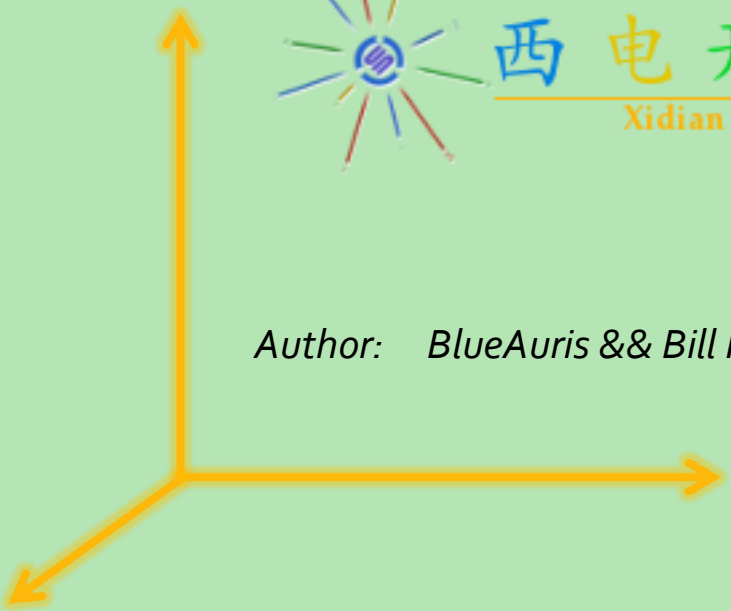
Introduction to Programming

-To the majority of the programming novice



西电开源社区
Xidian Open Source Community

Author: *BlueAuris && Bill Ryan*

Introduction to programming

Author: Bill Ryan

Email: billryan@xdlinux.info

Version: 1.4

Publication date: Wednesday, January 25, 2012

版本升级说明：

V0.9 全书正文内容基本定型，附录部分未完善，打印了一份后发现蓝色字体打印后很浅

V1.0 全书整体内容基本确定，打印了 3 份，送给了大三的一些朋友，有附带银杏叶做的亲笔签名噢~

V1.1 在 V1.0 的基础上对 Appendix-A 部分做了两段的增删，面向通院科协实验室大一内部发行。

V1.2 在 V1.1 的基础上对 Appendix-A 部分增加了一小段针对大二的内容，修复了部分解释文字的字体设置。

V1.3 在 V1.2 的基础上对 Appendix-A 部分删除了一段考虑尚未周到的文字，确定对大二这边发布。

V1.4 在 V1.3 的基础上做了不少改进，征得原作者的同意后以创作共用约定在网络上发布，从此本书将以协作开发的模式开展

V1.4.1 将 V1.4 部分英文改为中文，降低初学者门槛

改编自《编程及 C/C++初学者 FAQ》

作者：碧蓝右耳

联系方式：BlueAuris@Gmail.com

版本号：3.0

完成日期：2008-03-10

[1] http://creativecommons.org/licenses/by-nc-sa/3.0/deed.zh

序

弹指之间，从我进入西电到现在已有两年多了，从大一的懵懂少年到现在的一知半解，其中经历的酸甜苦辣实在是太多太多，不是在这一两句话就能说清楚的，而编程的学习就是众多酸甜苦辣回忆的一段，所以这本小册子在某种程度上也可以称为是我个人学习编程的学习笔记。

虽然本人从大一上学习 C 语言开始真正接触计算机，但我的编程水平确实不咋滴，Coding & Algorithm && Data Structure 都没有系统的学习过，小程序也没写过多少……

正如许三多当初选择在五班修路一样，我也抱着同样的信念选择为大家奉献这么一本小册子，虽然自身水平有限，但我还是非常愿意和大家一起分享我在成长路上所获得的点点滴滴……

OK，啰嗦到这里我也应该开始切入正题了，本书内容的主要素材源于网络上的资源，我在此做了大量的整合，以便于适应目前的实际环境。这本书谈的更多的是编程思想，或言之为一种方法论，目的是希望能为大家从全局去把握编程提供一定的思路，不为细节所困。本书是一篇散文，形散神不散——以编程为神，以各章节为形；它同时又是一篇科普文章，读起来自然不难理解，甚至是轻松幽默的。大家在看本文电子版的时候旁边若能添上一杯茶就更好了，最好能是杭菊+枸杞，清火明目。

最后想说的是鄙人才疏学浅，高中时语文和英语就是我最差的两项，曾在高三的一次月考中我的语文英语两科总成绩也不及数学一科就是一个很好的佐证。而这本书中既需要大量的语言来组织，同时又包含一定程度的英语，这两项融合在一块已经远非我能力之所及。还好英语这一块有小鱼儿倾力相助，问题已经得到了相当程度的改善。即便如此书中错误也在所难免，发现错误后请及时在西电开源社区相应主题上回帖，以供大家参考。

本书在开源社区 BBS & Wiki & 豆丁上公开发布，网址链接如下->

BBS: http://xdlinux.info/bbs/forum.php?mod=viewthread&tid=1232&extra=

Wiki: http://xdlinux.info/wiki/index.php/%E5%88%86%E7%B1%BB:Introduction_to_Programming

所有事项均第一时间在 BBS 上发布

西电开源社区：袁斌

二〇一二年一月二十五日星期三 18 时 37 分

Contents

Chapter0. Preface

0.0 Why wrote this book?

大约是两年前，也就是我当年大一的时候，对计算机各方面了解都不多，上了很久的计算机文化和 C 语言课也没搞懂编程到底是在干啥，只是觉得编程是一件很神奇的事，就像是做梦一样，不过雾里看花终隔一层，始终也未能亲身体验它的乐趣。那时多希望有个诸葛能扶一扶我这个阿斗，即便不是卧龙，凤雏也行嘛！我想现在一定也有那么一小撮人抱着这种想法，我只想对你说："该醒醒了！现在不是立波梦话板块！"**在任何关键时刻，能帮助你的人只有你自己，即便你能找到愿意帮助你的人，那也要你花心思去找不是么？**

作为一个曾经的新手，我深知从新手过渡到老鸟需要拨开天空的乌云&&翻山越岭……，正如你们所看到的，书店&网络上充斥着无数的编程教材，同时可以肯定的是，目前已经面世的教材，穷一人一生之力是不可能看完的。在这些书中，有大量的垃圾书，海量的平庸之作，还有少量的精品，而即使是这少量的精品，也不可能看全。既然书这么多，我为什么还要抽时间再来整理一篇呢？拿本书蓝本创作者碧蓝右耳的话来说，他还能拿这些时间多画几张效果图挣俩钱花咧 O(∩_∩)O 哈哈~

情况是这样的，市场上的书虽多，但其中几乎没有几本是面向初学者的。这样的书是如此之少，以至于要去购买或是阅读到它们都是很困难的事。他们要么是一下子告诉你所有的事，好像你能在千分之一秒中突然从菜鸟变成高手，要么就是认为有些事你早就应该知道，拿你当老鸟看，导致你有一种赤身裸体被抛弃于猛兽横行的非洲旷野的感觉。你还没有穿上衣服走出帐篷，连刀子都没有摸过，他们就试图告诉你草原上有多少可以捕获的猎物以及他们的位置，告诉你几百种武器和毒药的使用秘籍，告诉你两百条以上的陷阱安放要领。你没有经过丝毫的练习，甚至还没有杀死过一只刚出壳的小鸡，他们就要你独自去捕猎数十头饥饿的狮子。这种看似荒谬的情况从过去持续到今天，至今仍然存在。这并不是说那些写教材的朋友都是傻瓜，而是你的水平太次。They assume that you already have an idea of what programming is, why you want to do it, what's basically involved, and so on. 他们面向的读者是程序员，程序员就像是猎人，他们更换语言就像猎人更换武器一样，不管他使用哪一种武器，捕猎的基本原理没有变化，变化的只是武器的使用方法。对一个成熟的猎人而言，再强调基本原理就没有必要，所以教材们对人所共知的一些事也就避而不提。

一个成熟的猎人，他心中的捕猎知识是浑然一体的，武器的选择，野兽的习性，陷阱的安放，怎样做和为什么这样做都结合在一起，没有哪一部分可以独立出来。一部分一部分的教给别人是极度困难的，要教就只能混杂在一起。编程的情况类似，它的知识体系是一个完整系统，谈到一个问题总会牵扯到另一个，只不过大家平时遇到的问题太简单了，以至于没有感觉到一个完整知识体系的存在，学起来自然只能是云里雾里。

O(∩_∩)O~是不是害怕了？不怕不怕，有西电开源社区为你掌灯引路呐~ :）

0.1 Intended Audience

读完上边的那段话你就应该能猜到本书（其实你现在看到的这个东西它既不是书，又不是一般的手册）的服务对象了，没错，就是为那些毫无编程经验的人准备的，甚至是第一次接触计算机的人。这本书的目的也正在于此——**为新手介绍一些编程所必需的背景知识，以便于更好地去阅读其它教材。**

如果你认为自己已有一定的编程经验，那你可以把你编程入门的这一段经历与大伙儿分享一下，完善一下 Wiki 或者在论坛上回复些自己的感想都很 Nice. With that in mind, let's begin! :）

0.2 Conventions, Typography and Others

本书中所采用的语言和句法与正规书籍并不完全一样，中英文混排贯穿全文，有些例子只是为说明问题方便而假设，严谨性可能不够，偶尔还会耍点小聪明或是利用编程中常用的符号，例如用&&代表和，||代表或者，更多的东西等着你们自己去挖掘。

章节名一律采用二级目录；正文内容采用 11 磅宋体，E 文尽量采用 11 磅 Times New Roman；重要内容加粗或黑体或红色表示；引用部分用五号蓝色华文正楷；解释说明用蓝色括号表示；英文书名用斜体表示。所有牵涉到他人隐私的地方我都尽量回避了，如有些许不合适的地方请及时 Email 我。

"尽信书则不如无书"——本书仅仅只是我把各种资料收集之后进行一次大的整合，但是水平毕竟有限，无法保证所有内容都是正确合理的，**信口雌黄和胡说八道仅一步之差**，部分评论我尽量保持中立态度，如有过于主观的观点就当我胡说好了，如果整篇都是 Paper 形式的文字，我想大家在看的时候也会比较累，不过引用参考的地方我会尽量标注出处以供进一步查阅，希望大家在读的时候带着怀疑的态度。

本书 V1.3 版本之前是本人在一个科技协会内部用于帮助他人学习用，V1.4 版本考虑在网络上发布，用以帮助更多编程新手。

0.3 Organization

首先说说书名吧——*Introduction To Programming* &&《计算机编程导论之 FAQ》，顾名思义，**本书只准备讨论编程中最为基础的部分，让初学者能建立起一个自己的最小学习系统。**

FAQ——Frequently Asked Questions，也就是常见问题解答。这玩意儿通常是一些所谓的高手&&前辈||公司的 support 为了节省回答新手的大量简单重复问题所耗费的时间精力而采用的一种手法。一旦完成，对于问与被问双方都轻松省事，效率甚高，实在是居家旅行杀人越货之必备良药，因此在网络上有如前期的泰国洪水一般泛滥成灾。

全书正文部分分为六章，由浅入深介绍编程，附录部分主要写了一些其它的注意事项和我的所感所想。附录 A 部分与编程无关，纯粹是在自言自语，考虑再三还是把它留下来了，因为我觉得没有附录 A 这本小册子似乎少了一种东西，至于少了什么东西那就仁者见仁智者见智喽~

0.4 Comments and Questions

有关本书的意见和问题在西电开源社区 BBS 和邮件列表相关主题下直接回帖，因为你的问题也许就是大家的问题，大家都会感谢你的；你一个好的建议也许就能帮助到一大批热爱编程的新手，这其中说不定也能出现不少高手噢；如有需要改进的地方也不妨直接提出，只要不涉及人身攻击…

0.5 Acknowledgements

- ✓ BlueAuris ——中文名：碧蓝右耳

  本书的原材料创作者，书中很多语句都出自他之手，我只是做了大量的组织和整合。可以说，没有他就没有这本书。谨在此致以崇高的敬意！

- ✓ 我的父母

  虽然你们没有在本书中给予我直接的帮助，但是你们在潜移默化中教会了我做人要有爱心，要懂得感恩。本书在很大程度上是基于此而发布的

- ✓ 西电开源社区

  社区提供了学习的平台和很好的资料，还有众多高手互相讨论交流，我从这里学到了不少东西, Thx~

- ✓ 我的好友——小鱼儿

  提供了本书全程英语翻译支持，还提供了不少改进意见

- ✓ 我的部员——zy & lanhao34 & zek & xqc

  提供了本书中 Linux & Windows 下编程工具的初步使用教程

- ✓ 我的嫂子——小马

  提供了本书 V1.3 之前编写过程中的网络支持——一枚 3G 联通无线上网卡，让我无论是在 A 楼还是图书馆里边都能无线上网

- ✓ 通院科协实验室所有的 XDJM 们

  没有你们就没有我组织编写这本书的机会，感谢在邮件列表 OR 短信 AND 电话给予我支持的所有 XDJM 们！

- ✓ 其他所有给予我支持和鼓励的人

  要感谢的人还是蛮多的，也许有些人只是在心里默默的鼓励我而没有表达出来，在此一并感谢。另外还需要特别感谢的是同寝的哥们，2011 年 11 月期间连续两周来我写这个文档经常熬到凌晨一点而影响到了你们的睡眠，但你们没有表现出任何的怨言，真心感谢你们的理解！

Chapter1. What is programming?

简单点来说，编程就是编制程序（呵呵，这不是废话么？）。程序是让计算机发挥功能的命令的集合，大体上有两种形式，让计算机真正执行的是电脉冲形式，叫机器码，譬如 0101 这种二进制数字。程序员编制的通常是文本形式，叫源代码。使用一个称为编译器的工具，可以把源代码转变为机器码。而编程就是产生那些源代码的工作。这个工作类似于谱曲、编菜谱、写工作手册。我们知道，谱曲并不是一蹴而就的，往往需要反复的修改，直到最后满意为止，编程也是一样。

1.0 Object of programming—Computer

如果你看到的这篇文章是电子版的话，你眼前的东西就是计算机，也就是俗称电脑的东西。这样的说法对普通用户来说是已经足够了，但是电气工程师认为一堆电路板、缆线、马达、和机壳的组合才算，联想的销售人员认为他们卖的那些方块才是计算机，而中央军委的人大概觉得银河 V 才能称得上。程序员的看法和他们并不完全相同，在程序员看来，只要能自动计算的东西，就是计算机。**这个说法的重点在于自动和计算这两个词。**广义的计算是指能对外界的某种输入做出反应，不一定就是数学运算。比如电梯就可以对按键做出反应，所以它也是能计算的。自动是说计算是通过自身的运作来完成的，不需外界干预。在有电的时候，电梯就可以自动运行，这样看来电梯也是计算机。算盘虽然能计算，但不是自动完成的，不过如果把打算盘的人一起算上，那就是不错的计算机了。使用指纹或虹模的智能锁、秦始皇陵里的机关和自动钢琴似乎也都是一种计算机，情况确实是这样，计算无处不在，计算机也无处不在。甚至可以把我们的世界看作一台巨大的计算机，然后就觉得我们是生活在 Matrix 里，这是一个很有意思的哲学问题，有兴趣就去看看黑客帝国三部曲吧。通常在实际编程中，程序员只考虑现代数字电子计算机，也就是使用电能为动力，在运算时以电子电路和逻辑代数为基础的计算机。他们用途广泛，种类也是极其繁多，手机、PC、服务器是比较常见的样子，电梯、收银台、智能门禁则是各种隐藏的版本。虽然设备的外在形式千变万化，但现代电子计算机的基本原理和体系结构并没有太大的变化。

1.1 Why use binary?

从计算机导论课程(或其他相关课程)我们知道在计算机底层是用二进制来表示各种信息的，有些人肯定会问计算机底层为什么不用咱们熟悉的十进制来进行各种运算？这样不是更接近人类的思考习惯么？这种想法还是不错的，你看咱们人类多聪明，让计算机使用十进制它是不是同样会变得很聪明呢？很遗憾的告诉你，不会，至少目前不会，那样只会拖累它，想想地球上有一半人改用火星文进行沟通，会出现什么情况？计算机的全称为电子计算机，既然是基于电子组成的，它就得遵循电子世界的规矩。

在数字电路中**一般**用 0 表示低电平，1 表示为高电平。电平是个逻辑量，只有逻辑高、低两种值，但逻辑的高低是用电压的大小来表示的。譬如与电源负极的电压差值很小的某一点可以表示为逻辑低电平——0，电压差值很大的那一点则可以表示为逻辑高电平——1。在目前的电路结构中，判断逻辑高、低电平两种状态是很容易做到的，而要判断多于两种状态则比较麻烦。

1.2 Why do we need programming?

编程这件事存在，完全是迫不得已，人类发明了计算机，想让它做事，仅此而已。但机器有三大特点，使得如果要让机器做事，完全不像吩咐人那么简单。

第一个特点就是机器很傻 - Computers are very, very stupid. 很多人希望自己像计算机一样聪明，真要是那样你就完了，赶紧卷好被子回家吧，不过待在竹园实验室也许是个不错的选择，至少还有你的难兄难弟陪着你——值班桌上那台破破烂烂的电脑。For example, 从三个数里选出最大最小值，人类可以一眼就看出，但机器只能先从两个里找出最大的，再把这个最大的和第三个比较，然后再这样重复一遍找出最小值。如果是七个八个数，机器也是这样反复操作，O(∩_∩)O~是不是想到了冒泡排序？这就类似工地上搬砖，人类的做法一次搬很多很多砖，用推车或者别的运输工具把砖码得高高的然后运到目的地，但计算机的做法是每趟只搬一块砖，你没看错，每趟一块，only one！

那为什么计算机这么厉害呢？**因为它的第二个特点，快而不乱。**机器可以不知疲倦地用同一方法，重复重复再重复地做某件事，而且每次重复都相当的快。这一点是人类做不到的，让一个人搬一堆砖，也许没什么问题，但是一车砖就会让人疲惫不堪，如果有一火车皮的砖，想必大多数人想都不想就放弃了，**即使有坚持去做的人，他的动作质量也只会越来越差。**但机器不同，他搬砖不是每趟一块么，但他每趟来回的时间很短，比如少于 0.00……001 秒，不管砖头有多少，十万块也好，十亿块也罢，按同一方法处理，既没有差别也不会厌烦，直到全部处理完。真是不怕苦、不怕脏、不怕累，新时代的劳动模范呀！所以从总体来比较结果，机器就比人强了。还是以选最大最小数为例，人可以用肉眼检视三四个数，但超过 100 个数就要用其它的方法。对机器来说，三个和三万个数只是重复次数的差别，单调快速的重复，这就是机器的诀窍。想想金庸笔下的绝招都是啥？——把最简单的招数练到极致，那就是绝招呀!!!

更重要的一点，机器和我们言语不通。就是说，我们不可能一抬手一挑眉毛就吩咐它做事。不要和我抬杠说你可以用鼠标画圈让机器做事，也可以声控，那都已经不是纯粹的机器了。机器和我们处在不同的感知空间，所以它不能理解我们的言语。我们要命令它，必须用它能够理解的形式。从根本上来说，就是电脉冲——也就是 **1.1** 节“Why use binary?”所说的数字信号“0”&“1”。

基于以上这三个特点，要让机器做事情，就必须让把我们的要求转化成最简单适于重复的命令集合，而且是电脉冲形式。比如我们要让机器在屏幕上显示一幅图，首先我们要把这幅图分解成很多很多的小方格，也就是所谓像素，每个像素只有一个颜色，每个颜色都用一个很长的数字表示，然后所有这些数字转化成内存里的电平信号，再用另外的电路信号来一个个的把这些电平送到显像电路。所有这些电平和信号的集合就是程序，而编程，就是制造这些程序的工作。

也许有一天，我们不再需要编程序，程序员这个职业也将不复存在。不过我想如果那天到来，那就是机器已经能完全理解人类的语言和思想的时代了。在肉眼所及的范围里，似乎还看不到。:）

1.3 Why should I learn programming?

这个问题有两层意思。首先是编程为啥要学，很简单，因为这玩意不是生来就会，就像木匠活一样属于一门手艺，自然要通过学习才能掌握。你怎么也得花上几年来学英语不是么？第二层涵义才是重点，为啥我要学它？It varies from person to person.

1) Programming is very intellectually rewarding.
2) Programming makes you feel superior to other people.
3) Programming gives you complete control over an innocent, vulnerable machine, which will do your evil bidding with a loyalty not even your pet dog can rival.

以上三种都属于自虐一族。。。

1) 学校开了 C 语言课程，而且还是必修，不得不学。ヽ(ﾍ _ ㄣ)ノ
2) Programmers make lots of money.
3) Programming is really fun.
4) Programming is a constructive art.[1]
5) So on…

相信在看过"Why do we need programming ?"后你对计算机的特性已有所了解。在我眼中，学习编程的意义更多的在于可以利用现代计算机技术来推动本专业的发展&通过计算机（这里泛指广义的计算机）这个平台相对方便地实现自己的想法，说白了它就是一个很好的工具。即使现代计算机领域的分工逐渐细化，但你也总得了解一些基本概念吧？

1.4 What exactly programming is?

计算机本身什么也做不了，必须依靠程序来指挥他做事。What exactly programming is?—The operation rather than the results. 程序就是操作流程的顺序，或者说是顺序排列的多个操作过程，它是方法的描述，同时又往往包含着《孙子兵法》中**分而治之&&各个击破**的思想——Programming is breaking a task down into small steps.

Long long ago, 有一个叫张三的人，是一个木匠，老本行是做家具的。一次李四让他帮做张摇椅，于是乎张三先把原木分割成木条木板，再把木条木板弯曲到指定的形状，然后把他们放置到适当的位置，接着设法固定他们，最后雕花抛光和上漆，一张漂亮的摇椅就做好了。这整个的流程如果记录下来，就是一个程序。任何程序都有三个要素，执行者、操作对象（也称为资源）和操作方法（指令）。在做摇椅的这个程序里，张三就是执行者，木头就是他所对付的资源，在指令的持续作用下，木头（资源）的状态（如形状大小、颜色 位置等）不断发生变化，最后，在程序结束时，木头变成了漂亮的摇椅。

电脑程序和上面一样，是方法的描述。只是这些程序的执行者不再是人，而是 CPU，命令也变成了 CPU 的指令（It's hard to imagine how to 给 CPU 下"去吃海棠盖浇饭"的指令），而资

[1] Wirth, Niklaus (1976). Algorithms + Data Structures = Programs. Prentice-Hall. ISBN 978-0-13-022418-7. 0130224189.

源则是 CPU 可以改变其状态的东西，通常是内存，当然端口&&硬盘等等也是，不过一般应用程序都只使用内存就可以完成工作。有的时候会听到别人说：“我的电脑内存是 500G 的，怎么玩起 DOTA 来还是很卡呢？”如果这种事情真的发生在你的电脑上，那我只能说：“恭喜你的电脑成功完成穿越，该电脑的准确生产年代不是史前，那应该就是在 2012 年之后。”一般来说个人电脑内存不超过 4G，硬盘 500G 倒是比较普遍了。CPU 并不直接从硬盘读取数据，而是通过内存间接获取。

有一点要注意，CPU 其实并不知道自己在做什么，是程序在指挥 CPU 的运作。这一点比较难理解，让我举例说明。来看一个算盘的计算，要使用算盘，只需要一件东西：口诀。记熟口诀（当然还有它对应的操作），就可以用算盘计算。在这个用算盘计算的过程里，口诀就是程序，指导着计算过程。算盘自身并不知道自己在计算，他只产生了物理上的一些变化（算珠位置的变化），做珠算的人同样也不需要知道，他只需要按照口诀调整算珠的位置。当程序结束的时候，算珠必定会处在某个位置上，这个位置的状态可以按照某种约定被读出，被读成某个数值，比如下面 4 个算珠全都在中档而上面的珠子没有落下的状态就是 4。

操作和储存状态的设备并不需要知道状态是怎么转换成信息的，转换由阅读者来完成。举个**不是非常准确**的例子，显示器在显示图像的时候，在显示屏上总是按照如下的规则进行：

坐标(1,1)黑色、(1,2)白色、(1,3)白色、(1,4)白色、(1,5)白色、(1,6)白色、(1,7)白色、(1,8)白色、(1,9)白色、(1,10)黑色……

坐标(2,1)黑色、(2,2)黑色、(2,3)黑色、(2,4)白色、(2,5)白色、(2,6)白色、(2,7)白色、(2,8)白色、(2,9)白色、(2,10)黑色……

虽然显示器只是在适当的坐标显示黑色或是白色，但我们却在显示器上看到了文字、图片和动画，你不会认为显示器知道这些是鸟山明的漫画吧？如果你能造出这样的显示器，那真是太有才了！

计算机也是一样。比如计算圆周率的程序，CPU 只是不断地对某一块内存进行操作，当程序结束的时候，这块内存恰好处在某种特殊的状态。而按照事先的约定，这个状态在被读出来的时候，它正好和圆周率相同，于是我们可以说，算出了圆周率。其实 CPU 只是在那里象手指头一样拨动内存的算珠而已。这种算珠极其简单，他只有两个位置，0 和 1，拨动它也很方便，电流就可以，但这种算珠实在太多，使得他们能组合起来表示很复杂的信息，就像只有黑白两色的屏幕点当数量足够多的时候，就可以用来表现有趣的漫画。

所以程序代表人期望电脑能做的事（注意不是电脑要做的事，这一直混淆着许多人），当人需要做这些事时，人提供指令，再给出某些资源以期电脑能对其做正确的改变。程序只是方法的描述，本身是不能发生任何效用的，直到它被执行，人为给定它一块内存，告诉它计算结果的精度及计算结果的存放位置后，他通过控制 CPU 才改变人为给定的这块内存的状态以表现出计算结果。

通常，我们把计算机的物理实体部分称为硬件包括电路板、机箱、键盘鼠标等，而把不可见的非实体部分称为软件，软件大体就是程序和主要由程序产生的数据。广义的说，乐谱、菜谱、工作手册、仪器的操作说明也是某种程序，我们不妨称之为类程序。

1.5 What is programming language？

如果以上所说的就是编程，那编程语言（比如 C 语言）又是怎么一回事？

菜谱、仪器的操作说明可以用法语来写、也可以用中文来写，不影响实质效果。因为人类生存在同一个四维物理时空中，具有相同或类似的感知。虽然人类的语言五花八门，但都可以通过翻译得到正解。

仪器操作说明、一般的菜谱，所描述的都是人类世界的事物，人类可以理解，因此它们可以用人类的语言来描述。但计算机程序显然与菜谱有不同，他是指挥计算机用的。首先 CPU 所能感受到的物理空间和与人类的感受严重不同，很多概念根本无法传达，其次没有大脑的计算机并不懂得人类的语言，何况人类的语言并不那么完美，很多事不能精确的描述，所以人类的语言不论英语还是中文都不能胜任这个任务。这个情况和音乐有点类似，解决方案是发明一种人造语言专门用于这个领域。比如五线谱就是一种专门的供音乐使用的人造语言（数字简谱也是，由于它与中国的工尺谱相当接近，所以在中国得到了最大程度的发扬光大，有点地方话的味道）。于是就有一些专门的纯粹用于计算机的语言被创造出来。

其中最早的一种基于电路原理，直接用 0 和 1 来表示电路的开关通断，不断的拨动开关，就形成了程序。这种语言就是机器语言，它可以直接被计算机听懂，但遗憾的是，人类虽然可以看懂这种语言，但它不符合我们通常的交流习惯，很难被人所阅读，更不要谈设计和修改了。

人类的智慧总是无穷的，后来人们提出这样一个方案，我们可以先按某种方法和规则，生成一个我们能看懂的指令序列（就是源代码），再通过某个转化的工具（就是编译器），把它变成机器可以运行的指令（也就是可执行程序）。这个我们能看懂的指令序列的规则的总和（也就是词汇和语法），就是我们通常说的计算机语言，为了和机器语言相区别，被称为高级语言。

章节开始所说文本形式的源代码其实是有些规定的，就像我们和老美用英语交流。首先你得说英文单词，不能冒法语词汇日文假名出来，其次你得按语法讲话，不能一个个单词往外蹦。程序也同样有词汇和语法上的一些规定，这些规定就构成一门语言[2]。显然任何一门编程语言都是人造语言，既然是人造的东西，因发明人的想法而不同，就形成了不同的语言。当然，编程语言的区分远不止以上所说的词汇和语法上的不同，还有其运行机制也不完全一样。

常见的编程语言有很多种，对于西电通信工程专业来说，本科阶段课表上明确规定需要学习的语言有 C & C++ & JAVA & Assembly language, 前三种语言近几年几乎一直占据着 TIOBE 编程语言排行榜的前三甲，其他著名语言有 PHP & C# & Python & Perl & So on…他们各有所长，在不同的领域发挥着各自的作用。**正如哲学有云：“存在即为合理。”**但由于计算机的体系结构大致相同，这些语言也大同小异，具有共通之处。这情况很好理解，通常真实世界的拳法看起来都有点相似，只有漫画这类幻想作品里才会有手脚飞出或者口吐火焰眼下喷水这种怪异的事情发生。

[2] 关于语言词汇和语法上的详细说明见《Linux C 一站式编程》 1.2 节 自然语言和形式语言

1.6 What is compiler？

前边不少地方提到编译器，那是什么东东？——代码翻译机，我在前面的章节中或多或少地解释了这个东西。不过我不介意再解释一次，是的，我刚刚说过，程序其实是电脉冲形式的指令的集合——对机器这是绝对正确的。但你认为人类可以直接操作电脉冲么？——当然不能。所以最早的时候，程序员们是通过反复的拨动开关或者插拔插头来做这件事的，就像老电影里的电话接线员和发报员。后来技术进化了，人们可以把脉冲信号设置在打了孔的纸带上，然后让纸带穿过有灯管的感应器，有孔电路就通，没有就是断，由纸带机实现了在电脉冲和纸带之间转换。老电影里的工程师们经常拿起一条长长的纸带来阅读——真是高深莫测啊！不过即使这样，要理解程序还是很困难，更别说阅读编写和修改了，人毕竟不是机器。其实人们最习惯用来表达思想的方法是文字，于是人们设想能否直接写出文字形式的程序。通过不懈的工作，这个目标实现了。今天人们可以写出文本形式的称为源代码的程序，然后再利用特定的工具把代码转换成机器能理解的电脉冲形式，也就是目标程序。**这种转换工具就叫做编译器，作用相当于翻译，以前是纯粹的机电设备，到了现代它也成了程序的一种。**

从某个角度来说，其实没有任何人能被称为程序员，编译器才是真正的程序制造者。人所制造的只是源代码，从这个角度往下想的话，其实是程序在制造程序，换言之，程序在借助人类之手自我进化。还记得前边说过的 Matrix 么？

在这里顺便讲解下编译和解释的差别。编译器的工作本质上类同于翻译，而我们知道其实翻译有口译和笔译两种工作模式。程序员写完所有的源代码，由编译器一次性转为可执行文件留待以后执行，这种类似笔译的模式我们称为编译。程序员每次输入一行或数行代码，编译器马上把他转换并执行，接着等待程序员的后续输入，这种类似即时口译的方式就称为解释，此时编译器就被叫作解释器。C/C++ & Pascal 等语言是编译型的，Perl & Python 等语言就是解释型的，Java 语言很特殊，他先编译成一种中间代码，然后在不同的机器上边解释边执行，这样就能实现跨平台运行，称为半编译模式，微软的.NET 也是这种机理。现代的新型解释性语言很多都是半编译的——兼顾了运行效率和跨平台性。一般来说，编译型要比解释型的运行效率高些——因为不需要等待程序员的输入，也便于系统优化。但解释型在编程的时候容易排错，界面友好，而且通常程序编写比较方便。不管用哪种语言编程，你总需要一个编译/解释器。

现代的编译器，往往不止是编译器，它还会包含有着色和搜索等功能的代码编辑器，支持单步调试并行调试的调试器，能够读入文件的多个版本并进行比较分析的版本控制，编辑图标等的资源编辑器，在大型项目中用于统一协调的项目管理，和用于自动化代码生成的向导工具等等。这样的编译器，我们就称它为集成开发环境(IDE : Integrated development environment)，代表就是开源的 Eclipse[3]&微软的 Visual Studio 系列[4]。

[3] Eclipse 是著名的跨平台的自由集成开发环境（IDE）。最初主要用来 Java 语言开发，目前亦有人通过插件使其作为 C、Python、PHP 等其他语言的开发工具。

[4]美国微软公司的开发工具套件系列产品。对高校学生有授权的 Professional 版本，一般在校内某 FTP 上有下载

Chapter2. How to learn programming ?

2.0 I heard that programming is very difficult. Can I understand it?

你觉得说中文难么？汉语是世界上最难学习掌握的语言之一，但你不是每天都能流利地用普通话和别人聊天么？任何一种计算机语言的难度都不会超过英语，更比不上汉语，编程这件事的难度基本上等价于指挥 IQ<20 的壮汉。那么你认为你能不能学会编程呢？

请记住学会和学好是两个层次，就像中学生作文和报刊出版物之间的差距。达到学会那个程度，只要有小学三年级的知识基础就可以开始了，也就是识字就行。而如果要学好，那需要初高中毕业的文化水准，也就是应该略懂英文和解析几何。当然还有更高的技术层次，比如在相当于职业作家的水平上，你将被称为专业程序员。而如果你被称为大师，那就是诺贝尔文学奖。抵达那个程度需要付出艰苦的努力，至于怎么达到，我想我还没资格谈论。

2.1 My English is poor

无法回避的事实，当今世界中的信息技术，绝大部分是欧美人的发明，更准确地说，是美国人的功劳。顺理成章的，各种资料文档技术手册，尤其是记载最新技术的， 都是用美国通用语也就是英语撰写的。可能在其它领域不懂英语没有什么大的问题，但在电子通信&&IT 业，不懂英语就是无法掌握最先进的技术，基本等于没有出头之日。不懂？——不懂就去查呗。

不过话说回来，不是说你一定要有个托福雅思的高分才能去学编程的。由于历史原因，大部分计算机语言借用了部分英语的词汇作为基本词汇，但绝不是说必须先学英语才能学习编程。就编程本身而言，它所需要的英语水平不过是死记硬背好几个单词而已。你在看好莱坞大片时，一部片子下来总能记住主角和主要配角的名字吧，C 语言全部关键字一共 32 个，而其中有 6 到 7 个的使用率超过 78%，这样你还有什么可担心的。但是，要想成为高手，阅读大量的相关资料是免不了的，这个时候，英语就显出它的重要性了。**能使用英文原版的软件就尽量不要用中文汉化版，能读经典的原版英文教材就尽量不要读翻译过来的中文版。**在这里不是崇洋媚外，而是使用汉化版的软件极易引起各种不可预料的问题，国人翻译过来的多多少少有失原味。下边来一段小插曲。

> 如果只因为那几个屈指可数的英文单词而放弃学习编程，我只能说这样的人趁早离开为好。顺便提一句，本文作者（这里指 BlueAuris），一向认为那个叫易语言的中文编程语言是个不折不扣的笑话。理由很简单，其他流行的语言都有自己的独到之处，就像武当剑少林拳打狗棒一样在江湖上占有一席之地，而这个语言除了有几个中文标识符之外，一无所长。而这几个中文标识符也不过是使用了文本替换的方式把 C 语言的几个特定词汇换成了意思相对应的中文而已，任何一个文本编辑器都可以做到这一点。形象的表现一下，这就是有个猥琐的家伙特地身穿全套阿拉伯长袍练了整路正宗少林长拳然后大声叫嚷这就是他发明的具有民族特色的中东石油大亨拳。

2.2 Can I learn programming well?

如果你已经看完前面的部分到达这里，显然你是确实想要学习编程的新手。不论你是为了什么目的来学习，在看了上面的话之后是不是很有信心呢？不过我要打击你一下，不是谁都学得好编程的。人人学开车，但不是谁都能上赛道，舒马赫更是只有那么一个。就像有些惊险刺激的游乐设施禁止高血压心脏病患者参与一样，编程作为一项耗费智力和体力的活动，对参与者也有一定的要求。先来看看吧，满足以下这些条件，你就可以放心大胆的开始了。:-)

先说体质要求。太祖说：身体是革命的本钱。健康的身体对编程大有益处，但并不是说残疾人就不能参与。就目前的技术水平，除了脑瘫和目盲这两项，其他的肢体残缺根本不妨碍。Stephen William Hawking 在这样严重的情况下还能持续研究的事实，还不够激励你么。\(^o^)/~

紧随其后的是足够的精力和时间。Dear friend，只要你能静坐半小时安静的看完这本小册子，你的体能就达标了。如果你想告诉我，你能够一天在电脑前连续操作 18 小时以上，我要说的是，Dear friend，虽然你很犀利，但请注意保护好身体和眼睛。只要每天你能抽出 30 分钟来学习编程，那你的时间也合格了，当然有更多时间确实会更好，不过也没必要每天 12 小时。编程并不是世界上最重要的事，我们还有别的事要做。必须认识到，学习编程重要的是持之以恒，而不是依靠爆发力，每天半小时比一周一次 6 小时效果好的多。

第三个要求是你要略微懂一点计算机。不错，只要略懂就可以了。因为这正是**本文的主要目的之一：向略懂计算机的人介绍编程。**那么，怎么才算略懂呢？能浏览网页&用文本编辑器输入代码&把键盘上的 26 个字母和十个数字挨个输入一遍就 OK 了！

最后一点，你需要有顽强的毅力。编程并不像你想象的那样轻松，不是野餐和聚会。尤其是对于职业程序员都应掌握的 C++，它可以用两句话来形容：三年不开张，开张吃三年。千万记住，C++被称为是真正的程序员使用的语言不是没有理由的，它的复杂度和性能超出你的想象。有无数的编程新手在第一个月不到就放弃了，你最好确定你不是他们中的一员。毅力没有尺子可以来度量，在这里我只能先祝愿各位都能坚持到最后。Good luck~

2.3 Must I have a computer?

到目前为止，是的。在老年时期，Beethoven 可以凭空作曲而不依靠钢琴之类的东西，但几乎没有哪个程序员不依靠电脑而只在大脑里编写，尤其是开始学习的阶段。但这也不意味着只能在电脑上才能编程。在电脑还属于稀有电子产品的那个年代，哪有那么多人能非常方便的在电脑上编程？但最后不同样也诞生了不少优秀程序员么？那他们是怎样编程的呢？——纸&笔&&大脑。当然，能有一台电脑显然更好。

用各种语言所编写的程序被运行在各种各样的机器和设备上，从掌上设备到巨型服务器，从台式电脑到微波炉，所有能够自动运行的地方都有程序的身影。但是很遗憾，并不是只要能够运行的设备就可以用来进行编程的，这就像虽然可能洗衣机也能发出悦耳的声音，但你不能指望用洗衣机来录制流行歌曲（额。。。也许 SONY 有这样的产品，我保留意见）。

目前大部分语言，所需要的电脑并不如你想象的那样高级。当然作为学习，我们还是需要有

比较称手的设备。不错，最重要的就是称手两个字，所谓称手，就是不会由于设备的问题，妨碍你的思考，不需要高速的处理器，也不需要巨大的显示器，更不需要海量的硬盘，只要称手。当然，如果你是游戏或者影音发烧友，那你可以当我说的是废话。编程不是豪华海上旅游，而是修行。修行不需要五星级宾馆，但也要有破屋以遮风雨，否则感冒发烧了，怎么来修行呢？

2.4 The first step of programming

首先确保你的健康状况和基本计算机操作水平（不会？不会就去学，健康状况不好？这个我只能建议你多锻炼身体）。

编程这件事上，没人可以无师自通，天才也不行。所以你要做的第一件事是找一本好的入门教材，最好是经典作品（后边会集中介绍）。反复地看教材，要牢记一点，你所提出的大部分问题，教材上都有解释，只是你没有认真看而已。反复地阅读，直到你觉得你已经可以编写出那本教材了，才可以丢弃它。教材比老师好的理由之一是，你可以带教材上厕所去卧室，而老师不行。当然，老师也有比教材优越的地方，那就是他可以给你解释教材上没有的问题。但记住老师也是人，**你不动脑子就去找 TA，TA 会厌烦的。**不要鄙视你的老师，即使 TA 其他方面不如你，但至少在编程这件事上 TA 能做你的老师就是 TA 比你强的硬道理。尊师重道是中华民族的传统美德，O(∩_∩)O 哈哈~

你以为光看语法书和背诵单词而不练嘴就能说好英语么？显然不对。编程也是，熟能生巧在任何地方都是一样的。首先看懂教材上的那些例子，确保看懂之后，按着他的思路把它默写出来，当你尝试过就会知道看懂和默写是两个完全不同的程度。然后就可以做书后面的习题，独立想，想好之后最好是能在电脑上敲一遍，**不到万不得已不要看答案或提示。**等到整本书后面的习题你都能做对的时候，你就算入门了。如果你的目的仅仅只是为了考试拿高分，现在就可以去睡大觉了，但如果想要用编程来解决一些实际问题，sorry，你还欠火候。这其中的差别就相当于大学英语四级作文和畅销英文小说之间的差异。

会做书后边的习题只是编程的第一步而已，你掌握了大量的单词和熟悉语法并不能让你写出优美的英语小说。你得学习修辞手法&谋篇布局这类文学技巧，还得掌握历史典故、谚语俗话以及文化背景这些文字外的东西。如果要畅销，还需要超凡的主题&生动的故事&&跌宕的情节。编程圈子里正好有和这些类似的玩意儿，这些都留到下一章细说。

2.5 If problems come up, who can I turn to help?

郑钧的《路漫漫》歌词中有：

记住没人会同情你 我亲爱的兄弟 你最好鼓起勇气才能活下去
因为路漫漫 其修远 我们要上下而战斗

编程这件事，基本上是不能指望有人帮你的，因为你很可能问了别人半天后才发现你们俩说的不是一个东西。当你编程遇到问题，首先应该是去看编译器提供的信息，它可是最直接的来源，相当于案发现场，很少有刑警不看尸检报告就直接破案的（当然，对于柯南这种我保持沉默）。现代的编译器已经不止编译那么简单，编译本身就能送出大量的提示，调试功能更是强大到可以让你检视程序运行的每一步都发生了什么变化，只要你能看懂。机器的问题， 就应该用机器来解决，而不是用人的肉眼去检查（在你还没练就火眼金睛前）。不过也不要过分依赖编译器，**尤其是初学编程的时候，**有错误时先在纸上用大脑跑一跑程序，实在搞不定了再单步跟踪，多试几次你的编程能力自然会有很大的提高。

然后就去查教材和文档。手边的教材能解决 70%以上的问题，Linux 下的 man 手册就非常不错，如果还不够，上网去查。如果找不到，请记住有个东西叫搜索引擎，首推 Google[5]，90%的问题可以得到解决，然后再推荐一个 Wikipedia，比起那个百度百科还是要好很多的，以前在一篇帖子中就指出：Wikipedia 是百度百科他爷。**后边如果遇到有不懂的术语请自行 Google || Wikipedia。**

如果还是不行，确认你已经努力过而没有答案，接下来可以向老师请教或者在邮件列表上提问。在你确定要提问之前，请先把《提问的智慧》[6]好好读一读。解决完一个问题后可以在开源社区 BBS 和邮件列表上和大伙儿分享你的经验，独乐乐不如众乐乐。:）

[5] Google 在大陆地区访问是存在干扰的，有关正常使用 Google 的各种服务方法另附
[6] 中英双语将会在开源社区 BBS 上发布，附录也会附上

Chapter3. Advanced Programming - Part I

3.0 Data Structures & Algorithms

首先让我们看看 Wikipedia 上对数据结构和算法的定义。

> 数据结构（Data Structure）是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以带来最优效率的算法。
>
> 算法（Algorithm）是指完成一个任务所需要的具体步骤和方法。也就是说给定初始状态或输入数据，能够得出所要求或期望的终止状态或输出数据。

算法和数据结构就是程序里的修辞手法&谋篇布局。人类编程虽然不过几十年，但运用程序所解决的问题，已经覆盖世界的每个角落各个方面。各种各样的问题，被前辈和大师提炼归纳，有些人直接找出了解决的方法，有些人找到了寻找解决方法的途径，还有些人索性证明了在现阶段是不可能解决的，这些解决方案就被统称为算法。**学习算法就是学习前人的智慧，少走弯路。禅语有云：不走弯路就是捷径!!!** 连牛顿爵士都是站在巨人的肩膀上的，除非你自我感觉比老牛还牛，凭空就能解决别人十几年才想清楚的问题。需要注意的是算法必须依附于具体的数据结构而存在，脱离了数据结构来谈算法意义不大。为了更形象的说明这一关系，下面我们来玩一个经典的数学游戏：

> 有三个瓶子，容量分别为两升、三升、七升，现在要快速准确量出六升的水，请问该怎样得到？
>
> 显然我们必须用这三个瓶子组合起来才能得到准确的六升水，那如何得到呢？我想聪明的你肯定很快就有了答案。接下来的问题就是怎么用算法表示出来呢？——这就是在考验我们的数学思维能力了。

瓶子装水这一类问题抽象出来其实就是一个求不定方程整数解的问题。OK，点到为止。按照 Wikipedia 上对数据结构的定义，我们可以把上边装水的瓶子类比为数据结构——瓶子在人的干预下干了装水&倒水的工作，通过三个步骤——你的算法，最终实现了在七升的瓶子中装了六升的水。通过三个瓶子才能得到六升水，这种事肯定不是聪明的人想出来的，那么聪明一点的人会怎么想呢？——找一个六升的瓶子不就一步完成了嘛！

前辈们已经总结出很多算法和产生算法的方法，我们可以直接学习。如果你积极进取，总有一天，你会发现有需要自己开创新的算法的时候。这个时候，数学功底会帮你很大的忙。也许只是数学工具在起作用，但更有可能的是你的大脑受过的数学思想训练在帮助你。**总之，为了前途着想，提高数学素养是没错的。这不是说多背数学公式和多做数学题，而是指一种数学的思维方式。**

学算法很简单，也是找教材，做习题。教材容易找，但新手往往找不到合适的习题。可以尝试在完成教材上的所有习题之后去找编程竞赛的练习题来做,也就是所谓的 Online Judge[7]，后续将深入讨论这个东西。

[7] An online judge is an online system to test programs in programming contests. The system can compile and execute codes, and test them with pre-constructed data.

3.1 Others

相当于历史典故、谚语俗话、文化背景的东西，就是各个编译器和平台上的接口和库了。假设现在有个程序要读写文件，不要误认为你需要亲自写个程序去控制硬盘的磁头伸缩，或者是光驱的透镜移动或者是U盘的地址定位，除非你是想做个Linux或者Window那样的操作系统。否则所有包括文件操作、网络通讯、人机界面这些，都是由操作系统提供的现成模块，只等着你来使用。这些模块通常称为应用程序接口(API - Application Programming Interface)，不同的操作系统提供的API不一样。在接口的基础上，很多编译器和程序员做了进一步的包装形成了库，你可以比直接使用接口更方便地使用这些库而达到同样的功能。

由于系统的不一致，编译器的不同，所以在学库之前，先要确定你所要工作的平台和环境，还有应用方向。Windows下编游戏和Linux下做数据库用的库是大相径庭的。然后就是同样的一套步骤，找教材，做练习。这个教材通常就是官方文档，Unix/Linux平台有大量的文档，分布于man页和各种手册上，Windows下最好最全的就是MSDN，其他平台自行搜索。可以去找习题，**但更好的方案是去找一个实际的小型应用，在使用中学习效果无与伦比。**编写一个QQ或者BT，绝对能让你对网络操作部分了如指掌。网络上有很多开源项目，有兴趣可以自己去找找。

学习库和学习算法可以同时进行，在你完成这两个阶段的时候，你已经是一个合格甚至是优秀的程序员了。

优秀的文学作品有个共同的特征，他们虽然立足于不同的民族文化，但关心的却是全人类共通的思想感情，体现着终极的人文关怀，我们都会为 *The Old Man and the Sea* 中的Santiago所感动不是么。优秀的程序虽然应用方向不同以及平台各异，但它们一定都完全符合计算机原理，并且用最合理的数学模型来展现。如果你想成为合格的程序员，**计算机原理和相关的数学知识是一定要补习的理论课。**

除了看书和做题之外，还有一个内容不可缺少，就是阅读别人的程序。没有哪个作家不大量阅读别人的作品，同样你也可以从别人的代码中吸取营养。代码就是程序的全部，是真实的实现方法，一切都在代码中，**甚至有时长篇累牍的说明还不如几行代码清晰明白**。今天的程序员是幸运的，开源运动的发展使得他们能够无偿而方便地得到世界上最优秀的并且是实际运作中的代码，几乎遍布任何应用领域。只要你有心，就可以找到任何想要的代码。但读代码也是一件考验你毅力的事，读优秀的代码更是一种享受。**但请阅读和你水平相当的代码，差距太大将会是严重的身心打击。**你不会告诉我你C都还没学就想精通Linux内核吧？

超凡的主题跌宕的情节——其实就是你的程序的应用方向。如果说前面都是练习的话，那现在就是你自主创新的时候了。很多人都只是在老板的安排下为了工资而被动做些既不喜欢又没有价值的流水线产品，少数人才有机会做自己喜欢的东西。想想看， 3DMAX、WOW、Firefox、Apache、Linux甚至Mac OS X都是多么知名的程序，也许有朝一日你的程序将会和它们一样知名。不过我还是不得不给你泼冷水，这需要不懈的努力&&敏锐的眼光&&少量的运气，只有极少数的程序员能做到这一点。**不过，有梦想才能不断前进，不是么？:-)**

Chapter4. Advanced Programming - Part II

4.0 Preliminaries to Algorithms

“能看懂别人的程序，但自己就是写不出来，Why？”——这个问题其实每个刚开始学习编程的人都会遇到，你所见到的各位达人大牛都曾经有过这么一段经历，所以完全不必为这种情况而怀疑自己的能力。**为什么会有这样的情况出现？——因为思维模式。**

在小学的数学教材里，有一种题型，叫应用题。它会给出很多生活中的场景，然后让你用数学知识来解决。在解这种题时，其实分为三个步骤，首先是要提取出数理模型，比如常见的追逐相遇这类问题，就要使用速度时间模型，然后把这个模型数学化，找出各个变量之间的关系，确定已知量和未知量，形成可求解的方程，最后求解。

编程的情况与此类似。首先要建立一个抽象描述模型，然后建立数学表达，接下来略有不同，不是亲自求解，而是给出求解的方法——也就是算法，最后是把算法转化为程序。而新手通常之所以会卡壳，是由于这个流程中有两个难关。建立模型不是问题，数学表达也不难，但找出算法却是非常艰难的事情，即使找到正确的算法，要把它写成正确的代码也不容易。新手常说我在学习 XX 语言,XX 语言真复杂啊。其实学习语言本身只能保证你在最后一步，也就是翻译代码（又名 coding）那里少出错误，即使你顺利的学习了一万种语言，你也会觉得编程很难，假如你没有学习算法的话。让我们找个具体的例子来说明，假设现在有个题目要找 N 个正整数中的最大值。显然这个题目模型很清楚，本身就是数学问题，也不需要数学表达了。接下来就是解法，新手这时就卡在这个地方了。

刚接手这个题目，很多人就会想用一种类似人类的快捷操作，比如三个数，瞥一眼就可以找出最大值，四个数也毫无问题，甚至十个数也是一下子。这时我问你，你怎么把这个瞥一眼的动作表示成程序，另外如果 N 大于 10000 怎么办？——哑口无言。原因是，人类的头脑过于聪明，可以同时处理很多事务，也就是可以并行处理一定量的数据（当然大规模数据就要另外对待）。而计算机——很遗憾，没有这种能力。这个时候你肯定会说：“现在不是有多处理器多核多线程等各种各样的并行处理的计算机了么？”我要告诉你，那些都是不同层次的概念。目前这个时代的计算机，在出现革命性的变化之前，从 CPU 指令的层次来说，都是单线程单参数工作的。再说明白一点，这些机器任何时候只能一次处理两个数，而且其中一个还必须已经在 CPU 内部了，任何 N>=3 个数相加都必须转化成持续的两个数相加，就是先把第一个第二个加起来得到结果之后，才能和第三个相加，照此重复求得所有的和——这是目前的科技无法改变的铁律。这个时候我要请你记住一个重要的思想：**编程中任何问题都要分解到足够小，小到机器可以一次解决的程度！**当然，这不是意味着我们每次编程都得从 CPU 执行一次指令的角度去解决问题。

回到刚才的那个题目：寻找 N 个正整数中的最大值。我们知道直接解决是不可能的。而按照刚才讲过的铁律，我们知道直接找到两个数中的最大值是一次可以做到的。那怎样从 2 个扩展到 N 个呢？这里就是算法的天下了。一种很常见的想法是，完全可以从两个中找出最大值，再让它和接下来的一个比较，这就是 N=3 的情况，再把三个中的最大值和第四个比较，这就解

决了 N=4，依此类推，我们似乎找到了通用的算法。是的，找到前 N-1 个中的最大值，然后与第 N 个比较——不要怀疑，这个算法方向是正确的。接下来就是把它细化使他能变成代码。你注意到，首先要设法从 1 增加到 N，而且每次前进一步都要做类似的操作，显然可以用一个循环来实现。每一次循环中，都需要将保留的最大值和当前的这第 n 个数比较，如果最大值比它大，那就保留，否则就要把最大值替换成新的——这就是条件语句的作用了。写完这个循环之后，还有些小细节，比如这个最大值在与第一个数比较之前应该是多少呢？太大的话，可能会比整个数列的数都大，这就会出问题。因此常用的做法是，就让他等于第一个数。然后包括读入那 N 个数，而输出最大值这些琐碎的细节就属于收尾工作了，没什么可多谈的。当然，即使是这样的小题，也不仅这一种算法。你记不记得有一种叫做单淘汰赛的机制——最后顶点的就是最大值。用在这个地方正合适。不过，如果要把这个淘汰赛算法实现成程序的话，如何实现分组，如何表达这个淘汰过程和取出顶点的值，正是算法描述里要解决的。这个就是排序里很有名的最大堆排序。**一旦算法（伪代码）描述齐备，程序编写不过是打字校对的工作。**

为什么你可以看懂别人的程序呢？——因为 TA 的算法隐含在程序中已经被实现了。就像你读王维的诗，总能在眼前浮现出一幅幅绝美的风景画，但轮到自己写，却描绘不出那样的画面。一方面是因为你束手无策，不知道怎样找到可实现的算法；另一方面是即使你找到了算法，也只是爱在心中口难开，不知道怎样去表达。

算法总是从问题出发，通过一定的模式，逐渐细化再细化，直到可以直接转成程序。新手很难一下子领会什么样的算法是可以实现的，但好在新手接触的问题一般不是很难，算法通常非常清楚明白，所以重点是要解决后面那个怎样把算法表达出来的问题。因此在这里建议各位新手默写教材上经典的例题程序。很显然对于那些例题，只要你用心看过就会领会它的算法。那么，你再默写一遍，即使和它的原程序样子不一样，也总算是把这个算法表达出来了。反复这样练习，这个表达问题不就解决了么？而且在这个过程中，至少你学到了一个算法。基于此原则，任何你遇到的可以看懂的例程（当然要是好的例程才行），都建议你默写它——尤其是开源的精品代码。

4.1 Algorithms + Data Structures = Programs

It is out of question that most people have heard the above statement. But do you understand it?

Algorithms + Data Structures = Programs[8] is a 1976 book written by Niklaus Wirth covering some of the fundamental topics of computer programming, particularly that algorithms and data structures are inherently related.[9]

算法与数据结构的基本概念已经在“Data Structures & Algorithms”章节介绍过了，这里再稍微深入一步探讨，对于新手来说不必在本小节纠结过多，本节的存在只不过是为了本章的完整性而充实的。

A way of solving a problem will (generally) only be accepted if we can demonstrate that it

[8] Wirth, Niklaus (1976) (in English). Algorithms + Data Structures = Programs. Prentice-Hall.
[9] http://en.wikipedia.org/wiki/Algorithms_%2B_Data_Structures_%3D_Programs#cite_note-0

always works. This, of course, includes proving that the efficiency of the method is as claimed.[10]

实际上，数据结构与算法解决的问题是整个编程中最有限，最底层的那些问题，(它没有涉及到设计，用户等编程三层架构中的最重要的后二层)。它只解决**对于计算机**在组织内存,支持对这些内存中数据进行操作(排序，查找)等有限问题的问题，它仅仅能很好地解决这些问题，所以说它是面向计算机的功能性方案，是计算机的科学，解决对于计算机来说最为迫切要解决的那些问题，比如效益敏感类问题。如果放在整个大编程中来讨论，那么它是颇为有限的那类东西。

承载计算机科学最最根本的东西，是数据结构跟算法，而不是语言(语言只是表达工具)。难怪宏大的 TAOCP[11]写的几乎全是数学和算法分析等内容，跟语言相比，语言本身并不能解决一个问题，它只是反映事物的工具，跟解决问题没有绝对的联系，数据结构与算法才是“真正能解决”计算机问题的手段与技术。你看路由器算法，这些底层的东西，都是数据结构与算法发挥作用的地方。

于是，当用数学和机器的眼光直面解决问题的时候，很自然地就产生了一门学科，即数据结构和算法。

算法与数据结构实际上就是计算机编程界将应用问题离散化建模的方案，这样，就可以将应用问题转化为软件上可用的抽象，而所有一切软件上的问题，不过都是抽象。[12]

4.2 Text Editor

不知道上面那一段话把读者绕晕了没，我希望没有，(*^__^*) 嘻嘻……打起精神让我们以轻松的姿态迎来下面的章节。

在“4.0 Preliminaries to Algorithms”章节中曾提到“一旦算法（伪代码）描述齐备，程序编写不过是打字校对的工作。”咱们平时学习工作都非常讲究效率，那么“打字校对”的工作是不是也可以也有一些比较好的办法来提高效率呢？答案是肯定的！一些聪明的程序员早已想好了各种办法来提高 Coding 的效率，其中之一便是选择一款适合自己的高效 Text Editor（用来输入程序代码的玩意儿，称为文本编辑器）。Windows 下大家最常见的恐怕就是记事本程序咯，够小巧，但我想应该没几个人愿意一直用这玩意儿吧？输几行代码进去它似乎什么反应也没有，No Keywords highlight. Linux 下最普通的便是 Gedit 了，但简单不代表简约，比起 Windows 那个 notepad 可要威武多了。不过比起下面即将出场的两位大神恐怕他们都得往一边站，OK,该轮到 Vim 和 Emacs 出场了。

Vim-Vi IMproved：Vim 是从 vi 发展出来的一个文本编辑器。代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。开始学习的时候可能会进展缓慢，但是一旦掌握一些基本操作之后，能大幅度提高编辑效率。详细介绍和教程自行 Google & Wikipedia || 查看 help 手册。

---

[10] Martin Richards “Data Structures and Algorithms” *Lecture notes* in Computer Laboratory University of Cambridge (October 11, 2001)
[11] *The Art of Computer Programming* Donald E. Knuth
[12]引自《编程新手真言》Ver3.0

Emacs：Emacs 即 Editor MACroS（宏编辑器），是一种文本编辑器，在程序员和其他以技术工作为主的计算机用户中广受欢迎。同上，相关详细介绍自行…

Vim-the god of editors, Emacs-the god's editor 两款神器我就不浪费时间多扯了，西电开源社区的 Wiki 有他人精心整理的学习笔记。

和平时大多数人用 wps OR word 写文档一样，用的熟练的人很快就能完成一份精美的文档，而对于不熟练的人来说就只能面对屏幕望洋兴叹喽，学习新事物是需要成本的，你现在的每一滴汗水都是在为你自己的将来投资。

4.3 Coding Styles

代码风格好不好就像字写得好不好看一样，如果一个公司招聘秘书，肯定不要字写得难看的，同理，代码风格糟糕的程序员肯定也是不称职的。虽然编译器不会挑剔难看的代码，照样能编译通过，但是和你一个 Team 的其他程序员肯定受不了，你自己也受不了，写完代码几天之后再来看，自己都不知道自己写的是什么。Thus, programs must be written for people to read, and only incidentally for machines to execute. 代码主要是为了写给人看的，而不是写给机器看的，只是顺便也能用机器执行而已，如果是为了写给机器看那直接写机器指令就好了，没必要用高级语言了。代码和语言文字一样是为了表达思想、记载信息，所以一定要写得清楚整洁才能有效地表达。正因为如此，在一个软件项目中，代码风格一般都用文档规定死了，所有参与项目的人不管他自己原来是什么风格，都要遵守统一的风格。在编程初期就要养成好的习惯!!! [13]

4.4 After programming languages - The OS & platform

在众多的程序里，有一大类特殊的程序，它们就叫操作系统（0S: Operating System）。操作系统是最基础的程序，它让计算机运行起来&所有的硬件都做好准备&接受别的程序给予的指令。相应的，其它程序就叫应用程序。操作系统和应用程序的关系，就像人的基本意识和数学水平一样。想让一个连基本意识都没有的人（譬如植物人）参加湖南数学高考——看来你和我都疯了。一般的计算机都是硬件、操作系统和应用程序相互分离的，需要的时候分别安装。有些特殊的设备直接把操作系统做在硬件里（比如嵌入式操作系统），比如各种电子游戏机，可以开机，但是要有游戏光盘或游戏卡才能玩，还有些计算机把操作系统和应用程序都做在一起，放在机器内部，N 年前很流行的俄罗斯方块掌上游戏机和电子宠物就是这样的设备。书写到这儿让我想到了“笨兔兔的故事”[14]，让我们一起来欣赏一下这段美丽的动人传说吧~

Long long ago, 就是上个世纪 60 年代的美国一个春天啦，那个时代的计算机是个新鲜玩意，非常笨重，家庭用户是没有的，都是商用或者试验，科学计算用的机器。你说你想买个电脑斗地主？把你卖给地主你也买不起呀。再说那时候的计算机不是随便一个人就会用的，那时候的计算机使用的时候是由人来输入一条条的指令来进行各种运算的。他们输入的指令大约相当于现在的汇编指令，所以这个效率和操作难度有多高就可想而知了。那时候计算机大都没有什么操作系统，顶多有个批处理系统，可以把要输入的指令记录在某种媒介上（比如纸带）一次性

[13] 本段引自《Linux C 编程一站式学习》，如与 GFDL 许可证相冲突，请作者及时与我联系
[14] http://forum.ubuntu.org.cn/viewtopic.php?f=112&t=162040

输入进去，让人们省去一条条重复输入指令的麻烦。后来慢慢有了很简单的操作系统，但并不像现在我们见到的操作系统这样通用。这个时候，卖计算机的厂商要为每一型号的计算机设计不同的操作系统，一个程序如果在这个型号的计算机上写好了，拿到其他型号的计算机上是运行不了的，因为这两台机器连操作系统都不一样，怎么可能程序通用呢。计算机要是老这样肯定是不行啦，否则你今天要玩斗地主，人家游戏公司就得专门派人到你家机器上现写出一个来——因为不同型号的计算机上的操作系统不同用嘛。这个斗地主的问题，终于还是被那个时代IT业界的大地主，蓝色的IBM公司率先着手解决了。1964年他们公司推出了一个系列的大型机，用途、价位各不一样，但他们上面运行的操作系统，都是System/360。（这360可不是卖鞋的，也不是跟QQ打架的那个）这一下获得了很大的成功，因为省去了为每一台电脑单独编写系统的成本嘛。直到今天，IBM 的大型机上依然可以运行这个360 系统，可见其当初设计时充分考虑的兼容性。然而我们要讲的主角不是360，而是另一个伟大的操作系统。那时候有个聚集了很多牛人的地方，叫做贝尔实验室，是 1925 年由 AT&T 公司成立的。一帮头脑发达四肢也不一定简单的家伙整天聚在那里，研究新奇的东西，什么任意门啊，竹蜻蜓啊，记忆面包啊——呃……都不是他们发明的（发明这些的人是个日本科学家）。那贝尔实验室那帮人研究什么呢？贝尔实验室的工作可以大致分为三个类别：基础研究，系统工程和应用开发。在基础研究方面主要从事电信技术的基础理论研究，包括数学、物理学、材料科学、行为科学和计算机编程理论，反正都是大学听不懂的那几门就对了。系统工程主要研究构成电信网络的高度复杂系统。开发部门是贝尔实验室最大的部门，负责设计构成贝尔系统电信网络的设备和软件。具体来说贝尔实验室研究出来过的东西有晶体管、发光二极管、数字交换机、通信卫星、电子数字计算机、蜂窝移动通信、有声电影、立体声录音，等等。（怎么样，不比机器猫那些东西差吧？）通信网的许多重大发明都诞生自这里。那时候还有个聚集了很多牛人的地方，叫做麻省理工学院（MIT），这是美国的一所综合性私立大学，有“世界理工大学之最”的美名。从这里走出的牛人很多，到2009 年为止，先后有76 位诺贝尔奖得主，都曾经在麻省理工学院学习或者工作。麻省理工学院的自然及工程科学在世界上享有极佳的盛誉，其管理学、经济学、哲学、政治学、语言学也同样优秀。另外，麻省理工研发高科技武器和美国最高机密的林肯实验室、领先世界一流的计算机科学及人工智能实验室、世界尖端的媒体实验室、和培养了许多全球顶尖首席执行官斯隆管理学院也都是麻省理工赫赫有名宝贵资产。有着毋庸置疑的实力，麻省理工自然非常不差钱，截至2008 年底麻省理工有101 亿美元的总资产，因为不差钱，于是该校对家庭年收入低于75000美元的学生一律免学费。那时候，又有个聚集了很多牛人的地方。（哪那么多地方阿！）这地方是个公司，叫做通用电气。这公司可是个大公司，当年是个卖灯泡的，他们的灯泡可非同一般，虽然不节能，虽然寿命不如现在的长，虽然价格比现在贵， 虽然外形也不一定好看，但是——他们是第一家卖灯泡的！因为他们的老大，就是大名鼎鼎的Thomas Edison。1876年，发明灯泡的爱迪生同学成立了爱迪生灯泡厂，为节约蜡烛和灯油做出了突出的贡献，估计那年的五一劳动奖章肯定是他的了。到1890年，爱迪生同学将灯泡厂重组，成立的爱迪生通用电气公司，到1892 年又与汤姆森－休斯顿电气公司合并，成立了通用电气公司。好，时间到了1965 年，这三个聚集着不少牛人的地方有一天忽然想合作一把。于是，大名鼎鼎的贝尔实验室，大名鼎鼎的麻省理工学院和大名鼎鼎的通用电气公司一起开始了一个制作操作系统的计划。为了结束长期以来计算机上面没有统一的操作系统的混乱局面，他们决定要创造出**一套前无古人后无来者，上得厅堂，下得厨房，念天地之悠悠，独怆然而泣下**的惊世骇俗的操作系统！具体来说吧，

这个操作系统应该是一个支持多使用者、多任务、多层次的操作系统，因为这三多，所以这个操作系统就起名叫做——MULTICS。（难道你以为叫许三多？人家讲的是英语好不好）有了这三家的强强联合，那开发的结果还用问么？这个 MULTICS 操作系统的项目在 1965 年成立，到了 1969 年就……被取消了。 咳咳，这个……其实编写操作系统也不是一件容易的事儿啦。毕竟道路是曲折滴，研究是辛苦滴，成绩还是有滴，失败捏……也是可以原谅滴嘛。项目失败了，大家都很沮丧。在这些沮丧的人中，Kenneth Lane Thompson 只是很普通的一个——1943 年出生在美国的新奥尔良，吃着烤翅长大的他没有辜负养育他长大的父母和那些没有了翅膀的鸡。1960 年，汤普逊考上了加州大学博克莱分校主修电气工程，顺利取得了电子工程硕士的学位。1966 年，他加入了贝尔实验室，参与了 MULTICS 项目。做项目是个很辛苦的事情，在疲劳的揉揉因熬夜而发红的眼睛后，他很想能有个电脑游戏来玩玩。然而那时候别说超级玛丽，连吃豆也没有啊！所以汤普逊同学就自己编了一个游戏，叫做星际旅行。这个星际旅行跟星际争霸那肯定是没的比的了，不过在那时候已经算是很有吸引力了。这个游戏自然是被设计运行在 MULTICS 系统上的，由于 MULTICS 系统还不完善，所以游戏运行的也不是很流畅，所以，能够顺畅的玩星际旅行，成为了汤普逊同学努力工作的动力。可是后来项目被干掉了，如果事情就这样结束，那么汤普逊同学就再也不可能流畅的玩他的星际旅行了，这是多么遗憾的事情阿。可是现实是残酷的，项目确实就是取消了，要想顺畅的玩游戏怎么办？毛主席教导我们说：自己动手，丰衣足食。我估计汤普逊没有背过毛泽东语录，但是他用自己的行动证明了这两句话的正确性。他在墙角淘换出一台 PDP-7 的机器，并且伙同其同事 Dennis Ritchie，打算将星际旅行移植到这台 PDP-7 上。当然，要想运行这游戏，还是得有个系统。有了固定的系统，那以后再编写别的游戏就更方便了。可是系统从哪里来？MULTICS？已经停工了，并且这系统绝对不是两个人可以搞定的。那怎么办？还得自己动手！于是 Ken Thompson 和 Dennis Ritchie 再次发扬自己动手的精神，用汇编语言写出来个系统，这就是最初的，非常简陋的，UNIX 的前身。这个系统不像 MULTICS 那么牛，不支持很多的用户，只能支持两个用户，（估计是为了避免做好了之后俩人抢机器玩的局面发生。也可能是为了以后俩人对战？）支持的进程也有限，其他功能也都没有 MULTICS 设计的那么复杂。相对于那个 MULTICS 系统——MULTiplexed Information and Computing System，Brian Kernighan 开玩笑地戏称他们的系统其实是："UNiplexed Information and Computing System"，缩写为"UNICS"。后来大家取其谐音，就诞生了 UNIX 这个词。这一年，已经是 1970 年，史称 Unix 元年。直到现在，计算机中都是用 1970 年 1 月 1 日 0 点 0 分 0 秒为原点来记录时间。（计算机中的时间记录的是自 1970 年 1 月 1 日 0 点 0 分 0 秒开始，到现在经过的总秒数，再用这个秒数计算出年，月，日）后来，Brian Kernighan 觉得用汇编写的系统不好维护，于是……他发明了 C 语言（符合大牛一切自己动手的风格），然后用 C 语言又重写了一遍。从此，Unix 走上了发展的快车道，并且一直用到现在。许多世界级的大服务器，用的都是 Unix 系统（其实就现在而言，很多服务器上是用的开源的 Linux 系统）。**而这一切的努力，就是为了玩个游戏。-_-b**

总体而言，没有操作系统的计算机，就像没有意识的身体，是无法动弹的。

是不是觉得操作系统很酷？想不想自己写一套？编写操作系统要比通常想象的困难的多，它涉及到大量的背景知识和底层操作。所有连这本书都还不能消化的新手应该完全打消诸如自己制作操作系统的念头，否则吐血身亡我可负责不起，有实力之后再来尝试也不迟。

提供给钢琴和手风琴的乐谱并不是完全一样的，这是一种共识。为什么？——因为这是两种不同的乐器。同样，在这个世界上有很多种的计算机，他们相互之间的差别也很大。每一种计算机都需要操作系统，而某一款计算机可能有好几种操作系统可以使用，就像我们可以说中文，也可以说英语。但也有可能它只能装某几种操作系统，比如 Mac OS X 就不是普通的电脑能原生支持的。**特定的计算机和特定的操作系统的组合，就被称为平台。**就像钢琴曲不是给京韵大鼓使用的，对印度土著说中国成语也不会有什么好效果，编程通常都要针对某种平台来做。有一些高级语言号称能够跨平台——也就是可以在很多平台下运行。跨平台并不意味着在每一种平台上都能运行，一般能支持一些主流平台就可以称为跨平台。

就像我在前面谈到库的时候已经提到过，如果你想要在编程上面有所成就，你不得不选定一个平台深入地钻研。个人认为，初学者不适宜同时在两个完全性质不同的平台上学习，不同的特性绝对会把你搞晕，在你确认掌握了一个之后再学习另一个，相互借鉴的作用才能体现出来。不过我个人还是蛮希望大家一开始就能在 Linux 平台学习编程。

4.5 Console & GUI

当你开始照书上的例子编写 Hello world 时，很快就会问出这个问题。回想平时见到的那些程序，他们通常都有标题栏，有菜单和工具栏，可以用鼠标在上面点来点去好像还有很多别的功能。可是这个程序似乎完全不同，难道出了什么问题？其实没有问题，这是一个控制台程序。

世界上有各种各样的程序。几乎所有的程序都要和用户交流，接受用户的输入，输出运行的结果，但它们接受和输出的方式是不一样的，程序与用户交流的方式被称为界面。还记得早期科幻电影里那种出现在计算机屏幕上可以和人对话的巨大人脸吗？或者《贝克街的亡灵》中被称为“茧”的游戏——那就是界面的一种。事实上这种界面到现在还没有实现，人类的想象力总是能超前实际技术很多。界面有很多种，有些程序不需要界面，因为它们根本不必和用户交流，它们和其它的程序交流，你可以叫它无界面，典型的例子就是驱动程序，你什么时候见过驱动程序运行的样子？一般在 Windows 和 Mac OS X 下的程序所使用的则称为图形用户界面（GUI: Graphics User Interface），Linux 下同样也有很多采用 GUI 的应用程序。简单来说，就是所有的输入和输出都使用图形的方式。它接受用户图形化的输入，譬如用户用定位设备（鼠标、轨迹球、手写板）输入坐标、绘图，把程序的输出反映在可以显示图形的设备上，譬如显示器、打印机、头戴式监视器，通常这种程序会提供菜单、工具条等方式而极大地方便用户。这种程序直观明了，一般用户能很容易的掌握使用，只需要点击就可以完成大部分的任务。魔兽世界和 WORD 就是典型的 GUI 程序。

虽然 GUI 程序便于一般人使用，但其实图形界面的编程是相当复杂的。你能想象每次你都要画出显示器上所有的东西有多困难。窗口移动缩放时，你需要重画窗口里的每一样东西；在多窗口并存的时候，如果你的窗口被别人的窗口挡住了一部分（这是很常见的情况），你需要控制窗口上哪些部分被显示、哪些被遮挡；鼠标移动的时候，你需要把被鼠标遮住的部分盖住，并重画鼠标；当鼠标点击时，你需要判断鼠标的位置，还要判断这个点击是你的程序的，还是别人的；当鼠标点击到菜单的时候，你要确定是菜单还是按钮，以及是哪个菜单项，然后执行相应的操作。所有这些情况，都必须一一解决。其中有些是可以由操作系统解决的，但你要知

道怎样利用操作系统提供的资源才能让它帮你做，还有些就必须你自己想办法，种种问题使得图形界面的编程变得异常复杂。

与此同时，由于图形处理的需要，图形界面对计算机硬件的要求也比较高。画面绚丽的 3D 游戏，远比记事本程序对系统的要求要高的多。虽然现代的计算机早已能够满足这些要求，但早期的计算机并不是这样强大的。所以，在早期（也不是很远，大约是上世纪七、八十年代）的时候，程序的界面并不是图形，而是字符的。用户在键盘上输入文字，比如 dir，系统找到相对应的命令，然后执行，执行的结果也是以一行行文字的形式输出在可以输出文字的设备上（当然也主要是显示器），用户阅读文字，进行下一步的操作。现代的一些科幻片，在表现黑客侵入或者是操作高级设备（比如美国国防部的核武器系统）时，往往出现操作员在啪啪啪啪快速打字，然后突然一回车，就大功告成的场景，很少会出现卡通化的菜单和工具栏。想想黑客帝国的那个接线员，你见过他什么时候抓着鼠标吗？这种方式就叫命令行界面（CLI: Command Line Interface），由于操作员通常是坐在一个操作台前，而这个操作台确实可以控制整个系统，所以也被称为控制台界面（Console Interface）。控制台下运行的程序，就是控制台程序；运行控制台程序的系统，就叫控制台环境。控制台看起来很高级，很酷很眩，黑客们操作的时候也显得很高深，其实反倒是比较容易编程实现的。因为你只要处理和输出字符就可以，系统自然会把字符放在屏幕上适当的位置。你不用去管字体大小颜色这些事，更甭提窗口&菜单&鼠标这些不存在的东西，只要把注意力全部放在程序的功能上。电脑系统也不用消耗资源来画图，系统自然比较高效。所以，命令行界面一出现，就得到了广泛的应用，它的历史可比图形界面古老多了。和 GUI 程序比起来，命令行的程序通常很难看，操作也不直观。**但由于它的高效和快捷**，命令行方式的程序迄今仍然在使用，命令行方式不但没消失，相反，有许多系统反而更加加强这部分的功能，譬如 Windows7 下的 Windows PowerShell[15]，按下“Windows”徽标键（键盘左下侧一个窗户的图案），输入 PowerShell 试试，进入后试试 ls & tab，可以像 Linux 的终端一样自动补全噢~ 在很多时候，人们宁可使用命令行的程序来完成某些工作，这一点在 BSD、 Linux 和其它的 UNIX 系统中体现较为明显[16]。DOS 就是一个标准的控制台环境，Windows 系列操作系统，也提供了控制台环境。而很多的 GUI 程序，借鉴控制台方式，仍然保留有直接使用键盘操作的方法。比如魔兽争霸，几乎每个命令都有快捷键以加速操作。

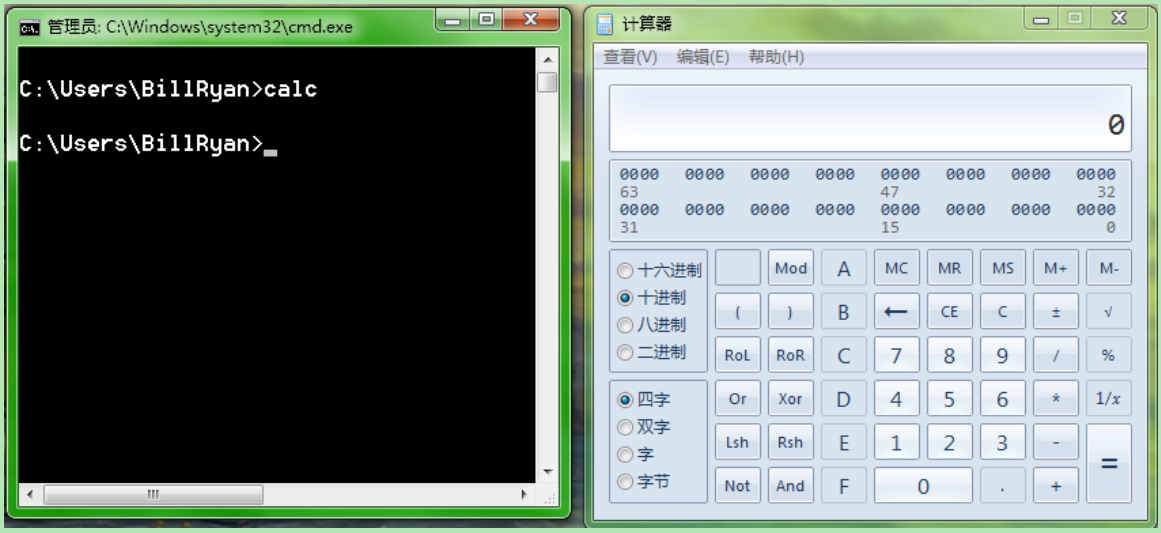
实际上，很多系统管理员更偏爱控制台程序，除了它比图形界面程序更高效之外还有一个原因：同样要增加实现一个选项，控制台只要增加一个输入字符作为开关，而图形界面至少需要增加一个按钮。想想看，屏幕上是可以增加的按钮多还是可以增加的字符数多呢？**所以控制台程序往往更能实现纷繁复杂的功能，只要你记得住相应指令。**

让我们在 Windows 下打开一个控制台环境并使用几个控制台程序感受一下。Linux 下的后边附录部分会单独截图演示。假如你使用的是 Windows XP，点击开始，找到程序->附件，在里面有一个叫“命令提示符”的快捷方式，点击运行它。Windows7 下类似，也可以点击开始后输入”cmd”. 还可以同时按住”Windows”徽标键和 R，在弹出的窗口中输入”cmd”.

你是不是看到了一个窗口，没有菜单，没有工具栏，只有标题栏和最大最小化的按钮，这就

[15] Windows PowerShell(TM) 是一种命令行界面和脚本语言，专门为系统管理而设计。
[16] 其实现在大部分的 Linux 对新手已经非常友好，可以在 GUI 下完成绝大部分操作，而且效果往往更好

是 Windows 下的控制台环境，在这里就可以运行控制台程序。现在，输入"dir"，然后回车。你看到了输出吧，这就是控制台程序的运行了。只要输入命令再回车，就可以运行了。再来运行一个，time，再来一个，ipconfig。最后再来一个，help。他给出了当前系统提供的各种命令，你可以看着提示分别试一试。其实不止可以运行这些，试试 notepad，你发现了什么？explorer, calc 等等都是可以这样运行的哦。玩够了，好吧，输入 exit，再回车，控制台环境就被关闭了。



控制台程序容易编写，易于理解，所以对于初学者，控制台程序是比较好的选择。在学会了控制台程序的基础上，再转到窗口程序就比较轻松了。

4.6 Project & a single file

有一件很明显的事情，当软件项目变得很大的时候，仅仅使用一个文件来完成所有的内容是不现实的。以《魔兽世界》这款游戏为例，游戏中需要声音、动画、图片这样的素材，也需要地图编辑、人工智能、光影渲染这些不同的模块，在制作的时候，需要有不同的部门完成不同的工作，工作中所使用的文件和工具都不相同，不可能生成同一个文件。这时，就存在一个协调的问题。因此，对于大型的开发工具来说，它们不是以单个文件为单位进行处理的，而是以项目为单位。一个开发项目包含一个到几个工程，每个工程都包含有大批的文件，有源代码，有程序所使用的图片音乐等资源，还有编译时需要记录的各种参数。每次编译都要完全编译整个工程（当然在优化状态可以只考虑更新变动的部分）。

显然工程对于类似《魔兽世界》这样的大型项目的编写是十分必要的，但在新手练习的小程序上，使用工程就像你只想在家给你的小狗搭个窝，却拉来了整个中国长江三峡工程开发总公司，实在有点小题大做。而且，以后我们要编写很多这样的小程序，如果每一个都新建工程，那会生成多少垃圾文件啊！所以，我们需要一种只编译单独文件的方法。幸运的是，不论哪种开发工具，它们都提供了这样的方法。虽然这种方法通常不会在菜单上直接出现，但它确实是常用和正确的方法。在 Linux 下这一切都非常方便，比起 Windows 来说个人感觉要方便多了。[17]

[17] 附录部分详细说明

4.7 Function, API, Class, Control, SDK & Software Reuse

还有件事也很明显，随着工程项目的越来越浩大，每一次都从零开始编写一个项目是不经济也是不现实的。我们注意到，不论是程序还是源代码，都有以下两个特性。第一，它们都是人类智慧的成果，每一行源代码都凝聚了程序员的聪明才智，花费了开发者的时间和金钱。第二：几乎每一行源代码本身都可以毫不费力的零成本的被复制到另一处，另一个代码块、另一个程序甚至另一个项目。基于这样的特性我们认识到：**如果能够重复利用已经编写过的程序和源代码，尤其是那些在使用中已经被证明健壮而正确高效的程序，就能够节约巨大的人力物力财力&时间。**即使只能使用其中的一小部分，也是了不起的成就，这就叫做软件复用。

软件复用最重要的好处就是能够让程序员不再把有限的智慧放到早已解决过的问题，而是投身新的问题，也就是：**不要再重复发明轮子。开放源代码的一个重要理由就是为了软件复用。**软件复用的一个行之有效的方法是使用函数。所谓函数，就是把一些具有固定功能的代码段组合在一起，并给予一个名字。在需要这些功能的时候，只要在适当的地方填入函数的名字。在编程时使用函数而不是纯粹手工打造，就像在盖楼时使用砖块而不是用粘土堆砌——已经是不小的进步了。盖楼时使用的砖块，既可以自己制造，也可以到市场上去购买。购买来的砖块，虽然有某些差异，但总是遵循某些共同特性。这些砖块虽然是一块块地使用，但总是一车车的大量购买。函数也一样，市场上总是有专业公司提供函数出售，他们出售的函数按照功能聚集在一起，成为函数库。虽然不同公司的函数库可能具有相同的名字和功能，但内部是怎样构成的却不尽相同。大部分现代编程语言，在规定了语言本身的语法和词汇（当然其中也包括了函数的使用方法）的同时，还会规定一个函数库。这个函数库只规定了每个函数的名称和用途。至于函数的具体实现方案，有的是由语言本身直接定义，有的是由实现该语言的编译器的厂商提供。而使用这种语言的用户，只要在需要的时候使用，而不用去管函数库内部的问题，大大提高了效率。这个库就叫做标准函数库。通常，现成的函数库都是经过了大量的实践检验证明是高效而健壮的，要比自己徒手打造的函数好用的多。**所以在可能的情况下，要充分使用已有的函数库，尤其是标准函数库。**C 语言之所以如此强大，原因之一就是它有一个强大完备的标准函数库。而 C 语言之所以如此难学，原因之一也是在于它有一个强大完备的标准函数库。

软件通常是运行在操作系统下的，从软件复用的角度看，操作系统如果能提供某些通用的服务，程序就可以集中注意力做自己的事。这些服务包括文件读写、设备操作、网络通讯、窗口绘制等等。否则，程序就会把大把的精力浪费在这些基础工作上。幸运的是，几乎所有的现代操作系统都提供了这样的服务。这些服务以函数的形式出现，程序使用这些服务，就像使用函数一样。这些函数形式的系统服务，就叫做应用程序接口 API（Application Programing Interface）。遗憾的是，不同的操作系统，提供的 API 通常是不同的。使用某个操作系统的 API 编写的程序，搬动到另一操作系统时，由于 API 不能相互对应，也就不能运行了。这就是编程要针对平台的原因之一。

有些大型程序，比如 Autodesk 公司的 AutoCAD，当你对它的某些功能不满或是感到有改进的必要时，它提供了编程改变的可能，相对于第一次开发来说这就叫二次开发。二次开发时，原始程序所提供的那些服务也被称为 API。

使用砖块盖楼确实很方便，但如果能使用预制板，那就更方便了！比函数更高一级的可以复用的程序模块叫做类，类的使用比函数复杂，但是它可以更广泛的复用，是更高级的软件复用形式。类，同样可以聚集成类库。能够使用类的语言，就会规定标准类库。C++比 C 更强大，更难学，原因之一就是因为，C++除了具有 C 所有的函数库之外，还有一个完整强大的标准类库。在类库当中，有一种很特殊的类，称为控件（control）。控件在快速的窗口程序开发中特别有用，它可以实现拖放式的编程。举例来说，想要编写一个有一个按钮的窗口程序，那么只要先创建一个窗口程序的工程，这个工程会自带一个窗口控件，而且直接显示在工作区里，然后再用鼠标把按钮控件从控件板上拖到窗口里需要的位置，按钮就摆放好了。至于按钮对应的功能，就需要程序员来编程实现。将来编译运行之后，程序的外观就和设计的外观一模一样。控件极大的提高了编程效率，但因为它需要自动生成某些代码，所以需要编辑环境的支持，支持控件的编辑环境总体来说比较少。控件的总和，就是控件库。

所有这些，函数、API、类乃至控件，开发必备，就像旅行时必需携带的行李一样，于是被统称为开发包（SDK : Software Development Kit）。

有些开发包是编译器自带的，如标准库等等，有些包是第三方厂商提供的。大部分开发包都需要集成开发环境（内带编译器）的支持，有的要编译器提供内部实现，有的要编译器给予连接，有的要编译器生成代码。不同的编译器提供的方式不太相同，这就造成了编译器之间的差异，有时甚至导致某个开发包不能在某款编译器上使用。为了进一步提高程序员的工作效率，很多集成开发环境还发明了另一些方法，相当于直接用一间间的房子叠在一起做成居民楼，你几乎只要刷刷外墙漆，就可以完成了，这被称为应用程序框架。可惜，不同的厂商的方法根本不同，这进一步扩大了集成开发环境之间的差异。不管未来怎样，至少现在，编程的规模变得越来越大，也越来越需要更多人的智慧。每个人的智慧都是有限的，不应该被浪费，充分使用每个人的智慧，才能取得最大的成功。软件复用就是整合所有人智慧的方式之一。从你编程的第一天开始，**请牢牢记住：软件复用。**要复用&复用&&再复用。

4.8 GUI Programming

黑客们很偏爱 Console，但对于普通用户来说，图形界面是他们更熟悉的。我们大多数人也喜欢编写 GUI 程序，因为那看起来似乎更友好也似乎更有成就感，那怎样才能编写 GUI 程序？

每一个操作系统，都会提供一套 API，如果该系统支持 GUI，那么它的 API 中就会有一个图形子系统和窗口管理子系统。其中图形子系统包括了基本图形元素的绘制，比如画点线面、显示文字图片和上色渲染等等，窗口管理子系统首先会包含图形子系统，再增加包括窗口和窗口元件的绘制、窗口的遮盖、移动调整，以及鼠标点击这类事件的传递处理等等这些内容。Windows 下，图形子系统就叫 GDI[18](Graphics Device Interface)。为了高效处理多媒体编程，微软在 Windows95 发布前夕开发出了 DirectX[19]——除了图形之外它还整合了很多其它功能，用它搞出了大量的 Windows 游戏。另一套著名的图形 API 则是跨平台&跨编程语言的 OpenGL[20]，渲染能力很强!!

[18] GDI+从 Windows XP 操作系统开始引入，提供二维的矢量图形，改进旧有的 GDI，加强的可视化属性_

[19] 目前最新版本为 DirectX 11，非 Windows 平台使用很困难

[20] 核心 API 没有窗口系统、音频、打印、键盘 / 鼠标或其他输入设备的概念。但可扩展性强，可通过其它库来弥补

另外值得一提的就是微软自 Vista 发布以来广泛用于界面开发的 WPF[21]，WPF 是下一代图形 API 在桌面上的延伸。以 WPF 撰写应用程序，具有更高的视觉品质，Win7 丰富的界面效果很大程度上要归功于 WPF 的应用。

即使有图形库窗口管理库，GUI 编程仍然是相当的繁琐。首先是要完成某个工作必须填写大量按部就班的代码，而且这些代码在程序内部和各个程序之间重复着，这提示人们应该在图形库的基础上作进一步的抽象。简单来说就是要在画点画线函数的基础上做出画二次曲线画立方体的模块。幸运的是，这些工作已经有人完成了，而且是不止一套。这些东西按照功能强弱，有的叫图形用户界面库，有的则叫应用程序框架。在 Windows 下，使用 VC++的人可以选择 MFC ATL，如果你选择.NET 阵营，.NET Framework 就是你最好的选择，结合 C#开发比较方便。

如果你希望程序不止在 Windows 使用，那么跨平台的 Qt & GTK+ & wxWidgets 都是你的好选择。它们在保证效率的同时，适当抽象，抚平了底层操作系统的差异。对于学习 C++的人来说，上述库都是可以用的，当然也有其它语言支持——比如 Python。这三个平台相对来说 QT 的资料会多一些，但是有一定的版权问题干扰，不过个人使用没有问题。Linux 下常见的 KDE 桌面环境下的软件就是基于 Qt 框架开发的，GTK+的广泛使用有一定的必然性，让我们来看一段历史：

> 1996 年 KDE 专案启动。KDE 是一个自由的桌面环境，但 KDE 依赖的 Qt 当时并未使用 GPL 授权。出于这种考虑，两个项目在 1997 年 8 月发起：一个是作为 Qt 库替代品的“Harmony”，另外一个就是创建一个基于非 Qt 库的桌面系统，即 GNOME 项目。GNOME 的发起者为米格尔·德伊卡萨和 Federico Mena.
>
> GIMP Toolkit(GTK+)被选中作为 Qt toolkit 的替代，担当 GNOME 桌面的基础。GTK+使用 LGPL，允许链接到此库的软件（例如 GNOME 的应用程序）使用任意的许可协议。GNOME 计划的应用程序通常使用 GPL 许可证。
>
> 在 GNOME 变得实用和普及之后，1998 年 Qt 加入 GPL 授权。Troll Tech 在 GNU GPL 和 QPL 双重许可证下发布了 UNIX 版的 Qt 库。Qt 加入 GPL 授权后，在 2000 年年底 Harmony 项目停止了开发，而 KDE 不再依赖非 GPL 的软件。2009 年 3 月，Qt 4.5 发布，加入了 LGPL 授权作为第三选择。
>
> **GNOME 桌面系统使用 C 语言编程，**但也存在一些其他语言的绑定使得能够使用其他语言编写 GNOME 应用程序，例如 C++，Java，Ruby，C#，Python，Perl 等等。

从以上部分我们可以看出，唯一支持用 C 语言编写 GUI 程序的主流图形库只有 GTK+[22]，但大部分都可以利用 C++和 Python 进行开发。

[21] 其实在 Linux 下的界面早已能做的比 WIN7 更好

[22] 也有一些其它支持 C 语言的图形库，*C How To Program* 一书中就有关于 Allegro 的介绍，Wikipedia 有详细介绍

Chapter5. Programming languages

这一章其实是为下边的第六章做准备的，如果读到这儿还不知道编程语言是啥的，请回头阅读“1.5 What is programming language？”下边的介绍仅仅只是蜻蜓点水式的，想详细了解的请自行 Wikipedia，存在即为合理，每种语言都有其优缺点和适用领域，不可一概而论

5.0 C

在“4.4 After programming languages - The OS & platform”中已经初步提及了 C 语言的产生历史。C 语言是一种通用的、过程式的编程语言，广泛用于系统与应用软件的开发。具有高效、灵活、功能丰富、表达力强和较高的移植性等特点，正是由于这些特点，在嵌入式领域也得到了最为广泛的应用。

C 语言是由 UNIX 的研制者 Dennis Ritchie 于 1970 年由 Ken Thompson 所研制出的 B 语言的基础上发展和完善起来的。目前，C 语言编译器普遍存在于各种不同的操作系统中，例如 UNIX、MS-DOS、Microsoft Windows 及 Linux 等。C 语言的设计影响了许多后来的编程语言，例如 C++、Objective-C、Java、C#等。

在发展的过程中，出现了许多略有差别的 C 语言版本。1989 年，美国国家标准研究所(ANSI)为 C 语言制定了一套 ANSI 标准，就是所谓的 C89。经过 10 年的发展，在 C89 的基础上又颁布了 C99 标准。大约又过了十年，2011 年 12 月，ISO 正式公布 C 语言新的国际标准草案：ISO/IEC 9899:2011。目前大多数编译器完全支持 C99，也可以指定为 C89 进行编译。

5.1 C++

C++和 C 的关系非比寻常。1979 年，当 Bjarne Stroustrup 在新泽西州的 Murray Hill 实验室工作时，发明了 C++。Stroustrup 最初把这种新语言称为“带类的 C”，1983 年，改名为 C++。C++通过增加面向对象的特性扩充了 C。因为 C++产生在 C 的基础之上，因此它包括了 C 所有的特征、属性和优点。在这个意义上，你可以认为 C++是 C 语言的进化，这个进化的原因是复杂性（complexity）。一旦一个程序的代码超过 25 000～100 000 行，就很难从总体上把握它的复杂性了。C++突破了这个限制，帮助程序员理解并且管理更大的程序。

当然今天的 C++已经不只是 C 的进化那么简单，它还包含了其它很多很多的新内容，C++对于 C 的进化着重体现在规模上，几乎 C++的所有新特性，都是为大规模编程服务的。如果说 C 语言编程是单兵小组作战的话，C++编程就是集团军运动。它是如此的复杂以至于你要花上几倍于学习 C 的时间来学习他。但由于它和 C 的历史渊源，所有支持 C++的编译器都能支持 C，因此他俩也就常被合称为 C/C++语言(大部分的 C 代码可以很轻易的在 C++中正确编译，但仍有少数差异，导致某些有效的 C 代码在 C++中失效，或者在 C++中有不同的行为。)。

标准化历经 C++98->C++03->C++TR1->C++11[23]

[23] http://en.wikipedia.org/wiki/C%2B%2B

5.2 Relationships among C/C++ & Other Programming languages

Java 只要用一句话就可以概括，它是一种简化了的跨平台的 C++语言（因此也被称为 C++--）。掌握了 C++的人，学习 Java 几乎是轻而易举。Java 舍弃了 C++语言中容易引起错误的指针，改以引用取代，同时移除原 C++与原来运算符重载，也移除多重继承特性，改用接口取代，增加垃圾回收器功能。不同于一般的编译语言和解释语言，Java 首先将源代码编译成字节码（bytecode），然后依赖各种不同平台上的虚拟机来解释执行字节码，从而实现了“一次编译、到处执行”的跨平台特性。在早期 JVM 中，这在一定程度上降低了 Java 程序的运行效率。但在 J2SE1.4.2 发布后，Java 的运行速度有了大幅提升（O(∩_∩)O~这当然也是相对的啦）。

Pascal, Perl, Python, PHP, C# 还有其他诸如此类的语言，或者与 C 处于同一档次，或者与 C++处于一个水平，学过 C/C++之后，再去学习他们是很容易的事。其实在 C 之后很多语言都可以称之为 C 系语言。

LISP, Scheme 是和 C 的思维方式完全不同的一类，被称为函数式编程语言，在人工智能等领域有奇妙的应用，有兴趣的朋友可以去领略。

5.3 Why need learn C/C++

考试的人不用讨论这个问题，他们没有选择。如果我们有选择，为什么我们学习 C 语言而不是别的。首先因为 C 本身是非常优秀的，它是世界上最伟大的编程语言之一。许多人认为 C 语言的产生标志着现代计算机语言时代的开始，它成功地综合处理了长期困扰早期语言的矛盾属性。C 语言是功能强大、高效的结构化语言，简单易学，而且它还包括一个无形的方面——它是程序员自己的语言。它的设计、实现、开发由真正的从事编程工作的程序员来完成，反映了现实编程工作的方法。它的特性经由实际运用该语言的人们不断去提炼、测试、思考、再思考，使得 C 语言成为程序员们喜欢使用的语言。

在编程规模越来越大的今天，用 C 往往会遭遇协作型的问题，而 C++在保持了 C 的高效的同时，实现了大规模协作的可能，因而成为了真正工业化的语言。C/C++的优秀，使得它是主流的，在 20 世纪 70 年代末和 80 年代初，C 成为了主流的计算机编程语言，至今仍被广泛使用。今天几乎所有的操作系统、大部分的的应用软件，90%以上的大型游戏都是用 C/C++编写的。在对运行速度和资源占用有严格要求的领域，比如游戏、即时控制、嵌入式系统，基本都是 C 语言内嵌汇编语言的天下。今天只有一种语言的性能比 C 强，那就是汇编，优化过的 C 程序的速度大约是汇编的 95%-98%。但汇编基本不是常人用的，所以实际上 C 就是最快的语言。

主流的就意味着资料丰富。不论是编程时所需要的文档，还是学习的示例代码，甚至是平台提供的接口和库，C/C++语言版本都是最丰富的。任何一家硬件软件公司，当他开发一款能够编程的设备或是软件，必然会提供 C/C++语言的接口函数，其他的语言，就不一定有这样的好处了。不论是计算机图形学、加密解密还是计算机编程的其他领域，C/C++简直就是编程界的普通话。根据 International Data Corporation 的统计，C/C++ 是全球开发者使用最多的编程语言。如果确实要投身编程界，不懂 C/C++的话，根本就是 Mission Impossible。

如果你还不明白，我只需举一个小例子。假设你现在有机会去学习一门外语，投入时间差不多，你会选择学习英语呢，还是斯瓦希里语呢？(斯瓦希里语流行于非洲东部，为肯尼亚坦桑尼亚等国的官方语种)。当然，并不是说其他语言不值得学习，存在即是合理，今天存在的各种语言，当然都有它存在的价值，多学一点没有坏处。没有人禁止你在学会英语的基础上再学习法语日语等等。况且为了找工作等现实原因，学习其他语言也是非常正常和必要的。无论过去、现在还是将来手，天底下不存在哪一门语言非学不可才能成为高手，思想最重要，“不会 XXX 语言不算真正的高手”之类的言论实在无聊之至。学 C/C++，主要还是通过他学习编程思想，**真正的武林高手，难道会局限于手里的那把宝刀吗？**

5.4 Python

虽然如上所述，C/C++语言有如此之多的好处，但它也不是完美无缺的，第一个致命弱点是复杂性。为了应付千奇百怪的需求，C++提供了很多奇妙的语法从而实现了各种现代编程特性，这就使得 C++变得异常复杂。如果有一个程序员对你夸口说他已经彻底掌握了 C++的话，你基本可以断定他接触 C++还没超过半年。幸好 C 语言还没有复杂性的困扰。但是开发效率也是C/C++的致命伤，因为过于接近硬件底层，C/C++程序在运行时几乎可以调动一切资源，取得最高的性能。但与此同时，C/C++程序员必须小心维护程序的运行状态，稍有不慎，轻则文件丢失&内存泄露，重则死机甚至导致整个系统软硬件崩溃也是有可能的（大多数人 C 语言上机时恐怕或多或少都会遇到野指针的问题）。所谓高收益必与高风险相伴就是这个道理。这就使得开发时必须小心谨慎，开发效率也就提不上去。

对于性能要求极高的程序，比如要求大量高速 3D 计算的计算机游戏、密集科学计算的工程软件、同时应付巨量用户的网页和数据库服务器，使用 C++是当仁不让的选择。但是对于很多应用来说，些许的延迟是可以忍受的。举例来说，在某个工程中想要得到圆周率的后十万位，如果此时我们不在乎 0.25 秒和 1.25 秒运算时间之间的差别的话，就可以选择开发效率极高，而计算速度相对较慢的其他语言来实现。在所有这些其他语言中，强烈推荐的是 Python 语言。Life is so short, you need Python. By lvzongting :）

由创始人 Guido van Rossum 在 1989 年圣诞节期间创造出来的 Python 语言，是一种面向对象、直译式计算机程序设计语言，也是一种功能强大的通用型语言，已经具有近二十年的发展历史，成熟且稳定。Python 具有脚本语言中最丰富和强大的类库，足以支持绝大多数日常应用。这种语言具有非常简捷而清晰的语法特点，适合完成各种高层任务，几乎可以在所有的操作系统中运行。Python 支持命令式编程、面向对象程序设计、函数式编程、面向切面编程、泛型编程多种编程范式。与 Scheme、Ruby、Perl 等动态语言一样，Python 具备垃圾回收功能，能够自动管理内存使用。它经常被当作脚本语言用于处理系统管理任务和 Web 编程，然而它也非常适合完成各种高阶任务。

Python 语言有两大关键特点。首先它被称为是一门清晰的语言。因为它的作者在设计它的时候，总的指导思想是，对于一个特定的问题，只要有一种最好的方法来解决就好了。这在由 Tim Peters 写的 Python 格言（称为 The Zen of Python）里面表述为：There should be one—and preferably only one obvious way to do it。这正好和 Perl 语言（另一种功能类似的高级动态语言）

的中心思想 TMTOWTDI（There's More Than One Way To Do It）完全相反。Python 语言是一种清晰的语言的另一个意思是，它的作者有意的设计限制性很强的语法，使得不好的编程习惯（例如 if 语句的下一行不向右缩进）都不能通过编译。这样有意的强制程序员养成良好的编程习惯。简单来说，几乎所有的 Python 程序看起来都是一个样子的，而相对的，人们说，到哪里去找一段随机字符，读一段 Perl 程序就行。

第二个特点是它被称为胶水语言。哦，这可不是说它会把你的手指给粘住。它的意思是它可以像胶水一样，把用其他语言制作的模块拼合起来。很多人是这样用 Python 语言制作大型程序的：他们先用 Python 做出一个可用的程序，再把其中最影响性能的部分用 C/C++语言重新写成模块（在已经满足要求的时候，根本就无需替换）。不用担心配合的问题，Python 的可扩充性完全可以胜任要求，可扩充性可以说是 Python 作为一种编程语言的特色。新的内置模块（module）可以用 C/C++或其他语言写成，同时也可为现成的 C/C++或其他语言模块加上 Python 的接口。这样一来，我们既能充分享受到 Python 快速开发带来的效率，又能够实现足够强大的性能。

再加上设计极为出色的三大内置数据类型：异构列表、元组、字典和从函数式语言学来的列表解析、对象自省、方法动态生成等突出特性，使得它的功能变得极为强大的同时程序又相当简洁。想想看，你能在 C 程序运行的同时，改变自身的代码让它再次以另一面貌运行么？Python 就可以轻松地做到。同样的功能，用 Python 实现所需的语句可能仅仅是 C 的十分之一到二十分之一，而开发（包括调试）所需要的时间更可能只需要百分之一。

Python 语言是少有的一种可以称得上既简单又功能强大的编程语言。对于工程师会计师这类非职业程序员，在平时需要某种语言来编写程序解决一些小问题的时候（例如做个土方计算，分析某种投资的收益，批处理一些文件的改名压缩之类），Python 是比 C 语言更好的选择。即使对职业程序员，Python 也是称手的工具。总之，Python 就是手边的语言。

Python 的应用很广，既可以当做脚本语言来使，又可用于 Web 开发，还可用于科学计算——《Python 科学计算》[24]，可以用轻便的 Python 来替代昂贵的商业软件 MATlab 噢~[25]

在 MIT Fall 2008 的 Introduction to Computer Science and Programming [26]课程中有不少内容就是用 Python 实现的，MIT 开放式课程官网有详细介绍，视频啥的也可以到 VeryCD 上下载。

[24] http://book.douban.com/subject/7175280/ 豆瓣主页
[25] http://www.zeuux.org/group/scipython/ 相关问题及讨论
[26] http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/

Chapter6. Textbooks & Other Cases

推荐好书当然是本书的重要任务之一，这次集中推荐一下。主要是 C/C++, Python, Java,数据结构算法等教材。Linux & Widows 编程以及图形库等读者可参考 BlueAuris 的原帖[27]，或者等待本书 Wiki 页面更新。

这些主题中任意一个的教材都是汗牛充栋，读一辈子都读不完。不过很显然，大家都公认的经典，也只是那么寥寥数本而已，下面推荐的都是这样的著作（其实只能说是适合大部分人）。当然，我推荐的书并不一定适合你，如果有更好的可以在开源社区 BBS 中跟帖，其它本书未涉及的都将在 Wiki 上 update。本书所推荐的教材不一定都要去读的，可以根据你自己的兴趣和精力去选择。以下排名不分先后，很大程度上考虑了排版因素_

6.0 Computer Science

- **NO.1** *Computer Science: An Overview* 10th edition
  中文书名：计算机科学概论 Author: Nell Dale, John Lewis
  计算机科学概论课程的经典入门教材，英文版已经到第 11 版了_
- **NO.2** *Computer Systems: A Programmer's Perspective* 2nd edition
  中文书名：深入理解计算机系统 Author: Randal E. Bryant, David R. O'Hallaron
  Amazon 五星图书，最伟大的计算机科学教材之一_
  看过此书你会对计算机原理、汇编和 C 语言有根本性的认识。同时也是一本非常好的入门教材，CMU 的计算机学科类导论教材，作者讲解很细致_

6.1 Data Structures and Algorithm

- **NO.1** *The Art of Computer Programming*
  中文书名：计算机程序设计艺术 Author: Donald E. Knuth
  4.1 节已有提及，这里详细介绍，简称 TAOCP，号称是经典中的经典，每一个想要掌握算法精髓的人都必须看的书，算法书中的圣经_
  1938 年初，高德纳出生于美国威斯康辛州。毕业于加州理工学院的他，目前是美国著名的计算机科学家，并且是斯坦福大学计算机系荣誉退休教授。高德纳的英文全名为：Donald Ervin Knuth。他的中文名是 1977 年到中国来之前，姚储枫为他取的。Knuth 从此用高德纳作为其在 Unicode 世界的名字。_
  TAOCP 这套关于算法分析的多卷论著已经长期被公认为经典计算机科学的定义性描述。迄今已出版的完整的三卷(第四卷已有部分出版)已经组成了程序设计理论和实践的惟一的珍贵资源，无数读者都赞扬 Knuth 的著作对个人的深远影响，科学家们为他的分析的美丽和优雅所惊叹，而从事实践的程序员已经成功地将他的“菜谱式”的解应用到日常问题上，所有人都由于 Knuth 在书中表现出的博学、清晰、精确和高度幽默而对他无比敬仰_
  他因这些成就以及大量富于创造力和具有深远影响的著作（19 部书，160 篇论文）而誉满全球。这么说吧，目前你所能听说过或者还活着的那些计算机软件大牛，没有哪个不直接或间接的受过 Knuth 的教导。他强悍的程度只用一件小事就可以说明，在撰写这套《计算机程序设计艺术》的过程中，由于感到原有排版系统的不足，他特地重新制作了一套新的计算机排

[27] http://www.vcgood.com/BBS/forum_posts.asp?TID=1559

版系统，这套称为 TEX 的东西目前已经是出版界的标准系统_

1968 年，刚刚进入 Stanford 的高德纳开始准备出版经典巨著《计算机程序设计艺术》，据说当时他一口气写了 3000 页，自此他计划写七卷（目前已经完成四卷）。这七卷分别为：基础算法、半数值算法、排序与查找、组合算法、造句算法、与上下文无关语言理论、编译器技术。1999 年底，该书被美国科学家期刊列为 20 世纪最佳 12 部学术专著之一_

建议到豆瓣和 Amazon 上仔细看看评论或者看看印刷版，这一套书虽然很多人都极力推崇，但并不见得他们都看过，个人觉得中文翻译的看起来有点别扭。纸上得来终觉浅，绝知此事要躬行_

- **NO.2** *Introduction to Algorithms*

  中文书名：算法导论 Author: Cormen, Leiserson, Rivest and Clifford Stein

  麻省理工学院计算机系的算法教材，已经成为世界范围内广泛使用的大学教材和专业人员的标准参考书_

  第 3 版已于今年年初出版，从第二版(英文原版)的内容来看，作者讲解非常详细透彻，中文翻译的我没读过，不好评价，盗版貌似挺严重_

  It includes careful mathematical treatment of the algorithms that it discusses, and would be a natural candidate for a reference shelf. Despite its bulk and precision this book is written in a fairly friendly and non-daunting style, and so against all expectations raised by its length it is my first choice suggestion_[28]

以上两本教材之前最好是能有一定的数据结构和离散数学的基础，当然咯，没有的话影响也不太大，只要你愿意去思考问题。下面推荐一些轻量级的书

- **NO.3** *Data Structures and Algorithm Analysis in C* 2nd edition

  中文书名：数据结构与算法分析-C 语言描述

- **NO.4** *Data Structures and Algorithm Analysis in C++* 3rd edition

  中文书名：数据结构与算法分析-C++语言描述

  Author: Mark Allen Weiss

  以上两本书都是 Weiss 的作品，算得上是数据结构和算法分析方面的经典教材了，国内严蔚敏的《数据结构-C 语言版》有不少地方参考了他的_

  个人觉得学习数据结构和算法时使用 C 和 C++不是太好，尤其是 C，需要考虑非数据结构和算法的地方太多了，似乎有点偏离主线_

- **NO.5** Algorithms 4th

  中文书名：算法 I～IV（C++实现）——基础、数据结构、排序和搜索

  Author: Robert Sedgewick, Kevin Wayne

  算法分析及实现方面较全面的一本书，比算法导论稍微薄一些，作者是普林斯顿大学的老师_

6.2 C

初阶读物：

- **NO.1** *C How To Program* 5th edition

  中文书名：C 大学教程 Author: H.M. Deitel, P.J. Deitel

  Deitel 父子所著的 C 语言的经典入门教材，他们从事语言教材的编写超过 40 年，谭浩强貌似

[28] Martin Richards “Data Structures and Algorithms” Lecture notes in Computer Laboratory University of Cambridge (October 11, 2001)

也写了不少书，但比起 Deitel 父子来说个人感觉编书水平要差不少。英文版已经到第 6 版了_
这本书的第五版包含了 C & C++两大块，前边介绍 C89，后边有一章节专门介绍 C99，再后边的一大块就是 C++了_

- **NO.2** *C Primer Plus* 5th edition
  中文书名：无 Author: Stephen Prata
  C 语言的百科全书，既有深度又有广度，如果觉得上边那本书过于简单可以看看这本（其实这本书写的也挺容易懂的）_
  第 6 版已经在国外出版，包含了最新的 C11 标准的介绍_
  本书的中文翻译版翻译质量不太好，大陆地区无英文原版发售_

- **NO.3** *C Programming: A Modern Approach* 2nd edition
  中文书名：C 语言程序设计·现代方法 Author: K. N. King
  书如其名——现代方法，讲解了一些适应现代大规模编程的方法。这本书的第一版有超过 225 所学校用过，比如 MIT, Stanford, UC Berkeley, Caltech 等等牛校_

- **NO.4** Linux C 编程一站式学习
  作者：宋劲杉
  终于看到有推荐国人写的书了，是不是很兴奋？_
  此书内容涵盖极广：C 的基本语法、简单的数据结构与算法、C 与汇编的联系、计算机系统结构、操作系统、正则表达式、TCP/IP 以及 Linux 系统编程，无所不包。 如此一来似乎样样通而样样不精，其实不是这么回事。作者将内容穿插得非常好，用十分简单的方式把每个方面最重要的东西都阐明清楚了。 所以，**其实这是本入门书**，当然也适合各个方面都了解之后总结用_
  今年 3 月该作者又出版了《一站式学习 C 编程》(升级版)，弱化了 Linux 和嵌入式的方向性，而且也变的更适合零基础读者阅读，在“它和前一版有什么不同”中作者有这么一段话：“虽然我在上一版中信誓旦旦地说这是一本面向完全零基础读者的书，但现实教育了我，要写出一本让任何零基础读者都看得懂的书是一门复杂的系统工程，我只能努力接近这个目标，而永远达不到这个目标。”很幸运的是这本编程导论小册子弥补了这一点 O(∩_∩)O 哈哈~_
  作者为本书第一版添加了 GFDL 许可证，网络上也可免费阅读，而且比纸质出版的内容要多
  http://learn.akae.cn

- **NO.5** *The C Programming Language* 2nd edition
  中文书名：C 程序设计语言（第 2 版•新版） Author: Brian W. Kernighan, Dennis M. Ritchie
  此书简称 K&R，由 C 语言的创建者撰写，其品质毋庸置疑。写的十分精简，不过并不适合入门，对于那些已经对 C 有一定了解之后的人来说是一种享受_

进阶读物：在没有读完初阶读物前最好不要来读此类书籍，否则身心将受到巨大的摧残

- **NO.6** *C Programming FAQs: Frequently Asked Questions* 2nd edition
  中文书名：你必须知道的 495 个 C 语言问题 Author: Steve Summit
  本书是 Summit 以及 C FAQ 在线列表的许多参与者多年心血的结晶，是 C 语言界最为珍贵的财富之一。适合对 C 有一定了解和一定编程实践后再来看， 1995 年出版了该书的英文第二版，中文版算是第一版吧，2009 年出的，所以说与英文原版有较大改进_

- **NO.7** *Expert C Programming*
  中文书名：C 专家编程 Author: Peter van der Linden
  书的内容虽然很有深度，但是作者语言幽默，读起来还是比较轻松的，中文译者徐波_

- **NO.8** *Pointers on C*

  中文书名：C 和指针 Author: Kenneth Reek

  同上，中文译者仍为徐波，翻译过来的读起来偶尔会有点拗口，全书通过指针这根主线来组织_

- **NO.9** *C Traps and Pitfalls*

  中文书名：C 陷阱与缺陷 Author: Andrew Koenig

  书很薄，英文原版是 1989 年 1 月 11 日出版的，那时候 ANSI 标准都还没出来。可想而知，书中所提到的缺陷已经在 C89 & C99 中得到了相当的改善，所以看之前得对 C89 & C99 有所了解才能吸收最大的价值。书虽然很老了，但是其中的思想大家还是可以借鉴的_

- **NO.10** *C: A Reference Manual* 5th edition

  中文书名：C 语言参考手册 Author: Samuel P. Harbison & Guy L. Steele

  适合在写程序的时候作为参考，对每一条函数都有 C89 & C99 等详细使用区别_

- **NO.11** *The Standard C Library*

  中文书名：C 标准库 Author: P.J. Plauger

  英文原版是 1991 年 1 月 11 日出版的，中文翻译版是 2009 年 7 月出版的_

  本书精辟地讲述了每一个库函数的使用方法和实现细节，同时还给出了实现和测试这些函数的完整源代码_

- **NO.12** *Writing Solid Code*

  中文书名：编程精粹·编写高质量 C 语言代码 Author: Steve Maguire

  英文原版是 1993 年 1 月 1 日出版的，2009 年人民邮电出了中文翻译版， 1993 年电子工业出版社出的基本没有了_

  本书篇幅不长，主要讲的是微软团队在开发大型软件过程中所总结的经验_

- **NO.13** *ISO/IEC 9899:1999, Programming languages — C* 2nd edition

  中文书名：ISO C99 规范

  一切关于 C 语言的疑问，只有一件东西最权威，那就是 ISO 的规范。但这不是正式出版物，是类似法律文本的技术说明，而且只有英文版。新手阅读，提防吐血而亡。。。_

  P.S. 前一阵子似乎发布了 C11 标准，估计还得过一阵时间才能普及开来吧

也说谭浩强《C 程序设计》

一般来说，很多人（比如学姐学长呀）都会推荐谭浩强的书，当时我也被推荐使用这本书，从我个人的实际使用效果来看，我是不会再给学弟学妹们推荐谭浩强的《C 语言教程》了。不可否认，他写的这本书确实比较简单，但个人感觉他这本书主要是针对**非计算机专业了解 C 语言**用的，某种程度上感觉它是应试教育的产物，而不是一本真正教你编程的书。

6.3 C++

C++比 C 复杂的多，所以教材也就纷繁芜杂。既有综合性入门性的，也有专项深入的。由于我 C++方面了解不多，所以也无法做出比较好的推荐，在此我只推荐我了解的三本：

1. *C++ How to Program* 7th edition——适合编程**基础十分不好**的人看，英文已到第 8 版了
2. *C++ Primer plus 5th* edition——C++入门一本很好的参考书，Amazon 上评价很高
3. *C++ Primer* 4th edition——名声在外，中文翻译质量确实不错

6.4 Java

- **NO.1** 疯狂 Java 讲义
  作者：李刚
  之所以把这本书放在 NO.1 当然不是因为作者是李刚…囧，而是因为本书确实比较适合大部分 Java 初学者使用，内容安排也和平时上课类似，读起来比较轻松，某些言语表达似乎有点疯狂，书中有不少案例学习的例子，到底适不适合你就要看个人接收知识的方式喽_
- **NO.2** *Head First Java* 2nd edition
  中文书名：深入浅出 Java Author: Bert Bates, Kathy Sierra
  Head First 系列中比较有影响力的一本书，各种插图比较多，所以读起来没有一般技术类书籍那么乏味_
- **NO.3** *Thinking in Java* 4th edition
  中文书名：Java 编程思想 Author: Bruce Eckel
  书如其名——已经上升到了 Thinking 的层次了，有些人说它适合入门，有些人说它适合进阶，就普遍而言，国外的教材内容即使有深度，一般也不会让读者读起来晦涩难懂，这本书（英文原版）给我的感觉是讲解清晰，又给读者留下了一定的思考余地，听说中文翻译过来的不太好，我没敢去碰，不好在这里评价_
- **NO.4** *Core Java™, Volume I–Fundamentals* 8th edition
  中文书名：Java 核心技术 卷一：基础知识
- **NO.5** *Core Java™, Volume II–Advanced Features* 8th edition
  中文书名：Java 核心技术 卷二：高级特性
  Author: Cay S. Horstmann, Gary Cornell
  如果说 Thinking in Java 侧重于思想的话，那么此书则侧重于 Java 技术应用的讲解_
- **NO.6** *Effective Java* 2nd edition
  中文书名：深入浅出 Java Author: Bert Bates, Kathy Sierra
  Effective 系列的书籍，我就不多介绍了，进阶用_

6.5 Python

码字到码这儿我确实有点累了，二来刚刚接触 Python，对一些参考书的质量也把握不准，不敢轻易推荐，今后在西电开源社区 Wiki 上更新。这里推荐啄木鸟社区：http://wiki.woodpecker.org.cn/moin/%E9%A6%96%E9%A1%B5 国内一个关于 Python 介绍及相关学习非常全面的一个网站，我就不废话了。

6.6 How to find classic books?

这个问题十分关键，也是我最应该分享给读者的。**授之以鱼，不如授之以渔。**全世界每天都在出版各种不同的书，我们不可能有那么多时间一本一本去挑。一般来说，经典书籍肯定是要经得起大家的检验的。

在国内出版电子计算机方面教材比较多的出版社有人民邮电出版社、电子工业出版社、清华大学出版社以及机械工业出版社，这四家出版社出好书的可能性相对会大一些，但并不意味着出的都是好书。国外的 Addison Wesley, Microsoft Press, McGraw Hill, Pearson & O'Reilly 也是尽出猛

书的地方，像 Addison Wesley, Pearson, McGraw Hill 这三家出的其它比如数理电子类教材经典的也比较多，图书馆到处都能看到这三家出版社的书。

以上是从大的范围去挑，书是人写的，书好不好在很大程度上是由编者水平决定的，所以说要去找某领域比较好的书籍可以先找找这个领域有哪些领军人物，再去看看 TA 都写了什么书。

如今的网络十分发达，信息的公开程度也越来越高，像国内的豆瓣[29]在书籍、电影评论这一块就做的非常不错，有很多很多圈子和小组供你查看，如果是比较好的书的话上边一般都会有较多评价，我们可以根据上边的评分以及评论进行抉择。国内买书推荐去卓越亚马逊[30]和当当[31]，但是这两家网店上的书评就不咋地了，用户参与评论的积极性不够高。国外的网站推荐到 www.amazon.com 看看，记住不要把国外的这个网站和国内的卓越亚马逊混一块了，Amazon 上的书评很有参考价值。再者就是一些论坛了，不过国内的论坛总体而言显得很乱，国内网民的参与积极性也不像发达国家那么高，很多回复只是灌水而已，毫无价值可言。国内的 CSDN[32]算是 IT 方面比较齐全的网站了，但是现在里边特别乱，提问题后得到比较满意的答复的可能性较小。在资料整合这一块个人认为大家网论坛[33]做的不错，我很多资料就是从它上边弄下来的，不过自己要注意资料的取舍。

最后还是想提的就是建议大家学会使用 Google[34] & Wikipedia[35], 这两个东西组合在一起能解决你绝大部分问题，更是你成长的催化剂。

有关 Google 的使用方法在它首页的帮助里边有，找时间看看。**工欲善其事，必先利其器！**

6.7 OJ

Online Judge 系统（简称 OJ）是一个在线的判题系统。用户可以在线提交程序多种程序（如 C、C++、Pascal）源代码，系统对源代码进行编译和执行，并通过预先设计的测试数据来检验程序源代码的正确性。[36]

OJ 的题目大部分是关于算法的。题目的输入输出通常是命令行方式，而非图形界面。也就是说，要关注的不是平台的兼容性、文件的格式抑或窗口的布置这种无关紧要的细节，而是问题本身的逻辑实现。

一个用户提交的程序在 Online Judge 系统下执行时将受到比较严格的限制，包括运行时间限制，内存使用限制和安全限制等。用户程序执行的结果将被 Online Judge 系统捕捉并保存，然后再转交给一个裁判程序。该裁判程序或者比较用户程序的输出数据和标准输出样例的差别，或者检验用户程序的输出数据是否满足一定的逻辑条件。最后系统返回给用户一个状态：通过（Accepted, AC）、答案错误(Wrong Answer, WA)、超时(Time Limit Exceed, TLE)、超过输出限制（Output Limit Exceed, OLE)、超内存（Memory Limit Exceed, MLE）、运行时错误（Runtime Error,

[29] www.douban.com
[30] www.amazon.cn
[31] www.dangdang.com
[32] www.csdn.net
[33] http://club.topsage.com/forum.php
[34] IPV4：www.google.com.hk IPV6：ipv6.google.com.hk
[35] 英文：en.wikipedia.org 简体中文：zh.wikipedia.org
[36] 本节部分内容引用 http://baike.baidu.com/view/1583090.htm

RE)、格式错误（Presentation Error, PE)、或是无法编译（Compile Error, CE），并返回程序使用的内存、运行时间等信息。

Online Judge 系统最初使用于 ACM-ICPC 国际大学生程序设计竞赛和 OI 信息学奥林匹克竞赛中的自动判题和排名。现广泛应用于世界各地高校学生程序设计的训练、参赛队员的训练和选拔、各种程序设计竞赛以及数据结构和算法的学习和作业的自动提交判断中。

以下几个 OJ 都很不错，请随意进入，也可以自行搜索，国内很多学校的 OJ 正在热火朝天的建设中，有的甚至直接作为了考试用系统。

西电 ACM- http://acm.xidian.edu.cn/index/

其它的 OJ 的介绍在此就不浪费空间了，见搜搜百科 http://baike.soso.com/v708668.htm

一个人孤单做题的滋味是很郁闷的，尤其是当你确实绞尽脑汁也搞不定的时候。幸好线上还是有很多同样在做题的朋友，他们通常会在论坛 BBS 或邮件列表之类的地方集中讨论解题心得，甚至有可用的答案。建议用 OI (Olympiad in Informatics 信息学奥林匹克)[37]为关键字搜索。

哪里有代码示例可看？这个可以自己 Google，如果你已经能够做一些 OJ 的题了，那么可以考虑看一些开源软件的代码。http://sourceforge.net/ http://github.com http://code.google.com 上有很多开源软件，确定一个应用主题上去找就会有收获。不过说实话，从 OJ 到项目是很大的跨越，所以要找到适合自己的项目来看是不容易的。可以尝试和他人合作做一些小型的项目[38]，边学边用，或者是找一些知名的又比较小规模的项目，然后找他的早期版本的代码，会比较清晰，比如 vim1.0/2.0 lua1.0 apache1.0 这种。

6.8 Usage of programming tools

前边说了那么多编程的东西，那么怎么把自己的想法变为最终的结果呢？——当然是选择合适的编译器（或者类似的东西）啦，首推 GCC[39]（能处理 C/C++、Fortran、Pascal、Objective-C、Java、Ada，以及 Go 与其他语言），关于编译器在这就不多介绍了，前边已经做过初步解释，初学者无需在此浪费过多时间，知道它能把你的代码翻译为计算机可执行的文件就 OK 了。下边针对 C/C++做些介绍，其它语言类似。

就国内的高校 C 语言教学来说，Windows 下的 VC6.0 使用率还是比较广的，估计大部分都是用的盗版汉化软件，我大一一开始学的时候老师上课演示也是用 VC6.0 示范的，不得不在这里小小的抗议一下，初学编程完全无需 VC6.0 这样的庞然大物，光建工程配置文件就能打击一大批新手的自信心，而且莫名其妙的错误是经常会有的！对于初学者，个人觉得 Linux + Terminal 就非常不错，能配合《Linux C 一站式编程》就更完美了。

如果你喜欢 IDE，那也没关系，免费开源的 **Code::Blocks**[40]，自由小巧的 Dev-C++也不错，如果你平时使用多种语言开发，开源的 Eclipse 当仁不让。当然咯，如果你钟情于 Windows，Visual

[37] 一篇介绍 OI 的文章 http://boj.5d6d.com/thread-1656-1-1.html
[38] 西电开源社区的 https://github.com/xdlinux
[39] http://zh.wikipedia.org/wiki/GCC
[40] http://www.d2school.com/bhcpp_book/2_2.php 和 http://club.topsage.com/thread-2221069-1-2.html 有更详细的信息

Studio 则比较适合你，初学时可以用免费的 Express 版，也可以使用微软授权给高校学生授权的 Professional 版本。

值得一提的就是软件调试，这本是一块很大的话题，在这里稍微提一下，《Linux C 一站式编程》中有关 gdb 的使用已经很详细了，有兴趣的去看看，其它 IDE 或多或少也有类似的排错功能，使用方法见各软件帮助文档或网络上的教程。

除了 IDE 这种开发方式，你也可以使用前边提到过的 Text Editor 配合编译器使用，配置好的话熟练后非常方便，效率也较高。更多的介绍请参考 Wikipedia 或者西电开源社区的 Wiki 页面[41]，BlueAuris 在 FAQ 原文也说的相对多一些，个人觉得这一块多说无益，二来我也不是什么 Coding 大神，就不浪费篇幅了误导大家了，现在的网络很发达，随便 Google 一下就能搜到很多答案。

6.9 Other cases

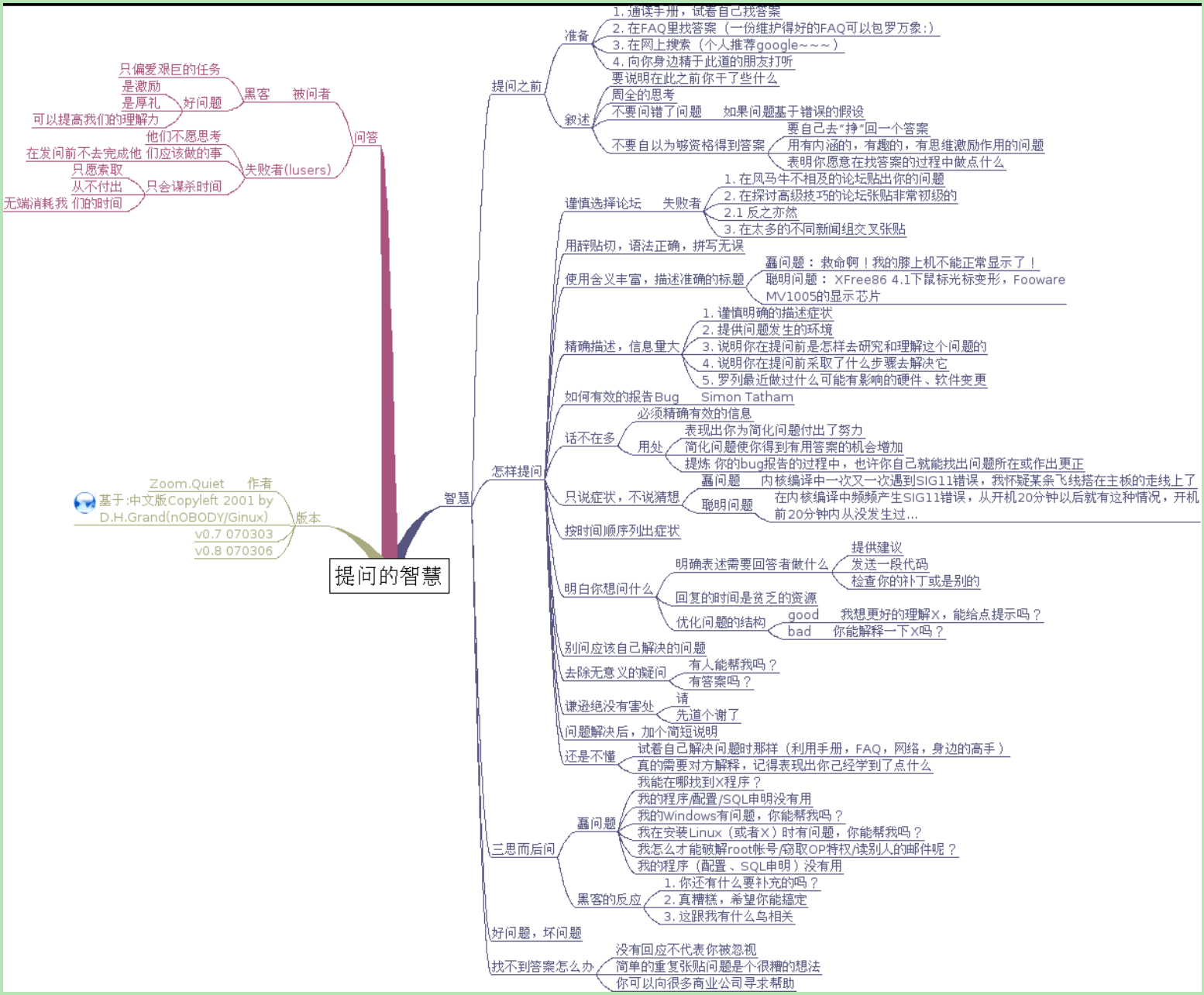
本小节主要讲述相关学习资源及网站介绍

1. 西电开源社区-xdlinux.info
2. 酷壳- coolshell.cn 享受编程和技术带来的快乐
3. http://sunxiunan.com/?p=1661sunxiunan 有关于 C 语言的相关学习博文
4. *C How to Program* 一书有关于 C 语言详细的资料链接，这里就不一一列出来了
5. 《Linux C 一站式编程》-http://learn.akae.cn 不要被书名给误导了，其实有很大一部分与 Linux 没有必然的联系，第三部分与 Linux 有较多联系
6. 《编程新手真言》- http://xisofts.sinaapp.com
7. http://club.topsage.com 计算机专区 资料特别多
8. http://www.d2school.com/ 给新手介绍编程的，主要是 C++相关的东西
9. …在 Wiki 上更新吧，我真的累了=_=

[41] http://xdlinux.info/wiki/index.php/%E5%A6%82%E4%BD%95%E7%94%A8C/C%2B%2B%E5%81%9A%E5%B7%A5%E7%A8%8B

Appendix B-How To Ask Questions in The Smart Way[42]

首先给大家看一张思维导图：看不清楚的话自行到大家网论坛下载清晰的图片



[42] 摘自 http://linuxmafia.com/faq/Essays/smart-questions.html http://club.topsage.com/thread-220478-1-1.html

=====中文翻译版=====

在黑客世界里，当提出一个技术问题时，你能得到怎样的回答？这取决于挖出答案的难度，同样取决于你提问的方法。本指南旨在帮助你提高发问技巧，以获取你最想要的答案。

首先你必须明白，黑客们只偏爱艰巨的任务，或者能激发他们思维的好问题。如若不然，我们还来干吗？如果你有值得我们反复咀嚼玩味的好问题，我们自会对你感 激不尽。好问题是激励，是厚礼，可以提高我们的理解力，而且通常会暴露我们以前从没意识到或者思考过的问题。对黑客而言，“问得好！”是发自内心的大力称赞。

尽管黑客们有蔑视简单问题和不友善的坏名声，有时看起来似乎我们对新手， 对知识贫乏者怀有敌意，但其实不是那样的。

我们不想掩饰对这样一些人的蔑视--他们不愿思考，或者在发问前不去完成他们应该做的事。这种人只会谋杀时间--他们只愿索取，从不付出，无端消耗我们的 时间，而我们本可以把时间用在更有趣的问题或者更值得回答的人身上。我们称这样的人为“失败者”(由于历史原因，我们有时把它拼作“lusers”)。

我们在很大程度上属于志愿者，从繁忙的生活中抽出时间来解惑答疑，而且时常被提问淹没。所以我们无情的滤掉一些话题，特别是抛弃那些看起来象失败者的家伙，以便更高效的利用时间来回答胜利者的问题。

如果你觉得我们过于傲慢的态度让你不爽，让你委屈，不妨设身处地想想。我们并没有要求你向我们屈服--事实上，我们中的大多数人最喜欢公平交易不过了，只 要你付出小小努力来满足最起码的要求，我们就会欢迎你加入到我们的文化中来。但让我们帮助那些不愿意帮助自己的人是没有意义的。如果你不能接受这种“歧 视”，我们建议你花点钱找家商业公司签个技术支持协议得了，别向黑客乞求帮助。

如果你决定向我们求助，当然不希望被视为失败者，更不愿成为失败者中的一员。立刻得到有效答案的最好方法，就是象胜利者那样提问--聪明、自信、有解决问题的思路，只是偶尔在特定的问题上需要获得一点帮助。

======== 提问之前 ========

在通过电邮、新闻组或者聊天室提出技术问题前，检查你有没有做到：

1. 通读手册，试着自己找答案。
2. 在 FAQ 里找答案(一份维护得好的 FAQ 可以包罗万象)。
3. 在网上搜索(个人推荐 Google~~~)。
4. 向你身边精于此道的朋友打听。

当你提出问题的时候，首先要说明在此之前你干了些什么；这将有助于树立你的形象：你不是一个妄图不劳而获的乞讨者，不愿浪费别人的时间。如果提问者能从答案中学到东西，我们更乐于回答他的问题。

周全的思考，准备好你的问题，草率的发问只能得到草率的回答，或者根本得不到任何答案。越表现出在寻求帮助前为解决问题付出的努力，你越能得到实质性的帮助。

小心别问错了问题。如果你的问题基于错误的假设，普通黑客(J. Random Hacker)通常会用无意义的字面解释来答复你，心里想着“蠢问题...”，希望着你会从问题的回答(而非你想得到的答案)中汲取教

训。

决不要自以为够资格得到答案，你没这种资格。毕竟你没有为这种服务支付任何报酬。你要自己去“挣”回一个答案，靠提出一个有内涵的，有趣的，有思维激励作用的问题--一个对社区的经验有潜在贡献的问题，而不仅仅是被动的从他人处索要知识--去挣到这个答案。

另一方面，表明你愿意在找答案的过程中做点什么，是一个非常好的开端。 “谁能给点提示？”、“我这个例子里缺了什么？”以及“我应该检查什么地方？”比“请把确切的过程贴出来”更容易得到答复。因为你显得只要有人指点正确的方向，你就有完成它的能力和决心。

======== 怎样提问 ========

☆ 谨慎选择论坛版块

小心选择提问的场合。如果象下面描述的那样，你很可能被忽略掉或者被看作失败者：

1. 在风马牛不相及的论坛贴出你的问题
2. 在探讨高级技巧的论坛张贴非常初级的问题；反之亦然
3. 在太多的不同新闻组交叉张贴

☆ 用辞贴切，语法正确，拼写无误

我们从经验中发现，粗心的写作者通常也是马虎的思考者(我敢打包票)。回答粗心大意者的问题很不值得，我们宁愿把时间耗在别处。

正确的拼写，标点符号和大小写很重要。更一般的说，如果你的提问写得象个半文盲，你很有可能被忽视。

如果你在使用非母语的论坛提问，你可以犯点拼写和语法上的小错--但决不能在思考上马虎(没错，我们能弄清两者的分别)

☆ 使用含义丰富，描述准确的标题

在邮件列表或者新闻组中，大约 50 字以内的主题标题是抓住资深专家注意力的黄金时机。别用喋喋不休的“帮帮忙”(更别说“救命啊！！！！！”这样让人反感的话)来浪费这个机会。不要妄想用你的痛苦程度来打动我们， 别用空格代替问题的描述，哪怕是极其简短的描述。

蠢问题： 救命啊！我的膝上机不能正常显示了！

聪明问题： XFree86 4.1 下鼠标光标变形，Fooware MV1005 的显示芯片。

如果你在回复中提出问题，记得要修改内容标题，表明里面有一个问题。一个看起来象“Re：测试”或者“Re：新 bug”的问题很难引起足够重视。另外，引用并删减前文的内容，给新来的读者留下线索。

☆ 精确描述，信息量大

1. 谨慎明确的描述症状。
2. 提供问题发生的环境(机器配置、操作系统、应用程序以及别的什么)。
3. 说明你在提问前是怎样去研究和理解这个问题的。
4. 说明你在提问前采取了什么步骤去解决它。
5. 罗列最近做过什么可能有影响的硬件、软件变更。

尽量想象一个黑客会怎样反问你，在提问的时候预先给他答案。

☆ 话不在多

你需要提供精确有效的信息。这并不是要求你简单的把成吨的出错代码或者数据完全转储摘录到你的提问中。如果你有庞大而复杂的测试条件，尽量把它剪裁得越小越好。

这样做的用处至少有三点。

第一，表现出你为简化问题付出了努力，这可以使你得到回答的机会增加；

第二，简化问题使你得到有用答案的机会增加；

第三，在提炼你的 bug 报告的过程中，也许你自己就能找出问题所在或作出更正。

☆ 只说症状，不说猜想

告诉黑客们你认为问题是怎样引起的没什么帮助。(如果你的推断如此有效，还用向别人求助吗？)，因此要确信你原原本本告诉了他们问题的症状，不要加进你自己的理解和推论。让黑客们来诊断吧。

蠢问题： 我在内核编译中一次又一次遇到 SIG11 错误，我怀疑某条飞线搭在主板的走线上了， 这种情况应该怎样检查最好？

聪明问题： 我自制的一套 K6/233 系统，主板是 FIC-PA2007 (VIA Apollo VP2 芯片组)，256MB Corsair PC133 SDRAM，在内核编译中频频产生 SIG11 错误，从开机 20 分钟以后就有这种情况，开机前 20 分钟内从没发生过。重启也没有用，但是关机一晚上就又能工 作 20 分钟。所有内存都换过了，没有效果。相关部分的典型编译记录如下...。

☆ 按时间顺序列出症状

对找出问题最有帮助的线索，往往就是问题发生前的一系列操作，因此，你的说明应该包含操作步骤，以及电脑的反应，直到问题产生。

如果你的说明很长(超过四个段落)，在开头简述问题会有所帮助，接下来按时间顺序详述。这样黑客们就知道该在你的说明中找什么。

☆ 明白你想问什么

漫无边际的提问近乎无休无止的时间黑洞。最能给你有用答案的人也正是最忙的人(他们忙是因为要亲自完成大部分工作)。这样的人对无节制的时间黑洞不太感冒，因此也可以说他们对漫无边际的提问不大感冒。

如果你明确表述需要回答者做什么(提供建议，发送一段代码，检查你的补丁或是别的)，就最有可能得到有用的答案。这会定出一个时间和精力的上限，便于回答者集中精力来帮你，这很凑效。

要理解专家们生活的世界，要把专业技能想象为充裕的资源，而回复的时间则是贫乏的资源。解决你的问题需要的时间越少，越能从忙碌的专家口中掏出答案。

因此，优化问题的结构，尽量减少专家们解决它所需要的时间，会有很大的帮助--这通常和简化问题有所区别。因此，问“我想更好的理解 X，能给点提示吗？”通常比问“你能解释一下 X 吗？”更好。

如果你的代码不能工作，问问它有什么地方不对，比要求别人替你修改要明智得多。

☆ 别问应该自己解决的问题

黑客们总是善于分辨哪些问题应该由你自己解决；因为我们中的大多数都曾自己解决这类问题。同样，这些问题得由你来搞定，你会从中学到东西。

你可以要求给点提示，但别要求得到完整的解决方案。

☆ 去除无意义的疑问

别用无意义的话结束提问，例如“有人能帮我吗？”或者“有答案吗？”。

首先：如果你对问题的描述不很合适，这样问更是画蛇添足。其次：由于这样问是画蛇添足，黑客们会很厌烦你--而且通常会用逻辑上正确的回答来表示他们的蔑视，例如：“没错，有人能帮你”或者“不，没答案”。

☆ 谦逊绝没有害处，而且常帮大忙

彬彬有礼，多用“请”和“先道个谢了”。让大家都知道你对他们花费时间义务提供帮助心存感激。

然而，如果你有很多问题无法解决，礼貌将会增加你得到有用答案的机会。(我们注意到，自从本指南发布后，从资深黑客处得到的唯一严重缺陷反馈，就是对预先道谢这一条。一些黑客觉得“先谢了”的言外之意是过后就不会再感谢任何人了。我们的建议是：都道谢。)

☆ 问题解决后，加个简短说明

问题解决后，向所有帮助过你的人发个说明，让他们知道问题是怎样解决的，并再一次向他们表示感谢。如果问题在新闻组或者邮件列表中引起了广泛关注，应该在那里贴一个补充说明。

补充说明不必很长或是很深入；简单的一句“你好，原来是网线出了问题！谢谢大家--Bill”比什么也不说要强。事实上，除非结论真的很有技术含量，否则简短可爱的小结比长篇学术论文更好。说明问题是怎样解决的，但大可不必将解决问题的过程复述一遍。

除了表示礼貌和反馈信息以外，这种补充有助于他人在邮件列表/新闻组/论坛中搜索对你有过帮助的完整解决方案，这可能对他们也很有用。

最后(至少？)，这种补充有助于所有提供过帮助的人从中得到满足感。

如果你自己不是老手或者黑客，那就相信我们，这种感觉对于那些你向他们求助的导师或者专家而言，是非常重要的。问题久拖未决会让人灰心；黑客们渴望看到问题被解决。好人有好报，满足他们的渴望，你会在下次贴出新问题时尝到甜头。

☆ 还是不懂

如果你不是很理解答案，别立刻要求对方解释。象你以前试着自己解决问题时那样(利用手册，FAQ，网络，身边的高手)，去理解它。如果你真的需要对方解释，记得表现出你已经学到了点什么。

比方说，如果我回答你：“看来似乎是 zEntry 被阻塞了；你应该先清除它。”，然后：一个很糟的后续问题：“zEntry 是什么？”

聪明的问法应该是这样：“哦~~~我看过帮助了但是只有-z 和-p 两个参数中提到了 zEntry 而且还都没有清楚的解释:<你是指这两个中的哪一个吗？还是我看漏了什么？”

======== 三思而后问 ========

以下是几个经典蠢问题，以及黑客在拒绝回答时的心中所想：

问题：我能在哪找到 X 程序？

问题：我的程序/配置/SQL 申明没有用

问题：我的 Windows 有问题，你能帮我吗？

问题：我在安装 Linux(或者 X)时有问题，你能帮我吗？

问题：我怎么才能破解 root 帐号/窃取 OP 特权/读别人的邮件呢？

提问：我能在哪找到 X 程序？ 回答：就在我找到它的地方啊蠢货--搜索引擎的那一头。天呐！还有人不会用 Google 吗？

提问：我的程序(配置、SQL 申明)没有用回答：这不算是问题吧，我对找出你的真正问题没兴趣--如果要我问你二十个问题才找得出来的话--我有更有意思的事要做呢。

在看到这类问题的时候，我的反应通常不外如下三种：

1. 你还有什么要补充的吗？
2. 真糟糕，希望你能搞定。
3. 这跟我有什么鸟相关？

提问：我的 Windows 有问题，你能帮我吗？ 回答：能啊，扔掉萎软的垃圾，换 Linux 吧。

提问：我在安装 Linux(或者 X)时有问题，你能帮我吗？ 回答：不能，我只有亲自在你的电脑上动手才能找到毛病。还是去找你当地的 Linux 用户组寻求手把手的指导吧(你能在这儿找到用户组的清单)。
提问：我怎么才能破解 root 帐号/窃取 OP 特权/读别人的邮件呢？回答：想要这样做，说明你是个卑鄙小人；想找个黑客帮你，说明你是个白痴！

======== 好问题，坏问题 ========

最后，我举一些例子来说明，怎样聪明的提问；同一个问题的两种问法被放在一起，一种是愚蠢的，另一种才是明智的。
蠢问题：我可以在哪儿找到关于 Foonly Flurbamatic 的资料？
这种问法无非想得到“STFW”这样的回答。
聪明问题：我用 Google 搜索过“Foonly Flurbamatic 2600”，但是没找到有用的结果。谁知道上哪儿去找对这种设备编程的资料？
这个问题已经 STFW 过了，看起来他真的遇到了麻烦。
蠢问题：我从 FOO 项目找来的源码没法编译。它怎么这么烂？
他觉得都是别人的错，这个傲慢自大的家伙
聪明问题：FOO 项目代码在 Nulix 6.2 版下无法编译通过。我读过了 FAQ，但里面没有提到跟 Nulix 有关的问题。这是我编译过程的记录，我有什么做得不对的地方吗？
他讲明了环境，也读过了 FAQ，还指明了错误，并且他没有把问题的责任推到别人头上，这个家伙值得留意。
蠢问题：我的主板有问题了，谁来帮我？
普通黑客对这类问题的回答通常是：“好的，还要帮你拍拍背和换尿布吗？”，然后按下删除键。
聪明问题：我在 S2464 主板上试过了 X、Y 和 Z，但没什么作用，我又试了 A、B 和 C。请注意当我尝试 C 时的奇怪现象。显然边带传输中出现了收缩，但结果出人意料。在多处理器主板上引起边带泄漏的通常原因是什么？谁有好主意接下来我该做些什么测试才能找出问题？
这个家伙，从另一个角度来看，值得去回答他。他表现出了解决问题的能力，而不是坐等天上掉答案。
在最后一个问题中，注意“告诉我答案”和“给我启示，指出我还应该做什么诊断工作”之间微妙而又重要的区别。
事实上，后一个问题源自于 2001 年 8 月在 Linux 内核邮件列表上的一个真实的提问。我(Eric)就是那个提出问题的人。我在 Tyan S2464 主板上观察到了这种无法解释的锁定现象，列表成员们提供了解决那一问题的重要信息。
通过我的提问方法，我给了大家值得玩味的东西；我让人们很容易参与并且被吸引进来。我显示了自己具备和他们同等的能力，邀请他们与我共同探讨。我告诉他们我所走过的弯路，以避免他们再浪费时间，这是一种对他人时间价值的尊重。
后来，当我向每个人表示感谢，并且赞赏这套程序(指邮件列表中的讨论--译者注)运作得非常出色的时候，一个 Linux 内核邮件列表(lkml)成员表示，问题得到解决并非由于我是这个列表中的“名人”，而是因为我用了正确的方式来提问。
我们黑客从某种角度来说是拥有丰富知识但缺乏人情味的家伙；我相信他是对的，如果我象个乞讨者那样提问，不论我是谁，一定会惹恼某些人或者被他们忽视。他建议我记下这件事，给编写这个

指南的人一些指导。

======== 找不到答案怎么办 ========

如果仍得不到答案，请不要以为我们觉得无法帮助你。有时只是看到你问题的人不知道答案罢了。没有回应不代表你被忽视，虽然不可否认这种差别很难区分。

总的说来，简单的重复张贴问题是个很糟的想法。这将被视为无意义的喧闹。你可以通过其它渠道获得帮助，这些渠道通常更适合初学者的需要。有许多网上的以及 本地的用户组，由狂热的软件爱好者(即使他们可能从没亲自写过任何软件)组成。通常人们组建这样的团体来互相帮助并帮助新手。另外，你可以向很多商业公司寻求帮助，不论公司大还是小(RedHat 和 LinuxCare 就是两个最常见的例子)。别为要付费才能获得帮助而感到沮丧！ 毕竟，假使你的汽车发动机汽缸密封圈爆掉了--完全可能如此--你还得把它送到修车铺，并且为维修付费。就算软件没花费你一分钱，你也不能强求技术支持总 是免费的。

对大众化的软件，就象 Linux 之类而言，每个开发者至少会有上万名用户。根本不可能由一个人来处理来自上万名用户的求助电话。要知道，即使你要为帮助付 费，同你必须购买同类软件相比，你所付出的也是微不足道的(通常封闭源代码软件的技术支持费用比开放源代码软件要高得多，且内容也不那么丰富)。