大学学长亲述的C/C++“绕坑”指南

脑洞大开

C语言另类攻略

刘隽良 编著
胡华 李万清 主审

西安电子科技大学出版社
http://www.xduph.com

脑洞大开——

C语言另类攻略

刘隽良　　编　著
胡　华　李万清　主　审

西安电子科技大学出版社

图书在版编目(CIP)数据

脑洞大开：C 语言另类攻略/刘隽良编著. —西安：西安电子科技大学出版社，2016.12

ISBN 978-7-5606-4381-6

Ⅰ.① 脑… Ⅱ.① 刘… Ⅲ.① C 语言—程序设计 Ⅳ.① TP312.8

中国版本图书馆 CIP 数据核字(2016)第 306999 号

策　　划　陈婷　马乐惠
责任编辑　陈婷
出版发行　西安电子科技大学出版社(西安市太白南路 2 号)
电　　话　(029)88242885　88201467　　邮　　编　710071
网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　虎彩印艺股份有限公司
版　　次　2016 年 12 月第 1 版　　2016 年 12 月第 1 次印刷
开　　本　787 毫米 × 1092 毫米　1/16　　印　张　13.75
字　　数　270 千字
印　　数　1～1000 册
定　　价　25.00 元

ISBN 978-7-5606-4381-6/TP

XDUP 4673001-1

序

知识学习应愉悦轻松，知识传授应以学生为本。

C 程序设计语言诞生至今已有四十多年的历史，对其研究介绍的著作和教材数不胜数。当前，C 程序设计语言教材大多是以传授者的视角编写的，内容也大都专注于语法规则的讲解，偏重于知识的灌输。就知识讲解而言，教材或者工具书采取这样的编写方式确有好处，但对于信息时代的学习者来说，学习难免枯燥乏味。因此，当一位老师向我推荐杭州电子科技大学一名在读大学生于大一时凭兴趣写的一本“很有趣、很有特点”的C语言教材时，我实在难以想象出他会写出什么新意来——直到浏览了其全部书稿。本书的作者捕捉了大量被其他书籍忽略但在实践中非常重要的细节内容，以一种与读者互动的姿态和语言娓娓道来，答疑解惑。同时我也非常赞赏其叙述的独特视角，认为确有理由向大众推荐和分享这本好书。

此书作者刘隽良同学是杭州电子科技大学信息安全专业的本科在读学生，在学习 C 语言程序设计课程时，感觉教材“刻板无趣”。他认为，如果不能以自己的方式准确地阐释所学的内容，就不能算是真正地掌握知识。为此，在学习过程中，刘隽良开始描绘自己心中的C语言面貌。经过两年的思索和积累，完成了这本书的初稿。有趣的是，这本书稿只是个开始。在学习数据结构和密码学课程的时候，刘隽良又以同样的方式完成了其他两本书稿的创作。更难得的是，在完成这三本书稿的过程中，他从未向他人透露过自己的创作历程。直到一个偶然的机会——他参加杭州电子科技大学华为企业奖学金评比，将这三本书稿的写作经历展示给评委老师时，师生们才知晓此事。

刘隽良同学特立独行的想法和坚持不懈的毅力，深深触动了我。作为一名教育工作者，我认为应该让更多的学生分享他这些有益的学习经验，并请专业老师辅助他完善了三本书稿。经过努力，西安电子科技大学出版社正式出版了本书。

本书以一个曾经的学习者的视角，从计算机硬件运行方式、软件执行方式、编程语言以及编译技术等多个层面展示了C语言，以诙谐幽默的语言生动形象

地向读者描述了C语言的精彩世界。

如前所述，本书的新颖之处在于以一种学习者的姿态与读者互动，并通过大量图片和逐步图解来辅助理解，将学习C语言变成一种享受。书中收录了大量被其他书籍忽略的但在实践中必须掌握的细节，巧妙地展现出C语言不常为人关注的一面，让读者在轻松愉快的氛围中，能够“知其然，后而知其所以然”。全书集C语言入门、进阶以及C++面向对象入门于一体，逻辑清晰，语言流畅，深入浅出，细节翔实，既通俗易懂又不失严谨。可以说，这本书从以学生为中心的视界与角度，引导学习者形成勤于思考的习惯，鼓励学习者将所学的知识用自己的见解表达出来，从习惯性地被动接受教材的灌输中脱离出来，这对我们反思教育教学改革不无裨益。

在此，让我们向刘隽良同学表示热烈祝贺！期待他再接再厉，在今后的人生中绽放更多的精彩！

胡　华

2016年春于杭州电子科技大学

阅读易误导，实践出真知

——前言什么的

貌似每本书都需要有个叫做前言的东西。嗯，写点什么好呢？

一、一点点不算感悟的感悟

阅读易误导，这个听起来有点匪夷所思啊～不都说书是人类进步的阶梯嘛～你怎么又说阅读易误导咧？

不假，对于编程书籍而言，聆听大师教诲的确很有必要，第一次看这类书的感觉的确是醍醐灌顶。不过，当看的书多了，你就会发现虽然大家说的都有道理，但是又各有差异。毕竟每个人对同一个问题的看法和见解都不一样，而书就是他们各自见解的合集，他们将自己的理解写出来供别人参考，然后看过这些感悟的人又有了自己的见解，便又可能另立新作，以此类推周而复始。

然后，麻烦就来了。

当你需要知道某个内容的时候，相应著作百花齐放，良莠不齐，它们或对或错，这都不重要，重要的是在这个过程中，你会不知不觉忘掉你自己的见解。这点就可怕了，你开始变得人云亦云，变得知其然而不知其所以然，你会觉得你所想出的一切都只不过是在翻版别人的感悟，而不是自己发自内心最想表达出来的东西。

这就麻烦了，毕竟学习编程最重要的不是你看过多少本书，而是你能够悟出多少奥义，你能将多少知识用自己的见解表达出来而不再只是因为教材就是这么写的所以你就这么做。

所以，从这个层面上来说，在编程方面，阅读易误导。

领悟，靠的是自己。书，永远只是辅助。或许，背下书中的知识可以考试不挂科（事实也确实如此），不过要是真想将这些内容变成自己的东西，只有躬亲实践自己领悟，别无他法。

所以在写这本书时，我更多的是希望读者能够学会独立思考感悟，而不是单纯的死记知识。编程是一门艺术，所以，很多东西，只可意会，不可言传，若欲意会，唯有躬亲。这也许就是我在写这本书时最大的感悟吧。

二、写作缘由与经历

这本书的初稿完成于 2014 年 8 月，是我第一次在学校学习完 C 语言课程的暑假。起初的原因是对所使用的教材的知识讲述方法有点“怨念”，觉得知识不应该是这样的枯燥，

应该是立体且很有趣的，觉得如果不能把所学的东西以自己的方式描述出来，就并不能算是真正的理解，因此，这本书的初稿就诞生了……作为第一次尝试，现在看当年的初稿不禁感叹自己的毅力。虽然初稿内容很浅，错误在所难免，但是作为当时自己的最高水平，真的已经是极限了；而且书中的语言风格和行文方式以及内容编排都有自己的特点，这也直接决定了这本书的与众不同。而后来由于机缘巧合获得此次出版机会后，我再次使用了近半年时间重新对初稿进行了多轮“骨灰级”修改——将原有初稿页数增加了近一倍，修改完善N多的错误和不足，使得内容更加准确严谨，更加符合最新标准。

由于本书的出发点不是作为一本“传统”的教材，所以全书的框架设计、内容逻辑相对于教材有较大区别。为了能够让大家更容易轻松地领悟C语言，我对本书的知识框架做了较大的调整——首先我们会从计算机体系结构入手，从计算机硬件运行方式、软件执行方式、编程语言以及编译技术等多个层面结合起来全方位立体展示，以便于更好理解语言本身，同时辅以大量图片辅助理解并搭配各种小问题一起研究，较好地摆脱了传统书籍的说教式知识传授过程。此外，在本书中我们将更加注重细节，对大量不被提及的细节不再人云亦云而是告诉你为什么会是这样，让你能够更好地理解和掌握语言本身。

希望这样的设计能给大家带来更好的学习体验。

三、致谢

感谢父母的支持，让我能够尽情做自己喜欢的事情。在本书的成书过程中，杭州电子科技大学胡华副校长和李万清老师对书稿进行过多次审核，提出了很多很有价值的修改意见，非常感谢他们的付出，使得这本书能够以更为完善的姿态展现在读者面前；同时要感谢西安电子科技大学出版社的出版支持，尤其感谢编辑陈婷老师和马乐惠老师在本书出版过程中提供的诸多帮助(尤其像我这种“不守规矩”的，真是辛苦她们了……)。

最后还要感谢某神秘人士K，作为最初版本的原始读者，是你向我提供了最初动笔的动力，从而才诞生了这本书。

四、本书结构

本书主要分成了5章：

第1章是一个开头总结和引导，简单介绍了计算机硬件运行方式、软件执行方式以及C语言代码从预处理到最终编译成可执行文件的过程，并总结了在C和C++中普遍通用的规范代码模式以及一些要注意的点。

第2～4章是对C语言的总结，作为一门历经40多年依然经典而坚挺的编程语言，它自然有着与众不同的魅力与风格，这三章分别从关键字、函数以及数组和指针等方面对C语言进行了多方面的剖析，并深入细节细化理解，让读者能够对细节做到知其然又知其所以然，让读者在多问些为什么的过程中进阶C语言水平。这部分内容适合C语言初学者快速入门，让入门者快速进阶，也适合初级进阶者查漏补缺。

第5章则是基于C++的面向对象模型快速过渡与理解，帮助读者在理解C语言面向过

程思维后向面向对象的入门级过渡，以一章的内容将C++中最主要的子集以最好理解的状态展现在读者面前，适合作为C++的初学入门指导。

五、求“勾搭”

当然，毕竟金无足赤，人无完人，更何况我自己也还远远达不到真正的高手水平……所以书中一定还会有不足和众多这样那样的问题，大家如果发现了什么瑕疵或者对这本书有更好的建议，随时欢迎沟通交流指(gou)教(da)。

联系邮箱：ddizxt@126.com

最后希望这本书能对你有所启发哦。 

刘隽良

2016/5/14

杭州电子科技大学

目　录

第 1 章　一点点想说在前面的话

1.1　计算机是怎样运行的?

在正式介绍 C 语言之前，简单讲解一下计算机的运行原理是很必要的。

至于为什么很必要咧？首先，要想了解 C 语言的本质，只从语言本身的层面讨论是远远不够的，我们需要一些对计算机硬件基础的了解。其次，即使不是为了学习语言，面对一个你整天都在用的东西，肯定都有一点想了解它内部运行机理的欲望吧。

所以今天，我们就来好好讲讲这些内容。

首先，我们需要知道的是，计算机并不“聪明”：它没有思想，没有自己的想法和认知能力，更“笨”的是它甚至自己都不知道自己能做什么，它需要人们给它命令才能够按照人们的命令去工作，按照人们给它的命令去处理人们给它的内容，再把结果反馈给人们，这就是它所能做的事。那么它如何才能听懂人类的命令呢？人类之所以能够沟通交互，是靠语言，自然机器也不例外，计算机能够与人交互，靠的是“机器语言”——而这个机器语言对我们人类而言可能很陌生，因为它是一大堆二进制代码，而为什么是二进制代码呢？

嘿嘿，原因还是因为计算机太“笨”啊！

因为计算机是一个机器，它是由各种电子元器件组成的。而这些电子元件一般只有两个状态：通电、断电，通常我们以通电表示二进制中的数字 1，断电表示 0。正是由于电子元件所能表示的状态如此有限，计算机没法“认识”1 以上的数字。如果你真的想让计算机能够认识 0～9 的十进制数字的话，那估计电子元件的状态要有下面这些：

0：断电；

1：通一点点电；

2：通比通一点点电多一点点的电；

3：通比通一点点电多一点点的电之后再多一点点的电；

⋮

9：通满电。

好吧，总觉得即使计算机对着这种进制方法不崩溃，设计电子元件的人也要崩溃。(囧)

正由于上述种种原因，计算机的机器语言是只有 0 和 1 的二进制形式的代码。那这些二进制代码是谁处理的呢？自然是我们刚才提到的那些用电子元件制造的硬件；那这些二进制代码是谁给硬件的呢？是硬件上运行的软件；那谁是硬件谁是软件呢？

什么是硬件，什么是软件，这个我们拆个机箱就知道了。不知道大家有没有拆过台式机机箱，拆过的话可能会发现，其实机箱里东西并不多，无非就是下面这些：

(1) 机箱(主机的外壳，用于固定各个硬件)。

(2) 电源(主机供电系统，用于给主机供电稳压)。

(3) 主板(连接主机内各个硬件的躯体)。

(4) CPU(主机的大脑，负责数据运算处理)。

(5) 内存(暂时存储电脑正在调用的数据)。

(6) 硬盘(主机的存储设备，用于存储数据资料)。

(7) 声卡(处理计算机的音频信号，有主板集成和独立声卡)。

(8) 显卡(处理计算机的视频信号，有核心显卡(集成)及独立显卡)。

(9) 网卡(处理计算机与计算机之间的网络信号，常见个人主机都是集成网卡，多数服务器是独立网卡)。

(10) 光驱(光驱用于读写光碟数据)。

(11) 软驱(软驱用于读写软盘数据，然而软盘如今已经彻底淘汰= =)。

(12) 散热器(主机内用于对高温部件进行散热的设备)。

像这些我们看得见摸得着的有实体存在的部件，我们称其为硬件；相反，对于没有实体存在的部件，我们称之为软件。比方说操作系统(Windows、Linux 等)以及我们安装的各种应用程序(QQ、IE 等)，它们没有实体，即为软件。

刚才我们说过硬件处理二进制机器语言指令，其中最重要的部件应属 CPU，它负责计算机内部绝大部分的数据运算，并将运算结果传输给其他硬件最终呈现给我们不同的内容。为了更为简单地描述这段工作，我们上张图片(见图 1-1)。

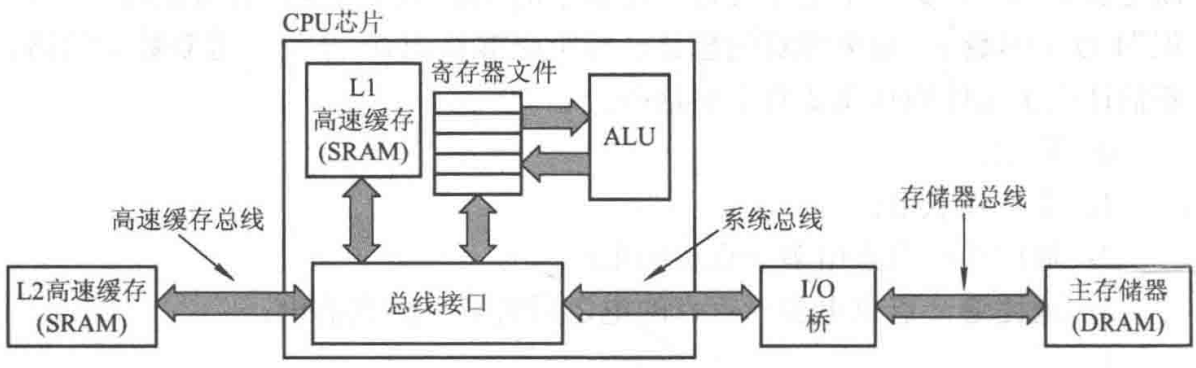

图 1-1

首先，CPU 执行运算一定需要有数据，那这个数据是怎么传给 CPU 的呢？这中间是一个很长的过程。首先我们已经知道的是，计算机内存储的数

据都是以二进制形式存储在外存(如硬盘)的，它们在被运行时会被调入到主存储器 DRAM 中(这个 DRAM 就是我们插在主板上的内存条)。

之后计算机会根据 CPU 的运行需要依次分块将这些数据通过 I/O 桥存入高速缓存 SRAM 中(这个 SRAM 是有好几个等级的，现在的一般是三级缓存。其中第三级向第二级提供数据，第二级向第一级提供数据，级别越高的缓存速度越快，但存储空间越小)，SRAM 中的内容将会按需传输给寄存器，CPU 在执行运算时直接从寄存器获取数据，运算完成后再将结果写回特定寄存器，寄存器再将结果根据用途通过总线接口传输给特定部件。

看了这么长一个步骤，你一定会问啦：为什么要搞得这么麻烦？

嗯哼，这就要提到一个速度和成本的问题了(见图 1-2)。

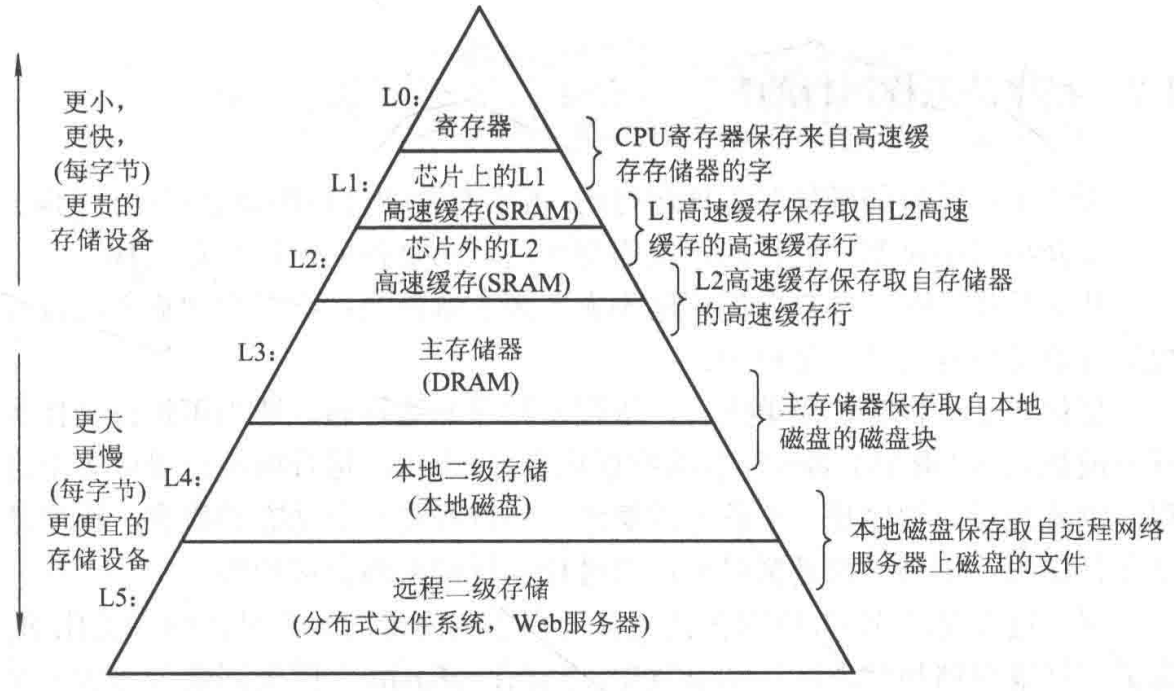

图 1-2

在图 1-2 所示的金字塔中，越接近顶端的设备速度越快，但存储空间越小，单位空间大小的制造成本越高。我们平时用的最多的是本地磁盘，即我们的硬盘。它动辄几百 GB(1 GB = 1024 MB)或者几 TB(1 TB = 1024 GB)，但是成本不过几百元。与它同价格的主存储器 DRAM 只能有几 GB 的大小，而现在的主流高速缓存则只有 1～6 MB 不等。寄存器则更远小于这个量级，仅有数十 KB(1 MB = 1024 KB)。

这样的层次性设计是为了得到效率和成本间较好的平衡，我们需要的存储空间自然是越大且越便宜越好，但这种存储空间的运行速度不够啊，所以我们就把需要运行的部分转入到速度更快但没那么大的设备里运行，这样既满足了我们需要更大存储空间的诉求，又能够保证需要运行的部分的运行速度。因此我们把最需要经常访问的数据放在速度最快容量最小的寄存器和高速缓存里，访问量最少的数据放在最慢容量最大的内存条和硬盘里，最终就形成了图 1-2 所示的金字塔层次结构。然而现在随着 CPU 运算能力的日益强大，硬盘速度成了运行速度的最大阻碍，所以现在出现了新型的速度比目前

流行的机械硬盘速度更快的固态硬盘，虽然现在固态硬盘存储空间还是略小，成本也略高于机械硬盘，但相信在不久的将来随着工艺的提升会大有替代机械硬盘的势头。

为什么要讲这些东西呢？因为我希望大家能知道以下知识：

(1) 计算机是执行输入、计算、输出的机器。

(2) 计算机内存储的一切数据其实都是二进制形式的，而且硬件也只认识二进制数据。

(3) 计算机运行设备有金字塔的层次结构，这样的层次性设计是为了得到效率和成本间较好的平衡。

那为什么要知道这些知识呢？后面你就会知道了。

1.2 程序是怎样运行的?

说完了计算机的硬件是怎样运行的，再来看看软件程序是怎样运行的吧。

要介绍程序是怎样运行的，就需要知道什么是程序，什么是进程。

什么是程序呢？最直白但可能不太严谨的解释是：存在你硬盘上的没有被运行的可执行文件就是程序。

那什么是进程呢？简单而言，进程是程序的运行态，即当可执行文件执行时被载入到 DRAM 的运行态的数据集合。一个程序运行时可以对应多个进程，也就是说，程序是一个静态的概念，而进程是一个动态的概念，程序是永久存在的，而进程则是暂时的，当进程运行结束即会被销毁。

因为进程是动态的，所以在其运行过程中它内部的数据是在时刻变化的。为了更好地存储和管理这些动态数据，进程在 DRAM 中的空间要分为十多个段，不同的段存放不同类型、状态或用途的数据。这些段中的数据要参与运算时便会被转存入高速 SRAM 缓存，最后进入寄存器以供 CPU 进行运算。当然，进程还不是程序运行的最小单位，比它小的是线程。进程在运行时是线性的，其中每一条“线”就是一个线程。这些线程共享进程的数据，帮助进程更高效地执行。(当然，其实线程还不是进程的最小组成结构，其下还有纤程，这里就不做介绍了。)

这样一来，整个程序的运行过程就出来了：

一切可执行文件，在没运行前，它叫程序。运行后，在 DRAM 中出现了属于它的进程。这些进程在 DRAM 上有一块内存空间，这段内存空间中又有很多个段，这些段存储了不同类型、状态或用途的数据。每个进程还可能会有多个线程，这些线程共享该进程的数据，帮助进程更加有效率地完成任务。当特定数据需要被执行运算时，它将会被转入速度更快的 SRAM，并根据我们上一节讲到的金字塔层次结构最终转入寄存器并被 CPU 执行运算。之后再回传给 DRAM 表示此次运算完成，最后进程执行完毕被销毁，而硬盘上的程序则依然以原状态存在。至此，程序便运行完毕。

如果图解这个过程，它大致会是这样的(见图 1-3)。

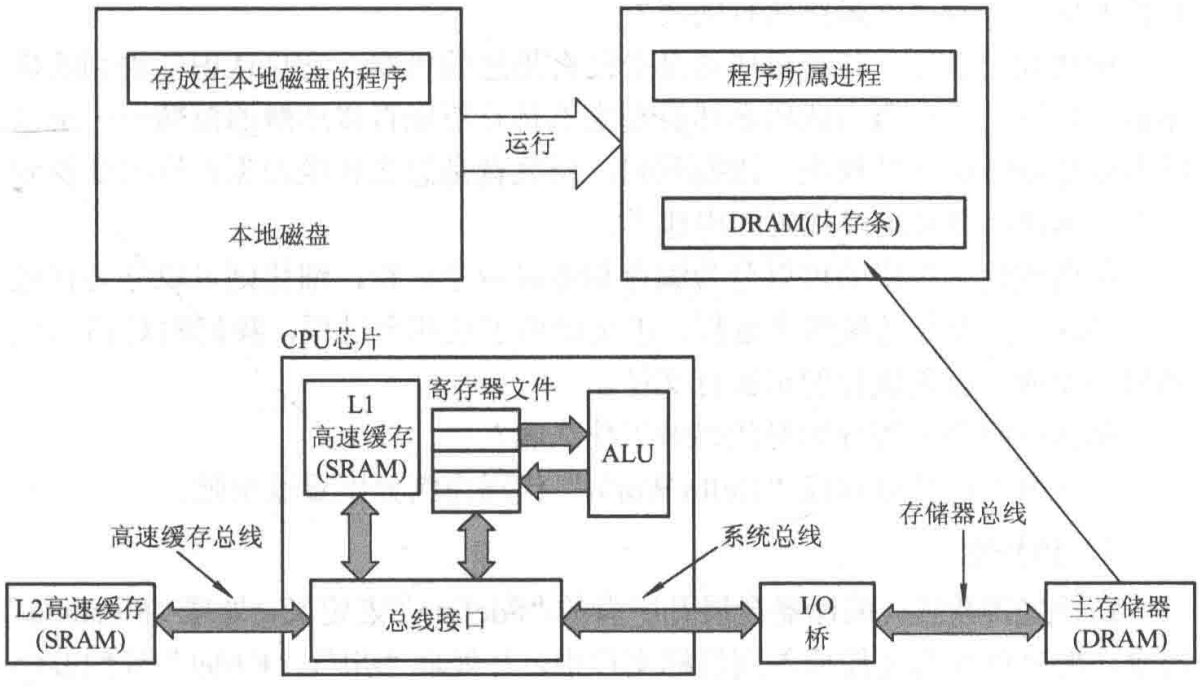

图 1-3

为什么要告诉你这些咧？因为我希望大家有这样几点认识：

(1) 程序运行态是进程。

(2) 进程是程序运行在 DRAM 上的临时存在的状态。

(3) 进程内部分为很多个段，这些段存储了不同类型、状态或用途的数据。

那为什么要知道这些知识呢？嘿嘿，和上一节一样，都与下面的一节(1.3 节)有关，接下来我们就来看看这些知识跟 C 语言到底有什么关系吧。

1.3　前面两节与 C 语言有什么关系？

在看了两节“不知所云”的内容后，相信你已经迷茫了，到底为什么要知道那些内容呢？这节就会知道了。

说到 C 语言，不得不提的就是“Hello World”了。作为一个经久不衰的标志，几乎成为了所有程序员入门某种语言后输出的第一句话。接下来就以输出“Hello World”的 C 语言代码为例，介绍一下 C 语言与前面两节软硬件运行机制的关系。

```
#include <stdio.h>

int main(void)
{
    printf("Hello World");

    return 0;
}
```

好的，我知道大家都能够写出这样一段代码，但是大家知道这段代码是怎样变成一个程序并最终执行的呢？

前面我们说过，计算机毕竟是个没有思想的机器，它只认识二进制的机器语言代码，而 C 语言代码显然看起来不是它所能直接理解的范畴……所以就需要想办法把它转换成二进制代码，而究竟是怎么转换的呢？这就要说到 C 语言编译环境对这段代码的构建了。

所谓构建，粗略的可以分为编译和链接两个过程，细化则可以分为预处理、编译、汇编和链接四个过程。正是经历了这四个过程，我们的 C 语言代码最终变成可以被执行的可执行文件。

那么这四个过程分别对代码做了什么呢？

接下来我们就以这段“Hello World”代码为例分别看过来吧。

1．预处理

在预处理阶段，编译器会展开所有的“#define”宏定义，处理“#include”指令，将要包含的文件插入到目标文件中，并处理“#if”、“#ifdef”等预编译指令，删除所有使用“//”和“/* */”注释等。对于 "Hello World" 代码而言，这一步要做的就是处理“#include <stdio.h>”指令，即删除这句话并且将 stdio.h 等要包含的文件插入到我们的代码中，这样 "Hello World" 代码就算是预处理完成了。经过此步后，"Hello World" 代码文件已经从 "Hello World.c" 变为了 "Hello World.i"～“i”即“input”之意，表示这个文件已经可以作为编译的输入文件了。

2．编译

通过上一步，我们将 C 语言代码扩展成了“完全体”形态，但它还是 C 语言代码，计算机依然不认识……这就需要我们继续进一步对其进行转化。编译就是将 C 语言代码翻译成汇编代码的过程，在这个过程中，编译器要分析代码的语法语义，并根据语法语义的意愿编写合适的汇编代码。比方说在 "Hello World" 代码中，编译器就要分析 main()函数的起始位置，并以该位置作为程序的入口点，再分析

```
printf("Hello World");
return 0;
```

分别要执行怎样的操作，并将其翻译为汇编代码。

经过编译步骤，"Hello World.i" 将会变为 "Hello World.s"。

3．汇编

编译过后的代码虽然已经不是 C 语言代码而是更低一层的汇编代码，但依然不是计算机所能直接“看懂”的内容，所以需要再来一步。汇编的目的便是将汇编代码变成完全二进制的机器语言。前面 1.1 节说过，计算机硬件只认识二进制，所以我们的程序的最终形态也一定是二进制的机器语言。这些二进制机器语言在计算机眼里就是一条条有序的指令，按照这个指令执行便可以得到我们预期的结果。

经过此步骤我们的"Hello World.s"将会变为"Hello World.o"，“o”即“output”之意，即最终编译完成的输出文件。

4．链接

链接的作用是将同一项目中的多个文件间的相互引用关系一一进行处理，保证各引用的正确性与稳定性。

经过这一步，"Hello World.o"将最终变为可执行文件，我们看到的结果就是通过这个可执行文件(.out、.exe 等)呈现出来的。

这样一来，前三节的内容就真正的连成一环了：起点是我们编写的 C 语言代码，重点是 CPU 如何执行运算。总结起来就是我们写过的 C 语言代码经过预处理、编译、汇编和链接最终变为可执行程序，当它运行时将会在 DRAM 产生进程执行程序内容，这个进程内部分为多个段，其中几个段我们将在后面章节着重介绍。之后进程不断将要执行的数据通过金字塔的多级缓存最终到达寄存器，被 CPU 取出并运算，并不断将运算结果返回给进程，直到最终进程运行结束(见图 1-4)。

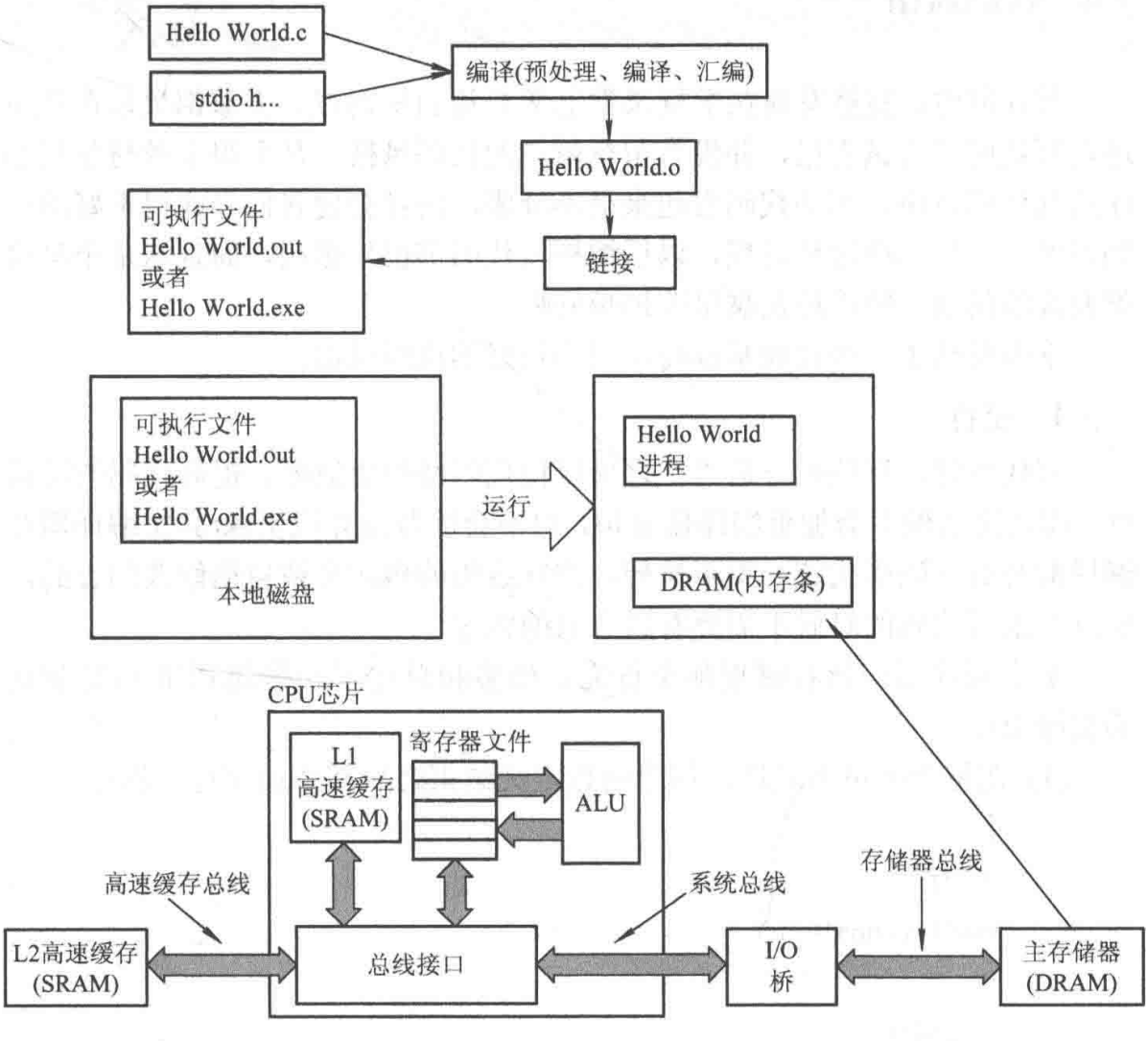

图 1-4

至此，我们成功地介绍完了 C 语言如何从代码变成程序、程序如何运行以及它运行时计算机是如何执行运算的了。