内容简介

本书按照快速了解、详细解读、深入进阶的顺序展开，目的是使读者尽快领略 C 语言的全貌，进而产生详解的兴趣和深入探讨的愿望。依据此思想本书分为三篇组织内容：第一篇感知篇，通过一个简单任务驱动，让读者在很短的时间了解 C 语言的主要知识点及 C 语言所能完成的任务；第二篇详解篇，逐步展开对每个知识点的详细研究，按照理解、语法规则、使用方式进行深入的探讨，学会计算机语言的基本要素；第三篇进阶篇分析、设计、实现一些有一定难度的案例，深层次运用各个知识点，进而培养读者结构化程序设计的能力。本书组织方式完全符合人类对语言学习过程：模仿、理解、应用，也符合软件工程迭代式开发过程的思想，对读者从事软件研发大有裨益。

前　言

从事了多年的C语言教学工作，怎样提高教学水平一直是萦绕笔者脑海的问题。总结学生学习中的问题，大致有以下三点：第一，学习过程的初期对计算机语言没有整体的认识，不了解C程序所能完成的功能，以及个知识点所扮演的角色，学习时没有明确的目的性；第二，C语言本身知识点繁多、晦涩难懂，对于初次接触计算机语言的读者来说难于掌握，久而久之，产生畏难情绪，因而丧失了学习的兴趣；第三，只能初步了解各知识点语法规则，不能够综合运用、融会贯通，更谈不上程序设计。

C语言具备一般语言的特点，笔者想计算机语言的学习能不能借鉴人类自然语言的学习方式？人类对自然语言的学习过程，总是从模仿开始，这个阶段只知道这么说，并不理解其语法规则；然后在生活、学习当中逐步懂得语言的结构及语法；最后，通过阅读、分析文章和撰写文件才能够熟练地应用语言。

C语言的学习是为了软件的研发，回顾多年的软件研发经验，其中最困难的当然是了解用户的需求。如果不熟悉问题领域的需求，学习起来很困难。如何解决呢？我们通常采用采用迭代式的软件开发方法，因为开始很难祥知开发问题领域的所有问题，可以选一条主线完成一个基本结构，然后经过多次迭代，每次软件有一定增量，使得每次的迭代产品都越来越接近目标系统。

于是，笔者萌生这样一个想法，C语言对于初学来说，知识点繁多、且难于理解，想一次完全掌握对于大多数读者来说相当困难。C语言的学习与人类学习自然语言以及软件开发过程中理解问题域的知识非常相似，就可以把人类对自然语言的学习过程和软件开发过程的迭代方法应用到C语言学习当中来，我们称之为“三次迭代法”。

三次迭代法采用三个周期，每个周期有一定的增量，递增的学习各个知识点。具体方法如下：感知部分（第一次迭代）让读者在很短的时间内亲身感受到C语言的全貌，建立起对C语言的兴趣，给出一些样例，并给出相似的模仿习题；详解部分（第二次迭代）逐步展开对每个知识点的详细研究，对每个知识点按照理解、语法规则、使用方式的次序做深入的探讨；进阶部分（第三次迭代）综合运用各个知识点分析、设计、实现一些有一定难度的案例，不但再次深层次运用各个知识点，而且培养了程序设计能力。这就完全符合了人类学习自然语言的过程，同时，三次迭代过程每次迭代都是对前一次的深化。

三次迭代法的教材组织与教学方法对传统模式的变革，类似于软件开发过程由“瀑布式开发”到的“迭代式开发”演变，对读者从事软件研发也会大有裨益。

有了以上想法，笔者异常兴奋，废寝忘食，奋笔疾书，望早日呈书于读者面前，希望为C语言的学习者提供些许帮助。

鉴于时间仓促，笔者水平有限，书中难免有纰漏，欢迎广大读者多提宝贵意见。

编　者

2010年6月

目　　录

第 0 章 概述

0.1 计算机的由来及组成

1. 计算机发展史

社会上对先进计算工具多方面迫切的需要，是促使现代计算机诞生的根本动力。20 世纪以后，各个科学领域和技术部门的计算困难堆积如山，已经阻碍了学科的继续发展。电子计算机的开拓过程，经历了从制作部件到整机从专用机到通用机、从“外加式程序”到“存储程序”的演变。1938 年，美籍保加利亚学者阿塔纳索夫首先制成了电子计算机的运算部件。1943 年，英国外交部通信处制成了“巨人”电子计算机。这是一种专用的密码分析机，在第二次世界大战中得到了应用。

1946 年 2 月美国宾夕法尼亚大学莫尔学院制成的大型电子数字积分计算机（ENIAC），最初也专门用于火炮弹道计算，后经多次改进而成为能进行各种科学计算的通用计算机。这台完全采用电子线路执行算术运算、逻辑运算和信息存储的计算机，运算速度比继电器计算机快 1000 倍。这就是人们常常提到的世界上第一台电子计算机。但是，这种计算机的程序仍然是外加式的，存储容量也太小，尚未完全具备现代计算机的主要特征。

新的重大突破是由数学家冯·诺伊曼领导的设计小组完成的。1945 年 3 月他们发表了一个全新的存储程序式通用电子计算机方案—电子离散变量自动计算机(EDVAC)，推动了存储程序式计算机的设计与制造。

1949 年，英国剑桥大学数学实验室率先制成电子离散时序自动计算机(EDSAC)；美国则于 1950 年制成了东部标准自动计算机(SFAC)等。至此，电子计算机发展的萌芽时期遂告结束，开始了现代计算机的发展时期。20 世纪中期以来，计算机一直处于高速度发展时期，计算机由仅包含硬件发展到包含硬件、软件和固件三类子系统的计算机系统。计算机系统的性能—价格比，平均每 10 年提高两个数量级。

计算机器件从电子管到晶体管，再从分立元件到集成电路以至微处理器，促使计算机的发展出现了三次飞跃。

第一代电子管计算机时期(1946～1959)，使用真空电子管和磁鼓做主存储器。主要用于科学计算。其特点是操作指令是为特定任务而编制的，每种机器有各自不同的机器语言，功能受到限制，速度也慢。

第二代晶体管计算机时期 (1959～1964)，主存储器均采用磁心存储器，磁鼓和磁盘开始用作主要的辅助存储器。计算机中存储的程序使得计算机有很好的适应性，可以更有效地用于商业用途。中、小型计算机，特别是廉价的小型数据处理用计算机开始大量生产。

第三代集成电路计算机 (1964-1972)，集成电路使得更多的元件集成到单一的半导体芯片上，半导体存储器逐步取代了磁心存储器的主存储器地位，磁盘成了不可缺少的辅助存储器，计算机变得更小，功耗更低，速度更快。这一时期的发展使用了操作系统，使得计算机在中心程序的控制协调下可以同时运行许多不同的程序，推动了微程序技术的发展和应用。

第四代大规模集成电路计算机 (1972-现在)，大规模集成电路 (LSI) 可以在一个芯片上容纳几百个元件。到了 80 年代，超大规模集成电路 (VLSI) 在芯片上容纳了几十万个元件，后来的 (ULSI) 将数字扩充到百万级。1981 年，IBM 推出个人计算机 (PC) 用于家庭、办公室和学校。 80 年代个人计算机的竞争使得价格不断下跌，微机的拥有量不断增加，计算机继续缩小体积。与 IBM PC 竞争的 Apple Macintosh 系列于 1984 年推出，Macintosh 提供了友好的图形界面，用户可以用鼠标方便地操作。20 世纪 70 年代以后，计算机用集成电路的集成度迅速从中小规模发展到大规模、超大规模的水平，微处理器和微型计算机应运而

生，各类计算机的性能迅速提高。随着字长 4 位、8 位、16 位、32 位和 64 位的微型计算机相继问世和广泛应用，对小型计算机、通用计算机和专用计算机的需求量也相应增长了。

新一代计算机是把信息采集存储处理、通信和人工智能结合在一起的智能计算机系统。它不仅能进行一般信息处理，而且能面向知识处理，具有形式化推理、联想、学习和解释的能力，将能帮助人类开拓未知的领域和获得新的知识。

2. 计算机组成

计算机由运算器，控制器，存储器，输入装置和输出装置五大部件组成，每一部件分别按要求执行特定的基本功能。

⑴ 运算器或称算术逻辑单元（Arithmetical and Logical Unit）

运算器的主要功能是对数据进行各种运算。这些运算除了常规的加、减、乘、除等基本的算术运算之外，还包括能进行“逻辑判断”的逻辑处理能力，即“与”、“或”、“非”这样的基本逻辑运算以及数据的比较、移位等操作。

⑵ 存储器（Memory unit）

存储器的主要功能是存储程序和各种数据信息，并能在计算机运行过程中高速、自动地完成程序或数据的存取。存储器是具有“记忆”功能的设备，它用具有两种稳定状态的物理器件来存储信息。这些器件也称为记忆元件。由于记忆元件只有两种稳定状态，因此在计算机中采用只有两个数码“0”和“1”的二进制来表示数据。计算机中处理的各种字符，例如英文字母、运算符号等，也要转换成二进制代码才能存储和操作。

存储器是由成千上万个“存储单元”构成的，每个存储单元存放一定位数（微机上为 8 位）的二进制数，每个存储单元都有唯一的编号，称为存储单元的地址。“存储单元”是基本的存储单位，不同的存储单元是用不同的地址来区分的，就好像居民区的一条街道上的住户是用不同的门牌号码来区分一样。

⑶ 控制器（Control Unit）

控制器是整个计算机系统的控制中心，它指挥计算机各部分协调地工作，保证计算机按照预先规定的目标和步骤有条不紊地进行操作及处理。

控制器从存储器中逐条取出指令，分析每条指令规定的是什么操作以及所需数据的存放位置等，然后根据分析的结果向计算机其它部分发出控制信号，统一指挥整个计算机完成指令所规定的操作。因此，计算机自动工作的过程，实际上是自动执行程序的过程，而程序中的每条指令都是由控制器来分析执行的，它是计算机实现“程序控制”的主要部件。

通常把控制器与运算器合称为中央处理器（Central Processing Unit-CPU）。工业生产中总是采用最先进的超大规模集成电路技术来制造中央处理器，即 CPU 芯片。它是计算机的核心部件。它的性能，主要是工作速度和计算精度，对机器的整体性能有全面的影响。

⑷ 输入设备(Input device)

用来向计算机输入各种原始数据和程序的设备叫输入设备。输入设备把各种形式的信息，如数字、文字、图像等转换为数字形式的“编码”，即计算机能够识别的用 1 和 0 表示的二进制代码(实际上是电信号)，并把它们输入到计算机内存储起来。键盘是必备的输入设备、常用的输入设备还有鼠标器、图形输入板、视频摄像机等。

⑸ 输出设备(Output device)

从计算机输出各类数据的设备叫做输出设备。输出设备把计算机加工处理的结果（仍然是数字形式的编码）变换为人或其它设备所能接收和识别的信息形式如文字、数字、图形、声音、电压等。常用的输出设备有显示器、打印机、绘图仪等。

通常把输入设备和输出设备合称为 I/O 设备（输入／输出设备）。

0.2 计算机程序

计算机每做的一次动作，一个步骤，都是按照以经用计算机语言编好的程序来执行的，程序是计算机要执行的指令的集合，而程序全部都是用我们所掌握的语言来编写的。所以人们要控制计算机一定要通过计算机语言向计算机发出命令。

计算机所能识别的语言只有机器语言，即由 0 和 1 构成的代码。但通常人们编程时，不采用机器语言，因为它非常难于记忆和识别。

目前通用的编程语言有两种形式：汇编语言和高级语言。

汇编语言的实质和机器语言是相同的，都是直接对硬件操作，只不过指令采用了英文缩写的标识符，更容易识别和记忆。它同样需要编程者将每一步具体的操作用命令的形式写出来。

高级语言是目前绝大多数编程者的选择。和汇编语言相比，它不但将许多相关的机器指令合成为单条指令，并且去掉了与具体操作有关但与完成工作无关的细节，例如使用堆栈、寄存器等，这样就大大简化了程序中的指令。同时，由于省略了很多细节，编程者也就不需要有太多的专业知识。

高级语言主要是相对于汇编语言而言，它并不是特指某一种具体的语言，而是包括了很多编程语言，如目前流行的 VB、VC、FoxPro、Delphi 等，这些语言的语法、命令格式都各不相同。

高级语言所编制的程序不能直接被计算机识别，必须经过转换才能被执行，按转换方式可将它们分为两类：

解释类：执行方式类似于我们日常生活中的“同声翻译”，应用程序源代码一边由相应语言的解释器“翻译”成目标代码(机器语言)，一边执行，因此效率比较低，而且不能生成可独立执行的可执行文件，应用程序不能脱离其解释器，但这种方式比较灵活，可以动态地调整、修改应用程序。

编译类：编译是指在应用源程序执行之前，就将程序源代码“翻译”成目标代码(机器语言)，因此其目标程序可以脱离其语言环境独立执行，使用比较方便、效率较高。但应用程序一旦需要修改，必须先修改源代码，再重新编译生成新的目标文件(＊ .OBJ)才能执行，只有目标文件而没有源代码，修改很不方便。现在大多数的编程语言都是编译型的，例如 Visual C＋＋、Visual Foxpro、Delphi 等。

计算机语言是人与计算机进行对话的最重要的手段。目前人们对计算机发出的命令几乎都是通过计算机语言进行的。与人之间的交流不仅仅依靠计算机语言，还有一些其它的方式，比如人的自然语言、手势、眼神等。由此我们可以推测，在不久的将来，计算机与人类的交流将是全方位的，而不再仅仅依靠计算机语言。那时，人们将更方便、更容易地操纵和使用计算机。

0.3 C 语言发展史

C 语言是目前世界上流行、使用最广泛的高级程序设计语言。

C 语言对操作系统和系统使用程序以及需要对硬件进行操作的场合，用 C 语言明显优于其它高级语言，许多大型应用软件都是用 C 语言编写的。

C 语言具有绘图能力强，可移植性，并具备很强的数据处理能力，因此适于编写系统软件，三维，二维图形和动画它是数值计算的高级语言。

C 语言的发展原型 ALGOL 60 语言。

1963 年，剑桥大学将 ALGOL 60 语言发展成为 CPL(Combined Programming Language)

语言。

1967 年，剑桥大学的 Matin Richards 对 CPL 语言进行了简化，于是产生了 BCPL 语言。

1970 年，美国贝尔实验室的 Ken Thompson 将 BCPL 进行了修改，并用它的第一个字母命名为“B 语言”。并且他用 B 语言写了第一个 UNIX 操作系统。

而在 1973 年，美国贝尔实验室的 D.M.RITCHIE 在 B 语言的基础上最终设计出了一种新的语言，他取了 BGPL 的第二个字母作为这种语言的名字，这就是 C 语言。

为了使 UNIX 操作系统推广，1977 年 Dennis M.Ritchie 发表了不依赖于具体机器系统的 C 语言编译文本《可移植的 C 语言编译程序》。

1978 年 Brian W.Kernighian 和 Dennis M.Ritchie 出版了名著《The C Programming Language》，从而使 C 语言成为目前世界上流行最广泛的高级程序设计语言。

1988 年，随着微型计算机的日益普及， 出现了许多 C 语言版本。由于没有统一的标准，使得这些 C 语言之间出现了一些不一致的地方。为了改变这种情况，美国国家标准化协会(ANSI)为 C 语言相继制定了一系列标准，现在用得最多是 1989 年制定的 ANSI X3.159-1989（简称 C89），1990 年，国际标准化组织 ISO 接受 C89 为为国际标准 ISO/IEC9899：1990（简称 C90）。C90 和 C89 基本上是一致的。本书以 C89 为标准进行论述。

1999 年，ISO 组织在保留 C 语言基本特征基础上，又增加了面向对象的支持，命名为 ISO/IEC9899：1999（简称 C99）。

各个软件厂商开发的 C 语言的编译系统，并不是完全支持标准 C。不同的厂商有的为了效率、有的为了方便，对 C 语言的功能和语法规则可能略有更改，所以，学习时要了解具体的编译系统。

0.4 C 程序基本结构

任何一种程序设计语言都具有特定的语法规则和规定的表达方法。一个程序只有严格按照语言规定的语法和表达方式编写，才能保证编写的程序在计算机中能正确地执行，同时也便于阅读和理解。

为了了解 C 语言的基本程序结构，先介绍几个简单的 C 程序。

例 0.1 第一个 C 程序

```
#include<stdio.h>
int main()
{
   printf("Welcome to C Language World!");
   return 0;
}
```

这是一个最简单的 C 程序，其执行结果是在屏幕上显示一行信息：

```
Welcome to C Language World!
```

每个 C 语言程序必须包含一个 main 函数（有且仅有一个，好比每个家庭只能有一个一家之主一样），程序从 main 函数开始执行，main 函数执行完毕则程序结束。在例 1.1 中，main 函数只包含一条语句 printf()。程序执行到 printf()时，去使用（专业词汇称为“调用”）printf 函数。printf 的说明在头文件 stdio.h 中，如同读一篇文章时，不明白句子“学而时习之，不亦悦乎？”的含义，去查《论语》一样。

例 0.2 通过函数计算两个整数之和

```
#include<stdio.h>
/*计算两个整数的和*/
int add(int i1,int i2)
```

```
{
   int i3;
   i3=i1+i2;
   return i3;
}
int main()
{
   int i4=1;
   int i5=2;
   int i6=add(i4,i5);
   printf("%d",i6);
   return 0;
}
```

例 0.2 程序从 m a i n ( )处开始执行，定义两个变量 i4、i5 并且赋值为 1 和 2；然后调用 add 函数对，把 i4、i5 的值传递给 add 函数的 i1、i2，i1、i2 的值为 1 和 2，add 函数中计算出 i1 和 i2 的和赋值给 i3，通过语句“return i3”返回给 main 函数；main 函数把 add 函数返回的值赋值给 i6，通过 printf 函数输出 i6，“return 0”结束 main 函数。

从上面程序例子，可以看出 C 程序的基本结构。

C 程序为函数模块结构，所有的 C 程序都是由一个或多个函数构成，其中必须只能有一个主函数 m a i n ( )。程序从主函数开始执行，当执行到调用函数的语句时，程序将控制转移到调用函数中执行，执行结束后，再返回主函数中继续运行，直至程序执行结束。C 程序的函数包含编译系统提供的标准函数（如 printf 等）和用户自己定义的函数（如 add 等）。函数的基本形式是：

```
函数类型  函数名(形式参数)
{
   数据说明部分；
   语句部分；
}
```

其中：

函数首部包括函数类型（如 int）、函数名（如 main、printf、add）和圆括号中的形式参数（如 int i1,int i2）。

函数体包括函数体内使用的数据说明（如 int i3;int i4）和执行函数功能的语句（如 i3=i1+i2;），花括号{和}表示函数体的开始和结束。数据说明和执行语句都必须以分号（;）结束。

0.5 C 程序开发步骤

开发一个 C 程序，按照顺序包括以下四步：

1. 编辑。程序员用任一编辑软件（编辑器）将编写好的 C 程序输入计算机，并以文本文件的形式保存在计算机的磁盘上。编辑的结果是建立 C 源程序文件，扩展名为 c（如 welcome.c）。

2. 编译。编译是指将编辑好的源文件翻译成二进制目标代码的过程。编译过程是使用特定环境的编译程序（编译器）完成的。不同操作系统下的各种编译器的使用命令不完全相同，使用时应注意计算机环境。编译时，编译器首先要对源程序中的每一个语句检查语法错误，当发现错误时，就提示错误的位置和错误类型的信息。此时，要再次调用编辑器进行查

错修改。然后，再进行编译，直至排除所有语法和语义错误。正确的源程序文件经过编译后在磁盘上生成目标文件，如（welcome.obj）。

3．连接。程序编译后产生的目标文件是可重定位的程序模块，不能直接运行。链接就是把目标文件和其他分别进行编译生成的目标程序模块（如果有的话）及系统提供的标准库函数（如 printf）连接在一起，生成可以运行的可执行文件的过程。连接过程使用特定环境的连接程序（连接器）完成，生成的可执行文件存在磁盘中（如 welcome.exe，连接的文件名不一定和源文件同名）。

4. 运行。生成可执行文件后，就可以在操作系统控制下运行。若执行程序后达到预期目的，则 C 程序的开发工作到此完成。否则，要进一步检查修改源程序，重复编辑--编译—连接--运行的过程，直到取得预期结果为止。

0.6 集成开发环境

为了编译、连接 C 程序，需要有相应的 C 语言编译器与连接器。目前大多数 C 环境都是集成的开发环境（IDE），把程序的编辑、编译、连接、运行都集成在一个环境中，界面友好、简单易用。

目前 C 语言的主流集成环境有 Visual C++6.0、DEV-C++、TurboC 等。Visual C++6.0 是 Windows 下的图形界面的集成环境，编辑、编译、调试等都可以可视化地进行，对 C 语言的学习者非常容易上手。Visual C++6.0 是微软的产品（微软有时不太遵守标准），对 C89 支持不是很好。但是从易用性角度考虑，本书采用 Visual C++6.00 环境进行介绍，所有例子都在 Visual C++6.0 中调试通过。

Visual C++6.0 为用户开发 C 和 C++程序提供了一个集成环境，这个集成环境包括：源程序的输入和编辑，源程序的编译和连接，程序运行时的调试和跟踪，项目的自动管理，为程序的开发提供各种工具，并具有窗口管理和联机帮助等功能。

使用 Visual C++集成环境上机调试程序可分成如下几个步骤：启动 Visual C++集成环境；建立和编辑源程序；编译连接源程序；运行程序。下面详细介绍一下 Visual C++的上机操作方法。

1. 环境窗口介绍

Visual C++6.0 启动后，就产生如图 0.1 所示的 Visual C++6.0 集成环境。

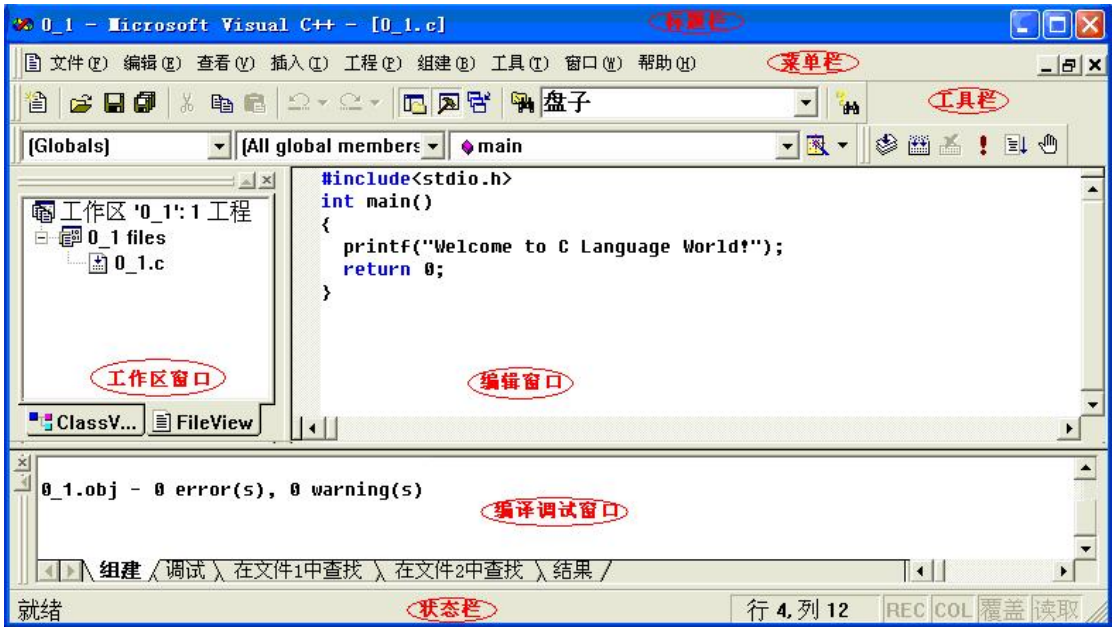


图 0.1 Visual C++6.0 集成环境

Visual C++6.0 集成环境是一个组合窗口。窗口的第一部分为标题栏；第二部分为菜单栏，其中包括文件（File）、编辑（Edit）、视图（View）、插入（Insert）、工程（Project）、组建（Build）、工具（Tools）、窗口（Windows）、帮助（Help）等菜单。第三部分为工具栏，其中包括常用的工具按钮；第四部分为状态栏。还有三个子窗口：工作区窗口、编辑窗口、编译调试窗口。

2. 建立 C 源程序

生成源程序文件的操作步骤为：

（1）选择集成环境中的“文件（File）”菜单中的“新建（New）”命令，产生“新建（New）”对话框，如图 0.2 所示。

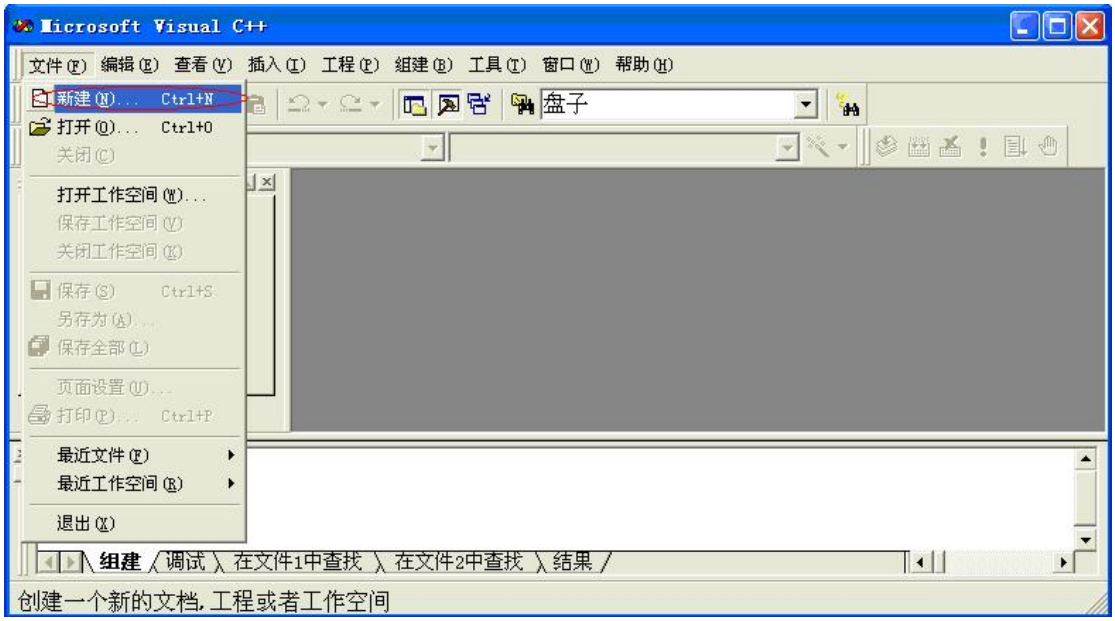


图 0.2 新建 C 源文件

（2）单击此对话框的左上角的 File（文件）选项卡，选择 C++ Source File 选项。如下图所示：

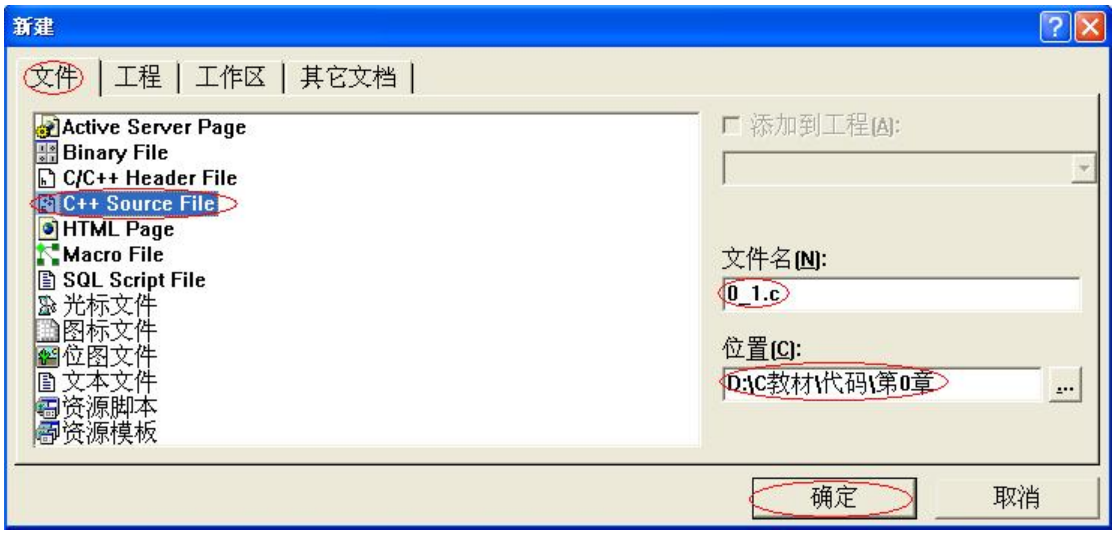


图 0.3 新建 C 源文件选项

（3）我们在右上方的文件（File）文本框输入准备编辑的源程序文件的名字，例如图 0.3 中我们给源程序文件命名为 0_1.c。注意：指定的文件名后缀为.c，如果输入的文件名为 0_1.cpp，则表示要建立的是 C++源程序。如果不写后缀，系统会默认指定为 C++源程序文件，自动加上后缀.cpp。为学习 C 程序的的语法规范，建议扩展名采用 c。

（4）设置源文件保存路径

若将源文件保存在默认的文件存储路径下，则可以不必更改位置（Location）文本框，但如果想在其他地方存储源程序文件则需在对话框右半部分的位置（Location）文本框中输入文件的存储路径，也可以单击右边的省略号(…)来选择路径。图 1.3 中源文件的保存路径就为：D:\C 教材\代码\第 0 章。

（5）单击确定（OK）按钮，进入编辑界面。

3. 编辑 C 源程序

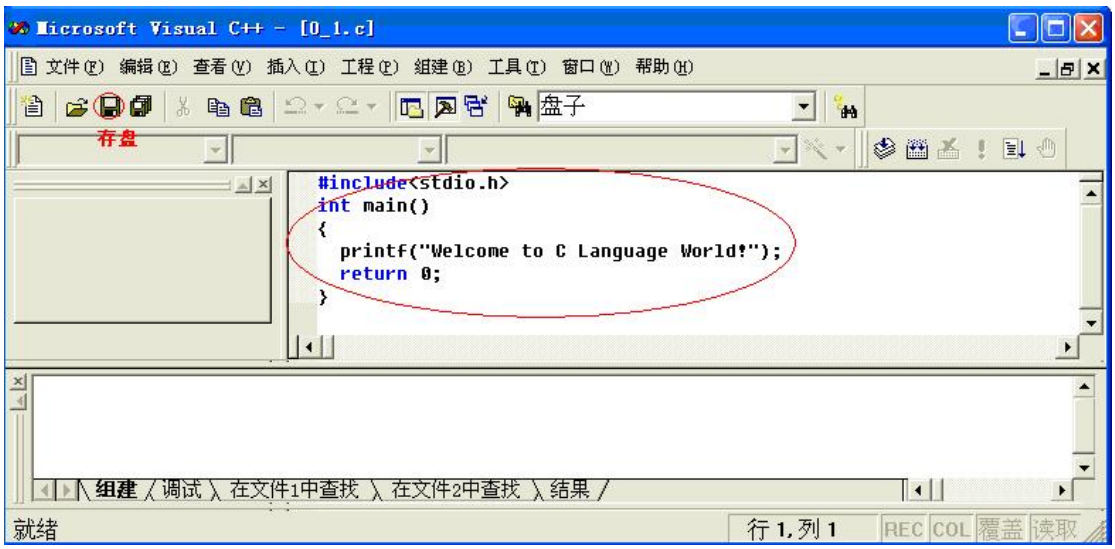


图 0.4 编辑 C 源程序

在图 0.4 所示的编辑窗口中输入源程序代码，输入完毕后单击存盘工具按钮保存程序

4. 编译 C 源程序

单击主菜单栏中的组建（Build），在其下拉菜单中选择“编译 0_1.c(Compile frist.c）项”，或者单击工具栏编译按钮，则开始编译程序。如图 0.5 所示。

编译过程中，编译命令要求一个有效的项目工作区，你是否同意建立一个默认的项目工作区），单击是（Y）按钮，表示同意由系统建立默认的项目工作区。这个过程中还将提示你是否保存变动，单击是（Y）按钮。

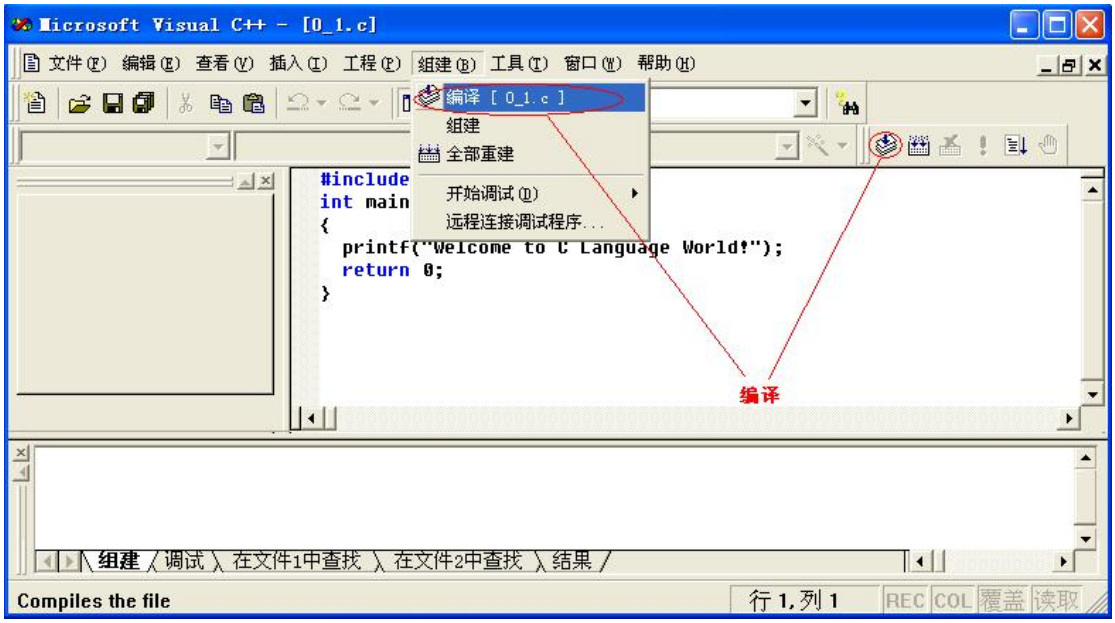


图 0.5 编译 C 源程序

若源代码无错误，编译成功后，会在编译调试窗口显示生成 0_1.obj 文件，如图 0.6 所示。若源代码有错误，会在编译调试窗口显示有多少个错误、多少个警告，而且还会有详细的错误列表（每个错误位置、错误号、错误类型、错误代码）和警告列表。双击错误项，光标会停在该错误对应的源代码处，修改后存盘，在重新编译，直到没有错误为止。

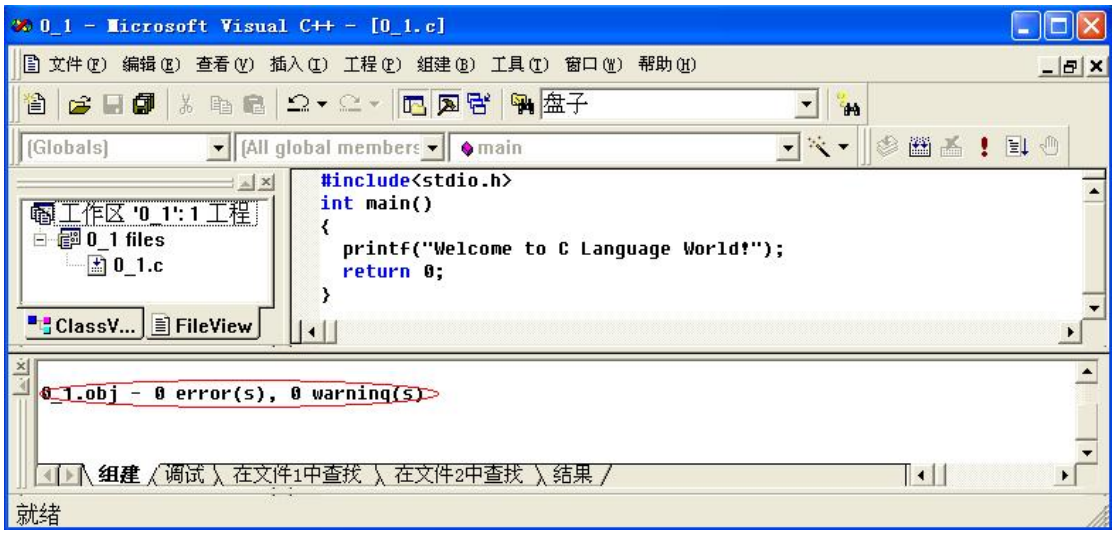


图 0.6 编译结果

5. 连接目标程序

在得到了目标程序后，就可以对程序进行连接了。选择主菜单上的组建（Build），在其

下拉菜单中选择“组建 0_1.Exe（Build 0_1.exe）项”，或者单击工具栏连接按钮，则开始连接程序。如图 0.7 所示。

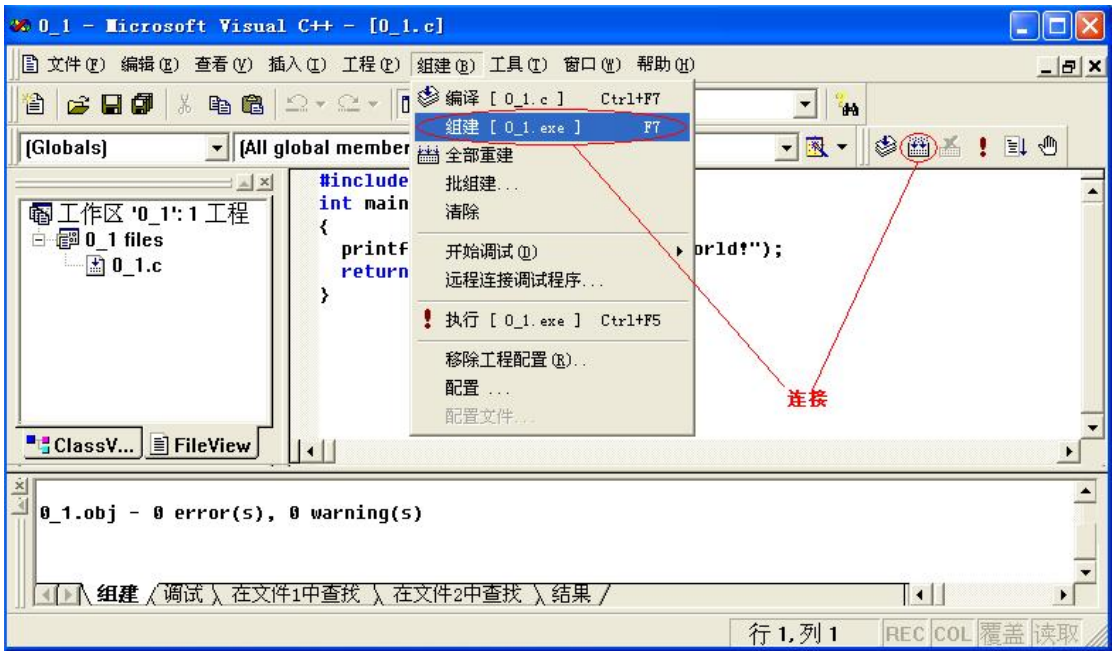


图 0.7 连接 C 目标程序

若连接各个模块没有错误，连接成功后，会在编译调试窗口显示生成 0_1.exe 文件，如图 0.8 所示。若连接各个模块有错误，会在编译调试窗口显示有多少个错误、多少个警告，而且还会有详细的错误列表（错误号、错误描述）和警告列表。这时一般是连接资源找不到，修改对应的源代码错误处，修改后存盘，在重新编译，重新连接，直到没有错误为止。

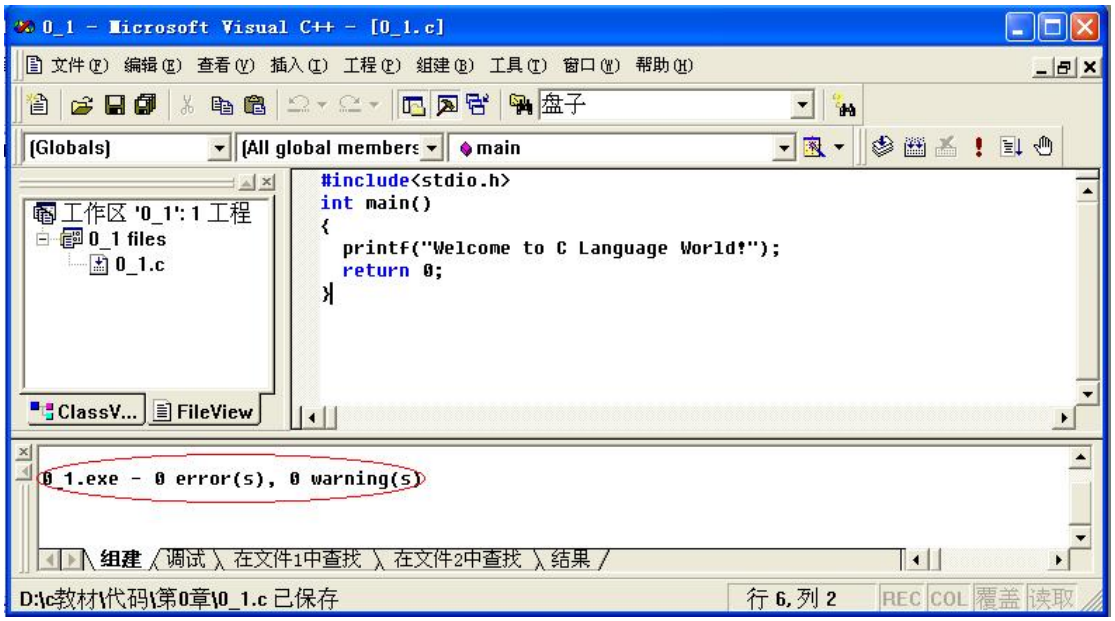


图 0.8 连接结果

6. 运行程序

选择主菜单上的组建（Build），在其下拉菜单中选择“执行 0_1.exe（Execute 0_1.exe）

项”，或者单击工具栏运行按钮，则开始运行程序。如图 0.9 所示。

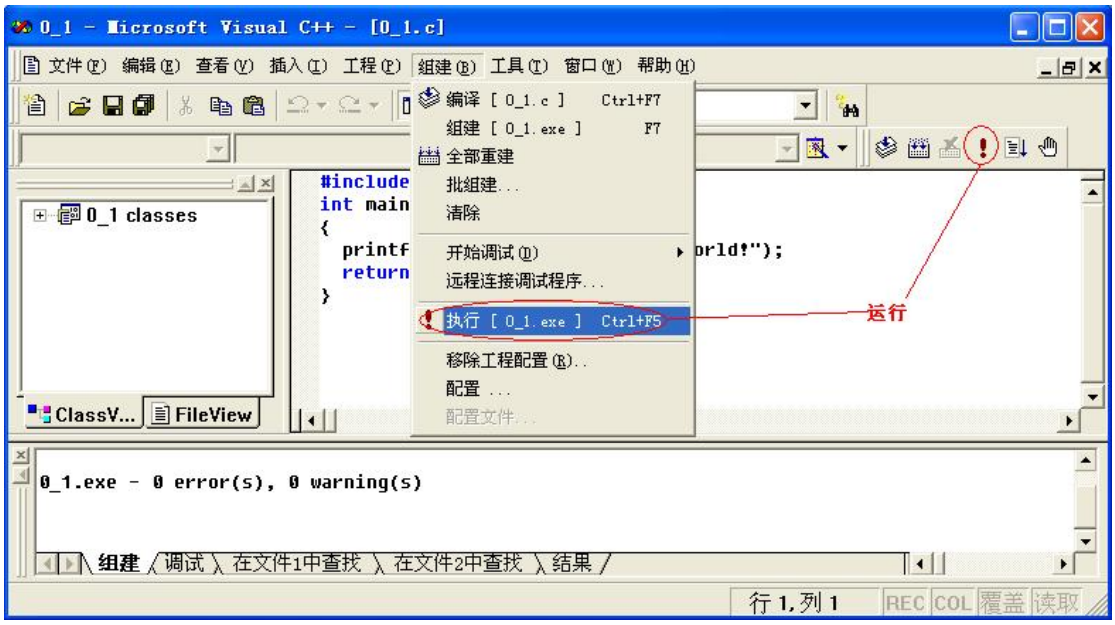


图 0.9 运行 C 程序

被启动的程序在控制台窗口下运行，与 Windows 中运行 DOS 程序的窗口类似。图 0.10 是执行程序后，弹出 DOS 窗口中显示的程序执行结果。

图 0.10 运行结果

注意：“Press any key to continue”并非程序所指定的输出，而是 VC6.0 在输出完运行结果后系统自动加上的一行信息，通知用户："按任何一键以便继续"。当你按下任何一键后，输出窗口消失，回到 VC6.0 主窗口。

习题

1. 计算机由那几部分组成？
2. 什么是计算机程序？
3. 说明计算机高级语言与计算机低级语言的异同。
4. C 语言的程序的开发过程包括哪几个步骤，每一步生成什么文件？
5. 说明 C 语言程序的结构。
6. 编写一个输出你姓名的 C 程序，在环境中调试。

第 1 篇 感知篇

人类自然语言的学习过程总是从模仿开始的，而对语言的理解不一定能面面俱到。模仿的过程可以提供对语言足够的感性认识，因而形成自然的思维方式和习惯。基于此思想，本篇从任务驱动的角度出发，逐步展开，应用各个知识点，解决任务中的问题。这样，读者快速了解 C 语言的主要知识点，对 C 语言就有一个全面的认识，亲身感受到利用 C 语言解决问题的工作过程，进而产生进一步深入学习 C 语言的愿望。

本篇以阿 Q 的成绩管理过程入手，让读者快速了解 C 语言的输入输出、程序控制、数据组织等，让读者了解 C 语言所能完成的工作。同时，以“解决大毛日常生活中的一些小问题”为例贯穿整篇，供读者模仿练习，亲手体会 C 语言的魅力。本篇的最后，在读者了解 C 语言基本开发过程之后，系统介绍算法描述和编码规范，以便在深入学习 C 语言各要素之前就养成一个好的表达习惯。

第 1 章 数据的基本操作

《C 语言程序设计》（以下简称《C 语言》）是阿 Q 接触的第一门计算机基础课程，那么如何用对该课程的成绩进行管理呢？，程序又是如何实现处理课程成绩的功能的呢？本章通过几个简单的小程序讨论数据的基本操作方法，引入数据的存储方式、数据的输入与输出操作以及数据的基本运算方法。

1.1 数据的存储与输出

为了使新手对程序设计有基本的认识，从简单的需求开始设计小程序，使其对变量的概念和输出函数的用法能够有直观地了解。

【抛砖引玉】

假设阿 Q 本学期的《C 语言》成绩为 92 分，例 1.1 实现成绩的存储和输出功能。

例 1.1 指定并输出阿 Q 的《C 语言》成绩。

```
/*  例 1_1.c    输出成绩*/
#include <stdio.h>                                  /*引入库函数*/
int main( )                                         /*主函数*/
{
   int iScore ;                                    /*定义整型变量存放 C 成绩*/
   iScore = 92 ;                                    /*将 92 赋值给变量 iScore*/
   printf("Mr. Ah Q's score of C language:");   /*输出一串字符*/
   printf("%d\n", iScore);                         /*输出 C 成绩*/
   return 0;                                       /*结束函数，返回 0*/
}
```

程序运行结果：

```
Mr. Ah Q's score of C language:92
```

通过这个例子，可以了解数据的存储和使出方式

1．变量的概念

程序运行时要对一些数据进行加工处理，变量是程序中存储数据的基本单位。每个变量都有类型、名字和值。为了将数据正确的存储在内存中，必须选择合适的类型，如 int(整型)，float（浮点型）和 char(字符型)。好比要将物品保存起来，需要选择合适的容器，既能够装下物品又不浪费空间。变量的名字是识别该数据的标识符，取个合理规范的名字便于维护程序。变量的值在程序运行过程中可以改变，程序中通过对变量 iScore 进行赋值操作改变其值。

2．输出函数 printf（）

printf 函数能够将其参数按照指定的格式输出，方便程序员或者程序用户直观的观测内存中存储的二进制数据。本程序中将变量 iScore 的值按％d 的格式（十进制整数）显示在屏幕上，并输出转移符\n 即回车。该函数还可以直接输出引号中的字符串。

【乘胜追击】

为了进一步了解变量和输出函数的使用方法，我们将 1.1 进行扩展。要求输出阿 Q 的个人信息，包括姓名、年龄和身高。

例 1.2 输出阿 Q 的个人信息

```
#include <stdio.h>
int main( )                                         /*主函数*/
{
```

```
    /*定义变量  */
    int     iAge ;
    float   fHeight;
    /*变量赋值*/
    iAge = 18 ;
    fHeight = 1.78;
    /*输出字符串和多个数据*/
    printf("Mr. Ah Q's age and height:\n %d %f\n", iAge, fHeight);
    return 0;
}
```

运行结果：

```
Mr. Ah Q's age and height:
18 1.780000m
```

通过该程序可以了解不同类型数据的表示和输出方法，注意几下几点：

1. 类型不同数据的定义

对于年龄和身高，分别用整型和单精度的小数（浮点型 float）变量存储数据，且必须在两个语句中分别定义，不能将其定义在同一语句中。

2. 输出多个数据

标准输出函数 printf 可以用于连续输出多个数据，它们的类型可以不同。在本程序中，按照双引号中的格式，首先输出字符串，再输出回车，接着输出变量 iAge 和 fHeight 的值，对于 float 型的数据必须按%f 的形式输出。

3.关于注释

/* …*/中的语句为注释，可以是一行或者多行，可以是任何语言或者符号。注释并不是编译器需要编译的代码，它可以起到对代码进行说明的作用。为了增加代码的可读性，并且便于程序的修改和维护，要对一些关键代码或者数据进行注释。

【同步练习】

阿 Q 的邻居家有个孩子叫大毛，请定义不同类型的变量，并对其进行赋值，编程输出大毛的姓名、性别、年龄和身高。

1.2 数据的输入与运算

在程序中用变量存储数据，变量的值在程序运行的过程中可能发生改变，通常改变变量值有两种方式：其一，数据通过各种运算处理后，由赋值符号“=”赋予新的值；其二，需要用户与程序进行交互，通过输入函数实现。

【抛砖引玉】

在计算阿 Q 的《C 语言》成绩时候，需要考虑考试成绩与平时成绩两部分，期末总成绩为 70%的考试成绩和平时成绩之和，它们的值分别由教师输入，实现总成绩的计算及输出功能。

例 1.3　计算并输出期末总成绩

```
#include <stdio.h>
int main( )
{
   int iScore1 = 0, iScore2 = 0 ;                  /*定义变量  考试成绩  与  平时成绩*/
   float fTotalScore ;                             /*定义变量总成绩*/
   printf("输入考试成绩与平时成绩:\n");          /*输出提示信息*/
```

```
    scanf("%d ",&iScore1) ;                    /*分别输入成绩*/
    scanf("%d",&iScore2) ;
    fTotalScore   =   iScore1*0.7 + iScore2;     /*  计算总成绩*/
    printf("总成绩为%f\n", fTotalScore);          /*输出浮点型的总成绩*/
    return 0;
}
```

运行结果：

输入考试成绩与平时成绩:

75

26

总成绩为 78.500000

通过这个例子继续分析变量的使用方法，了解输入函数 scanf 的功能。

1. 变量的定义

使用变量前一定要先定义，对变量的类型及名字进行声明，通知编译器为其分配内存。C 语言对变量的声明必须在函数开始处，将要用到的所有变量逐一声明。本程序中分别定义考试成绩变量和平时成绩变量，并对其赋初值 0，定义 fTotalScore 时没有在声明时候初始化，而是通过“=”赋予。初始化变量不是必须的，但是个好的习惯，最好设置合理的变量初值。

2. 输入函数 scanf

函数 scanf 也是 C 语言中的一个标准库函数，其基本功能是将从标准输入设备（如键盘）中获取的数据存储在变量中，调用该函数需要包含标准输入输出头文件<stdio.h>。

程序执行 scanf("%d ",&iScore1) ; 语句时，屏幕的光标闪动，等待用户输入一个整数，当用户在键盘上输入数据 75 和回车键后，数值 75 按照%d（表示整型）的格式被存储在整型变量 iScore1 中，一般输入数据时用回车键作为结束符。

调用函数 scanf 时涉及两个参数：其一，输入格式，引号中的字符串表示输入数据的类型，如%d 整型，%f 浮点型，%c 字符型等；其二，变量列表，以取地址运算符&作用于变量名前，表示要将输入的数据传送要某个变量对应的内存空间。

3. 运算符与表达式

C 语言提供了丰富的运算符，如用于计算的算术运算符(+、-、*、/)和赋值运算符（=），用于逻辑判断的关系运算符（>、<、≤、≥、==、!=），以及其他运算符。运算符以简洁灵活的形式提供强大的功能，有效增强程序的可读性。运算符将操作数连接起来构成表达式，每个表达式都有某种类型的值，在求解表达式的值的时候，按照不同运算符的规则计算。

【乘胜追击】

为了继续了解对数据进行输入和运算的方法，我们通过程序 1.4 进一步理解输入函数 scanf 的用法，并介绍其他运算符的使用规则。

例 1.4 输入阿 Q 的《C 语言》期中考试成绩和期末考试成绩，计算并输出平均成绩。

```
#include <stdio.h>
int main( )
{
    int iScore1, iScore2 ;                     /*期中成绩 与 期末成绩*/
   double dAverage = 0.0;                      /*平均成绩*/
   printf("输入期中成绩与期末成绩:\n");
   scanf("%d%d",&iScore1,&iScore2) ;       /*连续输入成绩*/
   dAverage =    iScore1 + iScore2 / 2;       /*  计算总成绩*/
   printf("平均成绩为%f\n", dAverage);       /*输出平均成绩*/
```

```
    return 0;
}
```

运行结果：

输入考试成绩与平时成绩:

88 91

平均成绩为 133.000000

标准函数 printf 可以连续输出多个类型不同的数据，函数标准输入函数 scanf 也可以实现多个数据的输入。将要输入值的变量用逗号分隔开，并在双引号中的指定格式。一般输入数值型数据时，用回车、空格和 Tab 键作为结束符或者分隔割符。本程序运行时，输入的两个整数 88 和 91，用空格作为分隔符，并用回车作为结束符，数据按顺序依次传递给 iScore1 和 iScore2。注意输入数据的类型、个数和顺序，必须和指定格式参数完全一致，才能使变量获得正确的数据。

观察运行结果，发现并没有输出正确的结果，原因是计算平均成绩的表达式错误。算术运算符的优先级和代数四则的运算规则相似，应将表达式改为（iScore1 + iScore2）/ 2，用括号运算符（）改变“先乘除后加减”的计算顺序。这样程序就会输出准确的平均成绩了吗？细心的读者会发现，输入 88 和 91 后输出的平均成绩为整数 89。C 语言中整数和整数运算其结果还是整数，因此 179/2 得到结果的小数部分就被无情的丢弃了，这就是出现此逻辑错误的原因所在。那么请聪明的你将这个 bug 解决掉。

【同步练习】

大毛是个一年级的“小豆包”，期末考试后阿 Q 问他考的怎样，编写程序帮助大毛计算数学、语文和英语三门成绩的平均分。

1.3 数据的比较与判断

现实生活中，有合理的比较才能进行准确的判断。计算机不仅能够进行复杂的数学计算，还能如人一样进行逻辑分析和判断，这些都是通过程序实现的。C 语言提供关系运算符和逻辑运算符，将复杂的逻辑问题映射成逻辑表达式，并通过选择结构实现对不同分支的处理。

【抛砖引玉】

根据平均成绩判断阿 Q 的成绩是否合格。由用户输入期中考试和期末考试成绩，计算平均成绩，若平均成绩高于 60 分（包括 60 分）为合格，否则输出不合格。

例 1.5　　计算平均成绩，判断是否合格

```
#include <stdio.h>
int main( )
{
    int iScore1, iScore2 ;                          /*期中成绩 与 期末成绩*/
    double dAverage = 0.0;                          /*平均成绩*/
    printf("输入期中成绩与期末成绩:\n");
    scanf("%d,%d",&iScore1,&iScore2) ;              /*分别输入成绩*/
    dAverage =   (iScore1 + iScore2)/ 2.0;          /* 计算平均成绩*/
    printf("平均成绩为%.2f\n", dAverage);           /*输出平均成绩*/
    /*判断成绩是否合格*/
    /*如果满足条件 dAverage >=60, 则执行{}中的语句*/
    if(dAverage >=60 )
    {
      printf("成绩合格");
```

```
    }
    if (dAverage<60 )
    {
       printf("成绩不合格");
    }
    return 0;
}
```

运行结果：

输入期中成绩与期末成绩:

88,91

平均成绩为 89.50

成绩合格

前面的程序中代码是逐条执行的，但实际上程序中的代码不一定都要执行到。程序可以根据某个条件，有选择地执行某些语句，也可以跳过一些语句不执行。在本例程中，根据计算出的平均成绩，选择输出“成绩合格”或者“成绩不合格”。下面来了解简单选择结构的用法。

1. if 语句

生活中我们常说“如果…则…”，程序中可以用 if 语句构成选择结构来实现。程序中如果满足条件 dAverage >=60，则执行输出函数 printf("成绩合格")，否则跳过该行继续下面的操作。用 if 语句构造单分支的选择结构，根据圆括号中条件来判断，是否执行花括号{}中的语句。

2. 关系运算符与表达式

在 if 语句中出现的表达式 dAverage >=60 和 dAverage<60，都是用于判断的关系表达式，此类表达式中使用关系运算符<、>、>=（大于等于）、<=（小于等于）、==（等于）、!=（不等）。这种表达式的值只有真或假两种，常用数值 1 和 0 来表示逻辑真（true）和假(false)。若满足该关系即此条件成立，则表达式的值为逻辑真。

3. 格式化输入输出

在使用函数 scanf 的时候，必须注意数据输入的格式，如果没有按照双引号中要求输出，则会产生匪夷所思的结果。不妨作一个有趣的实验，运行这个程序的时候输入两个数据，用回车或者空格作为分隔符，看看能否输出正确的结果。

不论是 float 还是 double 型的小数，若用%f 的格式输出，都输出小数点后 6 位。然而在输出平均分时，显示的结果为 89.50，而不是将小数点后的 89.500000。这是由于使用了%.2f 的格式输出 dAverage，限制只输出两位小数。

【乘胜追击】

输入阿 Q 本学期的平均成绩，判断成绩为优秀、合格还是重修。在输入成绩时校验数据的合理性，假设正确的成绩在 0～100 之间，当用户输入不合适的数据时输出错误的提示。假设平均成绩高于 85 分为优秀， 高于 60 分并且低于 85 分为合格格式，低于 60 分则需要重修。

例 1.6 输入成绩判断是否优秀

```
#include <stdio.h>
int main( )
{
  double dAverage = 0.0;          /*平均成绩*/
  printf("输入平均成绩:\n");
```

```
    scanf("%lf",&dAverage) ;
    if(dAverage>0 && dAverage<100)
    {       printf("平均成绩为%.2f\n", dAverage);    }
    else
    {     printf("成绩无效!\n");       }

    /*判断成绩*/
    if(dAverage > 85 )            /*  成绩高于 85*/
    {       printf("优秀\n");     }
    else if (dAverage>=60 )    /*成绩在 60~85*/
    {       printf("合格\n");    }
    else                              /*成绩低于 60*/
    {       printf("重修\n");    }

    return 0;
}
```

运行结果：

输入平均成绩:

77.675

平均成绩为 77.67

合格

通过该程序可以进一步了解其他形式的选择结构，掌握 if-else 构成的多分支选择语句的用法：

1. if-else 双分支选择结构

严谨的程序应该对输入数据的进行合法性校验，对于非法输入提示出错或者不予处理。程序 1_6 在校验平均分时，使用 if-else 语句构造了一个双分支的选择结构。其语句的执行顺序为：当 if( )中用于判断的表达式值为逻辑真，则执行 if 后{}中的语句，否则执行 else 后{}中的语句。

2. 逻辑运算符与表达式

对于简单的逻辑问题，可以用一个关系表达式构造用于判断的表达式，若比较复杂的逻辑问题，则用逻辑运算符（逻辑与&&、逻辑或||和逻辑非!）将多个关系表达式连接起来。当成绩在 0～100 之间时为有效，可以将算术表达式 0<dAveraged<100 映射成逻辑表达式 Average>0 && dAverage<100，当同时满足 Average>0 和 dAverage<100 两个条件时，该表达式值为真。

3.if-else if-else

若判断阿 Q 的成绩是优秀、合格合适还是需要重修，这是一种三选一的操作，程序用 if-else if-else 实现了三分支的选择结构。当 if()中的表达式为真时，执行 if 后{}中的语句；否则继续判断：当 else if（）中的表达式为真时，执行 else if 后{}中的语句。只有当两个条件都不满足时，执行 else{}中的语句。其实可以将这段代码改写为三个 if 语句：

```
    if(dAverage > 85 )                          /*    成绩高于 85*/
    {       printf("优秀\n");     }
    if (dAverage <=85 && dAverage>60 )   /*  成绩在 60~85 */
    {       printf("合格\n");    }
    if (       dAverage<60    )                 /*  成绩低于 60*/
```

```
{       printf("重修\n");   }
```

构造此类多分支结构要注意边界问题，如 85 分和 60 分应该属于哪个分支。合理构造判断表达式，以免出现逻辑错误。若需划分为更多的区间，可以添加其他 else if 分支，注意只能有 1 个 if 分支和 1 个 else 分支，并且 else 后没有用于判断的表达式。

此外，细心的读者通过测试可能已经应该发现该程序不够完善。如果输入非法数据，会出现怎样的结果？对程序应该进行怎样的修改才更合理？若构造更为复杂的逻辑关系，可将 if-else 语句进行嵌套，即 if()后的{}中包括 if-else 结构，或则 else{}的语句中包括 if-else 结构。为了增强程序的可读性，请注意分支结构中语句的缩进，。并增添空行使程序结构清晰易懂。

【同步练习】

1. 阿 Q 答应大毛，期末考试中数学、语文和英语都高于 98 分，就带他去动物园。输入大毛的三门功课成绩，输出能否去动物园。

2. 阿 Q 和大毛到了动物园，门票为 20 元。规定如果身高不足 120 厘米的儿童免票，如果身高在 120 厘米到 140 厘米之间要买半票，超过 140 厘米的就要全票。根据大毛的身高，阿 Q 需要拿所多少钱买票？

3. 学校对大一新生的英语选修课进行分班，根据高考英语成绩和入学摸底考试成绩，将分为“高级班”、“中级班”和“加强班”，高考成绩高于 120 分可以直接进入高级班，其他学生要参加入学摸底考试。如果摸底考试成绩高于 85 分则分入高级班，如果其成绩低于 60 分则分入加强班，其他学生在中级班学习。编写程序实现分班处理，学生按要求输入高考成绩以及摸底考试成绩，输出该生加入的班级。

第 2 章 结构化程序设计初探

在第 1 章中我们对阿 Q 同学的《C 语言》课程的成绩进行了简单的处理，实际上他选修了多门课程，若需要对这些课程的成绩进行深入而全面的统计分析，涉及到的数据将会增多，程序的规模将会扩大，导致设计的复杂度提高，因此应该采取更为科学有效的方法来规划程序。

结构化程序设计是一种程序详细设计的基本原则，其观点主要包括：以模块化设计为中心；采用自顶向下、逐步求精的程序设计方法；使用三种基本控制结构构造程序。这种设计方式可使程序层次清晰，便于使用、维护以及调试，是软件科学和产业发展的重要的里程碑。

C 语言以函数为基本功能模块，并且任何算法都可以通过顺序结构、选择结构和循环结构的组合来实现，从而使程序具有结构化的特点。本章我们了解结构化程序设计的思路，学习基本结构的组合方法，并讨论如何通过函数这种功能模块构造程序。

2.1 重复与循环语句

到目前为止的程序中，运行是时每条语句只被执行一次，然而我们在再使用大部分软件时，同一过程往往被重复多次。例如在超市收银系统中，收银员在结算时要反复输入商品的条码，一个顾客付款完后又有新的顾客来结算，又要重复刚才的过程。为了反复执行同一功能，可以使用循环结构来实现。

【抛砖引玉】

若阿 Q 同学本学期选修了《工程数学》、《中级英语》、《C 语言》、《计算机文化基础》和《体育》5 门课程，要求输入所有课程的成绩的总分。

如果按照前章的思路，可以分别定义 5 个变量存储每门课程的成绩，并 5 次调用 scanf 函数分别输入成绩，最后求得总分和平均分。但是若阿 Q 是个超人，一学期就选修了所有课程，课程数目 N 比较大，程序中将会有很多重复的代码，这些冗余的代码会使一个简单的程序臃肿而乏味。使用 while 语句构成的循环结构，轻松而简洁的处理多门课程的成绩。

例 2.1 求多门课程的总成绩

```
#include <stdio.h>
#define N 5                    /*课程数目*/
int main( )
{
   int iGrade = 0;          /*考试成绩*/
   int iSum = 0;            /*总成绩*/
   int iCounter=0;          /*计数器  */
   while( iCounter<N )      /*计数器控制循环*/
  {
    iCounter = iCounter + 1;
    printf("Input grade%d: ",iCounter);
    scanf("%d",&iGrade);
    iSum = iSum + iGrade;      /*/累加成绩*/
   }
   printf("Sum = %d ",iSum );
   return 0;
}
```

运行结果：

Input grade1: 78

Input grade2: 82

Input grade3: 79

Input grade4: 88

Input grade5: 90

Sum = 417

通过这个例程，对循环结构作初步了解。

1. 循环的的基本概念。

程序设计中的“循环”，是为解决某一问题或计算某一结果，在满足特定条件的情况下，重复执行一组操作。在 C 语言中，循环结构一般由 while 语句、do…while 语句和 for 语句来实现，三种形式可以实现同样的功能。

2. while 语句

while 语句圆括号()中的表达式是进行循环操作的条件，只要其值为逻辑真（非 0），程序的流程即可进入到循环体中，执行花括号{}中的语句序列；然后再判断条件表达式的值，若为真则重复执行循环体的操作；直到判断条件表达式的为逻辑假真（值为 0），则循环结束。

在求解总分的问题中，例 2.1 用 while 语句构成的循环结构。循环条件为 iCounter<N，只要满足该关系，就反复执行输入和累加成绩的操作。当到达循环体末尾时，程序的流程回 while 语句的‘{’后，再次执行循环体中的 4 条语句。

3. 计数器与循环条件

程序通过变量 iCounter 记录循环的次数，一般将这种整型变量称为计数器。iCounter 的值从 0 增加到 5，当第 6 次计算表达式的值时，iCounter 值为 5，不满足 iCounter<N 循环条件，循环终止。从程序中可以直观的看出循环应执行 N 次，完成 5 次输入和累加求和操作。计数器在以于执行次数确定循环结构中，常用于构造循环条件。

设计循环结构时应当注意，要保证循环执行次数是有限的，不能使其无限循环下去。驴拉磨是周而复始的，但总有将面磨好的时候，可以让它休息后拉车运货。程序中的循环是“有条件的循环”，需要谨慎设计循环条件，并在循环体中适当修改相关变量的值，使循环条件的值趋向零。

【乘胜追击】

根据学校规定，统计平均分时要考虑课程的学分，总分按照加权求和的方法计算。例如，阿 Q 同学本学期选修了 4 门课程，《工程数学》3.5 学分、《中级英语》2.5 学分、《C 语言》3 学分，《体育》1 学分，各门课程分数分别为 73、82、88 和 90，则本学期的总学分为 10，各门课程成绩占平均成绩的权重分别为 3.5/10、2.5/10、3/10 和 1/10，加权总分为（73×3.5 + 82×2.5 + 88×3 + 90×1，平均分为加权总分/总学分。编程求本学期选修课程的平均分，分别输入选修课程的数目、学分和成绩，输出平均分。

例 2.2 求 n 门课程的平均成绩

```
#include <stdio.h>
int main( )
{
  int iGrade , iNum ,   /*课程数目*/
      iCounter=0;
  float fAverage =0,    /*平均分*/
```

```
        fNum, fSum = 0 ;     /*学分和总学分*/

    printf("输入选修功课数目：");
    scanf("%d", &iNum);

    do     /*循环语句*/
    {
      printf("输入学分和分数:");
      scanf("%f,%d",&fNum, &iGrade);
      fSum += fNum;                  /*  加权和 fSum = fSum +fNum; */
      fAverage+= fNum*iGrade;    /* fAverage =fAverage+ fNum*iGrade; */
      ++iCounter ;       /* iCounter+=1; */
    } while(iCounter<iNum);   /*计数器控制循环

    fAverage /=    fSum ; /*加权平均分    fAverage = fAverage /fSum*/

    printf("平均分=%.3f\n", fAverage);
  return 0;
  }
```

运行结果

```
输入选修功课数目：4
输入学分和分数:3.5,73
输入学分和分数:2.5,82
输入学分和分数:3,88
输入学分和分数:1,90
平均分=81.450
```

通过这个例程进一步理解循环结构的用法，了解以下几点：

1. do…while 语句

和 while 语句一样，do{…}while()语句也可以构成循环结构。这种循环的特点是先执行 do 花括号{}中循环体中的操作，后计算 while（）中表达式的值，从而判断循环是否继续。注意 do…while 语句中，while()后必须以分号结束。将两种循环语句进行对比，有一系列的问题值得思考：两者的执行顺序有何和区别呢？哪种循环结构的循环体中的操作至少执行 1 次？本程序用哪种语句构造循环更合适？

2. 计数器控制循环

和例 2.1 相同，该程序也使用是用计数器控制循环次数，但程序执行前无法预测循环，需要根据用户输入的值确定循环次数。构造这种循环必须合理设置计数器初值，并在循环体中对其值作适当的修改。本程序中计数器 iCounter 的初值为 0，每次进入循环体后计数器自动增加 1，表达式的值向循环终止条件 iCounter== iNum 靠拢。若忘记修改 iCounter 的值，或者错误的修改 iCounter 的值，运行程序时会出现怎样的现象？另外此外请思考：在程序开始定义变量的时候，变量的初值的设置与计算结果有怎样的关系，哪些变量是必须进行初始化的？

3. 新的运算符

该程序使用了几个新的运算符：自增运算符（++）和复合赋值运算符（+=和/=）。第 15 行通过循环累计总学分 fSum 时，fSum += fNum 操作相当于 fSum = fSum+fNum，复合赋值

运算符+=将+和=的操作结合起来，简化化简了表达式，类似的复合运算符还有-=、*= 、/=和%=。对于自增运算符++，用于计数器 iCounter 的加 1 操作，++iCounte 相当于 iCounte += 1，也可使表达式更为简洁。

【同步练习】

1. 阿 Q 的邻居家有四个孩子：大毛、二毛和三毛，还有一个妹妹毛毛。他们每天帮助妈妈做家务，获得一毛、五毛或 1 元的零用钱。他们想送阿 Q 生日礼物，买个编写程序帮他们计算零用钱一共有多少元。分别输入四个孩子积攒的不同硬币的数目，输出零用钱总数。

2.学校实行个性化培养方案，根据学生的特点弹性选修课程。例如，对于高考英语成绩达到一定要求的学生，可以免修《中级英语》；有一定计算机基础的学生通过测验后可以免修《计算机文化基础》。在本学期计算平均分时考虑课程的学分，按加权求和的方法计算总分，平均分为总分/总学分。并且规定选修课程不得少于 3 门，其的总学分不得少于 8 学分。编程求本学期选修课程的平均分，分别输入选修课程的数目、学分和成绩，输出平均分及总学分，并判断选修学分是否达标。

2.2 基本结构的组合

按照结构化程序设计的观点，任何算法都可以通过顺序、选择和循环三种基本结构组合实现。 顺序结构中的语句无条件的依次执行，选择结构和循环都是以判断为前提的。为了使程序结构简洁清晰，结构化程序只用单入人/单出的控制结构，即每个控制结构只有一个入口和一个出口。三种基本结构通过不同的方式进行组合，利用这样的控制结构构造单入单出的程序，就很容易编写出结构良好、易于调试的程序来。

顺序、选择和循环结构可以按两种简单的方式组合：堆栈和嵌套。控制结构只是在程序中一个接一个地罗列，称为控制结构堆栈形式。在 2.1 节的同步练习中，可将程序分解为三个部分，由控制结构按堆栈的形式实现：

1. 利用循环结构经加权累加计算总分
2. 利用顺序结构计算平均分并输出
3. 利用选择结构判断是否满足学分条件

程序由三种基本结构顺序组合而成，就如同搭积木一样，将各个模块罗列起来构成房子。而有时候，程序由基本结构嵌套而成，即一个控制结构中包含一个或者多个控制结构，就如同一个套一个的俄罗斯套娃。

【抛砖引玉】

阿 Q 有几个好朋友原来都是编程的菜鸟，经过一学期的努力学习基本掌握了程序设计的思想和方法，想在《C 语言》考试中一较高下。例程 2.3 实现求若干人最高分的的功能，请分析这几种基本结构是如何嵌套的。

例 2.3 求最高成绩

```
#include <stdio.h>
int main( )
{
  int iGrade = 0,    /*考试成绩*/
  iNo = 0,           /*成绩序号*/
  iMax = 0;         /*  最高分*/
  /*  没有初始化或者初值为 100 是否合适？ */
  do
  {
    ++iNo;
```

```
        printf("Input grade %d: ",iNo);
        scanf("%d",&iGrade);
        if(iGrade>iMax) /*计算最大值*/
           iMax = iGrade;
    }while(iGrade!=-1);   /*标记控制循环*/
    printf("Highest grade is %d\n",iMax);
    return 0;
}
```

运行结果：

```
Input grade 1: 89
Input grade 2: 90
Input grade 3: 74
Input grade 4: -1
Highest grade is 90
```

程序的结构很清晰，利用循环结构分别输入成绩，利用选择结构记录最高分，由 if 语句构成的选择结构嵌套在由 do…while 语句构成的循环结构中，循环结构中包含了一个完整的选择结构。

例 2.3 中循环的执行次数是不确定的，采用标记控制循环的方法。当变量 iGrade 为特定标记值-1 时，循环条件 iGrade!=-1 的值为 0，即当 iGrade ==-1 时循环结束。用标记控制循环时，必须保证标记值为特殊值，不能和正常的有效数据混淆，如 iGrade 存储的合法分为 0～100 之间的正数，标记可设置为-1 或其他负数。当输入 3 个有效分数后输入-1，循环体被执行了 4 次。

循环结构中在求最高分时，利用 if(iGrade>iMax) iMax = iGrade 实现。变量 iGrade 存放当前分数，变量 iMax 存放历史最高分，比较两者的值，决定是否用当前分替换以往的最高分。这类似于好比冠军挑战比赛，守擂者在台上接受挑战，攻擂者总是一个个更换，若攻擂者胜利则升级为守擂者，这样所有的挑战者都比试完后，留在擂台上的是最终的冠军。但是这个 if 语句并不完善，设想当输入的成绩超过满分值，最高分是否有效呢？应该如何修正条件表达式？

【乘胜追击】

学校对大一新生的英语选修课进行分班，根据高考英语成绩和入学摸底考试成绩，将分为“高级班”、“中级班”和“加强班”，高考成绩高于 120 分可以直接进入高级班，其他学生要参加入学摸底考试。如果摸底考试成绩高于 85 分则分入高级班，如果其成绩低于 60 分则分入加强班，其他学生在中级班学习。编写程序实现分班处理，学生按要求输入高考成绩以及摸底考试成绩，输出该生加入的班级。

例 2.4 根据英语成绩进行分班

```
#include <stdio.h>
int main( )
{
    int iGrade,   /*  高考成绩*/
       iTest;      /*测试成绩*/
    printf("输入高考英语成绩(0~150):");
    scanf("%d",&iGrade);
    while(iGrade<0 || iGrade>150)   /*有效数据校验*/
    {
```

```
            printf("输入无效成绩，再次输入!");
            scanf("%d",&iGrade);
        }
        if(iGrade>120)                /*根据高考成绩分班*/
        {
            printf("直接进入高级班");
        }
        else
        {
            printf("输入摸底考试成绩(0~100):");
            scanf("%d",&iTest);
             /*根据摸底考试成绩分班*/
            if(iTest>85)
            {
               printf("进入高级班");
            }
            else if(iTest>=60)
            {
                  printf("进入中级班");
            }
            else
            {
                  printf("进入加强班");
            }
        }
        return 0;
    }
```

运行结果：

输入高考英语成绩(0~150):110

输入摸底考试成绩(0~100):88

进入高级班

在本程序中首先要输入高考分数 iGrade，用 while 语句实现数据的合法性校验，如果 iGrade 的值不在 0～150 之间，则重新输入，直到数据合法为止。接着用 if-else 语句构造双分支选择结构，初步实现分班操作。若不满足 iGrade>120 时，需要输入摸底考试成绩，并根据该成绩进一步判断分班情况。在 else 分支中，嵌套了另一个 if-else if-else 语句，构造了选择结构之间的嵌套关系。

请分析该程序中的循环结构和选择结构的组合关系，是堆栈还是组合。摸底考试成绩没有进行合法性校验，如果模仿对高考分数 iGrade 的处理方法，应该如何修改代码？新的循环结构和选择结构又是怎样的组合关系？

【同步练习】

1. 比较大毛家的四个孩子的零用钱，输出攒的最多和最少的钱数之差。

2. 阿 Q 的班级里有若干个同学，要求输入所有学生的出生年份，以负数结束，请编程统计班级人数以及所有学生的平均年龄。

2.3 模块化编程

在处理一个棘手的问题的时候，我们往往会采用“化繁为简，触类旁通，各个击破”的方式，将复杂问题分解成若干个较为简单容易的小问题，再逐个研究解决，如果遇到类似的问题，会借鉴以往的经验处理。 在前面处理成绩的程序实现的功能比较单一，所有的代码都在主函数中。但当设计规模较大的系统时，若代码都集中在一起，main 函数的负担就太重了，程序的层次结构也不清晰。可以将复杂功能分解成简单的功能，用不同的功能模块构造程序。结构化程序设计是以模块化设计为中心，将待开发的系统划分为若干个相互独立的模块，这样使完成每一个模块的工作变单纯而明确，为设计一些较大的软件打下了良好的基础。

函数是的 C 语言程序的基本功能模块，不难发现程序这是由各种的函数构成的。其实我们已经享受了函数的好处，例如要在显示器上输出数据时，我们无需知道数据是如何在内存和输入输出设备传输的，也无需知道调配数据的指令和代码是如何构造的，只需调用 printf 函数即可。除了利用标准库中的已有的函数，也可以根据量体裁衣的自定义函数，将实现一定功能的相关语句封装在函数中，方便调用和修改。

对于阿 Q 成绩的处理，若将前面几个例子的功能综合起来，既即要求阿 Q 某门课程的分数，又要输出他的各门课程中的最高分，还要将他的成绩与其他同学进行比较，都可以分别定义不同的函数分别实现。现在我们需要求阿 Q、孔乙己和祥林嫂三个人的总分和平均分，并要得到谁的平均成绩最高，例 2_5 中调用了自定义函数 sum 与 max，实现求总分和最高分的功能。

例 2.5 函数的定义和调用

```
#include <stdio.h>
/*  函数定义*/
/*求 3 个小数的最大值*/
float max( float f1, float f2, float f3)
{
   float fMax = f1;       /*定义最大值变量*/
   /*计算最大值*/
   if(f2>fMax)
     fMax = f2;
   if(f3>fMax)
     fMax = f3;
   return fMax;     /*返回最大值
}
/*函数定义，求 iNo 个整数的和*/
int sum ( int iNo )
{
   int iCounter = 0,    iNum = 0,    iSum = 0;    /*定义变量*/
   /*输入 iNo 个数据并求和*/
   printf("Input %d numbers:",iNum);
   while(iCounter<iNo)
   {
      scanf("%d",&iNum);
      iSum += iNum;
```

```
        ++iCounter;
    }
    return iSum;   /*  返回和*/
}
int main( )
{
   int iSum1,iSum2,iSum3;     /*总成绩*/
   float fAve1,fAve2,fAve3;     /*平均成绩*/
   iSum1 = sum(4);                /*函数调用，求 4 门成绩的总分*/
   fAve1 = (float)iSum1 / 4;     /*强制类型转换，求平均分*/
   printf(" average2 = %.2f\n",    fAve1);
   iSum2 = sum (5);                /*函数调用，求 5 门成绩的总分*/
   fAve2 = iSum2 / 5.0f ;          /*自动类型转换，5.0f  为 float 类型常量*/
   printf(" average2 = %.2f\n",    fAve2);
   fAve3 = (iSum3 = sum(3)) / 3.0f;           /*函数返回值参与表达式运算*/
   printf(" averaged = %.2f\n",    fAve3);
   /*函数 max 返回值作函数 printf 的参数*/
   printf("\nHighest average is %.f\n",    max(fAve1,fAve2,fAve3));
   return 0;
}
```

运行结果：

```
Input 4 numbers: 65 77 90 56
 average2 = 72.00
Input 5 numbers: 81 60 75 91 84
 average2 = 78.20
Input 3 numbers: 89 94 78
 averaged = 87.00
Highest average is 87
```

通过本例可以初步了解函数的使用方法。如果某个代码段实现特定功能，那么可以将这些相关代码用花括号{}括起来，并为这段代码其取一个合适的名字，这就构成了函数。在使用时候，只需知道函数名，并将要处理的数据传递给函数即可。用代码实现函数功能的过程称为函数定义，而使用函数的过程称为函数调用。

1. 函数的定义

函数在使用前必须进行定义，函数定义分为函数头和函数体两部分，一般语法形为

```
类型   函数名  （  形式参数列表  ）          /*  函数头*/
  {
        语句序列                          /*  函数体*/
  }
```

函数头包含三要素：函数名、参数列表和函数类型。函数名是一种标识符，要符合命名规范并保证唯一性。圆括号（）中是函数要处理的一些数据，要将数据的类型、，名字和个数依此列举出来，多个参数用逗号分割，将它们称为形式参数（简称形参）。执行函数后得到的结果称为函数返回值，函数类型即为返回值的类型。为了比较同学们的平均分，程序中定义函数 float max(float f1, float f2, float f3)，三个人的平均分通过参数分别传递给函数，返回值为三者的最大值（类型为 float）。

函数体为实现算法的语句序列，可由三种基本结构组成。函数将执行结果通过 return 语句返回，返回值可以是变量或者表达式，其类型通常和函数类型一致，且必须是唯一的。注意函数中要用到的数据，如果不是参数列表中的变量，必须在函数开始处定义。

2. 函数的调用

函数调用时，需要准确指明函数名及和要传递的参数。函数调用时圆括号（）中的参数为实际要交给函数处理的数据，称其为实际参数（简称实参），一般实参的数量和类型与函数定义时的形参列表保持一致。函数调用时将实参的值传递给形参，实参变量名与形参变量名可能不同，但顺序必须一一对应。程序 2.5 中调用函数 max 求三个平均分的最高值时，实参为变量 fAve1、fAve2 和 fAve3 将值分别传递给形参 f1、f2 和 f3。实参也可以为常量或者表达式，程序第 33 行调用 sum 求总分时，实参考试的科目数为常量，形参获得的值为 4。

一般函数调用会产生返回值，该结果可以在表达式中参与运算，如

```
iSum2 = sum (5)     /*将 5 门考试成绩的总分赋给变量 iSum2*/
fAve3 = (iSum3 = sum(3)) / 3.0f;   /*  将 3 门考试成绩的总分赋给 iSum3，并计算平均分*/
```

函数的返回值也可以作为其他函数的实参，如

```
printf("Highest average is %.f\n",   max(fAve1,fAve2,fAve3));
```

在求三个同学平均值的最高分时，输出函数 printf 的实参为 max 的返回值。若要显示阿 Q 的平均分，也可以直接利用 sum 函数实现，如

```
printf("Average = %lf\n",   sum(4) / 4.0 );
```

有的函数没有返回结果，其类型为 void，函数体中没有 return 语句。无类型函数的调用形式不能出现在表达式中，常常直接构成函数调用语句。

程序中的 main 函数中分别求 3 个人的总分，函数 sum 中的代码被反复利用，减少了主函数中的冗余代码。同一个函数还可以用在不同的代码中，实现相似的功能。如果要求 3 个人总成绩的最高分，也可通过调用 max 函数实现。如果要求三个人的平均分，也可利用 sum 函数。我们可以充分利用现有的函数作积木式的扩展，这样设计程序就变得轻松而有章法了。

【同步练习】

1. 定义并调用函数，分别实现如下功能，在主函数中输出结果：

（1）求 1 个学生 n 门功课的成绩，求他的平均分。

```
double average ( int n);     /*输入学生成绩，返回平均分
```

（2）求 3 个学生平均分最高者的序号。

```
int max( float f1, float f2, float f3);
```

（3）求全班《C 语言》的不及格率，参加考试人数不确定。

```
float count (    );       /*  输入若干学生成绩(0~`100)
```

2. 阿 Q 带大毛家的四个孩子到动物园玩，门票为 20 元。规定如果身高不足 120 厘米的儿童免票，如果身高在 120 厘米到 140 厘米之间要买半票，超过 140 厘米的就要全票。每个孩子都有若干有 5 毛和 1 元的硬币, 根据孩子们的身高和零用钱数，计算孩子们零钱的总数，阿 Q 需要再拿所多少钱买票？定义以下函数实现相关功能：

```
float ticketMoney（float height, float ticket）;  /*身高 height, 票价 ticket，返回门票钱*/
float totalMoney（int n ）;    /*   孩子数目 n，  返回零用钱总数*/
```

第 3 章 数据结构

第 2 章中初步了解结构化的程序思想，对阿 Q 成绩进行简单的处理。如果需要将他和同学们的成绩近一步的比较和分析，程序中只使用基本类型的变量会有很多困难和麻烦。为了使程序功能更加完善和实用，需要利用更为灵活的形式来存储和操作多个数据。

结构化程序设计的首创者是计算机科学家沃思(Nikiklaus Wirth)，作为 Pascal 语言之父，他提出一个著名的公式：数据结构 + 算法 = 程序。这对计算机科学的影响程度足以媲美物理学中爱因斯坦提出的质能方程，沃思因此获得 1984 年的图灵奖。这个公式揭示了程序的两个基本要素，数据结构(data structure) 是对数据的描述，算法(algorithm)即为操作步骤。为了使程序实现特定的功能，首先需要选择合适的数据类型和数据的组织形式，在此基础上构造算法。本章引入数组、结构体和文件的概念，对不同类型的多个数据进行操作。

3.1 数组

很多时候程序要处理许多个类型相同的数据，如统计期末的总成绩和平均成绩时，要录入各门课程的成绩并进行累加。阿 Q 本学期选修了 5 门课程，在例 2.1 中我们只用了 1 个变量接收成绩，但是并没有将所有数据保存下来，累加后的成绩就被后录入的成绩覆盖了。如果程序中需要再次引用这些成绩，就需要定义 5 个整型变量分别存储成绩。若要计算大学四年他所有课程的平均成绩，需要定义二、三十个变量，这样会很繁琐。若要处理全班所有学生的成绩，或者统计全校更过学生的成绩，难道要不厌其烦的定义成百上千个变量吗？为了避免定义多个变量的繁琐操作，程序中常使用数组（array）存储多个数据。数组是一种自定义的数据类型，它可方便的对类型相同的多个数据进行操作。

【抛砖引玉】

例 3.1 求 N 门课程的平均成绩

```
#include <stdio.h>
#define N 5                    /*课程数目*/
int main( )
{
   int aGrade[N] ;             /*成绩数组*/
   int i = 0;                  /*计数器*/
   float fAve = 0;             /*平均成绩*/
   printf("Input %d grade:",N);
   while( i<N )                /*遍历数组*/
    {
      scanf("%d",&aGrade[i]);      /*输入数组元素值*/
      fAve = fAve + aGrade[i];
      ++i;
    }
    printf("Average = %.2f\n", fAve / i);
    return 0;
}
```

运行结果：

```
Input 5 grade:87 74 66 91 56
Average = 74.80
```

该程序演示了数组的使用方法，学习以下两点：

1. 数组的定义

利用整型数组 aGrade 实现多个成绩的存储，定义数组时需要指定数组的大小和类型。类型 int 表示数组中所有数据都为整数，而方括号[]中必须是正整型常量，表示数组的长度，即包含多少个数据。现在可以不必单独定义多个变量 iGrade1、iGrade2 …. iGrade5，统一定义成 int aGrade[5] 即可。

2. 数组的引用

数组中的每个数据称为数组元素，aGrade[i]表示数组中的第 i+1 个元素。引用数组元素时方括号[]中的整数称为下标，可以为整型常量或者变量，注意下标的范围为 0~ N-1。 数组元素的使用形式像变量一样，可以通过运算符进行各种运算，也可以作为参数实现输入输出操作。

【乘胜追击】

阿 Q 和他的 4 个好朋友约定，期末成绩排名最后的请成绩最好的吃饭。如何利用数组比较 5 个人成绩，并输出最高分和最低分？

例 3.2 求 5 个学生的最高成绩和最低成绩

```
#include <stdio.h>
int main()
{
    float   aGrades[5]={ 78.81, 88.56, 67.0,91.2,56.4}; /*  平均分数组*/
    float   fMax = 0, fMin = 100 ;
    int i = 0;
   /*  遍历数组*/
    do
    {
        printf("%6.2f",aGrades[i]);
        if(aGrades[i]>fMax)     /*  比较最高分*/
            fMax = aGrades[i];
        if(aGrades[i]<fMin)     /*  比较最低分*/
            fMin = aGrades[i];
        ++i ;
    }while( i < 5);

    return 0;
}
```

运行结果：

```
78.81 88.56 67.00 91.20 56.40
max = 91.20,min = 56.40
```

通过程序可进一步了解数组的使用方法：

3. 数组的初始化

变量在定义时可以获得初值，如 float fMax = 0, fMin = 100，同样定义数组平均分数组 aGrades 时，可用花括号{}列举出数组元素的值，这称为数组初始化，形式如下：

```
float   aGrades[5] = { 78.81, 88.56, 67.0,91.2,56.4};
```

初始化后第一个数组元素 aGrade[0]的值为 78.81，最后一个数组元素 aGrade[4]的值为 56.4。注意花括号中数组的个数不能超过数组长度。

4. 数组的遍历

定义数组时可以统一对所有元素赋值，通常使用循环结构逐个访问数组中的所有元素，并用计数器变量控制循环。例 3.1 中，用 while 语句访问数组，在循环体中依次输入 5 个元素的值并进行累加；例 3.1 中使用在 do…while 语句访问数组的每个元素，这称为数组的遍历。

【同步练习】

大毛要参加歌唱比赛，有 10 个评委给她打分，实际得分是去掉最高分和最低分的平均分。利用数组编写程序，输入 10 个分数，输出最高分、最低分和实际得分。

3.2 结构体

当需要描述阿 Q 的姓名、年龄、成绩时，需要分别定义变量 aName[20]、iAge、fScore 保存数据，而这几个变量之间关系逻辑上是松散的，并不能体现出他们在逻辑上是个有机的整体，也就是说不能反应出他们是在描述同一个食物的不同属性。那么怎么能够使得他们形成一个整体呢？很自然地希望有一种类型，这种类型是个复杂一点的结构 Student，在这个结构 Student 中包含姓名、年龄、成绩三个成员，每个成员不能独立存在，他们都要依附于 Student，这样同一个 Student 的姓名、年龄、成绩，就会成为逻辑上相互依存的有机整体。Student 采用数组是不可以的，因为数组要求他的所有成员必须为相同数据类型，为此 C 语言引入了结构体数据类型。

在例 3.3 中定义一个描述学生阿 Q 的结构体，然后输入并输出其姓名、年龄和成绩。

【抛砖引玉】

例 3_3 定义阿 Q 的结构体，然后输入并输出其姓名、年龄和成绩

```
#include <stdio.h>
/*开始声明结构体 Student*/
/*结构体声明只是在声明一种数据类型（作用相当于 int、float），
并没有存放数据的空间，只有用该类型定义变量时才有存放数据空间*/
struct Student
{
    char aName[20];              /*姓名*/
    int iAge;                    /*年龄*/
    float fScore;                /*成绩*/
}; /*不要忘记此处的分号（;）*/
/*结束声明结构体 Student */
int main( )
{
    struct Student sAhQ;     /*定义结构体变量 sAhQ，具备数据空间*/
    /*输入 sAhQ 的姓名，注意此处不需要取地址符&，数组名本身为地址
    sAhQ.aName 中的.为成员运算符，表示 sAhQ 的 aName*/
    printf("输入阿 Q 的姓名、年龄、成绩:\n");
    scanf("%s",sAhQ.aName);     /*输入 sAhQ 的姓名*/
    scanf("%d",&sAhQ.iAge);   /*输入 sAhQ 的年龄*/
    scanf("%f",&sAhQ.fScore);   /*输入 sAhQ 的成绩*/
    printf("输出阿 Q 的姓名、年龄、成绩:\n");
    printf("%s\n",sAhQ.aName); /*输出 sAhQ 的姓名*/
    printf("%d\n",sAhQ.iAge); /*输出 sAhQ 的年龄*/
```

```
    printf("%.2f\n",sAhQ.fScore); /*输出 sAhQ 的成绩*/
    return 0;
}
```

运行结果：

输入阿 Q 成绩:

AhQ

20

92.5

输出阿 Q 成绩:

AhQ

20

92.50

通过例 3.4，对结构体的使用要注意以下几点：

1. 结构体类型用途

结构体是用来描述具有多个成员数据的自定义数据类型，并且不要求所有成员数据类型一致。

2. 结构的声明与结构体变量定义

结构的声明是用来说明一个数据类型，并没有数据空间，好比是制作一个模具。结构体变量定义才真正开辟了空间，好比用模具制作了一个产品。

3. 结构体成员的引用

结构体成员的引用通过成员运算符.引用。具体格式为：

结构体变量.成员

4. 结构体输入、输出

结构体不可以整体输入输出，只能使用结构体成员分别输入、输出，但是结构体变量间可以整体赋值。如：

```
struct Student s1,s2;
s2=s1;
```

【乘胜追击】

在例 3.2 中，假如阿 Q 和他的 4 个好朋友约定，期末成绩排名最后的请成绩最好的吃饭，利用数组解决。但是，其中只是处理了成绩数据，并没有记录每个人的姓名和年龄信息，因此无法在程序中输出谁请谁吃饭。有了结构体之后，把数组的每个成员定义为学生结构体，就可以输出姓名信息。

例 3.4 计算 5 个学生的最高成绩和最低成绩

```
#include <stdio.h>
struct Student
{
    char aName[20];             /*姓名*/
    int iAge;                   /*年龄*/
    float fScore;               /*成绩*/
}; /*不要忘记此处的分号（;）*/
/*结束声明结构体 Student*/
int main()
{
    struct Student aStudents[5]={{"AhQ",19,78.81f},{"AhX",20,88.56f},{"AhY",18,67.0f},
```

```
        {"AhZ",20,91.2f},{"AhD",20,56.4f}}; /*  学生数组*/
        float fMaxValue=0, fMinValue=100;/*成绩最高值和最低值*/
        int fMaxNumber=0, fMinNumber=0; /*假定成绩最高和最低同学的下标都为 0*/
        int i = 0;
        /*  遍历数组*/
        do
        {
            printf("%s\t",aStudents[i].aName);     /*输出姓名，跳过一个制表位*/
            printf("%d\t",aStudents[i].iAge);      /*输出年龄，跳过一个制表位*/
            printf("%.2f\n",aStudents[i].fScore);  /*输出成绩，换到下一行*/
            if(aStudents[i].fScore>fMaxValue)      /*比较最高分*/
            {
                fMaxValue=aStudents[i].fScore;     /*获取新的最高值*/
                fMaxNumber = i;                          /*获取新的最高值下标*/
            }
            if(aStudents[i].fScore<fMinValue)      /*比较最低分*/
            {
                fMinValue=aStudents[i].fScore;     /*获取新的最低值*/
                fMinNumber = i;                          /*获取新的最低值下标*/
            }
            ++i ;
        }while( i < 5);
        printf("%s 成绩最高,%s 成绩最低\n",aStudents[fMaxNumber].aName,
            aStudents[fMinNumber].aName);
        printf("%s 请%s 吃饭\n",aStudents[fMinNumber].aName,
            aStudents[fMaxNumber].aName);
        return 0;
    }
    运行结果：
    AhQ      19      78.81
    AhX      20      88.56
    AhY      18      67.00
    AhZ      20      91.20
    AhD      20      56.40
    AhZ 成绩最高,AhD 成绩最低
    AhD 请 AhZ 吃饭
```

在例 3.4 中，数组的每一个成员都为一个学生结构体类型 Student，注意对数组 aStudents 的初始化采用的是用 5 个大括号作为每个结构体的初值。

【同步练习】

大毛有兄弟五人，分别叫大毛、二毛、三毛、四毛、五毛、每个人都有自己的零用钱，写一个程序，输出零用钱最多和最少者的姓名。

3.3 动态数组

在数组部分强调过，数组定义时必须用常量指定大小，因此，不可以在程序运行过程中

根据需要定义数组的大小。如：

```
int i1;
scanf("%d",&i1);
int aArr[i1];
```

用变量 i1 来定义数组的大小是不允许的。

注：事实上 C89 标准中，也不允许在一个执行语句之后定义变量、数组等。

【抛砖引玉】

阿 Q 和同学们每个人学习的课程门数是不同的，希望写一个程序，首先输入自己所学课程门数，然后输入每门课程的成绩并用数组保存起来，最后计算平均成绩并输出。为了让阿 Q 和同学都能够使用该程序，而每个人的课程门数不一样，所以程序中保存成绩的数组的长度必须是可变的。

例 3.5 变长数组计算平均成绩

```
#include<stdio.h>
#include<stdlib.h>     /*使用 malloc 函数时需要此头文件*/
int main()
{
    int i1;
    int iNumber;            /*可课程门数*/
    float fSum=0,fAve;
    float *pScore;
    printf("输入课程门数：\n");
    scanf("%d",&iNumber);
    /*申请 iNumber 个 float 类型大小的空间，用 pScore 记载该空间起始地址*/
    pScore=(float*)malloc(sizeof(float)*iNumber);
    /*输入 iNumber 门课程成绩*/
    printf("输入%d 门课程成绩：\n",iNumber);
    for(i1=0;i1<iNumber;i1++)
        scanf("%f",&pScore[i1]);
    /*计算 iNumber 门课程平均成绩*/
    for(i1=0;i1<iNumber;i1++)
        fSum=fSum+pScore[i1];
    fAve=fSum/iNumber;
    printf("%d 门课程平均成绩为%.2f",iNumber,fAve);
    free(pScore); /*释放有 malloc 申请的空间*/
    return 0;
}
```

例 3.5 中需要了解以下几个问题：

1. 用到动态空间申请与释放时，需要头文件 stdlib.h。

2. sizeof 函数为计算某种数据类型所占字节数的函数。

3. malloc 函数申请的是一块连续的空间，返回值为该空间的起始地址，是一个空类型的地址需要转换为指定类型，括号为强制转换运算符。

4. float *pScore 定义一个用来存储 flaot 类型空间的起始地址，与数组名类似，可以与数组名一样使用。

5. 当空间使用完毕后，应该使用函数 free 释放。

【乘胜追击】

阿 Q 所在学校每个班级的学生人数目是不同的，希望写一个程序，首先输入班级的学生人数，然后输入该班每个同学的姓名、年龄、成绩并用数组保存起来，最后计算并输出成绩最高和最低的同学的姓名。为了让阿 Q 所在学校的每个班级都能够使用该程序，而每个班级的学生人数不一样，所以程序中保存同学的数组的长度必须是可变的。

例 3.6 输出某班级学生的最高成绩和最低成绩的姓名

```
#include <stdio.h>
#include <stdlib.h>
struct Student
{
    char aName[20];                 /*姓名*/
    int iAge;                       /*年龄*/
    float fScore;                   /*成绩*/
}; /*不要忘记此处的分号（;）*/
/*结束声明结构体 Student*/
int main()
{
    struct Student *pStudents;          /*指向学生结构体的指针*/
    float fMaxValue=0, fMinValue=100;/*成绩最高值和最低值*/
    int fMaxNumber=0, fMinNumber=0;   /*假定成绩最高和最低同学的下标都为 0*/
    int i1 = 0;
    int iNumber;                            /*学生人数*/
    printf("输入学生人数：\n");
    scanf("%d",&iNumber);
    /*申请 iNumber 个 Student 类型大小的空间,用 pStudents 记载该空间起始地址*/
    pStudents=(struct Student*)malloc(sizeof(struct Student)*iNumber);
    /*输入 iNumber 个同学信息*/
    printf("输入%d 个同学信息：\n",iNumber);
    for(i1=0;i1<iNumber;i1++)
    {
        printf("输入同学信息：姓名、年龄、成绩\n");
        scanf("%s",pStudents[i1].aName);          /*输入某同学的姓名*/
        scanf("%d",&pStudents[i1].iAge);          /*输入某同学的年龄*/
        scanf("%f",&pStudents[i1].fScore);       /*输入某同学的成绩*/
    }
    /*遍历数组*/
    i1=0;
    printf("输出同学信息：姓名、年龄、成绩\n");
    do
    {
        printf("%s\t",pStudents[i1].aName);     /*输出姓名，跳过一个制表位*/
        printf("%d\t",pStudents[i1].iAge);       /*输出年龄，跳过一个制表位*/
        printf("%.2f\n",pStudents[i1].fScore);   /*输出成绩，换到下一行*/
        if(pStudents[i1].fScore>fMaxValue)       /*比较最高分*/
```

```
            {
                fMaxValue=pStudents[i1].fScore;     /*获取新的最高值*/
                fMaxNumber = i1;                    /*获取新的最高值下标*/
            }
            if(pStudents[i1].fScore<fMinValue)      /*比较最低分*/
            {
                fMinValue=pStudents[i1].fScore;     /*获取新的最低值*/
                fMinNumber = i1;                    /*获取新的最低值下标*/
            }
            ++i1 ;
        }while( i1 < iNumber);
        printf("%s 成绩最高,%s 成绩最低\n",pStudents[fMaxNumber].aName,
            pStudents[fMinNumber].aName);
        free(pStudents);
        return 0;
    }
```

运行结果：

```
输入学生人数：
3
输入 3 个同学信息：
输入同学信息：姓名、年龄、成绩
AhX 20 78
输入同学信息：姓名、年龄、成绩
AhY 21 87
输入同学信息：姓名、年龄、成绩
AhZ 22 99
输出同学信息：姓名、年龄、成绩
AhX        20          78.00
AhY        21          87.00
AhZ        22          99.00
AhZ 成绩最高,AhX 成绩最低
```

在例 3.6 中数组成员换成了学生结构体（Student），空间的分配与回收道理是一样的。

【同步练习】

大毛有兄弟五人，分别叫大毛、二毛、三毛、四毛、五毛、每个人都有自己的故事书（每本书有书名、页数），但是每个人的本数不一样。写一个程序，使得 5 个兄弟都能用该程序找出自己的页数最多和页数最少的书的书名。

3.4 文件

首先介绍一下文件的概念，所谓“文件”一般是指存储在外部介质上数据的集合。文件是以数据的形式存放在外部介质（如磁盘）上的，操作系统是以文件为单位对数据进行管理的，也就是说，如果想找存在外部介质上的数据，必须先按文件名找到指定的文件，然后再从该文件中读取数据。

操作系统中的文件标识包括三部分：

（1） 文件路径：表示文件在外部存储设备中的位置

（2） 文件名：文件命名规则遵循标识符的命名规则

（3） 文件扩展名：用来表示文件的性质(.txt .dat .c)

例如： d: \ c++ \ temp \ file1.dat

↑ ↑ ↑

文件路径 文件名 文件扩展名

在利用数组和结构体撰写程序时，运行时输入了许多数据，但是下一次运行时数据都没有了，还需要重新输入，怎么才能把数据永久保存起来呢？这就需要用文件来解决。

【抛砖引玉】

阿 Q 希望把自己班的的 5 个同学（姓名、年龄、成绩）的信息输入，并永久保存起来，以后使用时，不需要重新输入。

例 3.7 把结构体数组写入文件

```
#include<stdio.h>
struct Student
{
    char aName[20];                 /*姓名*/
    int iAge;                       /*年龄*/
    float fScore;                   /*成绩*/
};
/*结束声明结构体 Student*/
int main()
{
    struct Student aStudents[5]; /*定义结构体数组*/
    FILE *fp;
    int i1;
    float f=1.0;
    /*输入 5 个同学信息*/
    for(i1=0;i1<5;i1++)
    {
        printf("输入第%d 同学信息：姓名、年龄、成绩\n",i1);
        scanf("%s",aStudents[i1].aName);          /*输入某同学的姓名*/
        scanf("%d",&aStudents[i1].iAge);          /*输入某同学的年龄*/
        scanf("%f",&aStudents[i1].fScore);       /*输入某同学的成绩*/
    }
    if((fp=fopen("file1.txt","wb"))==NULL)       /*打开磁盘文件*/
    {
        printf("\nCannot open file strike any key exit!");
        getch();          /*等待敲键盘，为显示上一句话*/
        exit(1);          /*结束程序*/
    }
    /*把 5 条学生记录写到文件中，每条记录大小为 sizeof(struct student)，写出时并不
    考虑内容是什么，把内存空间的 5 倍的 sizeof(struct student)个字节写到文件中*/
    fwrite(aStudents,sizeof(struct Student),5,fp);
    fclose(fp);             /*关闭磁盘文件*/
    return 0;
```

```
}
```

运行结果：

```
输入第 0 同学信息：姓名、年龄、成绩
AhA 19 60
输入第 1 同学信息：姓名、年龄、成绩
AhB 20 75
输入第 2 同学信息：姓名、年龄、成绩
AhC 21 70
输入第 3 同学信息：姓名、年龄、成绩
AhD 20 90
输入第 4 同学信息：姓名、年龄、成绩
AhE 21 88
```

在例 3.7 中，首先把 5 个同学的数据写到内存数组中，然后通过 fopen 函数打开文件，其中“file1.txt”为磁盘文件名，“wb”说明文件是以写二进制文件的方式打开，也就是说文件是用来写入数据的。if 是用来判断文件是否成功打开，若不成功返回 NULL，程序通过 exit 退出，不执行下面的文件写操作了。若文件打开成功，把文件地址用 fp 存储，以后就可以利用 fp 使用文件。fwrite 函数的作用是把从 aSttudents 的内存地址开始 5 倍的 sizeof(struct student)个字节写到 fp 所指的文件中。文件使用完毕，利用 fclose 函数关闭，目的是断开程序与磁盘文件 file1.txt 的联系，一遍其他程序能够使用 file1.txt 文件。

可以看出，文件操作分 3 部分：

1. 打开文件
2. 读写文件
3. 关闭文件

把数据写进磁盘文件后，以后使用数据时就可以直接从文件中读出来，不必要再重新输入。

例 3.8 把文件中的同学数据读出并输出到标准输出设备上

```
#include<stdio.h>
struct Student
{
    char aName[20];                 /*姓名*/
    int iAge;                       /*年龄*/
    float fScore;                   /*成绩*/
};
/*结束声明结构体 Student*/
int main()
{
    struct Student aStudents[5]; /*定义结构体数组*/
    FILE *fp;
    int i1;
    if((fp=fopen("file1.txt","rb"))==NULL)       /*打开磁盘文件*/
    {
        printf("\nCannot open file strike any key exit!");
        getch();          /*等待敲键盘，为显示上一句话*/
        exit(1);          /*结束程序*/
```

```
    }
    /*从文件中读 5 条学生记录写到数组 aStudents 中*/
    fread(aStudents,sizeof(struct Student),5,fp);
    /*输出 5 个同学信息*/
    for(i1=0;i1<5;i1++)
    {
        printf("输出第%d 同学信息：姓名、年龄、成绩\n",i1);
        printf("%s\t",aStudents[i1].aName);        /*输出某同学的姓名*/
        printf("%d\t",aStudents[i1].iAge);         /*输出某同学的年龄*/
        printf("%.2f\n",aStudents[i1].fScore);     /*输出某同学的成绩*/
    }
    fclose(fp);          /*关闭磁盘文件*/
    return 0;
}
```

运行结果：

```
输出第 0 同学信息：姓名、年龄、成绩
AhA     20      60.00
输出第 1 同学信息：姓名、年龄、成绩
AhB     21      75.00
输出第 2 同学信息：姓名、年龄、成绩
AhC     22      70.00
输出第 3 同学信息：姓名、年龄、成绩
AhD     21      90.00
输出第 4 同学信息：姓名、年龄、成绩
AhE     22      88.00
```

例 3.8 中，文件打开方式“rb”，是读二进制文件的方式打开。fread 函数的作用是从 fp 所指的磁盘文件中读出 5 倍的 sizeof(struct student)个字节写到数组 aSttudents 对应的内存地址中。最后对数组遍历输出所有元素。

【乘胜追击】

时间过了一年，阿 Q 的所有同学都长了一岁，希望把文件中所有同学的年龄增加 1。

例 3.9 把文件中的数据读出修改后再写回文件*/

```
#include<stdio.h>
struct Student
{
    char aName[20];          /*姓名*/
    int iAge;                /*年龄*/
    float fScore;            /*成绩*/
};
/*结束声明结构体 Student*/
int main()
{
    struct Student aStudents[5]; /*定义结构体数组*/
    FILE *fp;
    int i1;
```

```
    /*********************************************
          读文件数据写到结构体数组
    *********************************************/
    if((fp=fopen("file1.txt","rb"))==NULL)/*以读方式打开文件*/
    {
        printf("\nCannot open file strike any key exit!");
        getch();          /*等待敲键盘，为显示上一句话*/
        exit(1);          /*结束程序*/
    }
    /*从文件中读 5 条学生记录写到数组 aStudents 中*/
    fread(aStudents,sizeof(struct Student),5,fp);
    /*********************************************
          /*更改结构体数组中的年龄信息*/
    *********************************************/
    for(i1=0;i1<5;i1++)
        aStudents[i1].iAge=aStudents[i1].iAge+1;     /*年龄加 1*/
    fclose(fp);           /*关闭磁盘文件*/
    if((fp=fopen("file1.txt","wb"))==NULL)/*再次以写方式打开文件*/
    {
        printf("\nCannot open file strike any key exit!");
        getch();          /*等待敲键盘，为显示上一句话*/
        exit(1);          /*结束程序*/
    }
    /*********************************************
          写到结构体数组数据到文件中，会覆盖掉原来的数据
    *********************************************/
    fwrite(aStudents,sizeof(struct Student),5,fp);
    fclose(fp);           /*关闭磁盘文件*/
    return 0;
}
```

例 3.9 从功能角度分为 3 部分：

1. 以读方式打开文件，读文件数据到内存数组中，然后关闭文件。
2. 更改数组中的数据。
3. 以写方式打开文件，这时文件会清空，然后把数组写到文件中，相当于更改了原来文件中的数据。

【同步练习】

大毛有兄弟五人，分别叫大毛、二毛、三毛、四毛、五毛。

1. 把每个人的姓名和零用钱数写到文件中。
2. 从文件中读出数据，输出零用最多者的姓名。
3. 过年了，父母给每人 100 元压岁钱，更改文件中原有的零用钱数。

第 4 章 算法描述和编码规范

通过前几章的编码练习，对编写程序已经有了一个基本的了解。但是，为了使编程过程规范有序需要对算法和编码规范有进一步的了解。

4.1 程序设计与算法描述

4.1.1 程序设计与算法

盖房子，首先要有想法，然后画建筑图纸，最后才一砖一瓦地盖房子。编写程序同样道理，首先仔细考虑解决问题思路并描述出来，然后画出流程图，最后才是一行一行的编码。切忌上来就编写代码，因为没有算法描述和流程图就编码如同盖房子没有图纸就直接盖，通常会以失败结束的。

1. 算法的概念

为解决某一个问题而采取的方法和步骤，称为算法。或者说算法是解决一个问题的方法的精确描述。

2、算法的特点：① 有穷性：必须在执行了有穷个计算步骤后终止；② 确定性：每一个步骤必须是精确的、无二义性的；③ 可行性：可以用计算机解决、能在有限步、有限时间内完成；④ 有输入：⑤ 有输出：

例 4.1 交换两个大小相同的杯子中的液体（A 水、B 酒）。

算法：

1. 再找一个大小与 A 相同的空杯子 C；
2. A->C；
3. B->A；
4. C->B

例 4.2 输入 1 个数给计算机，打印其绝对值。

算法：

1. 输入 1 个数->X；
2. 若 X>0

   则　打印 X；

   否则 打印-X。

例 4.3 计算 5 的阶乘。

算法：

1. 找两个容器 T 和 I；T 为累乘器，初值为 1；I 为计数器，初值为 1。
2. 当 I 小于等于 5 时循环

   2.1 T*I->T；

   2.3 I+1->I；
3. 输出 T

以上只是一些小问题，对于一些规模较大的软件，通常将待开发的软件系统划分为若干个相互独立的模块，这样使完成每一个模块的工作变单纯而明确。这就是结构化程序设计。结构化程序设计由迪克斯特拉(E.W.dijkstra)在 1969 年提出，是以模块化设计为中心，将待开发的软件系统划分为若干个相互独立的模块，这样使完成每一个模块的工作变单纯而明确，为设计一些较大的软件打下了良好的基础。由于模块相互独立，因此在设计其中一个模块时，不会受到其它模块的牵连，因而可将原来较为复杂的问题化简为一系列简单模块的设

计。模块的独立性还为扩充已有的系统、建立新系统带来了不少的方便，因为我们可以充分利用现有的模块作积木式的扩展。

按照结构化程序设计的观点，任何算法功能都可以通过由程序模块组成的三种基本程序结构的组合：顺序结构、选择结构和循环结构来实现。顺序结构就是从头到尾依次执行每一个语句；分支结构根据不同的条件执行不同的语句或者语句体；循环结构就是重复地执行语句或者语句体，达到重复执行一类操作的目的。

结构化程序设计的基本思想是采用“自顶向下、逐步求精”的程序设计方法和“单入口单出口”的控制结构。自顶向下、逐步求精的程序设计方法从问题本身开始，经过逐步细化，将解决问题的步骤分解为由基本程序结构模块组成的结构化程序框图。“单入口单出口”的思想认为一个复杂的程序，如果它仅是由顺序、选择和循环三种基本程序结构通过组合、嵌套构成，那么这个新构造的程序一定是一个单入口单出口的程序。据此就很容易编写出结构良好、易于调试的程序来。

描述一个结构化程序设计的算法通常有以下几种表示形式：

（1） 文字描述：二义性，如：小王个子很高，到底多少算高呢？应该小王身高超过1.8 米。前面对三个例子的描述就是文字描述。

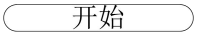
（2） 伪代码：用符号，不直观，比如类 C 语言描述。

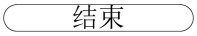
（3） 流程图：简洁、直观、无二义性。主流的流程图有传统的 FC（Flow Chart）流程图、NS 盒图、PAD 判断图。下面介绍 FC 流程图和 NS 盒图。

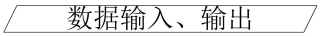
4.1.2 FC 流程图

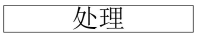
程序流程图是人们对解决问题的方法、思路或算法的一种图形化描述。

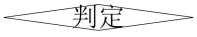
1. 流程图采用的符号

（1）起始框 

（2）终止框 

（3）输入输出 

（4）处理框 

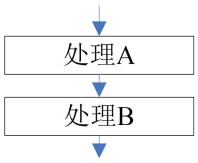
（5）判别框 

（6）流程线

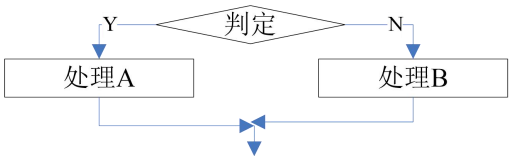
（7）注释 

2. 用程序流程图描述三种基本结构

顺序结构：

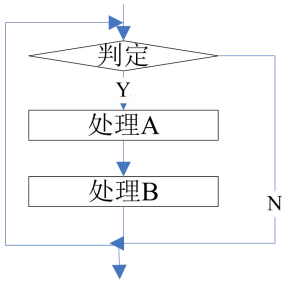


选择结构：

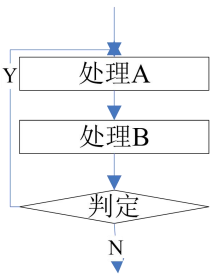


循环结构：

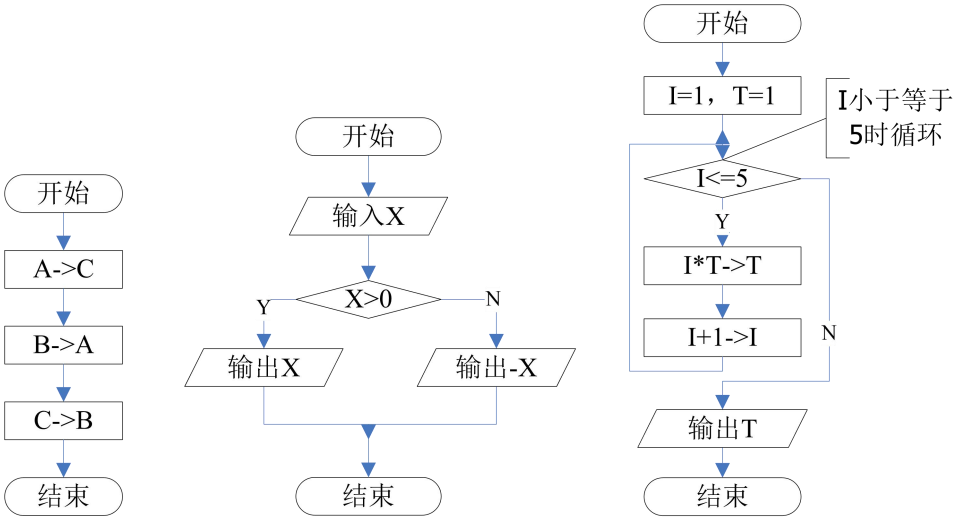
当型循环



直到型循环



例 4.1、4.2、4.3 的流程图如图所示。



程序流程图优点：独立于任何一种程序设计语言，比较直观、清晰，易于学习掌握。

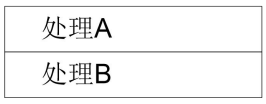
程序流程图缺点：流程图所使用的符号不够规范，常常使用一些习惯性用法。特别是表示程序控制流程的箭头可以不受任何约束，随意转移控制。当上下的流程线比较多时，像一团乱麻，阅读性非常差。

4.1.3 NS 盒图

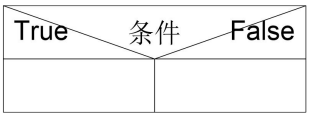
1973 年美国学者 I.Nassi 和 B.Sneiderman 提出了一种新的流程图形式。流程图中去掉了带箭头的流程线，全部算法写在一个矩形框内，在该框内还可以包含其他的从属于它的框。这种流程图称为 N-S 流程图（N 和 S 就是这两位美国学者的英文姓氏的首字母）。

NS 盒图描述三种基本结构

顺序结构：



选择结构：

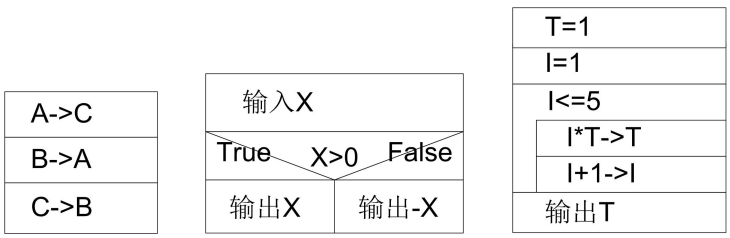


循环结构：

当型循环

直到型循环

例 4.1、4.2、4.3 的 NS 盒图如图所示



NS 盒图优点：首先，它强制设计人员按结构化方法进行思考并描述他的设计方案，因

为除了表示几种标准结构的符号之处，它不再提供其他描述手段，这就有效地保证了设计的质量，从而也保证了程序的质量；第二，NS 图形象直观，具有良好的可见度。例如循环的范围、条件语句的范围都是一目了然的，所以容易理解设计意图，为编程、复查、选择测试用例、维护都带来了方便；第三，NS 图简单、易学易用。

NS 图盒的缺点：手工修改比较麻烦，这是有些人不用它的主要原因。

本书采用 NS 图盒描述程序流程。

4.2 C 语言编码规范

写文章需要一定格式规范，编写程序也需要有一定的编码规范。具有良好编码规范的程序应该是结构清晰、容易阅读的，因此便于调试和修改。每个人的编码习惯各不相同，但是都应以编码的结构清晰、便于阅读为目的。下面总结一些编码过程中的规范。

1. 排版

（1） 程序块要采用缩进风格编写，缩进一个 TAB 键或 4 个空格或两个空格（用 TAB 键最好便于对齐，本书中为便于排版，使用的是两个空格）。

（2） 相对独立的程序块之间应该加注释。

（3） 较长的语句（>80 字符）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

（4） 循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

（5） 若函数中的参数较长，则要进行适当的划分。

（6） 尽量不要把多个短语句写在一行中，即一行只写一条语句。

（7） if、while、for、default、do 等语句自占一行。

（8） 函数的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求。

（9） 程序块的分界符（如'{'和'}'）应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、循环、分支、结构的定义、枚举的定义采用如上的缩进方式。

（10）在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如->、*），后不应加空格。

（11）程序结构清析，简单易懂，单个函数的程序行数不得超过 100 行（一般函数尽量不要超过一屏，便于阅读和调试）。

2 注释

（1） 一般情况下，源程序有效注释量必须在 20%以上（根据算法难度）。注释的原则是有助于对程序的阅读理解，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

（2） 说明性文件（如头文件.h 文件、.inc 文件等）头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

（3） 源文件头部应进行注释，列出：版权说明、版本号、生成日期、作者、模块目的/功能、主要函数及其功能、修改日志等。

（4） 函数头部（在函数之前）应进行注释，列出：函数的目的/功能、输入参数、输出参数、返回值、调用关系（函数调用）等。

（5） 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

（6） 注释的内容要清楚、明了，含义准确，防止注释二义性。

（7） 避免在注释中使用缩写，特别是非常用缩写。

（8） 注释应与其描述的代码相近，对代码的注释应放在其上方或右方（对单条语句的注释）相邻位置，不可放在下面。

（9） 对于所有有物理含义的变量、常量，如果其命名不是充分自注释的，在声明时都必须加以注释，说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方。

（10） 数据结构声明（包括数组、结构、类、枚举等），如果其命名不是充分自注释的，必须加以注释。对数据结构的注释应放在其上方相邻位置，不可放在下面；对结构中的每个域的注释放在此域的右方。

（11） 全局变量要有较详细的注释，包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明。

（12） 注释与所描述内容进行同样的缩排。

说明：可使程序排版整齐，并方便注释的阅读与理解。

（13） 将注释与其上面的代码用空行隔开（本书为了内容紧凑位采用）。

（14） 对变量的定义和分支语句（条件分支、循环语句等）必须编写注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

（15） 对于 switch 语句下的 case 语句，如果因为特殊情况需要处理完一个 case 后进入下一个 case 处理，必须在该 case 语句处理完、下一个 case 语句前加上明确的注释。

说明：这样比较清楚程序编写者的意图，有效防止无故遗漏 break 语句。

3 标识符命名

（1） 标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

如：

```
char studentName[16];
float area;
```

（2） 命名中若使用特殊约定或缩写，则要有注释说明。

如：

```
float pi;     /*数学中的π*/
```

（3） 自己特有的命名风格，要自始至终保持一致，不可来回变化。

（4） 对于变量命名，禁止取单个字符（如 i、j、k...），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但 i、j、k 作局部循环变量是允许的。本书在第一篇读者还不熟练时，采用类型加变量含义的方式（iNum、fScore），这样写输入输出不容易犯错误；在第二篇和第三篇，直接采用变量含义。

（5） 命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用 UNIX 的全小写加下划线的风格或大小写混排的方式，不要同时使用大小写与下划线混排的方式。本书采用的是大小写混排的方式。

习题

1. 对例 2.5 分别画流程图和 NS 盒图。
2. 对例 2.5 按照本节注释规范编写注释。