当中一个某某大学的研究生，拼命地说那个塔是由他一个人想出来的创意，一遍又一遍地，着实觉得他脸皮很厚。但是可笑的是他最后好像通过了群面，也许思科太看中他的自我表达能力吧。

一次次地我想寻找着一种共赢的模式，可是一次次地我发现总是把这个世界想象得像童话一样。我总是表现得文质彬彬，于是总有人在我就要张口之前开口了，于是我总是被别人抢走了话。在对其他组的作品进行质疑的时候，也是这样的。我拼命地举着手要起来发言，可是总有人速度比我快地站起来发言。抢得很惨烈。我始终挥动着我的双手示意着要发言，但始终没有机会。我不习惯于这样的抢。“群殴”结束后，心中那是一种怎样的委屈啊。那种没有展现个人才能的委屈，那种很不服输的委屈，那种凭什么的内心质问。面试结束后我在广场上散步，这时我只是想到重温童年时代的期待，在重温过去的时候寻找着激情，继续着向上的梦想，以淡化这种委屈。

初一的数学课，每每老师在黑板上写下简单的题目时，下面的我们就开始叫了起来，把该怎么做的步骤说了出来。每每老师在黑板上写下复杂的题目时，下面的我们开始鸦雀无声。时间久了，这位很有个性的老师说：“最烦你们这些没有技术含量的叫喊。发言的时候要说别人没有想过的东西，别人都想到的东西就不用说了！”他的这句话给我留下了深刻的印象，并在以后的成长过程中实践之。于是我在和多人讨论的时候，总是先倾听，那么地谦虚！于是在群面过程中我总觉得插不上嘴，于是总是很郁闷地被“刷”了下来。

这是我一次群面被刷的经历，感觉为人不可过谦，否则这么激烈的环境下怎有容身之地。主动出击，先发制人，才是顺利通过的根本。

深圳移动

2005 年 11 月底的时候，广东移动下的十多个分公司就陆续开始校园招聘，我也正是在那个时候做了网申。深圳移动通过了我的网申，通知我参加面试。就这样我参加了深圳移动在西安的首轮面试。

我当时应聘的是技术类，当时去的是长安城堡大酒店。大厅里坐满

了人，我就感觉自己很渺小。然后我们排队参加面试。队排得很长，有点像是去食堂排队打米饭的感觉。由于人很多，每个人只有 3 分钟的机会——我觉得就凭 3 分钟给一个人下结论实在太武断了。问到我的时候，面试官说让我用 1 分钟阐述我的优势。接着又问我在校的一些情况，估计说了 2 分钟。说是 3 天内会给我打电话，就这样结束了面试，事后杳无音信。

后来我回想，我觉得我参加的深圳移动的面试不是很对口的面试，毕竟我的专业是计算机软件，而他们需要的是网络或者硬件方面的人才。之所以给那么多人发面试通知但只面试 3 分钟的原因，可能是想做宣传，现在很多企业都是这样想的。

“霸王面”

“霸王面”，是说心理素质很强，没有接到面试通知也要闯进去面试的情况。在求职中采取“霸王面”是一种无奈之举，但这毕竟是你自己的人生，工作将来是属于你自己的，别人不会管你。机会要去主动求来，获得职位才是硬道理。在求职竞争激烈的环境里，找到一个适合自己的职位已是不易，谁还去计较用什么方式呢。当然，前提一定是合法的方式。

现在的毕业生这么多，对于 HR 来讲，学生和学生的差别不是很大，都只是一个一个的名字符号。而我们要做的事情就是要让他们一看见这个符号就能想起我们这个人。比如投了简历之后如果没有消息就打个电话问一下，多参加宣讲会，对于想去的公司争取面谈的机会。在 11 月的时候，我手里只有华为和神州数码的 offer，当时 Sybase 开始一面了，没有通知我。我是挺想去 Sybase 的，于是就去霸王面。跟着一个被邀请去面试的同学到了那里，当我看到有面试官出来的时候赶快抓住机会上去和他谈，终于同意我参加面试。虽然最后没有通过第四轮的面试，这也是我能做的所有了。虽然失败，我不后悔，因为我已经尽了力。轻叹一口气，再找下一个吧！

“霸王面”也是一个坚持自己理想的过程，坚持也是做任何事情必备的条件之一。从 9 月开始求职到 12 月底，大约 4 个月时间，这期间虽然有不少面试的机会（其中包括很多次“霸王面”），但有很多我想去的公司都把我给拒了，连拒信我都能背出来了。提高个人的“抗击打能力”，

不要太顾及所谓的“面子”，因为现在不是我们有面子的时候。求职被拒实在是一件再平常不过的事情，现在看来似乎很严重，可是我能预见到以后我会觉得远没有现在所想象的那么严重。就像我现在回想中学时期遇到的困难和挫折，有很多都是十分可笑和幼稚的。我所要做的就是尽量不让被拒影响我的心态，因为我坚信我的付出可以换来好的回报。我们是求职者，处于这个角色就要遵守游戏规则，要找到好的工作就要经历这些必要的过程。现在还仅仅是填填表格，回答回答问题，以后一定会有更加烦躁的事情我们不得不去做，谁能够坚持到最后谁就是胜利者。

“霸上面”最后的成功与否主要取决于以下两点。

一、是你所应聘工作招聘工作的严密程度。

许多安排周密的面试根本没有可能“霸上面”，人家会很委婉地回绝你，甚至笔试都是不可能的。这并不是一份卷子、几分钟谈话的问题。我曾经没有笔试直接参加葡萄城的面试，被婉言请出来了。不过临走前，HR 工作人员赠给我一件葡萄城的 T 恤，说是感谢我对该公司的关注，的确是一个很有人情味的企业。

二就是你的诚意和你能利用机会给人留下印象的能力了。

如果你了解到这个公司的招聘工作不是很周密，有面试的可能，那么就尽全力展示你自己吧，不要害怕开始碰钉子。你要坚持到了最后，机会也许就属于你了。吃“霸上面”不能全靠一股子热情，要靠智慧。因此，求职者必须对企业进行全面了解，特别需要了解很多细节。伯乐也是很重要的，并不是每个人都能碰到欣赏“霸上面”的人力资源经理。不同企业文化有不同的用人理念，应聘者要适合企业发展的需要，适应企业文化。

第二点是不能迟到。我原来参加 SAP 电话面试没有通过，于是去强面。但是当时人家进行的是群组面试，每 8 个人为一组，正好少一个人没来。如果我不迟到，完全可以补上这个“缺”。但是我是在人家开始后 30 分钟才来的，强面+迟到=失败。事后我得出结论，如果确实对该公司的职位感兴趣，一定不能迟到，要让用人单位感觉你的诚意是扑面而来的，这样才会有机会。一般来说，外企在这方面更加不拘一格。我朋友参加 SPSS、西门子、施耐得的“霸上面”都获得了成功，后两个还顺利拿到了 offer。

几点建议

答题

笔试的时候，在时间允许的条件内尽量认真答题，不要答得少，着急交卷。要记得，这也是有的公司考核员工，态度是否认真，值得聘用。面试时的礼仪装扮不是很重要。应聘程序员这样的行业，去面试时不需要西装革履。普通的衣服，干净整齐就行。面试最关键的就是回答问题。面试只有 10~15 分钟，如何在短短的时间内让人对你记忆深刻是关键。我个人认为关键是提出比较创新的提议。千篇一律是不好的，自己当然要与众不同。前期的准备工作，收集好资料，针对公司现有的情况，多想一些你应聘的职位的工作计划，以及提出一些你自己的想法，这些都是吸引 HR 眼球的地方。

在面试结束时，对方会问你还有没有什么要问他们的，你可以问两三个问题：关于公司对你这个职位的需要是什么样的；或者公司比较热点的问题，充分表现出你对该公司的了解做足了功课，有备而来，是个有心人。

选择

如果是为了职业生涯的前景，就一定要进行选择。不要看广告和随便投递简历。在对现有的工作产生了厌倦的时候，就要着手开始准备。主要是锁定工作单位目标。根据自己的学历、经验、能够胜任且符合自己的工作来决定。名牌公司未必适合自己，是做人公司中的一滴水，还是做中小公司的中坚地位，哪一个是自己现阶段需要的，就要锁定这个工作单位作为目标。一般来说，公司越大，管理制度越健全，分工越细，自己学到的东西也相对有限。

调查

有些公司在招聘的时候挂出某某跨国公司的旗号，其实只是与这个公司有一点合作关系的小公司，只是利用了大公司的招牌，里面的内容是截然不同的。再就是有的公司根本就是空壳，不具备经济实力，但包装却十分诱人。最后就是要了解公司内部的福利待遇，是怎么样的考核制度。总而言之，就是要确定公司的真实情况。

工作的时候，很多人不会想得这么仔细，有的时候明知道吃亏也只想先委屈一下。可是，这种做法从长远来看是不正确的。

比如，进了一家公司工作了几个月，甚至更短的时间，发现公司根本没有发展前途，更不利于自己的职业发展，那么只能辞职离开。这样频繁地更换工作，非常不利于自己的职业生涯，学到的东西也极有限，大部分的时间都在求职过程中度过了。

所以，宁可稳一些，也绝不能急躁。

确认

在锁定工作目标，并对公司进行调查之后，还要把未来新公司自己从事的工作内容认识清楚。要了解公司的企业文化。每个公司都有自己的特点和风格，培养出来的员工通常会带有公司的特色，比如思考问题的方式和做事情的风格。如果不了解这些贸然进去，会发现两种企业文化相抵触，自己很难融入现有的团队，以前的工作经验和做事方法根本无法发挥出来。最后搞不好自己还是要离开。很吃亏的。

失业

在职场生涯中，很多人都恐惧失业，跳槽的时候都喜欢骑驴找马，都喜欢找到了一个新工作才辞职。我个人不是很赞成这一点。因为工作一段时间后会发现原有的知识储备急速下降，很多年轻人都是边工作边学习，非常辛苦，虽然很上进，但是自己都不知道自己到底需要的是什么。很多人热衷于考证，压力又大又忙碌。这个时候也总是会出现一些迷茫的情况。

所以，如果对工作厌倦时，可以给自己一个放松的好机会。旅游，看影碟，逛逛街。整理好自己的思绪，规划出下阶段的目标，然后确定方向，再继续投入到工作当中。当然，失业期间也是进修的好时间。不单是为了拿证，而是让自己能够真正有所学。知识是用来用的，不是用来当摆设唬人的。

当然，最重要的，是我们自己要把握好自己，把握好自己要走的路。其实任何一份工作都需要我们努力工作，任何一份工作都无法钦定我们的终生。