第 1 章　计算机、因特网和万维网导论

学习目标：

在本章中，读者将学习以下内容：

- 计算机的基本概念。
- 不同类型的程序设计语言。
- C 语言的发展历史。
- 引入 C 标准库函数的目的。
- 对象技术的基础。
- 一个典型的 C 程序开发环境。
- 在 Windows、Linux 和 Mac OS X 上测试并运行一个 C 应用程序。
- 因特网和万维网 (WWW) 的一些基础知识。

提纲

1.1 引言

欢迎学习 C 和 C++! C 是一个简明但却功能强大的计算机程序设计语言，无论是几乎没有编程经验的技术人员，还是经验丰富的程序员，都喜欢用 C 语言来构建实际的软件系统。对于每一个想学习 C 语言的读者，本书都是一个有效的学习工具。

这本书的核心思想是通过已经证明是行之有效的方法，如 C 语言的结构化程序设计(Structured Programming)和 C++的面向对象程序设计(Object-Oriented Programming)，来强调软件工程的理念，使读者从一开始就按照正确的方式进行程序设计，并按照清晰而直接的方式来编写程序，从而降低软件开发成本。

本书提供了几百个完整的、可执行的程序，并以插图的形式显示了这些程序在计算机上运行后所显示的输出结果。我们称这种教学方法为“活代码方法”(Live-code approach)。所有这些例子程序可以从 www.deitel.com/books/chtp8 网站下载。

绝大多数人都熟悉计算机提供的令人激动的功能。学习本书后，将学会如何命令计算机去实现这些功能，因为控制计算机(常称为**硬件**)的是**软件**(即编写的驱动计算机执行**操作**和做出**决策**的指令)。

1.2 计算机硬件和软件

计算机进行计算和逻辑判断的速度比人类要快很多。很多当今的微型计算机在 1 s 内能完成几十亿次运算——这是一个人一辈子不停地计算也完成不了的。**超级计算机**(Supercomputer)的速度已经达到每秒执行几千万亿条指令。中国国防科技大学研制的天河 2 号超级计算机的速度已经超过了每秒 33 千万亿次运算(每秒 33.86 千万亿次浮点数操作)！这可以形象地比喻为，天河 2 号 1 s 完成的计算量平均分给地球上所有人的话，每个人能分到 300 万次运算。目前超级计算的“记录”还在快速地上升。

计算机在被称为**计算机程序**(Computer program)的机器指令序列的控制下对数据进行处理。程序指挥计算机完成由被称为计算机**程序员**(programmer)的人事先指定好的有序操作。

一台计算机由被称为硬件的各种物理装置组成(如键盘、显示器、鼠标、硬盘、内存、DVD 和处理器等)。得益于迅速发展的硬件和软件技术，计算成本急剧下降。几十年前，价值数百万美元、需要一个很大屋子来安放的计算机现在已经收缩到一个比指甲盖还要小的、售价仅几美元的硅片中。有趣的是，硅是地球上常见的材料之一，它是处处可见的沙子的主要成分。硅片技术的发展使得计算机便宜到已成为日常用品的程度。

1.2.1 摩尔定律

对于大多数商品或服务的价格，可能习惯于它们每年都会或多或少地上涨一些。但是在计算机和通信领域，特别是对于其中的硬件而言，事实却是相反。在过去的几十年间，硬件的价格一直在急剧下降。

每一到两年，在价格不变的情况下，计算机的性能几乎翻一番，这个重要的发展趋势被称为**摩尔定律**(Moore’s Law)。该定律是由今天计算机和嵌入式系统中微处理器的最先进的制造商——英特尔(Intel)公司的创始人之一戈登·摩尔(Gordon Moore)在 20 世纪 60 年代发现的，故以他的名字命名。摩尔定律及相关观点主要针对计算机内存和辅存(如硬盘)的容量以及处理器的速度这三者的发展变化，其中内存用于存储程序，辅存用于长时间保存程序和数据，处理器的速度就是它执行程序(即干工作)的速度。

同样的增长趋势也出现在通信行业：对通信带宽(bandwidth，即传输信息的能力)不断增长的需求而引发的激烈竞争，导致通信设备价格直线下降。除此之外，我们还没有看到其他行业出现过类似“摩尔定律”的现象。如此迅速的发展催生了所谓的“信息革命”(Information Revolution)。

1.2.2　计算机组成

无论以何种外在形式出现，计算机都可以划分为 6 个**逻辑单元**(如图 1.1 所示)。

逻辑单元	描　述
输入单元	这个“接收”单元通过**输入设备**(Input devices)获取信息(数据和计算机程序)然后将其交给计算机的其他单元处理。绝大多数的用户输入是通过键盘、触摸屏和鼠标进入计算机的，其他的输入方式有接收语音命令，扫描图像和条形码，从辅助存储器[如硬盘驱动器、DVD 驱动器、蓝光光盘(Blu-ray Disc，简称 BD)驱动器和 USB 闪存驱动器——也称为拇指盘或存储棒]读入，从网络摄像头获得视频，以及让你的计算机从因特网上接收信息(如从 YouTube 下载视频、从 Amazon 下载电子书，等等)。新的输入方式有从 GPS 设备获取位置数据，从智能手机或者游戏控制器(如微软公司面向 Xbox 的 Kinect、Wii Remote 和索尼公司的 PlayStation Move)中的加速度仪(一个能够反映向上/向下、向左/向右和向前/向后加速度的仪器)获取运动和方位信息
输出单元	这个“送货”单元取出计算机处理好的信息，然后将其放到各种**输出设备**(Output devices)上供计算机外部的用户使用。今天，绝大多数从计算机输出的信息都是显示在屏幕(含触摸屏)上、打印在纸上(“迈向绿色”组织反对这样做)、以音频或视频形式播放于微机和媒体播放器(如苹果公司的 iPod)以及体育场馆的大屏幕上，传播到因特网上或者用来控制其他机电设备，如机器人和智能家电。信息还常被输出到辅存中，如硬盘驱动器、DVD 驱动器和 USB 闪存驱动器。眼下最流行的输出手段是智能手机和游戏控制器的振动器，以及像 Oculus Rift 头盔式显示器那样的虚拟现实装置
内存单元	这个可以快速访问的、容量相对较小的“仓库”存储着经输入单元输入的、当需要处理时可迅速到位的信息。处理过的信息在被输出单元传送到输出设备之前，也保存在存储单元中。内存中的信息是**易失的**(Volatile)——一旦计算机断电，信息将会丢失。内存单元称为**存储器**(Memory)、“**主存**”(Primary memory)或者 **RAM**(Random Access Memory，随机访问存储器)。桌面计算机和笔记本电脑的主存最多包含 128 GB 的 RAM，其中 2 ~ 16 GB 最常见。GB 代表 gigabytes，1 gigabytes 约 10 亿字节，1 个**字节**(byte)有 8 个位，1 个位要么是 0 要么是 1
算术逻辑单元(ALU)	这个“加工车间”执行诸如加、减、乘、除这样的运算。它还具有决策机制，如让计算机比较两个取自内存单元的数据项以判断它们是否相等。在今天的计算机中，ALU 常被做成下一个逻辑单元 CPU 中的一部分
中央处理单元(CPU)	这个“管理员”协调和监督其他单元的工作。CPU 会告诉输入单元何时将信息读入内存单元，告诉 ALU 何时将来自内存单元中的信息用于计算，以及告诉输入单元何时将信息从内存单元传送到一个指定的输出设备。当前，很多计算机都拥有多个 CPU，可以同时执行多种操作。一个**多核处理器**(Multi-core processor)在一个集成电路芯片上实现了多个处理器——如一个双核处理器(dual-core processor)拥有两个 CPU，而一个四核处理器(quad-core processor)拥有四个 CPU。今天的桌面计算机拥有每秒执行几十亿条指令的处理器组
辅存单元	这是一个长期、大容量的“仓库”。当前没有被其他单元使用的程序和数据通常被保存在辅存设备(即硬盘驱动器)中，直到几小时、几天、几个月甚至几年后它们再一次被使用。辅存中的信息是**持久的**(Persistent)——即使计算机断电，信息依然存在。相比于主存中的信息，辅存中的信息需要更长的时间才能被访问到，但辅存的每个存储单元的价格要比主存的便宜得多。常见的辅存设备有硬盘驱动器、DVD 驱动器和 USB 闪存驱动器，有些设备的存储容量超过 2 TB(TB 代表 terabytes，1terabytes 约 1 万亿字节)。典型的桌面计算机和笔记本电脑的硬盘容量都达到 2 TB，个别桌面计算机的硬盘容量更是高达 6 TB

图 1.1　计算机的逻辑单元

1.3　数据的层次结构

计算机处理的数据构成了一个数据层次结构(data hierarchy)，这个结构正在变得越来越大，越来越复杂。底层是最简单的数据项(所谓的“位”)，高层有字符和域。图 1.2 描述了数据层次结构的一部分。

位(Bit)

计算机处理的最小数据项是数值 0 或 1。它称为 1 位(bit)，bit 是 binary digit 的缩写——一个二进制数字(binary digit)。显然，计算机最核心的功能就是对 0 和 1 进行最基本的操作——检测 1 个位的值、设置 1 个位的值和翻转 1 个位的值(将 1 改为 0 或将 0 改为 1)。

字符(Character)

与以“位”这样底层形式表示的数据打交道是很令人烦恼的。相反，人们更愿意处理十进制数(0~9)、字母(A ~ Z 和 a ~ z)和特殊符号(如$、@、%、&、*、(、)、–、+、 " 、:、？ 和/)。数字、字母和特殊符号被称为**字符**(Character)。计算机的**字符集**(Character set)就是可以用来编写程序和表示数据项的全体字符的集合。由于计算机只能处理 1 或 0，所以一个计算机的字符集就是用 0 和 1 组成的码点(pattern)来表示一个字符，不同的码点表示不同的字符。C 语言支持多种字符集(包括 **Unicode**)，这些字符集由用 1 个、2 个或者 4 个字节(8 位、16 位或者 32 位)表示的字符组成。Unicode 包含世界上大多数语言的字符。ASCII(American Standard Code for Information Interchange)字符集是 Unicode 的最常见的子集，能够表示大/小写字母、数字和常用的特殊符号，想了解 ASCII 字符集的更多信息请参阅附录 B。

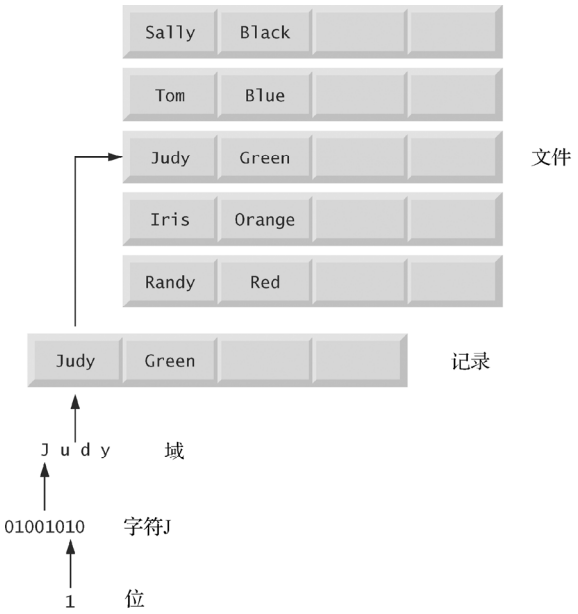

图 1.2 数据的层次结构

域(Field)

就像字符是由位组成一样，域由字符或字节组成。一个域表达某种含义的一组字符或字节。例如，一个由大/小写字母组成的域可以用来表示人名，一个由十进制数字组成的域可用来表示一个人的年龄。

记录(Record)

若干个相关联的域可以组成一个**记录**。例如在一个工资系统中，一个员工的记录可能由如下的域组成(每个域的数据类型标注在括号中)：

- 员工编号(一个整数)
- 姓名(一个字符串)
- 住址(一个字符串)
- 每小时工资(一个带小数点的数)
- 年初至今的收入(一个带小数点的数)
- 已缴税款(一个带小数点的数)

可见，一个记录就是一组相关联的域。在上面这个例子中，所有的域都属于同一个员工。一个公司可能有很多员工，每个员工有一个工资记录。

文件(File)

一个**文件**是一组相关联的记录［注：广义上说，一个文件可以包含任意格式的任意数据。在某些操作系统中，文件被看成是一个字节序列——文件中字节的组织结构，就像将数据组织成记录一样，是由应用程序员创建的视图来决定的］。文件的组织结构一般要支持文件包含几十亿甚至几万亿个字符的信息。

数据库(Database)

一个**数据库**是为了便于访问和操纵而组织在一起的数据集合。最流行的组织模型是关系数据库(Relational database)，在该模型中数据是存储在简单的表(Table)里。一个表由记录和域组成。例如，一个学生表包含了姓、名、专业、年级、学生编号(ID)和年级平均成绩等域。每个学生的数据就是一个记录，每个记录中独立的信息片段就是域。可以针对数据在多个表或数据库中的关系，对数据进行查找、排序等操作。例如，一所大学就需要将学生数据库中的数据与课程、校内宿舍和配餐等数据库中的数据联合在一起使用。

大数据(Big Data)

世界范围内产生的数据总量是庞大且快速增长的。根据 IBM 公司的报告，目前每天会有将近 2.5 百万万亿(2.5 EB)数据出现而且全球 90%的数据是在近两年内产生的！IDC 公司的研究表明，到 2020 年全球每年的数据供给量将达到 40 ZB(相当于 40 万亿 GB)。图 1.3 展示了一些常用的字节单位。**大数据**应用就是处理海量数据，目前这个领域发展很快，为软件开发人员创造大量的机会。美国加特纳集团(Garter group)的研究表明，到 2015 年全球有超过 4 百万个 IT 工作岗位将投入到大数据领域。

单　位	字　节	近 似 值
1kilobyte(KB)	1024 bytes	10^3（准确地是 1024 字节）
1megabyte(MB)	1024 kilobytes	10^6（1 000 000 字节）
1gigabyte(GB)	1024 megabytes	10^9（1 000 000 000 字节）
1terabyte(TB)	1024 gigabytes	10^{12}（1 000 000 000 000 字节）
1petabyte(PB)	1024 terabytes	10^{15}（1 000 000 000 000 000 字节）
1exabyte(EB)	1024 petabytes	10^{18}（1 000 000 000 000 000 000 字节）
1zettabyte(ZB)	1024 exabytes	10^{21}（1 000 000 000 000 000 000 000 字节）

图 1.3　字节单位

1.4　机器语言、汇编语言和高级语言

程序员可以用多种程序设计语言编写计算机指令，这些语言有的能被计算机直接理解，有的需要经过**翻译**(Translation)这个中间步骤才能被计算机理解。如今还在使用的计算机语言有几百种。这些语言可以分成三类：

1．机器语言
2．汇编语言
3．高级语言

机器语言(Machine Language)

一台计算机能够直接理解的仅仅是它自己的**机器语言**，该语言是由计算机的硬件设计所定义的。机器语言是由数字(最终简化为 1 或 0)串组成的，它指挥计算机在一个时刻执行一个最基本的操作。机器语言是**机器相关的**(即一个特定的机器语言只能用于某一类计算机)。机器语言对人而言如同“天书”。例如，加班工资加上基本工资再计算总工资的早期机器语言程序段如下：

```
+1300042774
+1400593419
+1200274027
```

汇编语言和汇编程序(Assembly Language and Assembler)

对绝大多数程序员而言，编写机器语言程序既费时又费力。所以程序员尝试着用英语风格的缩写词来表示计算机的基本操作用以代替计算机能够直接理解的数字串。这些缩写词就构成了**汇编语言**的基础。同时，被称为**汇编程序**的能够将汇编语言源程序按照计算机的速度转化成机器语言程序的**翻译程序**(Translator program)也被开发出来。例如，下面这段汇编语言源程序就是将加班工资加上基本工资再计算总工资：

```
load    basepay
add     overpay
store   grosspay
```

虽然这样的程序对人来说清楚多了，但计算机还是无法理解，除非它们被翻译成机器语言。

高级语言和编译器(High-level Language and Compiler)

借助于汇编语言，计算机的应用得到了迅速的发展。但是完成一个即便是最简单的任务，程序员仍然需要编写很多条汇编指令。因此，为了加快程序开发的速度，**高级语言**就应运而生了。在高级语言计算机程序中，完成好几个任务，可能只需要编写一条语句。被称为**编译器**的翻译程序负责将高级语言源程序翻译成机器语言。使用高级语言时，程序员可以用看上去非常类似日常英语并包含有常用的数学记号的语句来表示指令。例如，采用高级语言编写的工资程序可能就只用下面这样一条语句：

```
grossPay = basePay + overTimePay
```

从程序员的观点看，高级语言要比机器语言和汇编语言更受欢迎。C 是目前应用最广泛的高级语言之一。

解释器(Interpreter)

将一个大型的高级语言源程序编译为机器语言需要花费相当长的计算机时间。为此，一种称为**解释器**的翻译程序被研发出来，它直接执行高级语言源程序，避免了编译的延迟。不过与执行编译好的程序相比，通过解释器来执行高级语言源程序要慢得多。

1.5 C 程序设计语言

C 语言是在 BCPL 语言和 B 语言的基础上发展起来的。BCPL 是 1967 年由 Martin Richards 为开发操作系统和编译器而设计的。参照 BCPL 中的很多功能，Ken Thompson 提出了功能更强的 B 语言，并于 1970 年在贝尔实验室用 B 语言开发出 UINX 操作系统的早期版本。

贝尔实验室的 Dennis Ritchie 在 B 语言的基础上提出了 C 语言并于 1972 年首次实现。C 语言最早是作为 UINX 操作系统的开发语言而闻名的。今天，很多先进的操作系统都采用 C 或 C++语言来开发。目前 C 语言已被应用于绝大多数计算机。C 语言几乎不与具体的计算机硬件相关——只要精心设计，就可以编写出在绝大多数计算机上**可移植的**(Portable) C 程序。

为高性能而生

C 语言广泛应用于开发对性能要求较高的系统，如操作系统、嵌入式系统、实时系统和通信系统(参见图 1.4)。

20 世纪 70 年代后期，C 已经发展为目前所谓的“传统 C”。随着 Kernighan 和 Ritchie 编写的《C 程序设计语言》于 1978 年的出版，使 C 语言得到了更多的关注。该书也成为迄今为止最成功的计算机科学丛书之一。

标准化

C 语言在各种类型的计算机［也称为各种**硬件平台**(Hardware platform)］上的迅速扩张。导致出现了相似但却常常相互不兼容的很多种 C 语言版本。这对想编写能够在多个平台上运行的代码的程序员来说是一个很严重的问题。因此，推出 C 语言标准版本的呼声日益强烈。1983 年，美国国家标准委员会(American National Standards Committee, ANSC)下属的计算机与信息处理部(X3)成立了“X3J11 技术委

员会”,专门负责“提出一个无二义性的与机器无关的 C 语言定义”。1989 年,**美国国家标准学会**(American National Standards Institute, ANSI)推出了编号为 ANSI X3.159-1989 的“标准 C”，并通过**国际标准组织**(International Standards Organization, ISO)在世界范围内推广。1999 年，这个标准被更新为“INCITS/ISO/IEC9899-1999”，简称“C99”。该标准可以向美国国家标准委员会(www.ansi.org)订购，订购网址为：webstore.ansi.org/ansidocstore。

应用领域	描　述
操作系统	C 语言的可移植性和高性能使得它非常适合于开发操作系统，例如 Linux、微软的 Windows 的一部分以及谷歌公司的 Android。苹果公司的 OS X 是用由 C 衍生出来的 Objective-C 开发的。我们将在本书的 1.11 节介绍一些主流的台式计算机和笔记本电脑的操作系统以及若干移动终端操作系统
嵌入式系统	每年生产的微处理器绝大多数都嵌入到各种设备而不是通用计算机中。这些嵌入式系统包括导航系统、智能家电、家用安防系统、智能手机、平板电脑、机器人、智能交通信号系统。C 语言是嵌入式系统开发领域最流行的程序设计语言之一，这个领域要求程序运行越快越好，同时还要少占用内存。例如，当出现异常情况时，小汽车的防抱死刹车系统必须能够立即响应，在不打滑的情况下将汽车的速度降下来或者将汽车停下来。面向视频游戏的游戏控制器必须消除控制器与游戏中动作的时间延迟，同时确保动画画面的平滑切换
实时系统	实时系统常用于“任务攸关”的应用领域，这类应用要求可预测的几乎瞬间的响应时间。实时系统还要求能够持续地工作——例如一个飞行流量控制系统必须持续地监测飞机的位置和速度，并且不带任何延迟地向飞行流量控制人员报告这些信息。这样才能在可能发生碰撞时向飞机报警，使其改变航向
通信系统	通信系统需要快速地为海量数据确定通向它们目的地的路径，以确保能够平滑地无延地传输诸如音频和视频这样的信息

图 1.4　常见的面向性能的 C 语言应用领域

C11 标准

最新的 C 标准是在 2011 年推出的，故称为 C11。C11 对“标准 C”的功能进行精炼和扩展。在先进的 C 编译器中实现了很多新的功能，我们将在本书正文和附录 E(可关注也可忽略)中逐步介绍。

可移植性提示 1.1

因为 C 语言是与硬件无关的、被广泛配备的程序设计语言，所以用 C 语言编写的应用程序，无须修改或只需少许修改，就可以运行在很多不同的计算机系统上。

1.6　C 标准库

在本书的第 5 章将会了解到，C 程序是由被称为“**函数**”(Function)的程序模块组成。我们可以编写自己所需要的所有函数来构成一个程序，但是大多数程序员更愿意借用“**C 标准库**”(C Standard Library)中提供的大量现成的库函数。因此，学习 C 语言编程可以分成两部分——学习 C 语言本身与学习使用 C 标准库。本书将介绍 C 标准库中的大多数函数。若想深入地了解这些库函数，读者一定要参阅 P. J. Plauger 编写的《标准 C 库》(The Standard C Library)。该书介绍这些函数是如何实现的、怎样用它们编写出可移植的代码。在本书中，我们将使用并解释大多数的 C 标准库函数。

本书鼓励采用“**搭积木方法**”(Building-block approach)来开发程序。利用现成程序模块，而不是“重新发明轮子(即重复劳动)”——这就是“**软件重用**”(Software reuse)。在用 C 编程时，通常可以使用下列“积木”：

- C 标准库函数
- 程序员自己开发好的函数
- 别人(信任的人)开发好且可以获得的函数

创建自己函数的优点是可以清楚地知道这些函数是如何工作的，可以检查 C 代码。缺点是新函数的设计、开发、排错及性能优化都是花费时间的工作。

性能提示 1.1

使用标准 C 函数库中的函数，而非自己编写函数，能够改进程序的性能，因为这些函数是为高效执行而精心编写的。

可移植性提示 1.2

使用标准 C 函数库中的函数，而非自己编写函数，能够改进程序的可移植性，因为这些函数可以用在几乎所有的标准 C 的实现中。

1.7 C++和其他基于 C 的程序设计语言

C++是由 Bjarne Stroustrup 在贝尔实验室研发的。基于 C 语言，它还提供了使 C 语言显得更加“整洁漂亮”的许多功能。更重要的是，它提供“**面向对象程序设计**”(Object-oriented programming)的能力。“**对象**”(Object)是自然界事物的模型，是基本的可重用的软件**构件**(Component)。通过使用模块，面向对象的设计与实现方法能够大幅度地提高软件开发小组的生产效率。

本书的第 15 章至第 23 章中介绍 C++语言，这部分内容从我们的另外一本教材《C++大学教程(第九版)》中提炼出来的。图 1.5 介绍了其他一些流行的基于 C 的程序设计语言。

程序设计语言	描　　述
Objective-C	Objective-C 是一个基于 C 的程序设计语言。它是在 20 世纪 80 年代早期研发出来的，后来被 NeXT 公司收购，进而又被苹果公司收购。它已经成为 OS X 操作系统和所有 iOS 驱动的设备(如 iPod、iPhone 和 iPad)的主要的程序设计语言
Java	Sun 微系统公司在 1991 年资助了一个内部的公司研究项目，该项目的成果是产生了称为 Java 的基于 C++的面向对象程序设计语言。Java 的主要目标是使其编写的程序能够运行在一个很大范围的计算机系统和计算机控制的设备上。这有时称为“写一次，处处运行。” Java 被用于开发大规模的企业应用程序，加强 Web 服务器(提供我们在 Web 浏览器上所看到内容的计算机)的功能，为消费者的设备(如智能手机、电视机顶盒等)提供应用等目的。Java 还是开发 Android 系统上应用程序(app)的语言
C#	微软公司的三个核心的面向对象程序设计语言是 Visual Basic(基于原先的 BASIC 语言)、Visual C++(基于 C++语言)以及 Visual C#(基于 C++语言和 Java 语言，为将因特网和 Web 集成到计算机应用程序中而研发的)。还有一些非微软公司的 C#版本
PHP	作为面向对象的、开源的脚本语言，PHP 受到用户和开发者社区的支持，已经在几百万个网站上得到应用。PHP 是平台无关的——可以在所有主流的 UNIX、Linux、Mac 和 Windows 操作系统上实现。PHP 还支持包括流行的开源数据库 MySQL 在内的很多数据库
Python	作为又一个面向对象的脚本语言，Python 是在 1991 年公开发布的。Python 是由阿姆斯特丹的国家数学与计算机科学研究所(CWI)的 Guido van Rossum 研发的，主要借鉴了 Modula 3 语言——这是一个系统编程语言。Python 是可扩展的——它可以通过类和编程接口来扩展
JavaScript	JavaScript 是应用最广泛的脚本语言。它主要用于为网页增加动态的行为——例如，动画和改善与用户的互动性。所有主流的 Web 浏览器都提供该语言
Swift	Swift 是苹果公司为开发 iOS 和 Mac 上应用程序(app)而研发的新的程序设计语言，在 2014 年 6 月的世界开发者大会(World Wide Developer Conference, WWDC)上发布。尽管 app 仍然可以用 Objective-C 来开发和维护，Swift 已经是苹果公司未来 app 开发的语言。通过在某种程度上消除 Objective-C 的复杂性，它是一个现代化的语言，易于被初学者和从诸如 Java、C#、C++和 C 这样的高级语言转移过来的程序员掌握。Swift 强调性能和安全性，可以访问 iOS 和 Mac 提供的全部编程功能

图 1.5　流行的基于 C 的程序设计语言

1.8 对象技术

本节是专为有意学习本书后半部分内容 C++的读者提供的。在对新的功能更强的软件需求不断高涨的时代，快速、正确而且低成本地开发软件依然是一个难以实现的目标。

对象(Object)，更准确地说是产生对象的类(Classes)，本质上是可重用的软件模块。常见的有数据对象、时间对象、音频对象、视频对象、汽车对象、人员对象，等等。几乎任何一个名词都可以用属性(如姓名、颜色和大小)和行为(如计算、移动和通信)表示成一个软件对象。

软件开发人员发现，相比采用早期的软件开发技术，采用一个模块化的、面向对象的设计与实现方法可以大幅度地提高软件开发团队的工作效率——因为面向对象的程序更容易理解、更容易纠错和修改。

1.8.1　对象——以汽车为例

为了帮助读者更好地理解对象及其内涵，我们打一个简单的比方。假设你想驾驶一辆汽车并通过不断地踩油门踏板来使汽车越开越快。在实现愿望前会发生什么事情呢？首先，在你能够驾驶汽车前，必须有人去设计这辆汽车。一辆汽车通常是从一组工程图纸开始实现的，就像一个房子是从描绘它的蓝图开始实现的一样。这组工程图纸中包含有油门踏板的设计图。这个踏板向驾驶员隐藏了能够使汽车越开越快的复杂机理，就像刹车踏板隐藏了能够使汽车减速的机理，方向盘隐藏了能够使汽车转向的机理一样。这使得那些不知道发动机、刹车和转向机构是如何工作的人也可以轻松地驾驶汽车。

就像你不能在图纸上的厨房里做饭一样，你不可能驾驶一辆汽车的工程图样。在一辆汽车可以被你驾驶之前，必须根据它的工程图样把它建造出来。一辆完整的汽车必须有一个用于加速的油门踏板，但这还是不够的——汽车不能自己加速（但愿将来能做到），所以驾驶员必须用力踩下油门踏板才能使汽车跑得更快。

1.8.2　方法与类

让我们继续用汽车的例子来介绍一些面向对象程序设计的关键概念。程序中执行一个任务需要一个**方法**(Method)。方法的内部是执行该任务的程序语句。但方法向用户隐藏了这些语句，就像汽车的油门踏板向驾驶员隐藏了使汽车越开越快的机理一样。在面向对象的程序语言中，要创建一个称为**类**(Class)的程序单元，其内部是执行该类所有任务的一组方法。例如，一个表示银行账号的类就包含有向这个账号存钱的方法，从这个账号取钱的方法，查询这个账号当前余额的方法。在概念上，一个类等同于一张包含有油门踏板、方向盘等汽车部件的工程图纸。

1.8.3　实例化

就像在实际驾驶汽车前必须有人将它从工程图纸中建造出来一样，在程序执行某个类的方法定义的任务前，必须构建出这个类的一个对象。这个工作过程被称为实例化(Instantiation)。对象也就称为相应类的一个**实例**(Instance)。

1.8.4　软件重用

就像一个汽车的工程图纸可以反复使用来建造很多辆汽车一样，也可以多次重用一个类来构建多个对象。在构造新的类或程序时重用现成的类可以节省时间和工作量。由于现成的类或程序模块常常经历过大量的测试、纠错和性能优化，所以重用还有助于构建更可靠和更高效的系统。就像“可互换的零件”这个概念对工业革命是至关重要的一样，可重用的类对软件革命也是至关重要的，而这场软件革命正是被对象技术所驱动的。

软件工程视点 1.1

用搭积木的方法来开发程序。避免重新发明轮子——只要有可能就采用现成的高质量的部件。面向对象程序设计的好处之一就是这样的软件重用。

1.8.5　消息与方法调用

驾驶汽车时，脚踩油门就是向汽车发出一个消息(Message)，让它去执行一个任务——跑得更快些。同样，也需要向对象发出消息。每个消息被转换成所谓的**方法调用**(Method call)，通知相应对象的方法去执行它的任务。例如，一个程序调用特定银行账号对象的存款方法来增加该账号的资金余额。

1.8.6　属性与实例变量

除了能够完成相应的任务外，一辆汽车还会具有一些属性，如它的颜色，有几个车门，油箱的油量，当前速度以及行驶里程记录(即里程表读数)等。与汽车的功能一样，汽车的属性也会作为设计方案的一部分表示在工程图纸中(如包括一个里程表和一个油量表)。在驾驶汽车的过程中，这些属性与汽车是如影随形的。每辆汽车都在维护它的属性。例如，一辆汽车总是知道它自己油箱中还有多少汽油，但对其他车辆油箱中的汽油存量一无所知。

类似地，一个对象在被某个程序使用时，它的属性也是与它随影同行的。这些属性也被定义为相应类的对

象的一部分。例如，一个银行账号对象具有一个表示账号资金总额的余额属性。每个银行账号对象都知道它所表示账号的余额，但却对银行中其他账号的余额一无所知。属性被指定为类的**实例变量**(Instance variable)。

1.8.7 封装与信息隐藏

类(及其对象)封装(Encapsulate，也称为包装)它们的属性和方法。一个类(及其对象)的属性和方法是密切关联的。对象之间可以相互通信，但是它们通常不允许知道其他对象是如何实现的——也就是说，实现细节在对象之间是隐藏的。今后我们将看到，这种**信息隐藏**(Information hiding)对好的软件工程是至关重要的。

1.8.8 继承

通过**继承**(Inheritance)，可以很方便地创建一个新的对象的类——这个新类被称为**子类**(Subclass)，子类从一开始就具有一个现成类(被称为**父类**，Superclass)的特性，当然这些特性可能被定制或增加一些它自己独特的性质。在我们的汽车类别中，“敞篷车”类的对象肯定是更广义的“车辆”类的一个对象，只是较特殊的是，它的车顶是可以收起来或者降低的。

1.9 典型的 C 程序开发环境

C 系统通常包含三部分：程序开发环境、C 语言和 C 标准库。下面详细介绍如图 1.6 所示的典型的 C 程序开发环境。

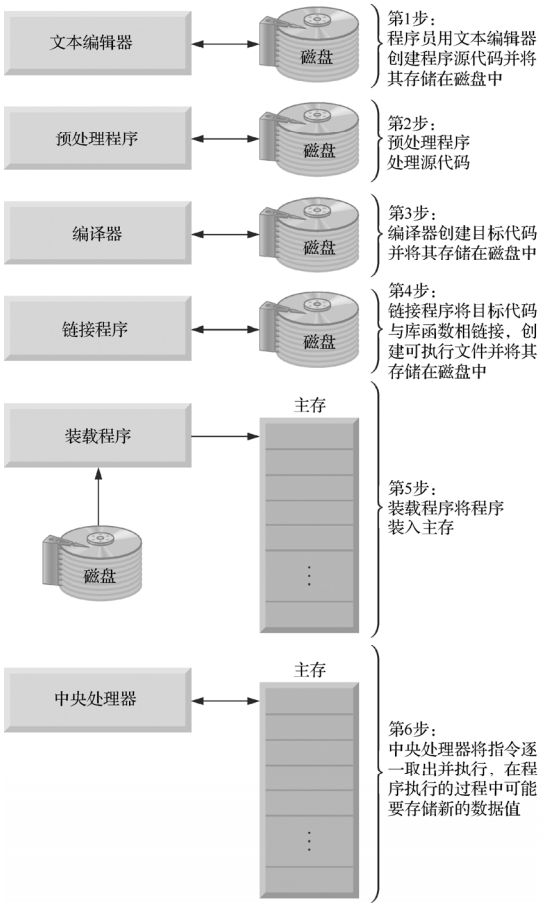

图 1.6 典型的 C 程序开发环境

C 程序通常要经过 6 个处理步骤才可以执行(参见图 1.6)。这些阶段依次是：**编辑**(Edit)、**预处理**(Preprocess)、**编译**(Compile)、**链接**(Link)、**加载**(Load)和**执行**(Execute)。尽管本书是一本通用的 C 语言教科书(其内容不依赖于任何一个具体的操作系统的细节)，但是在本节里我们将重点讲述典型的基于 Linux 操作系统的 C 语言系统(注意：本书的所有程序基本上不加修改就可以运行于绝大多数 C 语言系统，包括基于微软 Windows 操作系统的 C 语言系统)。如果目前使用的不是 Linux 系统，请查阅所使用系统的使用手册或向老师询问如何在你的系统上完成这些开发步骤，还可以在我们位于 www.deitel.com/C 的 C 资源中心中找到“入门指导”(getting started)去查阅常用 C 编译器及开发环境的辅导材料。

1.9.1　第 1 步：创建一个程序

第 1 步就是编辑一个文件，这要用一个**文本编辑器**(Editor program)来完成。Linux 系统中两个广泛使用的文本编辑器是 vi 和 emacs。诸如 Eclipse 和 Microsoft Visual Studio 这样的针对 C/C++集成开发环境的软件包，通常也有集成在开发环境中的文本编辑器。你的工作就是在文本编辑器中键入程序源代码，然后检查是否存在错误。若有错，则改正以确保程序正确。最后将程序源文件存入诸如硬盘这样的辅存中。C 源程序的文件名必须以扩展名“.c”结束。

1.9.2　第 2 步和第 3 步：预处理及编译一个 C 程序

在第 2 步，需要发出**编译**(Compile)程序的命令。编译器将 C 程序翻译成机器语言代码(也称为**目标代码**，Object code)。在 C 语言系统中，有一个**预处理程序**(Preprocessor)，它会在编译工作开始前自动执行。**C 预处理程序**(C preprocessor)执行所谓的**预处理命令**(Preprocessor directive)，这些命令指出在程序被编译之前应对程序进行某些特定的处理。这些处理通常有将其他文件包含进来一起编译以及进行各种文本替换。随后几章将会介绍最常用的预处理命令，而对预处理程序功能的详细讨论将会出现在第 13 章。

在第 3 步，编译器将 C 语言源程序翻译成机器语言代码。当编译器因某个程序语句违反了语言规则而无法识别该语句时，称发生了一个**语法错误**(Syntax error)。这时编译器会发出一个出错信息帮助定位并修改这个错误语句。C 标准并没有规定编译器发出的出错信息的用词，所以在系统上看到的出错信息可能会与其他人在其他系统上看到的不同。语法错误也称为**编译错误**(Compile error)或**编译时错误**(Compile-time error)。

1.9.3　第 4 步：链接

随后的步骤被称为**链接**(Linking)。C 程序通常包含有对在其他地方定义的函数的调用，例如在标准函数库或同一个软件项目组中其他成员编写的私人函数库中定义的函数。为此，C 编译器产生的目标代码中会为这些暂缺的部分留出“空隙”。**链接程序**(Linker)的任务就是要将目标代码与这些暂缺的部分链接成一个(无任何空隙的)**可执行映像**(Executable image)。在典型的 Linux 系统中，编译和链接一个程序的命令是 **gcc**(GNU C 编译器)。例如编译和链接一个名为“welcome.c”的程序，需要在 Linux 的提示符下键入：

```
gcc welcome.c
```

然后按“回车”键(或“返回”键)(注意：Linux 命令对字母的大/小写是敏感的，一定要确保 gcc 是小写字母以及文件名字母的大小写正确)。如果源程序编译和链接正确，系统将生成一个(默认的)名为“a.out”的文件，这就是 welcome.c 程序的可执行映像。

1.9.4　第 5 步：装载

下一步骤就是**装载**(Loading)。因为在一个程序能够被执行之前，该程序首先要被放置到内存中。

这项工作是由**装载程序**(Loader)完成的。装载程序将欲执行程序的可执行映像从硬盘中取出并传送到内存中。支持该程序执行的来自共享函数库的附加模块也要同时装入内存。

1.9.5 第 6 步：执行

最后，计算机在其 CPU 的控制下，逐条**执行**(Execute)程序中的机器指令。Linux 系统中欲装载并执行上述程序只需在 Linux 的提示符下键入“./a.out”然后按回车键。

1.9.6 程序运行时可能会出现的问题

在第一次操作时，程序并不是总能正确工作的。上面介绍的每个步骤都会因各种各样的错误而无法通过。下面我们就讨论一下可能会发生的错误。例如，一个正在运行的程序试图把零作为除数(就像在算术系统中一样，这在计算机上是一个非法操作)。这将引发计算机显示一条出错信息。只能再次回到编辑阶段，对程序做必要的修改，并再次重复余下的步骤，以判定所做的修改是否正确。

常见的编程错误 1.1

诸如“除数为零”这样的错误是在程序执行时出现的，所以它们被称为运行时错误(Runtime error)或执行时错误(Execution-time error)。“除数为零”属于一个严重的错误(Fatal error)，这样的错误将导致程序在尚未完成其工作的情况下被立即终止。与此相对，出现非严重的错误(Nonfatal error)时，程序仍然能继续运行直至结束，只是得到的结果往往是错误的。

1.9.7 标准输入、标准输出和标准错误流

绝大多数 C 程序都要输入和/或输出数据。特定的 C 函数从**标准输入流 stdin**(standard input stream)中获取输入信息。stdin 通常是键盘，不过也可以将 stdin 重定向到其他流上。数据一般是输出到**标准输出流 stdout**(standard output stream)上。stdout 通常是计算机屏幕，不过也可以将 stdout 重定向到其他流上。当我们说程序要打印一个结果时，实质上是指将结果显示在屏幕上。当然，数据也可以输出到诸如硬盘或打印机这样的设备上。还有一个就是被称为 **stderr** 的**标准错误流**(standard error stream)。流 stderr(通常与屏幕相连)用来显示错误信息。常规的数据输出，即 stdout，通常可被定向到除屏幕外的某个设备上，而让 stderr 始终与屏幕相连，这样用户就可以立即获悉错误信息。

1.10 在 Windows、Linux 和 Mac OS X 上测试并运行一个 C 应用程序

本节中，将运行第一个 C 应用程序并与该程序进行互动。将从运行一个猜数游戏开始，该游戏首先在 1 ~ 1000 之间随机地取一个数，然后邀请用户猜这个数。如果猜对了，游戏就结束了，如果猜错了，游戏就提示猜的数比正确结果高或者低，然后请你继续猜。游戏本身并不对你可以猜测的次数设置限制，但是肯定能够在不超过 10 次的猜测中猜对这个范围内的任何一个数。在这个游戏中蕴含有某些计算机科学的奥妙——在“6.10 节数组查找”中将学会“二分查找”(binary search)技术。

为了说明“测试驱动”(test-drive)，我们把本书第 5 章要求在练习中完成的这个游戏做一点修改。正常情况下，这个游戏是随机地选一个数作为正确答案。修改后的游戏是每次执行这个游戏时正确答案始终是同一个数(当然由于编译器不同，这个数也可能不同)。这样就可以玩我们在本节中使用的猜数游戏并看到同样的结果。

我们将依次在 Windows 命令提示符(Command Prompt)、Linux 的命令解释程序(Shell)和 Mac OS X 的“终端”(Terminal)窗口下演示这个 C 应用程序的运行。应用程序在这三个平台上的运行结果是一样的。在平台上执行完测试后，还可以尝试运行这个游戏程序的“随机版”，名为 randomizied_version 的“随机版”与该游戏的“测试运行版”存储在我们提供的同一个文件夹里。

可以用来编译、创建和运行一个 C 应用程序的开发环境是很多的，例如 GNU C、Dev C++、微软

的 Visual C++、CodeLite、NetBeans、Eclipse、Xcode 等。请向教师咨询你所使用的开发环境的信息。绝大多数 C++开发环境都可以同时编译 C 和 C++程序。

在后续学习步骤中，将会运行这个应用程序并输入不同的数来猜那个正确答案。在此过程中，你看到的元素和功能都是在本书中将要学习的典型内容。我们将通过字体来区分你在屏幕上看到的内容[如 Command Prompt(命令提示符)]和不直接与屏幕关联的元素。用“半黑体的 sans-serif Helvetica”字体来突出诸如标题和选单[如“文件”(File)选单]的屏幕内容，用“sans-serif Lucida”字体来突出文件名、程序显示的文本以及键入的数值(如 GuessNumber 或 500)。另外，你可能已经注意到了，每个关键词条的出现被设置成粗体。

本节中，对于 Windows 平台下的测试运行，将修改“Command Prompt”(命令提示符)窗口的背景颜色，使得该窗口看起来更舒服。要想修改系统中的“Command Prompt”窗口颜色，可以通过选择 Start > All Programs > Accessories > Command Prompt 来打开“Command Prompt”窗口，然后用鼠标右键点击标题栏，选择“Properties”(属性)。在随后出现的“Command Prompt”的“Properties”(属性)对话框中，点击“Color”(颜色)标签来选择你喜欢的文本和背景颜色。

1.10.1　在 Windows 命令提示符下运行一个 C 应用程序

1. **检查你的设置**。为了确保你已经将本书的例子正确地复制到你的硬盘驱动器中，请仔细阅读 www.deitel.com/books/chtp8/中的“开始之前”是很重要的。
2. **确定完整程序的位置**。打开一个“Command Prompt”(命令提示符)窗口。为了修改完整的猜数应用程序 GuessNumber 的目录，键入 cd C:\examples\ch01\GuessNumber\Windows，然后按“回车”键(参见图 1.7)。“cd”命令是用来改变文件目录的。

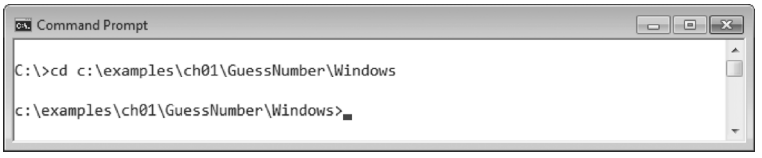

图 1.7　打开一个“Command Prompt”(命令提示符)窗口并修改文件目录

3. **运行 GuessNumber 应用程序**。现在就在包含 GuessNumber 应用程序的目录中，键入命令 GuessNumber(参见图 1.8)然后按“回车”键(注：这个应用程序的确切名字是 GuessNumber.exe。不过，窗口默认命令带有“.exe”扩展名)。

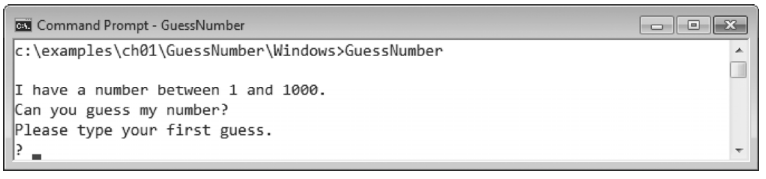

图 1.8　运行 GuessNumber 应用程序

4. **输入你的第一次猜测**。应用程序显示“Please type your first guess”(请输入你的第一次猜测)，然后在下一行显示一个问号(？)作为提示。在提示下，输入 500(参见图 1.9)。

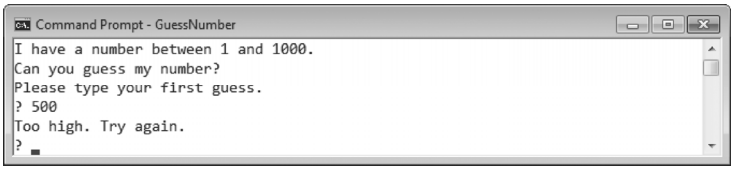

图 1.9　输入你的第一次猜测

5. **输入另一个猜测**。应用程序显示“Too high, Try again”(太大了，再猜一次)，这表示你输入的数比应用程序选择作为正确答案的那个数大。所以，下次猜测时应该输入一个较小的数。好的，在提示下，输入 250(参见图 1.10)。由于你输入的这个数还是比程序选择的那个数大，应用程序显示“Too high, Try again”(太大了，再猜一次)。

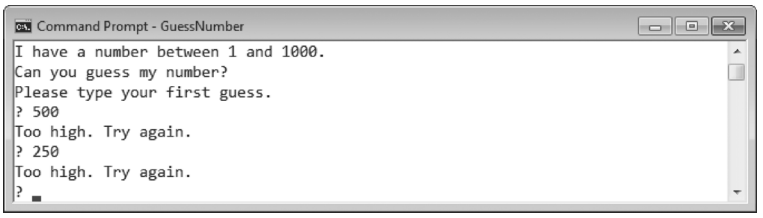

图 1.10 输入你的第二次猜测并获得反馈

6. **输入下一个猜测**。通过不断的输入整数继续玩这个游戏，直到你猜中了正确的数。应用程序将显示“Excellent! You guess the number”(太棒了！你猜中了这个数)(参见图 1.11)。

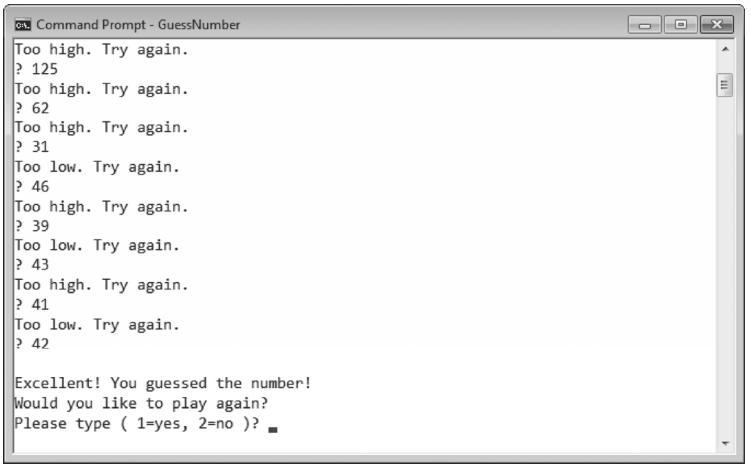

图 1.11 输入你的下一个猜测并继续猜正确的数

7. **再玩一次这个游戏或者退出应用程序**。在猜对后，应用程序会问你是否想再玩一次(参见图 1.11)。在提示下，输入 1 将导致应用程序选择一个新的数并显示后面带有一个问号的提示信息“Please type your first guess”(请输入你的第一次猜测)(参见图 1.12)，这样就可以在新一轮游戏中输入你的第一次猜测了；输入 2 将结束应用程序并返回到“Command Prompt”(命令提示符)窗口中应用程序所在的目录下(参见图 1.13)。每次从头(即第 2 步)开始执行这个应用程序，它都选择同一个数让你猜测。
8. **关闭“Command Prompt”(命令提示符)窗口。**

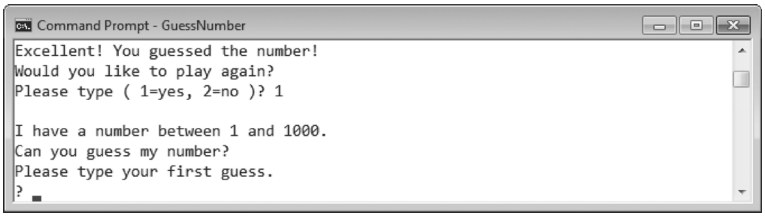

图 1.12 再玩一次这个游戏

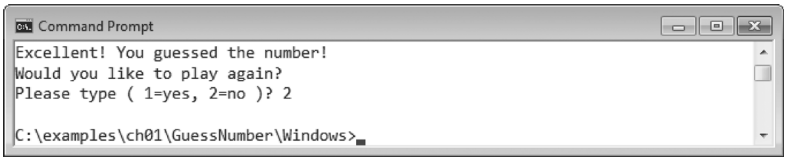

图 1.13　退出这个游戏

1.10.2　使用 Linux 中的 GNU C 来运行一个 C 应用程序

在本节的插图中，我们使用粗体字来显示每一步用户应邀输入的数字。在测试阶段，我们认为你是知道如何将这些例程复制到根目录下。如果在将这些文件复制到你的 Linux 系统时存在疑问，请联系你的教师。另外，在本节的插图中，使用粗体字来显示每一步用户应邀输入的数字。我们所用系统的命令解释程序(Shell)的提示符使用“波浪符”(~)来表示根目录，用一个“美元符”($)来表示提示符结束。不同的 Linux 系统的提示符的形式可能是不同的。

1. **检查你的设置**。为了确保你已经将本书的例子正确地复制到你的硬盘驱动器中，请仔细阅读 www.deitel.com/books/chtp8/中的“开始之前”是很重要的。
2. **确定完整程序的位置**。在 Linux 的 Shell 下，通过键入如下字符来修改完整的猜数应用程序 GuessNumber 的目录(参见图 1.14)：

   ```
   cd examples/ch01/GuessNumber/GNU
   ```

   然后按“回车”键 “cd”命令是用来改变文件目录的。

   ```
   ~$ cd examples/ch01/GuessNumber/GNU
   ~/examples/ch01/GuessNumber/GNU$
   ```

   图 1.14　修改应用程序 GuessNumber 的文件目录

3. **编译 GuessNumber 应用程序**。要想在 GNU C++编译器上运行一个应用程序，必须先键入如下字符来编译它：

   ```
   gcc GuessNumber.c -o GuessNumber
   ```

   如图 1.15 所示。这个命令将编译应用程序。其中“-o”选项的后面是你为可执行文件起的名字——GuessNumber。

   ```
   ~/examples/ch01/GuessNumber/GNU$ gcc -std=c11 GuessNumber.c -o GuessNumber
   ~/examples/ch01/GuessNumber/GNU$
   ```

   图 1.15　使用 gcc 命令来编译应用程序 GuessNumber

4. **运行 GuessNumber 应用程序**。要想运行可执行文件 GuessNumbe，只需在下一个提示符下键入命令“./GuessNumber”，然后按“回车”键(参见图 1.16)。

   ```
   ~/examples/ch01/GuessNumber/GNU$ ./GuessNumber

   I have a number between 1 and 1000.
   Can you guess my number?
   Please type your first guess.
   ?
   ```

   图 1.16　运行 GuessNumber 应用程序

5. **输入你的第一次猜测**。应用程序显示“Please type your first guess”(请输入你的第一次猜测)，然后在下一行显示一个问号(?)作为提示。在提示下，输入 500(参见图 1.17)。

```
~/examples/ch01/GuessNumber/GNU$ ./GuessNumber

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

图 1.17 输入你的首次猜测

6. 输入另一个猜测。应用程序显示“Too high, Try again”(太大了，再猜一次)，这表示你输入的数比应用程序选择作为正确答案的那个数大(参见图 1.17)。在提示下，你又输入 250(参见图 1.18)。由于你输入的这个数比程序选择的那个数小，应用程序显示“Too low, Try again”(太小了，再猜一次)。

```
~/examples/ch01/GuessNumber/GNU$ ./GuessNumber

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
?
```

图 1.18 输入你的第二次猜测并获得反馈

7. 输入下一个猜测。通过不断地输入整数继续玩这个游戏(参见图 1.19)直到你猜中了正确的数。当你猜中时，应用程序将显示“Excellent! You guess the number”(太棒了！你猜中了这个数)。

```
Too low. Try again.
? 375
Too low. Try again.
? 437
Too high. Try again.
? 406
Too high. Try again.
? 391
Too high. Try again.
? 383
Too low. Try again.
? 387
Too high. Try again.
? 385
Too high. Try again.
? 384

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )?
```

图 1.19 输入你的下一个猜测并继续猜正确的数

8. 再玩一次这个游戏或者退出应用程序。在猜对后，应用程序会问你是否想再玩一次(参见图 1.11)。在提示下，输入 1 将导致应用程序选择一个新的数并显示后面带有一个问号的提示信息“Please type your first guess”(请输入你的第一次猜测)(参见图 1.20)，这样就可以在新一轮游戏中输入你的第一次猜测了；输入 2 将结束应用程序并返回到 shell 中应用程序所在的目录下(参见图 1.21)。每次从头(即第 4 步)开始执行这个应用程序，它都选择同一个数让你猜测。

```
Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 1

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

图 1.20　再玩一次这个游戏

```
Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 2

~/examples/ch01/GuessNumber/GNU$
```

图 1.21　退出这个游戏

1.10.3　使用 Mac OS X 终端来运行一个 C 应用程序

本节的插图中，我们使用粗体字来显示每一步用户应邀输入的数字。你将使用 Mac OS X 的“Terminal”(终端)窗口来执行测试。欲打开一个“Terminal”(终端)窗口，只需点击位于屏幕右上角的高亮度的“Search”(查找)图标，然后输入“Terminal”即可找到 Terminal 应用程序。在高亮度的查找结果 Applications 的下方，选中“Terminal”即可打开一个“Terminal”(终端)窗口。“Terminal”(终端)窗口中，表示用户目录的提示符格式是 hostName:~userFolder$。本节的插图中，我们将去掉“hostName:”部分并使用通用名“userFolder”来使用用户账号的文件夹。

1. **检查你的设置**。为了确保你已经将本书的例子正确地复制到你的硬盘驱动器中，请仔细阅读 www.deitel.com/books/chtp8/中的“开始之前”是很重要的。我们假设这些例程存放在用户账号的 Documents/examples 文件夹中。
2. **确定完整程序的位置**。在“Terminal”(终端)窗口，键入如下命令即可改变完整的猜数应用程序 GuessNumber 的目录(参见图 1.22)：

   ```
   cd Documents/examples/ch01/GuessNumber/GNU
   ```

   然后按“回车”键。“cd”命令是用来改变文件目录的。

```
hostName:~ userFolder$ cd Documents/examples/ch01/GuessNumber/GNU
hostName:GNU$
```

图 1.22　修改应用程序 GuessNumber 的文件目录

3. **编译 GuessNumber 应用程序**。要想运行一个应用程序，必须先键入如下字符来编译它：

   ```
   clang GuessNumber.c -o GuessNumber
   ```

   如图 1.23 所示。这个命令将编译应用程序并生成一个名为 GuessNumber 的可执行文件。

```
hostName:GNU~ userFolder$ clang GuessNumber.c -o GuessNumber
hostName:GNU~ userFolder$
```

图 1.23　使用 gcc 命令来编译应用程序 GuessNumber

4. **运行 GuessNumber 应用程序**。欲运行可执行文件 GuessNumber，只需在下一个提示符下键入命令“./GuessNumber”，然后按“回车”键(参见图 1.24)。

```
hostName:GNU~ userFolder$ ./GuessNumber

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?
```

图 1.24　运行 GuessNumber 应用程序

5．**输入你的第一次猜测**。应用程序显示“Please type your first guess”（请输入你的第一次猜测），然后在下一行显示一个问号（？）作为提示。在提示下，输入 500（参见图 1.25）。

```
hostName:GNU~ userFolder$ ./GuessNumber

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
?
```

图 1.25　输入你的首次猜测

6．**输入另一个猜测**。应用程序显示“Too low, Try again”（太小了，再猜一次）（参见图 1.25），这表示你输入的数比应用程序选择作为正确答案的那个数小。在提示下，输入 750（参见图1.26）。由于你输入的这个数还是比程序选择的那个数小，应用程序显示“Too low, Try again”（太小了，再猜一次）。

7．**输入下一个猜测**。通过不断的输入整数继续玩这个游戏（参见图 1.27）直到你猜中了正确的数。当你猜中时，应用程序将显示“Excellent! You guess the number”（太棒了！你猜中了这个数！）

```
hostName:GNU~ userFolder$ ./GuessNumber

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
? 750
Too low. Try again.
?
```

图 1.26　输入你的第二次猜测并获得反馈

```
? 825
Too high. Try again.
? 788
Too low. Try again.
? 806
Too low. Try again.
? 815
Too high. Try again.
? 811
Too high. Try again.
? 808

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )?
```

图 1.27　输入你的下一个猜测并继续猜正确的数

8．**再玩一次这个游戏或者退出应用程序**。在你猜对后，应用程序会问你是否想再玩一次（参见图 1.11）。在提示下，输入 1 将导致应用程序选择一个新的数并显示后面带有一个问号的提示信