第 1 章　程序设计基础

学习程序设计是一件非常辛苦的事情，要有非常强的耐心。学习程序设计除要求具有一定的数学知识外，要学习和熟练掌握一门与人们习惯使用的自然语言非常不一致的程序语言，要能用程序语言熟练描述问题求解的方法，要学习程序设计的许多常规方法，要求不断学习计算机的新知识和新内容，还要求反复地上机实践。对于初学者来说，最困难的可能还是很难适应描述计算机算法的思维习惯，人们几乎无法承受程序必须将算法描述得几乎绝对地精细和精确。但对计算机来说，这又是非常必要的。一个计算机系统能正确完成预定的任务，除要有性能高、质量可靠的计算机等硬件设备外，还要为计算机系统设计功能齐全、使用方便的程序。计算机运行的过程就是计算机执行程序的过程。要让计算机去完成一项新的任务，就必须为它编写一个能让计算机正确完成该项新任务的程序。开发一个程序，特别是一个软件系统，是一件非常复杂的工作，要经历许多阶段。对于一个功能相对简单的计算任务来说，编写一个相对简单的程序，则是程序设计和程序编码阶段的任务。培养开发软件系统的能力，这要从设计和编写简单的程序开始，所以学习程序设计是达到掌握开发软件系统技术这个大目标的第一步。

1.1　程序设计基本概念

在介绍程序设计技术之前先介绍一些与程序设计有关的基本概念。

程　　序

要使计算机能完成人们预定的工作，就必须把要完成工作的具体步骤编写成计算机能执行的一条条指令。计算机执行这个指令序列后，就能完成指定的功能，这样的指令序列就是程序。所以，程序就是供计算机执行后能完成特定功能的指令序列。

通常，一个计算机程序主要描述两部分内容：描述问题的每个对象及它们之间的关系；描述对这些对象作处理的处理规则。其中关于对象及它们之间的关系是数据结构的内容，而处理规则是求解的算法。针对问题所涉及的对象和要完成的处理，设计合理的数据结构常可有效地简化算法，数据结构和算法是程序最主要的两个方面。

计算机程序有以下共同的性质：

- 目的性——程序有明确的目的，程序运行时能完成赋予它的功能。
- 分步性——程序为完成其复杂的功能，由一系列计算机可执行的步骤组成。
- 有序性——程序的执行步骤是有序的，不可随意改变程序步骤的执行顺序。
- 有限性——程序是有限的指令序列，程序所包含的步骤是有限的。
- 操作性——有意义的程序总是对某些对象进行操作，使其改变状态，完成其功能。

程序设计

程序设计就是根据要计算机完成的任务，提出需求，设计数据结构和算法，编制程序和调试程序，使计算机程序能正确完成需求所设定的任务。简单地说，程序设计是设计和编制程序的过程。

程序首先应能正确完成任务，且是可靠的。同时，由于程序在使用过程中，因使用环境改变或需修改其原来的功能等原因，可能会经常修改。因此，除了为程序编写详尽正确的文档外，编写容易阅读的结构化程序也是对一个好程序的要求。总的来说，一个好的程序有可靠性、易读性、可维护性等良好特性。为达到这些目标，应采用正确的程序设计方法，以便从方法上更有助于设计出具有上述良好特性的程序。

机器语言、汇编语言和高级语言

程序设计语言是人与计算机进行信息通信的工具，是用来书写计算机程序的语言。只有用计算机指令编写的程序才能被计算机直接执行，而用其他任何指令编写的程序还需要通过中间的翻译过程。用于编写计算机程序的语言少说也有几百种，它们大致可分为三类，机器语言、汇编语言和高级语言。计算机的指令系统称为机器语言，所有的计算机都只能直接执行由其自身机器语言编写的程序。机器语言与计算机的硬件密切相关，机器语言中的计算机指令通常用一个二进制形式的代码，由若干位 1 和 0 组成。一条计算机指令指示计算机一次完成一个最基本的操作。

由于用机器语言编写程序实在太慢和太枯燥，用类英语单词的缩写代表计算机的二进位代码指令，这些类英语单词缩写的符号指令构成了汇编语言的基础。用汇编语言编写的程序比用机器语言编写的程序清晰了许多，编写程序也方便了许多。但用汇编语言编写的程序要在计算机上执行，先要将用汇编语言编写的源程序转换成机器语言程序，称完成这个转换功能的程序为“汇编程序”。

随着计算机应用范围的迅速扩大，各种应用程序也越来越大，为加速程序的开发速度，人们开发出用语句表示要计算机完成有一定意义任务的高级语言。而把用高级语言写的源程序转换成机器语言程序的翻译程序称为“编译器”。显然用高级语言比用机器语言或汇编语言编写程序更为简便。高级语言是一种既能方便地描述求解问题的处理对象和对象的处理规则，又能借助于编译器为计算机所接受、理解和执行的人工语言。高级语言习惯上称程序设计语言，它有统一的语法，独立于具体机器，便于人们编码、阅读和理解。本书将用 C 语言作为程序的描述语言，C 语言是功能强、应用范围广的高级语言之一。本书最后还将介绍目前广泛应用的 C++ 语言。

面向过程的语言

目前最流行最经常使用的程序设计语言属面向过程型的语言，如广为流行的 Fortran 语言、Pascal 语言、C 语言、Cobol 语言和 Ada 语言等。面向过程的语言虽可独立于计算机编写程序，但用这类语言编写程序时，程序不仅要说明做什么，还要非常详细地告诉计算机如何做，程序需要详细描述解题的过程和细节。

面向问题的语言

为了进一步推广和便于应用计算机，目前已研制了多种面向问题的语言。用面向问题的语言解题时，不仅摆脱了计算机的内部逻辑，也不必关心问题的求解算法和求解的过程，只需指出问题是要计算机做什么，数据的输入和输出形式，就能得到所需结果。面向问题语言又称为非过程化语言或陈述性语言，如报表语言、SQL(structured query language)语言等。SQL语言是数据库查询和操纵语言，能直接使用数据库管理系统。由于使用面向问题语言来解题只要告诉计算机做什么，不必告诉计算机如何做，能方便用户的使用和提高程序的开发速度。但实现面向问题语言的系统从最一般的意义下实现问题如何做，通常实现的效率较低。另外，面向问题语言要求问题已有确定的求解方法，目前其应用范围还比较狭窄。

面向对象语言

为克服面向过程语言过分强调求解过程的细节，程序不易复用的缺点，推出了面向对象程序设计方法和面向对象语言。面向对象语言引入了对象、消息、类、继承、封装、抽象、多态性等机制和概念。用面向对象语言进行程序设计时，以问题域中的对象为基础，将具有类似性质的对象抽象成类，并利用继承机制，仅对差异进行程序设计，这大大提高了程序的复用能力和程序开发效率。面向对象程序语言已是程序语言的主要研究方向之一。

算　法

算法就是问题的求解方法，一个算法由一系列求解步骤组成，算法的描述由经明确说明的一组简单指令和规则组成，计算机按规则执行其中的指令能在有限的步骤内解决一个问题或者完成一个函数的计算。正确的算法要求组成算法的规则和步骤的意义应是唯一确定的，是没有二义性的，由这些规则指定的操作是有序的，必须按算法指定的操作顺序执行，并能在执行有限步骤后给出问题的结果。

对求解同一个问题可能会有多种算法可供选择，选择算法的标准是算法的正确性和可靠性，算法的简单性和易理解性。其他的标准还有算法所需要的存储空间少，执行时间短等。

描述算法的常用工具有流程图，又称框图。流程图是算法的图形描述，由于流程图往往比程序更直观清晰，容易阅读和理解，所以它不仅可以作为编写程序的依据，而且也是交流的重要工具。在逐步求精的结构化程序设计方法中，目前多数采用结构化的伪代码来描述算法。本书采用与C语言控制结构一致的伪代码描述算法。

通常算法的开发过程是由粗到细分多步完成的。先给出粗略的计算步骤，然后对其中的粗略步骤作深入考虑，添加上一些实现细节，变成较为详细一些的描述。可能其中还包含有实现细节不明确的部分，则还需对其中不明确的部分作进一步的细化。直至算法所包含的计算步骤全部都足够清楚，能用某种程序语言完全描述为止。这种算法开发方法称为逐步求精开发方法。

数 据 结 构

计算机程序的处理对象是描述客观事物的数据，由于客观事物的多样性，会有不同形式

的数据，如整数、实数、复数和字符，以及所有计算机能够接收和处理的各种各样符号集合。在计算机程序中，形式不同的数据采用数据类型来标识。变量的数据类型说明变量可能取的值的集合，以及可能施于变量的操作的集合。例如，程序对表示苹果个数的变量，它们只能大于或等于零的整数，对它们能施行加减操作，不能施乘除操作，但这些变量能与整数一起施行乘除操作等。所以数据类型不仅定义了一组形式相同的数据集，也定义了对这组数据可施行的一组操作集。

数据结构是指数据对象及其相互关系和构造方法，程序中的数据结构描述了程序中的数据间的组织形式和结构关系。

数据结构与算法有密切的关系，只有明确了问题的算法，才能较好地设计数据结构；要选择好的算法，又常常依赖于合理的数据结构。数据结构是构造算法的基础。

对计算机而言，程序是一组供计算机执行的指令序列。程序告诉计算机如何对指定的数据进行操作，最终得到希望的结果。从问题求解来看，求解问题的程序是对一个抽象的求解算法的详细描述，这种描述是建立在数据的特定的表示方式和结构的基础上的。不了解对数据进行操作的算法就无法决定如何构造数据。同样，确定算法结构和算法步骤也依赖于作为基础的数据结构。也就是说，程序的构成是和数据结构不可分割的。程序在描述算法的同时，也必须完整地描述作为算法的操作对象的数据结构。对于一些复杂的问题，常因数据的表示方式和结构的差异，问题的抽象求解算法也会完全不同。所以程序、数据结构和算法三者之间的关系,可以说,程序是对数据结构和算法的描述,即

$$\text{程序} = \text{数据结构} + \text{算法}$$

1.2 结构化程序设计

对简单的问题进行程序设计，主要工作是选择或设计能解决问题的算法和确定数据结构。一旦算法和数据结构确定后，就可选用合适的程序设计语言编制程序。程序设计的大致步骤依次是设计数据结构和算法、用结构化的伪代码描述算法、编写程序和测试程序。

随着计算机应用的日益广泛，计算机软件的规模和复杂性不断增加，对软件的测试和维护也越来越困难。人们为克服软件危机，提出了结构化程序设计的方法和软件工程的概念，要求软件开发必须遵循一套严格的工程准则，以得到可靠、结构合理、容易维护的软件产品。

结构化程序设计主要包括程序结构的自顶向下模块化设计方法、模块算法的逐步求精设计方法以及用结构化控制结构描述算法和编写程序。

自顶向下模块化设计方法

限制程序的复杂性是程序设计的核心问题。程序的自顶向下模块化设计能有效地解决这个问题。程序结构自顶向下模块化设计方法就是把一个大程序按功能划分成一些较小的部分，每个完成独立功能的小部分用一个程序模块来实现。分解模块的原则是简单性、独立性和完整性。按模块化设计方法开发程序，能使程序具有较高的可靠性和灵活性，同时便于程序的测试和维护。在用模块化方法划分程序模块时，应尽量让模块具有如下所述的多项良好性质：

- 让模块具有单一入口和单一出口。
- 模块不宜过大，应让模块具有单一的功能。
- 模块的执行不能对环境产生副作用。
- 让模块与环境的联系仅限于明确定义的输入参数和输出参数，模块的内部结构与调用它的程序无关。
- 尽量用模块的名字调用模块。

根据由粗到细，由抽象到具体的原则，程序结构的开发应自顶向下进行，首先粗略地划分出程序的宏观结构，在深入分析的基础上，逐步细化划分出一个个独立功能的模块和小模块。

逐步求精设计方法

程序设计的基本方法是抽象、枚举和归纳。抽象包括算法的抽象和数据抽象。算法抽象是指算法的寻求（或开发）采用逐步求精、逐层分解的方法。数据抽象也指在算法抽象的过程中逐步完善数据结构和引入新的数据及确定关于数据的操作。

模块算法的设计采用逐步求精设计方法，即先设计出一个抽象算法，这是一个在抽象数据上实施一系列抽象操作的算法，由粗略的控制结构和抽象的计算步骤组成。抽象操作只指明‘做什么’,对这些抽象操作的细化就是想方设法回答它‘如何做’。采用逐步求精的方法,由粗到细,将抽象步骤（大任务）进一步分解成若干子任务。分而治之，对仍不具体的抽象子任务再进行分解。如此反复地一步步细化，算法越来越具体，抽象成分越来越少，直至可以编程为止。

结构化控制结构

在早期程序设计活动中，因没有统一的现成方法，因人习惯而异，编制的程序的控制结构混乱，很难阅读和修改。经计算机科学工作者多年努力，已归结出顺序、选择和循环三种结构化的控制结构，并在理论上证明了可计算问题的程序都可用这三种控制结构来描述。

1. 顺序结构

通常，一个复杂的计算过程不可能用一个操作步骤来表达，为此，需把复杂的计算工作分解成一系列逐条执行的操作序列。顺序结构为把一个复杂的计算用若干简单的顺序计算实现提供控制手段。顺序结构是一个操作序列，顺序结构执行时，从序列的第一个操作开始，顺序执行序列中的操作，直至序列的最后一个操作执行后完成。若顺序结构只有两个操作 A 和 B组成,可写成以下形式：

```
{
  A;
  B;
}
```

例如,为交换变量 x 和 y 的值,可分解为顺序执行的三个操作步骤,写成顺序结构：

```
{
  temp = x;      /*将 x的值暂存于 temp*/
  x    = y;      /*将 x置成 y的值 */
```

```
    y     = temp;  /* 将 y 置成暂存于 temp 的值 */
}
```

2. 条件选择结构

条件选择结构为描述按不同情况自动选择操作步骤执行提供控制手段。条件选择结构由一个判断条件 P 和两个供选择的分支操作 A 和 B 组成,可写成以下形式:

```
if (p)
   A;
else
   B;
```

条件选择结构执行时,先计算条件 P 的值,如果 P 的值为真,即条件成立,则执行分支操作 A;否则,若条件 P 的值为假,即条件不成立,则执行分支操作 B。注意,无论 P 为何值,条件选择结构只能执行 A 或 B 之一。

条件选择结构中的分支操作又可以是任何控制结构,特别当分支操作又是条件选择结构时,就呈现嵌套的条件选择结构。

3. 循环结构

循环结构为描述循环操作提供控制手段。循环结构有多种表现形式,最常见的有 while 型循环结构和 do-while 型循环结构。

while 型循环结构由一个条件 P 和一个循环操作 A 组成,可写成以下形式:

```
while (P)
   A;
```

while 型循环结构的执行过程是:每次循环前,先求条件 P 的值,当条件 P 成立(真)时,就执行操作 A,并接着再次求条件 P 的值,以确定操作 A 是否再次被执行。当 P 的值一开始为假,或某次循环后 P 的值为假,则结束循环操作。

do-while 型循环结构也由一个条件 P 和一个循环操作 A 组成,可写成以下形式:

```
do
   A;
while (P);
```

do-while 型循环结构的执行过程是:每次循环前,先执行操作 A,接着再求条件 P 的值,当条件 P 成立(真)时,再执行操作 A。如此反复,直到条件 P 的值为假,结束循环操作。

以上所说的操作 A 和 B 可以是一个简单的操作步骤,也可以又是某种控制结构。理论上,可计算问题的程序可以不用如 goto 那样的控制转移操作,而全由一些简单的操作和用上述三种控制结构反复嵌套来描述。如是这样,这种程序的结构性和可读性就比较良好,建议读者要编写结构良好、可读性高的程序。

1.3 C语言基础知识

1.3.1 几个简单的C程序

本节从C程序例子出发介绍C语言的基础知识,让读者对C语言有一个初步的印象。其中提及的概念、名称在以后的章节中都将有详细的介绍。对于已有其他高级语言使用经验的读者，可能有些概念的提法与已熟悉的语言稍有不同，只要浏览阅读一遍就行。

【例 1.1】一个只输出一行信息的C程序。

```
#include   <stdio.h>
void main()        /*  主函数 */
{
  printf("This book is  < Programming with C and c++  languages > . \n")   ;
}
```

该程序只输出一行信息：

This book is < Programming with C and C++ languages > .

以这样一个非常简单的C程序为例,对C程序可以指出以下几个特点：

(1) 一个C程序有一个名为 main 的函数,称它为主函数。实际上C程序可以由多个函数组成。例 1.1 程序只有一个主函数。组成程序的多个函数可存放在一个或多个源程序文件中。习惯将含有函数定义的程序文件名称的后缀定为 c。

(2) 主函数 main()前的关键字 void 表示该主函数的执行不返回结果。

(3) 在函数名之后有一对圆括号“()”,圆括号内可包含函数的参数。

(4) 函数体用花括号“{}”括起来。花括号可以用来括住任何一组C代码,从而构成复合语句或分程序。例 1.1 的函数体只有一个 printf() 函数调用。printf()函数是C系统的函数库中的输出函数，其中用双引号“""”括住的字符串用来指明 printf() 函数的输出格式。例中的输出格式表示以字符串原样输出，而“\n”是换行的意思,即输出以下字符列：

This book is < Programming with C and C++ languages > .

之后换行。

(5) 简单C语句之后有一个分号“;”,它是某些语句的结束符。

(6) 程序中的“/* */”表示程序的注释部分。程序中的注释是给阅读程序的人看的，对程序编译和运行都没有作用。它用来说明该程序的文件名称、程序的功能、使用方法、最后更改日期等；注释出现在程序代码行中或某行语句之后，用来说明一段程序代码或一个语句的功能、意义等。

(7)上述程序中的第一行

```
#include   <stdio.h>
```

是编译预处理命令行，它告诉编译系统将包含有关输入和输出标准函数信息的 stdio.h 文件

内容作为当前源程序文件的一部分。

【例 1.2】读入两个整数，输出它们的和。

```
/*1*/   #include <stdio.h>
/*2*/   void main()
/*3*/   {/*变量定义部分*/
/*4*/     int x,y,sum;                    /*  定义整型变量 x,y,sum*/
/*5*/       /* 以下为语句序列*/
/*6*/     printf("Input x and y \n");     /*  输出提示输入数据的字样*/
/*7*/      scanf("%d%d",&x,&y);           /*输入 x和 y 的值*/
/*8*/     sum = x + y;                    /*  完成 x + y 的计算,求 sum = x + y*/
/*9*/     printf("x + y = %d \n",sum);    /*  输出结果*/
/*10*/  }
```

例 1.2程序的功能是输入两个整数，求出它们的和并输出。为表示变化的数据对象，程序引入变量,程序中的第 4行就是用来定义变量，它定义了三个整型变量，分别命名为 x、y 和 sum。第 6行实现输出文字“ Input x and y” ,用来提示程序已开始执行,请输入 x 和 y 的值。第 7行的 scanf()是 C 函数库中的输入函数,其中“ %d”是十进制整数输入格式，用来指定输入数据的数据类型和格式，这里表示键入的数据以十进制整数理解，将它转换成整型量的机内表示。&x 和 &y 中的“ &”的含义是取地址，对于本例，意指顺序输入的两个十进制整数将它们分别存于用 x 和 y 命名的变量所对应的内存单元中，直观地理解就是输入值给变量 x 和 y。第 8行是完成 x + y 的计算，并将计算结果赋给变量 sum。第 9行的 printf()函数调用将输出“ x + y = ”字样和变量 sum 的值,并换行,其中“ %d ”意指将 sum 的值按十进制整数形式输出。该程序运行情况如下：

```
Input x and y
    12  15     (假定输入 12和 15 两个整数)
x + y = 27
```

上面三行中的第 1、3 行是程序输出的,第 2 行是用户键入的，12 与 15 之间用空格或回车分隔。

【例 1.3】利用公式 C = (5/9)(F - 32) 输出 F 氏温度与 C 氏温度对照表,设已知 F 氏温度取 0、20、…、200。

```
#include   <stdio.h>
void main()
{  float f, c;          /*变量定义*/
   int lower, upper, step;
   lower = 0;   upper = 200;   step = 20;   f = (float)lower;
   while (f <= upper) {   /*循环计算*/
     c = (float)(5.0/9.0*(f - 32.0));
     printf("%3.0f   %6.1f \n",f,c) ;
     f = f + step;
   }
```

```
}
```

为描述循环计算，程序用循环控制结构来描述。本程序对指定范围内的 F 氏温度进行循环计算。程序首先为有关变量设定初值，然后用循环语句实现循环计算。在循环体内，对每个 F 氏温度计算出对应的 C 氏温度后输出，然后增加 F 氏温度值。循环直至 F 氏温度超出要求后结束。循环控制用循环语句实现，上述程序用的是 while 循环语句。C 程序的循环控制也可有 for 语句或 do - while 语句实现。

【例 1.4】输入两个实数，输出它们中的小的数。

```
#include   < stdio.h >
void main()
{   float x,y,c;              /* 变量定义 */
    float min(float, float);/*  函数说明 */
    printf("Input x and y. \ n");
    scanf("%f%f",&x,&y)  ;
    c = min(x,y);           /* 调用函数 min() */
    printf("MIN(%.2f,%.2f) = %.2f \ n"",x,y,c) ;
}
/* 以下定义函数 min() */
float min(float a, float b)
{   float temp;/* 函数内使用的变量定义 */
    if(a < b)   /* 如果 a < b */
      temp = a;
    else
      temp = b;
    return temp;          /* 返回 temp 到调用 min() 函数处 */
}
```

为控制程序的复杂性，程序把一些功能相对独立的程序段编写成函数。例 1.4 的程序包含两个函数 主函数 main()和求两个实数中小的数的函数 min()。函数 min()的作用是将 a,b 中的较小的值赋给变量 temp，然后用返回语句 (return)将 temp 的值返回给对它的调用处(主函数中的 c = min(x,y))。本 例 main()函数要调用 min()函数，调用时将实参 x 和 y 的值分别传送给 min()函数中的形参 a 和 b。经执行 min()函数后，得到一个值，这个值被赋给变量 c。程序中的输入输出函数调用中的格式“%f”与前面例子中介绍的格式“%d”相似，格式“%f”是用于输入输出浮点型数据的。注意：主函数中的函数说明“ float min(float, float);”，它是用于说明名字 min 是一个返回浮点型 (float)值的函数，它有两个 float 型形参。

C 语言中遵守对象“先说明或定义，后使用”的原则。若主函数未对函数 min()作说明，则主函数中的“c = min(x,y)”的 min()函数被省缺设定为整型函数，而后面的 min()函数定义却是浮点型，导致类型不一致的错误。另请读者注意：C 语言中变量和函数的说明与定义的区别，直观的理解是说明只给一个名赋予某种意义，如变量说明只宣布变量的类型等特性、函数说明只宣布它的结果类型等特性；而定义除表明全部特性外，更主要的是确定实体，如变量定义要求为变量分配存储，函数定义是指定形参和实现其功能的程序代码。

本例说明 C 程序由若干函数组成。C 程序至少包含有主函数(main( )函数),还可包含其他别的函数。另外,一个 C 程序可由多个源程序文件组成 ,每个源程序文件可有一个或多个函数组成,C 语言的这个特点有助于程序的模块化设计。

函数定义的一般形式:

函数返回值类型　函数名 (形式参数说明表)

{　说明和定义部分;

　语句序列

}

一个函数定义由函数头和函数体组成。函数头包括函数属性、函数返回值类型、函数名、函数形式参数名、形式参数类型。

一个函数可以没有形式参数,但函数名之后的一对圆括号是必须的。函数体是函数头之后用一对花括号括住的部分。函数体用于描述实现函数功能的代码,它一般又可包括:

- 说明和定义部分说明数据结构 (类型)和定义函数专用的局部变量等。
- 语句序列由 C 语句和控制结构代码组成,是实现函数功能的 C 代 码。

【例 1.5】统计输入字符行中各数字符、空格符或制表符和其他字符的出现次数。

```
#include   <stdio.h>
void main()
{   intc,k,nWhite,nOther;
    int nDigit[10];/*定义有 10 个元素的数组,分别是各数字符的计数器 */
    nWhite = nOther = 0;
    for(k = 0;k < 10;k + + )/* for循环语句 */
      nDigit[k] = 0;
    printf("Enter string line. \n");
    while((c = getchar())!  = '\n')
      if(c > = '0'&&c < = '9')/*  如是数字符 */
         + + nDigit[c - '0'];  /*对应数字符的计数器增 1*/
      else if(c = = ' '||c = = '\t')
           + + nWhite;  /*空白符计数器增 1*/
      else
           + + nOther;  /*  其他字符计数器增 1*/
    for(k = 0;k < 10;k + + )  /*  输出对应数字符及其出现次数 */
      printf("%c:%d \n" , '0' + k,nDigit[k]);
    printf("white space:%d \ tother:%d \n \n",nWhite,nOther);
}
```

程序中常将一组相同数据类型的变量用数组来实现,便于对成组变量作统一的处理。上面例子中的数组 nDigit 有 10 个元素组成,它们分别作为 10 个不同数字符的计数器,程序能对它们作统一处理,如用循环语句给它们置初值、统一累计对应数字符的计数器和输出它们的结果。

从以上一些示例程序知道 .C 程序从 main( )函数开始执行,不管 main( ) 函数在程序中的

位置如何。C 程序的书写格式是自由的，即一行可写多个语句，一个语句也可分写在多行上。为了便于人们(包括自己)阅读你的程序，建议用一种良好的风格书写程序。本书采用的程序书写风格就是一种比较流行的书写风格。

从以上程序例子还看到，C程序的每个简单语句，说明及变量定义之后都必须以分号结尾，分号是它们必要的组成部分。在本书给出的 C 语言句法成分的一般形式中，若最后有分号，则该分号是它的必要组成部分。另外，程序要输入输出时，可调用输入输出函数库中的标准函数实现，如 printf( )函数和 scanf( )函数。

在 C 程序的任何部分都可插入注释。注释便于人们理解和阅读 C 程序，但有一点需要特别指出，注释一般不写成嵌套形式，否则要在编译系统中预置成允许注释嵌套的状态。

1.3.2 C语言的词汇、数据类型、常量和变量

基本词汇

任何一种高级语言，都有自己的基本符号，基本词汇和一套语法规则。C语言的基本符号有：

数字 10个(0～9)。

英文字母大、小写各 26 个(A～Z,a～z)。

下线符“_”。下线符起一个英文字母的作用，以构成标识符等语法成分。

其他用于构成特殊符号的字符集。

C语言的基本词汇由上述基本符号组成，C 语言的基本词汇有：

字面形式常量，如 100、15.0、'A'、"ABC"。

特殊符号，主要是运算符。

关键字。

标识符，用于命名程序对象。

其中运算符见附录 A.1。C语言为了清晰表达程序成分的意义，使用了一些英文单词，这些单词称为关键字。主要有：

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	static	struct	switch	typedef	union	unsigned
void	volatile	while				

下面几个虽不属于关键字，但建议把它们看作关键字，不要在程序中随便使用：

define	undef	include	ifdef	ifndef
endif	line	elif		

这些单词用在 C 程序的预处理命令行中。

标识符用来标识变量、常量、类型、函数、语句等程序对象，C语言用标识符给它们命名。在 C 语言中，一个合理的标识符由英文字母或下线符开头，后跟或不跟由字母、下线符、数字符组成的字符列。一般以下线符开头的标识符作内部使用。标识符作为程序对象的名

称,为了便于联想和记忆,建议读者给程序对象命名时,使用能反映该对象意义的标识符。

另外,请注意不同的C系统对标识符的有效字符个数有不同的规定。对于限制标识符8个有效字符的系统来说,两个超过8个字符的不同标识符,当前8个字符依次相同时,系统就认为它们是同一个标识符,而不加以区别。

利用基本词汇,按照给定的C语言的句法规则就可命名程序对象,构造语句、函数,直至整个程序。

C语言的数据类型

计算机程序要描述的主要内容包括两个方面:一是关于算法的操作步骤和控制结构的描述,即计算动作和过程的描述;二是关于操作对象的描述,即数据的描述。程序设计就是设计数据结构和算法。

在高级语言中,引入数据类型的概念。数据类型反映两方面的内容:数据能被如何表示(它的所有表示构成了数据类型的值的集合),如何处理它的值(它的所有处理方式构成了数据类型的操作集合)。例如,整型是系统预先设定的有限整数集和一个关于整数的运算集。高级语言为便于编写程序,预先设定了若干基本数据类型。又为了使程序能描述和处理现实世界中各种结构化数据结构的问题,高级语言还提供若干从基本数据类型出发构造各种结构化数据结构的设施。C语言的数据类型系统有基本数据类型、指针类型和一些结构化数据类型的构造设施组成。其中基本数据类型有三种,它们分别是:

整型(short,int,long)。

实型(float,double,long double)。

字符型(char)。

结构化数据类型的构造设施有:

数组、结构、联合和枚举。

为描述数据实体之间具有某种关系的数据结构,如链表、树、图等数据结构,C语言引入了指针类型。在C语言中,指针类型直接赋予数据对象在内存中的地址的概念。

使用以上数据类型构造设施,从基本数据类型和指针类型出发,能构造出各种复杂的数据类型。以上构造设施还能被反复应用,习惯以最终使用的构造设施来称呼得到的数据类型。

常　量

在程序运行过程中,其值不能改变或不允许改变的数据对象称为常量。常量按其值的表示形式区分它的类型,如15、0、-7是整型常量;5.0、-12.36为浮点型常量;而'a'、'b'为字符型常量;"C++ language"为字符串常量。

在程序中直接使用字面形式的常量,在某些情况下会有不便之处。若表示同一意义的常量在程序中出现多处,可能会不易保持一致性,也不便修改。如程序中多处使用π的值,不同之处书写的位数可能不易保持相同。当为了提高计算精度时,就需要多处作同样的修改,容易产生遗漏。另外,字面形式的常量出现在程序中,不能反映该值的意义。给常量命名是解决上述问题的好方法。程序用常量的名引用常量,能保证一致性,也便于修改。如给常量命名一个反映常量意义的名,还能提高程序的可读性。在C语言中,用宏定义给常量

命名,其一般形式是:

define 标识符　字符列

如

```
#define PI      3.14159
#define MAXN    100
```

define 是 C 语言的预处理命令，详细用法见第 6 章。注意:# define 命令行最后不要另用分号结束。另外，程序中不允许对常量标识符赋值。

变　量

在程序运行过程中，其值可以改变的数据对象称为变量。变量在它存在期间，在内存中占据一定的存储单元，以存放变量的值。可以给变量一个名字，但请注意变量名、变量所占据的内存地址和变量的值等概念的区别。

程序在使用变量之前,先要对变量作定义,如例 1.2 和例 1.3 所示。变量定义是指定变量的名字和数据类型，其中变量的名字以便程序对它引用；变量的数据类型用来告诉编译系统，为它分配多少存储单元，如何判定程序对它操作的合理性，并为合理的操作生成正确的目标代码。

与变量有关的概念有 变量名、变量数据类型、变量的值、变量所占据的内存地址、变量在程序中的有效范围、变量在程序执行期间的存在时间等。

程序通过变量定义引入变量，变量定义的一般形式是:

类型　变量名表;

其中变量名表由一个或多个变量名组成。例如:

```
int i,j,sum;        /* 定义三个 int 型变量 */
char str[100];      /*定义一个字符数组 */
float z;            /*定义一个 float 型变量 */
```

在 C 语言中，变量定义在为变量指定类型和名称的同时，还可以为变量指定初值。为变量指定初值称作变量初始化。例如:

```
int index = 100,big _ int = 10000;
```

1.4　高级语言程序开发环境基本知识

为了便于用某种高级语言开发程序，语言通常被包装成一个软件系统。一般地，语言的软件系统由三部分组成，即程序开发环境、语言和标准函数库。其中开发环境支持程序的开发过程，对程序从开发到运行的每个阶段给予有力的支持。以一个典型的 C 系统为例,C 程序从开发到运行大致要经历六个阶段 编辑、预处理、编绎、连接、加载和执行。

第一阶段是编辑。程序员用系统环境提供的编辑器编辑源程序，产生一个源程序文件，并将源程序文件保存在磁盘中。对于 C程序，执行代码的源程序文件习惯用 .c 作文件的扩展名；而将只提供定义说明信息的源程序文件习惯用 .h作扩展名。

第二阶段是调用预处理程序。源程序编辑完成后，程序员发出编译程序的命令， C 编译

器先自动调用预处理程序，C的预处理程序按源程序中包含的预处理命令，对源程序文件作文字转换，产生一个新的内部程序代码。

第三阶段是编译。预处理结束后，编译器的编译程序开始工作，对经预处理程序产生的内部程序代码进行编译。若编译过程中发现程序有错误，则输出错误的详细信息；对正确的源程序产生机器语言程序，称为源程序的目的代码。

第四阶段是连接。C程序通常会引用库函数和别的源程序中定义的函数，连接程序将目的代码和这些函数的目的代码连接起来，产生计算机可直接执行的程序映象文件。

第五阶段是加载。将要执行的程序装入内存。

第六阶段是运行。装入内存的程序在计算机的操作系统控制下执行。

一般来说，C程序都要输入输出数据，适应大多数程序的要求，规定了一组标准输入和标准输出方法，并提供功能完整的标准输入和标准输出函数库。用 stdin标识标准的输入数据流，stdout 标识标准的输出数据流。通常 stdin 是指来自键盘的输入数据流，stdout 是指输出到显示器的数据流。但它们也可被指定为别的数据文件。程序应能检查错误，当发现错误时，程序应输出相应的错误信息。错误信息流用 stderr 来标识，它的对应设备是显示器。

思考题与习题

1. 上机运行本章例题，熟悉计算机系统运行 C 程序的步骤。
2. 模仿例 1.3,编写已知 C 氏温度求 F 氏温度的程序。
3. 模仿例 1.4,编写程序,输入两个实数,输出它们中的大数的程序。
4. 模仿例 1.5,编写程序,统计输入字符行中各英文字母的出现次数。
5. 试指出为程序中出现的常量用宏定义命名的好处。
6. 试指出常量和变量的区别。
7. 试指出高级程序语言中，数据类型的主要含义。
8. 试指出 C语言的基本数据类型有哪些。

第 2 章　基本数据及其运算和输入输出

C 语言提供整型、实型和字符型三种基本数据类型让程序直接定义变量，本章详细介绍这三种类型的特性和使用方法。

2.1　整 型 数 据

因计算机只能表示整数的一个子集，C 语言考虑到一般应用和特殊应用的不同需要，将整型数据的数值范围分成三种：基本整型、短整型、长整型。对这三种整型数据的内部表示的最高位的不同理解又可分别分成两类：带符号和不带符号。

(1) 基本整型，类型符用 int 标记。

(2) 短整型，类型符用 short int 标记，简写为 short。

(3) 长整型，类型符用 long int 标记，简写为 long。

以上三种整型数据的类型符都是标记带符号的整型，若分别在它们之前冠以 unsigned 后，就分别为不带符号的整型。如 unsigned int 是无符号的基本整型，unsigned short 是无符号短整型，unsigned long 是无符号长整型。无符号整型是指存储一个整数的存储单元中的全部二进位都用作存放数据本身，而不存储符号位。例如：

```
int i, j;      /* 定义两个带符号的整型变量 */
unsigned short k; /* 定义一个无符号的短整型变量 */
long m, n;   /* 定义两个带符号的长整型变量 */
```

C 语言本身未规定以上各类整型数据应占的内存字节数，一般以一个机器字存放一个 int 型整数，而 long 型整数的字节数不小于 int 型整数的字节数，short 型整数的字节数不多于 int 型整数的字节数。如微机上的某些 C 语言，短整型整数和基本整型整数是 2 个字节，16 个二进位 长整型整数为 4 个字节，32 个二进位。

用 16 个二进位表示一个带符号的整数，它的数值范围是 -32768 ~ 32767 表示一个无符号整数的数值范围是 0 ~ 65535。若用 32 个二进位表示一个整数，带符号和不带符号整数的数值范围分别是 -2147483648 ~ 2147483647 和 0 ~ 4294967295。

在 C 程序中，经常会出现整型常量，C 语言整型常量的书写形式有三种：

(1) 十进制整数。如 0，123，-45。

(2) 八进制整数。以数字符 0 开头并由数字符 0 ~ 7 组成的数字符序列，为八进制整数。如 0123 表示八进制整数，其值等于十进制整数 1 * 8 * 8 + 2 * 8 + 3 = 83。

(3) 十六进制整数。十六进制整数以 0x(或 0X)开头的整数。表示十六进制数的数字符有 16 个，它们分别是 0 ~ 9 和 A、B、C、D、E、F，其中六个字母也可以小写。例如，0x123 表示十六进制整数，其值等于十进制整数 1 * 16 * 16 + 2 * 16 + 3 = 291；0xabc ，其值等于 10 *

16 * 16 + 11 * 16 + 12 = 2748。

整型常数也可特别指明为是 long 型整数，在整型常数之后接上字母 L(或 l),即为 long 型整型常数。例如,0L、132L 等。如此明确的写法多数用于函数调用中。如果函数的形参为 long 型整型的,则要求实参也为 long 型整型。此时,若用整型常数作实参,可后接' L",以确保提供 long 型的整型常数。

整型常数也可特别指明是不带符号的，在整型常数之后接上字母 U(或 u),则指明该整数常数是 unsigned 型的。例如,1u、122U 等。为指明不带符号的 long 型整型常数,则需在整型常数之后同时加上字母 U 和 L,表明该整型常数是 unsigned long 型的。例如,22Ul、35Lu 等。

2.2 字符型数据

字符型数据用于表示一个字符值，但字符型数据的内部表示是字符的 ASCII 代码(二进制形式),并非字符本身。字符型数据的类型符用 char 来标记,例如：

```
char c1, c2; /* 定义两个字符型变量 */
```

在大多数系统中,字符型数据占 1 个字节,用 8 位二进位表示,就字符的 ASCII 代码来说,相当于一个 8位的整型数据。

字符型常量用来表示单个字符，它的书写方法：

(1) 普通字符,用单引号括住一个字符,如 'a'、'b'、'B'、'$'。

(2) 特殊字符,用 '\字符或字符列' 来标记,如换行符用 '\n' 来标记。采用这种方法就能表示特殊字符，它们的标记方法见表 2-1。

表 2-1 中的最后两行是直接用字符的 ASCII 代码表示字符，所以用这两种书写方法可以表示字符型中的全部字符数据。例如,'\102' 即表示 'B'、'\12' 表示 '\n',等等。

注意 制表符(其常量用 '\t'标记)、换行符(其常量用 '\n'标记)和回车符(其常量用 '\r'标记)在输出时的确切意义。

制表符作用是使当前输出位置横向跳格至下一个输出区的开始列位置。系统一般预设每个输出区占 8 列(预设值可改变)位置。各输出区的开始列位置依次为 1、9、17、… 。如输出的当前位置在 1 至 8 列任一位置上遇制表符都使当前输出位置移到第 9 列上。换行符表示以后的输出从下一行首开始。回车符表示以后的输出从当前行首开始。另外，打印机上输出与显示屏上输出的组织方法稍有不同。对于打印机，仅当一行字符填满或遇换行符时才输出，即整行一次性输出。当输出空格符或制表符时，作跳格处理，不用空格符填充。而对于显示器，逐个字符输出，空格符及制表符经过位置都用空格符输出。以上区别，仅当输出字符列中有回车符时才会发生差异。

注意：字符型常量与字符串常量书写形式的区别。字符串常量是一对双引号括起来的字符序列。例如：

```
"I am a student.", "China", "a", "$1234.00"
```

都是字符串常量。字符型常量'a' 与字符串常量 "a" 不同，不允许将字符串常量赋予字符变量。假设 ch 被指定为字符型变量，对它赋值一个字符型数据是正确的：

```
char ch = 'a';
```

而对它赋予一个字符串数据是错误的：

```
ch = "a";
```

由于字符型数据以 ASCII 代码的二进制形式存储，它与整数的存储形式相类似。因此，在 C 程序中，字符型数据和整型数据之间可以通用，字符型数据与整型数据可混合运算。一个字符型数据可以用字符格式("%c")输出，显示字符本身 也可以用整数形式输出，显示字符的 ASCII 码值。

表 2-1

标记形式	功　能
\n	换行符(打印位置移到下一行首)
\t	制表符，横向跳格到下一个输出区首(区大小可设置)
\v	竖向跳格符(与实现有关)
\b	退格(与实现有关)
\r	回车(打印位置移到当前行的首)
\f	走纸换页
\a	产生响铃声(与实现有关)
\\	反斜杠符 \
\'	单引号符 '
\"	双引号符 "
\ddd	ddd 为 1 至 3 个八进制数字，以该值为 ASCII 码的字符
\xhh	hh 为 1 至 2 个十六进制数字，以该值为 ASCII 码的字符

【例 2.1】字符型数据能与整型数据通用。

```
#include <stdio.h>
void main()
{  char c1, c2; /* 定义两个字符型变量 */
   c1 = 97;   /* 'a'的 ASCII码值为 97 */
   c2 = c1 + 1;/* 字符型数据可与整型数据混合运算 */
   printf("c1 = %c, c2 = %c \n", c1, c2);
   printf("%c's ASCII code = %d \n", c2, c2);
}
```

程序输出：

```
c1 = a, c2 = b
b's ASCII code = 98
```

当字符型数据作为一个整数来处理时，对于只有 8 位(一个字节)表示的字符来说，其第 8 位是否为符号位是由具体系统规定的。在有些系统中一个字符的 8 位代码表示 -128 ~ 127 的整数，在另一些系统中表示 0 ~ 255 的整数。为了避免与系统有关的不确定性，