从BASIC到C

中国科学院软件研究所

引　言

本书是特为那些打算或正在学习使用C语言的BASIC程序员而编写的。学习C语言的原因是很多的，其主要目的在于提高程序的适应性和执行速度。事实上，对今天的微计算机而言，C语言可被称为最有效的语言工具。毫无疑问，很好地了解怎样使用C语言是非常有益的工作。一旦你学会了怎样用C来书写程序，你也就理解了从一个机器到另一个机器的软件移植问题。同时你也能把你的很多程序上升到适合各种微计算机上的C编译能处理的形式。

BASIC程序员似乎都感到把自己的BASIC程序转换到C程序是相当困难的事情。对此，原因很多，但普遍的原因之一是缺乏帮助BASIC程序员实现这种转换所需的指导。过去，C语言仅是系统程序员或某些少数软件设计者的工具。换句话说，C语言或多或少仅限于职业程序员来使用。现在，是到了普及C语言的时候了。另一方面，我们也必须承认BASIC语言又是在计算机程序设计范围内最普遍、最受欢迎且占主导地位的一种语言系统。

如果你知道怎样设计—BASIC程序，并能灵活地使用大多数或所有的BASIC语句、函数，那么转换到C语言上来也并非可怕和不可及的事情。在此

我们向你推荐一个优秀的指南。该指南就是"经典"的C。名为"C程序设计语言"，是由Brian W. Kernighan和Dennis M. Ritchie所著。作者不仅在C语言的发展上作出了详细的介绍，而且对BASIC程序员深感困惑的地方也都做出了必要的提示。虽然此书并非为BASIC程序员所作，也没有明确具有指导性的意义。它实际上仅是一C语言参考手册。但你一旦阅读了它，定会得到很大的启发。从此，这本书将成为你的非常珍贵的助手。

从BASIC到C一书既是一种入门的引导又是一种参考指南。主要针对那些要实现这种转换的BASIC程序员。本书中还包括了大量的BASIC程序段和等价的C语言程序段。我们衷心地希望此书能成为BASIC程序员的良师益友，能真正指导那些更习惯于BASIC，而感兴趣于C的程序员来完成这种转换。

对于特定的BASIC程序及它的环境，或准确的C语言付本，本书都尽力给以介绍。读者将快速而有效地掌握C。因为此书的后面将以BASIC环境进行讲授。故一般说来，是针对所有BASIC程序员的。作者希望这本书将来能给你从BASIC到C的转换实现提供有益于借鉴的经验。

Robert J. Traister

简　介

如果你是一个 BASIC 程序员，想使用C，但又不懂C，或没有足够的使用C的编程经验，那么这本书就是写给你的。“从 BASIC 到C”将一步一步地使你认识C的全貌。开始，先让你用C来写一些简单程序。然后，全面地理解 BASIC 语句和C函数之间的关系。再之，为了使你能更舒服而有效地使用高水平的C语言系统，我们建立了一个有足够能力的知识库。

对于微计算机来说，C已是一种最好的语言描述体系。C语言编译已能适应许多种微计算机类型，对此，约十年前 Bell 实验室就使用最初的C语言做了大量的工作。虽然C是适合各种计算机的，但我们还是要针对某一系统模式来进行讲解。这本书所涉及的系统和软件模式如下：

1. Lattice C 编译。
2. IBM PC 至少带320K 存贮器。
3. 2×360K 软磁盘驱动器。
4. IBM 单色显示器和适配器。
5. IBM 彩色图形卡。
6. 普林斯顿图形系统 HX-12 RGB 彩色监视器。
7. IBM 点矩阵打印机。

在本书中，并没有意思限定上述相同的系统方可运行程序。C是一个可移植性很强的语言，本书中提到的所有程序均采用标准的C语言版本和使用标准的C函数集。所以原则上是适合于任意系统和任意C编译的，对于所建立的C语言

运行程序，上述系统条件也并非必须的。只不过有助于本书中所涉及的输入、输出程序实例而已。同时也有助于照顾用BASIC书写的许多源代码。(主要是采用Microsoft BASIC)

在此我们鼓励读者尽量用C语言多做一些实验，先不要急于修改你的BASIC程序，等到你对C语言能完全接受了再来改动。那时相信你会很好地实现从BASIC到C的转换，从而避免一些由于不懂而带来的麻烦。

目　录

第一章　程序设计语言C

C语言是贝尔实验室开发出来的一较新语言系统。它在大多数高级语言中之所以被称为最有希望的，对开发者而言主要原因在于它的尺寸小，表达精练和面向一般目的的特征。那什么是C呢？大多数它的用户称之为汇编语言的速记版本。它的代码质量很高且执行速度非常快。对于很多程序设计语言来说C是有它独到和奇特的地方。它既不属于高级语言，象PASCAL、FORTRAN、COBOL和BASIC等，也不属于象汇编那样的低级语言。通常C被称为是一种“新型的低级语言”或“一低级语言的高级版本和高级语言的低级版本”确实，它是一种界于高级与低级语言之间的有最好描述力的中级语言。

一般来说，许多C语言程序员都熟悉汇编，而对汇编和C之间的转换是没有什么困难的。但对于那些最初接触计算机语言是FORTRAN或BASIC的人来说完成与C语言之间的转换，就显得有某些困难了。对于BASIC程序员来说尤为如此。

C是一种极为简捷的语言。其中包括的语句和函数的数量是相当少的，所以不见得比BASIC更难学。很多用高级语言编程的程序员对C语言感到难学通常来自于习惯和偏见。例如，很多精通于BASIC的程序员就带有这种偏见，而对于从BASIC到C的转换极感困难。他们接触微机是从BASIC开始的，不了解C的主要规律和它的益处，再加上固有的习惯，就成了陌生的主要原因。

事实上，如果微机程序设计从一开始就选用BASIC和C一起教授，学习过程会加快并且程序员将能很好地适应其它语言。可悲的事实是很多微机爱好者并没有学到更多的知识就进入到一种职业化应用水平，并且以BASIC而自傲。这就使得简单的再学习过程能理解其它语言知识变为困难了。

在此，我并非看不起BASIC，它确实是很有用且功能很强的语言。但它象所有的高级语言一样，在结构上都有很大的限制，以使很多人深感难以扩充。且程序函数死板。这些至少也是适合BASIC的。

首先让我们假定BASIC能做的事情C都能做。换句话说，BASIC的很多功能都可用C来直接实现，以后我们也仅以等价功能的程序模块来介绍这种互通性。下面我们用Microsoft BASIC书写的程序作为最简单的方法来演示MID$函数怎样用Microsoft BASIC的其它功能来构造。

```
10  REM BASIC PROGRAM EXAMPLE
20  REM EMULATION OF MID$ FUNCTION
30  DIM A$(5)
40  A$(1) = "H"
50  A$(2) = "E"
60  A$(3) = "L"
70  A$(4) = "L"
80  A$(5) = "O"
90  START = 2
100 ND = 5 + START - 1
110 FOR X = START TO ND
120 X$ = X$ + A$(X)
```

130 NEXT X

140 PRINT X$

此程序段中英文词“HELLO”包含在一个数组A$中。并且注意到它是放在串数组的相连串变量中。串变量在此是相同的，更准确地说，在这个例子中，数组A$中的每一个变量存放一个符号，所以整个数组与“HELLO”字相等。在Microsoft BASIC中的MID$函数可让我们把串数组中的某些元素放在一起。这样你就可以用MID$函数把一些实际的符号移出或再分配到某些变量中。从90行到130行与下式是相同的：

X$ = MID$(A$,2,3)

它的意思是把串数组A$中的第二、三、四元素值分配给X$这个串变量。还与下式等价：

X$ = A$(2) + A$(3) + A$(4)

在这段BASIC程序中，90行给数值变量START以值2。它等同于MID$函数进行分配的开始位置（即A$中的第二个符号）。数值变量ND存放着从A$中移出的符号个数值。110行开始了FOR-NEXT循环，循环变量为X，初值为START(等于2)，终值为ND。当循环变量X的值为ND + 1时，循环终止。120行使用了串变量X$，此行是循环体，简单地从A$串数组中获取元素。当循环次数出界时X$的值就等价于MID$(A$,2,3)的函数值。接着的程序行是把X$的值简单地送往屏幕予以显示。所以，90行到130行表示了一个子程序，完成从A$串数组抽取字符，且有与MID$完全等价的功能。在C中，这段等价行就叫作MID$函数。这个函数在使用中需要插入相应变量值，我们又称之为函数自变量或参数。在此，包括A$，START，和ND，它们都是

MID$函数的参数。如果在Microsoft BASIC中不能处理MID$函数，那你必须建立一个具有相同功能的子程序来替代它，当然你可以使用一些语句或同时利用其它的函数来进行构造。

C语言的工作方式是相当精巧的。它仅提供了有限的语句和函数，并且用它们你还可以建立起很多有用的其它函数。例如，用C我们可以模仿BASIC而建立出所有的BASIC命令，语句和它的函数。

就这一点上，某些人会问，“如果MID$函数是那么有用，为什么不作为一标准函数而包括在C中呢？回答是MID$函数对许多文本处理的应用是很有用的，但它对程序员来说并非所需的那样迫切。例如，假定你希望MID$返回的符号仅是字母，而跳过数字符号。在Microsoft BASIC中就不能直接使用MID$函数，你必须增加程序行才能跳出数字字符。这样额外的程序行必然会增加程序执行时间和加大程序规模。在C中，我们可以按上述标准MID$函数所描述的那样写出一段程序，并允许你对特定的任务按习惯的设计方法来建立一些函数。这样就防止了我们必须预先定义函数和对其分配所需的地址。我们可以在较小的空间下运行，使得程序十分精炼并有较高的执行速度。

C语言编译配备了一套标准C函数。虽然是有限的且数量较少，但是它们的适应面却很宽，能够构造出数百数千的其它函数。这些函数不仅结合了标准函数的功能，而且各个函数之间又是可分而独立的，用自己适当的所需参数来面向要完成的任务。这就是中级语言的关键进步。它不象汇编语言那样在完成所建的函数目的时按字位或字节来动作，C是高水平的。它允许我们做许多工作如同执行汇编语言一样有

极高的速度。所以，C很容易使用，运行效率高且不象高级语言那样有过多的限制。

许多程序员在编程过程中，最终总是把自己的注意力集中在所要完成的任务上，且有自己乐于使用的个人程序设计方法。这样用户便可以在C的标准函数基础上来定义自己的用户函数集。这些“个人函数”只需要定义书写一次并存于磁盘上，就可以被其它C语言程序调用。在此当然指的是有必要用的那些函数。打一个一般的比喻，我们可以把程序看作一个房子。汇编语言的房子是用一块块的砖、木板及条条管子等建成的。高级语言房子是用大型单元模块建成，其中包括屋顶、地基和墙等。C语言的房子介乎于这二者之间，如门框、嵌板和部份墙板等等。汇编房子给我们的感觉是建造绝对灵活；高级语言房子给我们以清晰的面目；而C语言房子既给予我们了灵活，又给我们了清晰，是提高程序各种效率的优秀语言。

C语言的历史背景

C语言和UNIX操作系统的开发应归功于新泽西州默里黑尔的贝尔实验室一程序员小组的多次比赛。这些比赛几乎仅限于他们自己，在最友好的基础上进行竞争，一次次的推翻前者而获得进步。UNIX就是在不断推陈出新的过程中产生的。这种开发过程能使各种不同的思想充分发挥，注重开发者的改进措施。最后，名叫Brian W. Kernighan和Dennis M. Ritchie成了这个组最引人注意的人物。但其它的人也是相当出色地做出了贡献：Ken Thompson在加州大学(博克利)时曾在PASCAL语言设计中做出大量的开发工作。

很多程序设计者都承认他是一位世界上最优秀的程序员，没有人否认他的天才。在离开博克利之后，便参加了 MULTICS 课题从而介入了麻省理工学院，赫尼威尔公司和贝尔实验室的冒险生涯。后来贝尔退出了此课题，但 MULTICS 最后在 GE-465 机上开发成功，并在赫尼威尔6045机上实现。MULTICS 提供给它的大量用户以共享文件和极强的管理方式。MULTICS 在 UNIX 操作系统的形成上起过突出的作用。Ken Thompson 在贝尔退出 MULTICS 项目之前还在其上写出了一有名的编辑程序，称为 QED，至今仍然具有生命力。

在贝尔退出 MULTICS 开发之后，Ken Thompson 曾利用这一段空闲时间来寻找其它的途径进一步开发，并终于把他的分时思想在 DEC PDPT 机上置于操作系统中，这就是用汇编书写的最初 UNIX 操作系统。许多系统程序员都公认它是重要和极有意义的。但不管怎么说这一事实说明 UNIX 是 MULTICS 的修改版，是沿着 MULTICS 单用户方式进行设计的。这就是第一个 UNIX 单用户版本。

当 Thompson 在这个基础上进一步开发时，他对此课题产生极大的兴趣，并着眼于多用户系统。在此开发期间Dennis Ritchie 也参加到这个课题中来。同时 UNIX操作系统已从 PDP-7转到 PDP-11上来了。（在这前后，贝尔实验室并没有重视计算机机器的开发。）

紧接着一系列的事情发生了。Ken Thompson 写了一个解释程序，叫做 B 语言，它类似于 BCPL。后来它一直作为假想的简单可移植语言使用在计算机上。结果，程序很容易地实现了从一个机器到另一台机器的移植。今天我们比较 BCPL 与 C 两语言之间仍有大量的概念类同。Thompson 非

常喜欢 BCPL 语言的简捷，易学的特征，并把这些主要特征移到B语言中。之所以称这个语言为B大概是因为它是BCPL的小而简明的子集，并且他所书写的编译要比 BCPL 编译更富于逻辑性和注重代码质量。

此时，虽然 UNIX采用汇编已写成了早期的版本并且移到 PDP-11机上，但在默尔黑尔的小组却有一种感觉，应在系统上建立一些新的与众不同的概念，明显地脱离一般的思想。基本逻辑方式是用C来重写 UNIX。并且要求这种做法是快速而高质量的。当然，当时 UNIX 仍然是单用户形式，也有一个简单记录方式的多用户 UNIX，但不很成功。对新系统的要求在逻辑上和效率上都很高了。

与此同时，具有非常强竞争力的是 BLISS语言和Dennis Ritchie。此人也是C语言的创始人。BLISS语言是一个复杂的系统，其包括有数量惊人的操作符。竞争是从 PDP-11机上选出最高质量的代码。结果大多数程序员认为C是最佳的选择。原因在于它有惊人简捷的代码，且有高度的表达能力。

从系统程序员立场来看待C是中意的语言系统。因为从源码角度看它几乎等同于汇编语言书写的程序长度。另一方面，它的目标码也接近汇编目标码，且它的高级语言特征给程序员带来方便和可靠性。而且，C允许程序员控制代码生成的质量。

当C第一次被介绍时，就以极高的水平面对复杂的操作系统，真正开始了有商业价值的用高级语言书写系统的局面。当然，这最有力的说明是用C书写出最终的 UNIX 版本，而取代了用汇编书写的旧版本。确实，有些操作系统也是用其它高级语言书写的，但是主要部份和特殊的设备驱动程序确仍用汇编来书写，这种状况一直没能在C语言之前得以改

善。而UNIX操作系统是全部用高级语言C写成的。无论是编程效率还是代码运行效率对系统程序员来说都是令人满意的，而且可以处理机器一级的特殊程序问题。

当然，所有这些描述在C上我们仅涉及了系统程序员，那么对普通的微机用户怎样呢？在此，我们首先不能再用“普通”这一词了，因为这些用户已从BASIC解释程序的全面使用扩充到使用BASIC编译或FORTRAN、PASCAL、COBOL编译。这些用户已具备了相当的编程技能。目前，个人计算机上的用户是空前大量的，虽然他们大部分不是职业的程序员，但他们会使用个人计算机，有不同程度的程序设计经验。C语言适合于他们吗？回答是只要在他们的背景基础上稍加提高即可用C来完成他们的程序。再让我们回到系统水平来看一下，现在许多有名的通用软件产品都用C语言进行了重写，新软件也多采用C语言书写，这说明了适应性。但仍然存在着问题，很多职业程序员承认在计算机爱好者中C语言的使用还是有限的，在功能上也有限制，比如大多数C编译不支持动态数组，虽然在许多情况下并不影响实现，但它必竟是一问题。当前主要的问题仍然是没有一个好的C解释系统。虽然解释系统的执行速度很慢，但对于调试一刚写好的程序是很有用的。所以在软件工业中有很多人希望看到一C语言的解释系统能运行在各种机器上。但至今这一问题没能得到高度重视。这一课题也并非难不可及的，可能是出在软件开发价格上的某些原因。所以这个语言在职业程序员中越来越受迎欢时，但它仍然对普通用户存在着一定的难度。一个真正完美的语言不应着眼于目前，而应该认真考虑将来所要发生的任何事情。

C语言会变为明天的标准程序设计语言吗？回答是多种

多样的。总之，有一些人的答案是肯定的而另外一些人则否定。我个人认为是同意前者的，至少C的用户会迅速增加。当有人问什么样的语言对程序员学习来说是最有效的时，我虽无法给出明确地回答。但C语言比其它高级语言要更周密是许多使用者的一致认识。通常任何语言都有它自己的毛病，但教授者总不希望把它的弊端和坏的程序设计习惯传给他的用户。从这一点来看，大家都知道PASCAL就是最典型的例子。对许多职业程序员来说用BASIC书写系统程序恐怕也有困难。所以，最实际的我们应该持一种批判的观点来看待计算机上的各种语言系统。既不能只停留在机器语言水平上，也不能盲目地使用高级语言。不仿你可试一试FC，它会是令你兴奋的中间级语言系统。

当一段BASIC程序在计算机上运行时，我们总想知道实际上是怎样工作的，但在BASIC系统中却没有让我们了解到机器一级究竟发生了什么。对此，许多高水平的程序员总有一种天性，当他们被几行BASIC搞糊涂时，总要追根求源来了解具体的机器码是什么样的，而不安于一种初学的水平。这样做带来的是更加完美的了解，应当提倡。

上面的阐述是给您所关心的C的一般了解。从这些信息中我们可以看到，似乎C并非某个主要公司企图产生一个极有销路的商品而开发出来的软件，以适应迅速膨胀的商业和科技市场。实质上它是出自职业程序员的消遣性产物。基本思想是结构简单，没有那么多限制并重视在机器上的最终效率，C并非某一个人的产物，而是在友好的竞争中，互相比较，淘汰而保留下来的优秀成果。实践证明它是胜人一筹且有极好的经济效益。

第二章 LATTICE C编译

本书的书写过程中所使用的C编译版本是由纽约市莱福勃德联合公司提供的。即LATTICE C编译2.0版。这个版本是在IBM-PC及兼容机上最受欢迎的一个版本。这是一个完整的C编译而不同于其它的C编译，仅是原来Kernighan提出来的C语言的子集。从这本书中你就会看到LATTICE C与Kernighan C之间只有很微小的差异，这些差异基本上可以忽略不计，而是更加简明和易于理解。这些差异对微处理机上C用戶来说只是更有益、更方便了。在这本书中，你可以找到Kernighan和Ritchie所阐述的主要C语言概念，且有大量的源程序示例。

首先我们应明确Lattice C的编译版本在此介绍的仅适应IBM-PC微计算机。你书写的源代码今后想移到标准C上是沒有问题的，但移到其它版本上恐怕难以保证正确运行。对此机器的基本要求是：两个360k双面驱动器，320k动态存贮器和标准IBM显示适配器和监视器。对于其它配置，只要高于它的兼容系统均可以。Lattice C编译所包括的磁盘文件大约是160k。这是一个两遍扫描编译系统，每一遍约50k程序部份，最小的数据部份为14k。所以这个版本可以运行在IBM-PC的老式160k磁盘驱动器上。编译整个需要运行在DOS操作系统下，不包括用戶空间，它所占用的存贮器为160k。用戶想试一下这个编译系统，就应当注意检查一下是否是符合上述基本配置。建议至少使用320k的磁盘驱动器，否则LINK EXE、EDLIN等系统文件安装在此磁

盘上，再加上你的C代码必然会造成频繁地更换磁盘或频繁地起动多驱动器，从而降低处理时间，增加出错概率。

书写程序，编译源代码和程序运行

如果你在计算机上的以往经验仅仅是BASIC解释系统，而没有更多的机会使用编译系统，那你应当明白下述事实。解释程序的工作是把一道程序解释到机器语言上。通常的方式是逐行读入某缓冲区，解释成机器码，运行这段机器码，然后再去取下一程序行。这样不断重复此过程。所以我们可以看到执行时间始终伴随着解释时间，严重地影响了程序执行效率。BASIC解释系统也不例外，是大量用户感觉它慢的主要原因，就这一点上，编译方式就完全不同了。它是在程序运行之前，做完全部编译工作。简单地说就是先产生全部的机器代码，然后再来执行，这样在执行时就不会伴随解释时间，从而大大地提高了运行效率。任何的编译版本均是这样工作的，如C、FORTRAN和编译BASIC等均如此。编译时要处理的工作很多，词法、语法、语义分析和代码生成，有的还备有各种优化过程。虽然这样的处理工作占用了很多机器时间，但却能得到运行的高质量效果也是值得的。编译后一般产生的是浮动目标码，并以独立文件形式记入磁盘。例如，如果你的源文件在磁盘上以PROGRAM.C为名，那么编译后将产生一个名叫PROGRAM.OBJ的新文件。它仍是一个不可执行文件。你必须再对它进行链接处理，因为你可能需要与其它模块、程序或一些函数相联，更重要的这一步通常是依赖机器的。链接操作后即产生真正可执行的PROGRAM.EXE文件。一旦产生了这个结果文件，你便可