如何入门程序设计

下面我们来讨论如何提高大家的编程能力。提高编程能力究竟有没有好的方法？有没有捷径？相信这个问题一直是广大程序员朋友们最关心的问题之一。

1. 理论学习

程序员要想在程序设计上有长足的进步，那么首先离不开程序设计理论的学习。理论学习是奠定程序设计基础的必经阶段，也是不可跨越的阶段。一个程序员不可能连 C 语言和数据结构都不懂，就能写出一个程序来。很多数程序员都是靠自学，这其中的艰辛自不必多言，他们有时会很迷茫，发现自己看了很多书，花了很多时间，依然无法写出像样的程序。这可能是学习的步骤不对。一个循序渐进的学习步骤，能够帮助程序员朋友在最短的时间内达到最好的效果。

首先我们要有一个意识就是，无论干哪行，学哪个方面的技术，在我们下决心学习它的时候，第一个需要解决的问题就是：学习它可以分为哪几个阶段？每个阶段都有哪几本经典的好书供学习？只有在每个阶段都找到了一两本相关的经典的好书，才为我们学习这个方面的技术找准了方向，学习起来也会事半功倍。所以，在我学习程序设计的时候，我对书的要求很高，不是随便拿一本来看，看一本差书，既浪费时间，又毫无收获，而是要找口碑最好，写得非常经典的书籍来看。

下面我们来介绍一个程序员的学习阶段，并同时向大家推荐在每个阶段相关的好的学习书籍：

1）学习 C 语言。这是程序设计入门的第一步。C 语言是一门非常有用而且经久不衰的语言。语言排行榜上仅次于 Java 而长期牢固占据第二的位置。足见 C 语言的生命力。C 语言对于数据类型的描述，全面，清楚，明白，而这些都是编程中的根本。C 语言开发出了操作系统，数据库，编译器，办公软件，杀毒软件等很多大型系统软件。因此将 C 语言作为程序设计的入门语言是正确的选择。以后也肯定会有用武之地，事实上，很多优秀的程序员只懂 C 和汇编。学习 C 语言的入门教材可以选择谭浩强老师编著的《C 程序设计》，或者《C 程序设计语言(K&R)》，《一站式学习 C 编程》。

2）理解结构化程序设计思想。学 C 语言，可以从数据类型，基本表达式，再到条件语句，循环语句，数组和结构体，然后学习函数。到现在为止，应该重点理解结构化程序设计的思想。所谓结构化程序设计，即程序设计以模块功能和处理过程设计为主，由顺序，选择，循环三种基本控制结构构造。这样的程序一般没有 goto 语句，一个入口一个出口，自顶向下，逐步求精。推荐书籍：《C 语言编程精粹》--姜静波等译--电子工业出版社。

3）学习汇编。汇编是一门很有价值的课程。有了程序设计和 C 语言的基础，我们可以紧接着学习汇编语言。与 C 语言相比，汇编语言显得晦涩难懂。汇编语言包括各种各样的寄存器，内存地址，寻址方法，8086 的常用汇编指令，再加上一些伪操作，还有 DOS 和 BIOS 中断的调用。有了这些概念和汇编基础，我们就可以看懂书上的汇编程序了。到此，你应该会对计算机的理解更深一层。学好汇编语言，对于理解计算机程序执行的底层机制，以及以后程序调试和分析，都有很大的帮助。推荐汇编学习书籍：《汇编语言》--王爽，《天书夜读》--邵坚磊。

4）学习指针。有了汇编的基础，再来学习 C 语言中的精华部分指针。现在你要对 C 语言全全面面的学一遍了并试着 C 语言和指针写一些小程序。推荐书籍：《高质量 C/C++编程》《C 语言编程精粹》。

5）学习数据结构。数据结构是算法的基础。算法又是程序的核心。所以，学完以上的内容，非常有必要研究学习数据结构。字符串，数组，链表，堆栈，队列，二叉树，排序，查找，HASH 表等等，都要牢固掌握。数据结构教材推荐使用清华严蔚敏老师的《数据结构（C 语言版）》教材。数据结构是程序设计中的关键。因此我们一定要多花精力把数据结构学好，打下牢固的基础。不能急于求成，数据结构都没学好，就想用 VC 等开发工具，写一个不错的软件出来。除了严蔚敏的教材，还推荐一本李春葆老师的《数据结构习题与解析》李春葆 C 语言版

6）学习网络与数据库。紧接着应该学习一些数据库的知识，网络的知识，还有一些常用的算法了。在实际的程序开发中，网络和数据库是应用非常广泛的两个领域。SQL 编程和 TCP/IP 协议，socket 编程等。可以在项目中使用到了数据库和网络编程的时候，及时补充相关的知识。开始可以先简单了解一下。

7）学习 C++，理解面向对象编程思想。如果以上的知识你都学精通的话，就可以开始 C++的课程了。找本 C++上手的书（推荐《C++ Primer》），用一个月的时间去了解 C++和面向对象语言程序设计思想，包括数据的封装，关系的继承，和多态等。C++的推荐书籍包括：《C++Primer》，《Effective C++》，《Thinking in C++》，《More Effective C++》。

8）学习 VC。学习了 C 和 C++，我们还不能快速的写出一个界面程序。写程序都需要有一个开发类库，来方便和加速我们的开发，比如 VC 的 MFC 库。因为程序设计中，不是每一项任务都需要从头开始，而完全可以重用一些代码和公认的库。因此，需要找一本 VC 上手的书（推荐《VC 技术内幕》），花一个星期的时间学习 VC 的界面和用法，学习消息映射机制，然后就可以做一些简单的应用了。要想学好 VC，没有扎实的 C++知识是不行的。以下书籍是 VC 程序员必须常翻常看的：《深入浅出 MFC》。VC 中最经典的是 VC6.0，然后微软逐渐对其版本进行升级，现在的开发版本已经是 Visual Studio 20XX 了。MFC 界面程序主要就是对控件和对话框的设计与使用。

能写一个简单的基于对话框的程序，学习常见的控件（如 CListCtrl）的使用就 OK。

9）深刻理解 C++面向对象的思想。在项目开发中不断实践和提高面向对象程序设计能力，多用多总结，并看一看设计模式的书。

10）编程的时候，遇到问题和各种错误是难免的，所以，搜索引擎和 MSDN，技术文档一定要熟练使用。

11）学习程序调试技术。熟练掌握各种调试方法和调试工具的使用。这个时候我们可以学习的调试工具包括 Windbg 调试器，OllyDbg 等调试器，IDA 等。推荐张银奎写的《软件调试》一书。

12）如果你严格的走好了以上的每一步，你就顺利的走上了程序员的发展道路。

13）此时，我们还应该花时间进一步学习操作系统原理，编译原理。通过对计算机的基本原理有一个认识和理解，这对我们以后写好程序，理解程序的执行和程序的调试都有很大的好处。

14）现在，如果再学习 Java，python,php,C#等别的语言，就非常容易了。因为语言都是触类旁通的，拿来就用了，而且 C 和 C++算是语言中最难掌握的，如果都学会了，那么其它语言都很容易的了。

学习了 JAVA，就可以进一步学习移动编程，比如 Android 应用编程（推荐《疯狂的 Android》）了，而在一开始又学习了 MFC 的消息映射的思想，学习 ANDROID 界面编程也应该很容易理解里面的事件和监听机制，Android 的界面布局与 MFC 不一样，是通过 XML 来进行独立布局的，这个需要重新理解。从长远来看，程序员起码应该掌握：C，C++，Java，Python 这几种语言，是很有必要的。

此时，还应该学习 LINUX 操作系统，比如 LINUX 操作系统的一些常用的命令，GCC

开发 C 和 C++程序，PYTHON 语言等。

15）最后，我们再深入系统底层，学习内核驱动，系统安全开发等。在这些领域，C 语言是唱主角的一个语言。在内核驱动和系统安全开发领域，我们可以为系统挂上钩子函数，或者利用过滤驱动，监控系统的执行。这个时候供我们学习的书有《Windows Internal》，《Windows 2000 设备驱动程序设计指南》,《寒江独钓：Windows 内核安全编程》,《Rootkits: Subverting the Windows Kernel》,《0day 安全：软件漏洞分析技术.第 2 版》,《Android 软件安全与逆向分析》,《Linux 设备驱动程序》,《Linux 内核设计与实现》以及赵炯博士的《Linux 源代码分析》等。

2. 上网交流

作为一个 IT 从业者，一定要利用好互联网。遇到问题，首先要利用搜索引擎（Google，Baidu）进行查询，你会发现网络是一个巨大的知识库，很多你遇到的理论和技术问题都能够找到你满意的答案，有的还对知识的总结得非常系统全面。其次，无论是做哪个专业领域的程序设计，你都能找到相关的讨论社区，里面有很多经验丰富的专家和高手。你可以就相关问题向他们请教，热心的高手都会给你帮助，当然，如果有别人不懂的问题如果你知道答案，你也应该帮助他们。缺乏交流的程序员，由于闭门造车，最终会落后于外界的步伐。

3. 阅读代码

要想快速提高自己的编程水平，光是理论学习是不够的。还应该多阅读一些别人写的程序代码，学习别人的编程技能。这对自己的编程水平的提高非常有好处。目前，很多网站提供各个领域的源代码下载，也有很多好的开源程序代码如 Linux，OpenSolaris 等都是开源的。我们可以利用开源的大好机会，下载别人的代码来学习编程技巧。对于代码不但要分析，还可以编译运行相应的程序，也可以修改其中的代码和错误，来快速的锻炼和提高自己的编程和调试水平。阅读代码可以按照模块进行。先理清程序中的数据结构，然后分析每个函数实现的功能，包括它的输入和输出，算法流程等。推荐使用的代码阅读工具是 SourceInsight。

建议大家可以阅读一下 Linux 的内核代码和 UNIX 的内核代码。看看这些经典的系统是如何写出来的。这个方面也有很多介绍这些源代码的书籍。比如赵炯博士写的《Linux 内核完全剖析》,《莱昂氏 UNIX 源代码分析》。

4. 动手实践

无论干哪行或者学习什么，实践都是重中之重。前面我们提到的都是一些理论和经验的准备，对于理论和技巧的掌握，最终需要落脚到动手实践。无论是我们实际学习过程中，还是我们有了一个好的想法，我们都要勇于去动手实践。实践是检验我们学习和想法最好的手段。只有亲自实践了，才能加深理解，才能加强印象。只有实践，才能发现问题，因此也才能更好的解决问题。因此，建议大家在学习的时候，可以将其中的代码亲自上机试验一下。在平时中对某个问题把握不是很准确，也可以通过上机验证一下。久而久之，就有了一个实践的良好习惯。

比如，我们拿不准在某个平台上 int，short 的长度，我们可以写一个简单的程序来研究：

```
int main(int argc, char *argv[])
{
    printf("sizeof (int) = %d, sizeof (short) = %d, sizeof(char) = %d\n", sizeof (int),
        sizeof (short), sizeof (char));
    return 0;
}
```

又比如，我们拿不准如果程序发生了内存泄漏，如果把程序结束了之后，系统是否会

为我们清理回收泄漏的内存，我们可以写如下程序验证：

```
int main(int argc, char *argv[])
{
    char *p = NULL;

    p = (char *) malloc(10*1024*1024);
    return 0;
}
```

5. 积累代码与经验

要想成为一个有经验的程序员，就应该经常总结编程经验。将自己在学习和项目参与中的各种经验总结下来，或者放在自己的 blog 上与人分享，或者单独写在一个笔记本上，或者自己形成一个属于自己的 C/C++代码库。对其中一些经常使用的代码和算法，最好能够熟记于心，随时能写得出。

在编写程序过程中遇到的常犯的错误或者有用的总结，都提倡写下来，供以后参考，避免以后遇到类似的问题又得从头解决。编程经验是宝贵的财富。

6. 参与项目（实习，开源项目）

如果说动手实践是我们对所学知识的检验和加深理解的过程，那么参与具体项目的开发则是一个运用与提高的过程，也就是学以致用。一个好的项目，能够将一个程序设计菜鸟直接培养成不可一世的编程高手，项目对程序员的影响是最大的。因此，必须要寻找参加项目开发的机会。如果还是学生而身边又没有什么项目可以参与，就可以找机会去各个公司实习以获得参与项目的机会，另外也可以寻找一些外包项目来参与，既可以获得提高，还可以有一笔收入。还可以直接参加某些开源社区，为某一个开源软件提供自己的代码。

首先，我们应该明白参加开源社区开发的意义。

参与开源社区的开发，有很多好处。

第一，我们的编程能力和软件工程思想得到了很大的锻炼。

第二，感受到了开源的精神，为开源做出了贡献。

第三，有利于个人发展。

第四，不虚度光阴。

英语与编程

有一句话说得好：英语是计算机的母语，是程序的母语，所以必然是程序员的母语。大家知道，计算机诞生于英语为母语的国家。这就注定了，计算机与英语是分不开的。英语能力对于计算机程序设计的学习无疑是非常重要的。英语不好，学习编程会受到很大的影响。所以，在学习程序设计的同时，提高自己的英语水平是必须的。那么，英语水平对于计算机编程有哪些影响呢？

程序设计里面的语言是以英语的形式存在的。其中就有很多关键字。更重要的是，写程序时候的变量起名也很讲究。用英文单词或者缩写起名字的可读性要比使用拼音的可读性强得多。能起可读性强的英语变量名字，是写程序的一个重要的方面。新生的小孩的父母为他取名会费劲心血，而程序员为变量取个可读性好的名字，也很重要。

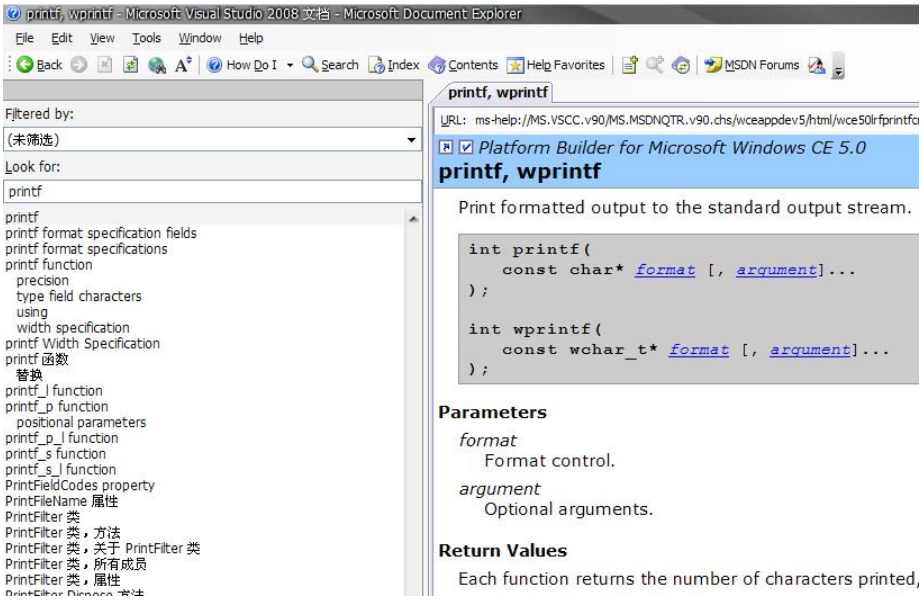
程序中的变量名、函数名起得好不好是决定代码质量和可维护性的最关键因素。高质量

的代码应该是这样的：代码基本上是自解释的（self-explanatory），不需要太多注释，不管代码的规模有多大，具有相关知识背景的读过文档的人都可以立刻上手、立刻参与维护和开发。要想使代码能够自解释，给变量和函数起个好名字很关键，很显然，只能用英文起名字，一是用汉语拼音起名字可读性很差，二是英文单词通常比汉语更 expressive（看吧，如果用汉语来说，就得说“表达能力更强”、“更有表现力”这么罗嗦），由于程序的复杂性，变量和函数往往表示一些很抽象的概念，起个既准确又简洁的名字并不容易，可是很多时候，用汉语需要很多字才能说清楚的一件事，用英文一个单词再加点词形变化就能说清楚了。

另外，大量的技术首先是通过英文技术书出版的。因此，想要第一时间获得新技术，就得首先要会英语，能够阅读英语技术书籍。虽然很多经典的英语书籍翻译成了中文，但是也并不是所有书籍能被翻译成中文的。而且中译本的质量也可能参差不齐。当然，如果有好的中文译本，建议也可以直接看中文，以提高学习效率。

除了大量的技术书籍之外，更多的技术文档，手册，以及 Windows 开发的技术参考 MSDN 都是用英语表述的。如果说书籍有人去专门翻译，那么这些技术文档也许就很难有人为你翻译了。你只能和英语面对面，去啃这些技术文档。甚至有的论坛里对一个问题的解决也是通过英语来讨论的。要想理解其中的解决方法，你也得会英语才能读懂。

其次，与计算机相关的很多问题，首先可以利用搜索引擎去解决。你遇到的每一个问题，都不可能是第一个遇到的，没准网上就有了现成的解决方案。所以，遇到问题，第一就是要会利用搜索引擎去解决问题。由于计算机是一门西方引入的科学，因此，有的问题在使用搜索关键词的时候，如果用中文很难搜到，没准使用对应的英文作为关键词，就很快能搜到相应的问题答案。因此，学会英语，对于使用用英文去向搜索引擎寻找答案也具有很大的优势。



既然英语这么重要，那么很多人可能就会比较着急的要问了，究竟要多好的英语才能有资格学习程序设计呢？英语不好难道就不能学习程序设计了呢？大家知道，英语能力分为听说读写这四项技能。而在计算机程序设计领域里，读最重要，写次之，说和听对程序设计并无太大的关联，除非你要进一个外企工作，必须与老外用英语交流。程序员需要使用英语阅读大量的文档资料，需要用英语来写程序，写注释等。因此，平常通俗的被大家所批判的哑巴英语，在程序设计里面，依然可以有用武之地，只要你会读会写，就不会影响大家。

实际上，高中英语的水平，就不会影响你进一步使用计算机相关的英语了。因为，高中之后，英语的语法基本上都涉及了，唯一需要补充的就是大家的英语词汇问题了。

那么，如果英语很差，怎么提高自己的英语水平呢？这要取决自己在英语的哪块比较欠

缺。如果语法比较差，那么请尽快找本讲语法的书籍，恶补一下英语语法。如果语法没什么问题了，那么请通过大量阅读来提高自己的词汇量和语感。笔者在大学学英语的时候，就明白了学习英语的最好的途径就是大量的阅读，并记住那些遇到的生词。有人建议在阅读过程中，遇到生词就放过。笔者持相反的观点。遇到生词，都应该去查词典，然后记住它，这样就能快速的提高自己的词汇量。

数学与编程

从接触计算机程序设计的时刻开始，身边的很多老师，师兄，同学，朋友都在告戒到，计算机与数学有很大关联，实际上，计算机就起源于数学理论。因此，嘱托大家要好好学习数学。计算机专业就开设了很多数学的专业课，计算机专业考研也把数学作为必考内容。

很多朋友，因为数学水平不足而没有信心学习计算机程序设计，畏惧不前。在这里，笔者想要提出不同的看法。实际上，数学确实对计算机很重要，这一点毋庸质疑。但是，数学一般应用在一些复杂的算法和复杂问题的解决上。比如搜索引擎，比如图形学等相关研究领域。这些问题的解决，一般是计算机领域的科学家致力于的。

而大多数情况下，大家学习计算机程序设计，并不需要去解决这么复杂的理论问题，大多数解决的是一些实际的应用工程问题。这些问题的解决，对数学的要求就没有那么高了。

通俗的讲，如果大家只打算做个程序员，而不是计算机科学家，那么数学知识的掌握，就不要求那么高了。更具体的讲，只要具备高中数学基础，学习计算机程序设计是没有任何问题的。

笔者从事计算机 IT 工作这么久，也很少在实际项目中用到很深的数学理论知识。

当然，数学作为一门思维体操，数学体现出来的思维方式，解决问题的思想和方法，是值得程序设计领域借鉴的。

但总的来说，不要因为数学不好，而不敢学习计算机程序设计。

此外，网上有一篇文章写得很好：

两个故事讲完了，究竟如何做技术方向的选择呢？答案就飘在风里……

1、操作系统、数据结构、算法、网络等基础技术应该在大学时代深入学习，如果毕业了你还没有掌握这些内容，那就随用随学好了。学习这些基础理论极为枯燥，只有实际工作中的需求才能给你最大的学习动力去掌握这些艰深的内容。

2、至少要掌握一门静态语言，比如 C、C++、Java、C#、Objective-C 等。至少掌握一门动态语言，比如 Python、Ruby、PHP 等。

3、推荐学习一些同时具备动态语言和静态语言的特性语言，比如 Go、Swift、Scala 等。这样你会对面向对象编程、面向过程编程、编译型、解释型语言有更深入的了解。

4、系统的构建自己的知识体系，而不是局限在某个点上。经常有读者问我，我前几年一直在写 VBA/ActionScript/Delphi/SQL ……现在项目组突然不再采用这些语言了，怎么办？很多人难以预料未来技术的走向，但是你至少要构建自己的技术壁垒和平台。学习 Java，就应该构建你自己的 JavaEE 平台；Objective-C 对应 iOS/OS X 开发平台；C#，对应 .Net 平台，SQL，对应数据库平台。如果你在用 ActionScript，那你不应该局限在 Flex 上，你对应的是整个前端平台。

立足平台，你会站得很稳。立足一个点，你可能摔的很惨，就是这样。

5、主动选择技术方向比被动等待好。根据自己的兴趣和技术的发展主动选择，就像小明一样，有时候放弃也意味着得到。

6、不要过于追新，不要每出一门「颠覆性」的语言或技术都投入精力物力。追新的后果很可能是该学的没学会，不该学的学完也忘了。我有一哥们，我们都在写 JavaScript 的时候，他认为 Java 新推出的 JavaFX 才是前端的未来……然后就没有然后了。我们都用 Java 的时候，他认为 ERlang 是才是编程语言的未来……然后就没有然后了，可谓一步早，步步早，让人扼腕叹息。

7、也不要过于保守，比如 Go、Swift、Docker 等技术，我个人以为是值得投入时间和精力的技术。

没有 8 了，写到这里，冬夜已经黑的不像样子。站在阳台望出去，仿佛看着某个巨大 IDE 的黑色编码主题，我想起了某位大牛的一句话：我不是懂得多，我只是学的快而已。