第1部分 语 言 篇

第1章 程序设计入门

学习目标

- ☑ 熟悉C语言程序的编译和运行
- ☑ 学会编程计算并输出常见的算术表达式的结果
- ☑ 掌握整数和浮点数的含义和输出方法
- ☑ 掌握数学函数的使用方法
- ☑ 初步了解变量的含义
- ☑ 掌握整数和浮点数变量的声明方法
- ☑ 掌握整数和浮点数变量的读入方法
- ☑ 掌握变量交换的三变量法
- ☑ 理解算法竞赛中的程序三步曲：输入、计算、输出
- ☑ 记住算法竞赛的目标及其对程序的要求

计算机速度快，很适合做计算和逻辑判断工作。本章首先介绍顺序结构程序设计，其基本思路是：把需要计算机完成的工作分成若干个步骤，然后依次让计算机执行。注意这里的“依次”二字——步骤之间是有先后顺序的。这部分的重点在于计算。

接下来介绍分支结构程序设计，用到了逻辑判断，根据不同情况执行不同语句。本章内容不复杂，但是不容忽视。

注意：编程不是看会的，也不是听会的，而是练会的，所以应尽量在计算机旁阅读本书，以便把书中的程序输入到计算机中进行调试，顺便再做做上机练习。千万不要图快——如果没有足够的时间用来实践，那么学得快，忘得也快。

1.1 算术表达式

计算机的“本职”工作是计算，因此下面先从算术运算入手，看看如何用计算机进行复杂的计算。

程序 1-1 计算并输出 1+2 的值

```
#include<stdio.h>
int main()
{
```

```
    printf("%d\n", 1+2);
    return 0;
}
```

这是一段简单的程序，用于计算 1+2 的值，并把结果输出到屏幕。如果不知道如何编译并运行这段程序，可阅读附录 A 或向指导教师求助。

即使读者不明白上述程序除了“1+2”之外的其他代码，仍然可以进行以下探索：试着把“1+2”改成其他内容，而不要修改那些并不明白的代码——它们看上去工作情况良好。

下面做 4 个实验。

实验 1：修改程序 1-1，输出 3-4 的结果。

实验 2：修改程序 1-1，输出 5×6 的结果。

实验 3：修改程序 1-1，输出 8÷4 的结果。

实验 4：修改程序 1-1，输出 8÷5 的结果。

直接把“1+2”替换成“3-4”即可顺利解决实验 1，但读者很快就会发现：无法在键盘上找到乘号和除号。解决方法是：用星号“*”代替乘号，而用正斜线“/”代替除号。这样，4 个实验都顺利完成了。

等一下！实验 4 的输出结果居然是 1，而不是正确答案 1.6。这是怎么回事？计算机出问题了吗？计算机没有出问题，问题出在程序上：这段程序的实际含义并非和我们所想的一致。

在 C 语言中，8/5 的确切含义是 8 除以 5 所得商值的整数部分。同样地，（-8）/5 的值是-1。那么，如果非要得到 8÷5=1.6 的结果怎么办？下面是完整的程序。

程序 1-2　计算并输出 8/5 的值，保留小数点后 1 位

```
#include<stdio.h>
int main()
{
    printf("%.1f\n", 8.0/5.0);
    return 0;
}
```

注意：百分号后面是一个小数点，然后是数字 1，最后是小写字母 f，千万不能输入错，包括大小写——在 C 语言中，大写和小写字母代表的含义是不同的。

再来做 3 个实验。

实验 5：把%.1f 中的数字 1 改成 2，结果如何？能猜想出“1”的确切意思吗？如果把小数点和 1 都删除，%f 的含义是什么？

实验 6：字符串%.1f 不变，把 8.0/5.0 改成原来的 8/5，结果如何？

实验 7：字符串%.1f 改成原来的%d，8.0/5.0 不变，结果如何？

实验 5 并不难解决，但实验 6 和实验 7 的答案就很难简单解释了——真正原因涉及整数和浮点数编码，相信多数初学者对此都不感兴趣。原因并不重要，重要的是规范：根据

规范做事情，则一切尽在掌握中。

提示 1-1：整数值用%d 输出，实数用%f 输出。

这里的“整数值”指的是 1+2、8/5 这样“整数之间的运算”。只要运算符的两边都是整数，则运算结果也会是整数。正因为这样，8/5 的值才是 1，而不是 1.6。

8.0 和 5.0 被看作是“实数”，或者说得更专业一点，叫“浮点数”。浮点数之间的运算结果是浮点数，因此 8.0/5.0=1.6 也是浮点数。注意，这里的运算符“/”其实是“多面手”，它既可以做整数除法，又可以做浮点数除法[①]。

提示 1-2：整数/整数=整数，浮点数/浮点数=浮点数。

这条规则同样适用于加法、减法和乘法，不过没有除法这么容易出错——毕竟整数乘以整数的结果本来就是整数。

算术表达式可以和数学表达式一样复杂，例如：

程序 1-3　复杂的表达式计算

```
#include<stdio.h>
#include<math.h>
int main()
{
  printf("%.8f\n", 1+2*sqrt(3)/(5-0.1));
  return 0;
}
```

相信读者不难把它翻译成数学表达式$1+\frac{2\sqrt{3}}{5-0.1}$。尽管如此，读者可能还是有一些疑惑：5-0.1 的值是什么？“整数-浮点数”是整数还是浮点数？另外，多出来的#include<math.h>有什么作用？

第 1 个问题相信读者能够“猜到”结果：整数-浮点数=浮点数。但其实这个说法并不准确。确切的说法是：整数先“变”成浮点数，然后浮点数-浮点数=浮点数。

第 2 个问题的答案是：因为程序 1-3 中用到了数学函数 sqrt。数学函数 sqrt(x)的作用是计算 x 的算术平方根（若不信，可输出 sqrt(9.0)的值试试）。一般来说，只要在程序中用到了数学函数，就需要在程序最开始处包含头文件 math.h，并在编译时连接数学库。如果不知道如何编译并运行这段程序，可阅读本章末尾的内容。

1.2　变量及其输入

1.1 节的程序虽好，但有一个遗憾：计算的数据是事先确定的。为了计算 1+2 和 2+3，

[①] 但也有不少语言会严格区分整数除法和浮点数除法。

下面不得不编写两个程序。可不可以让程序读取键盘输入，并根据输入内容计算结果呢？答案是肯定的。程序如下：

程序 1-4　a+b 问题

```
#include<stdio.h>
int main()
{
  int a, b;
  scanf("%d%d", &a, &b);
  printf("%d\n", a+b);
  return 0;
}
```

该程序比 1.1 节的复杂了许多。简单地说，第一条语句“int a, b”声明了两个整型（即整数类型）变量 a 和 b，然后读取键盘输入，并放到 a 和 b 中。注意 a 和 b 前面的“&”符号——千万不要漏掉，不信可以试试[①]。

现在，你的程序已经读入了两个整数，可以在表达式中自由使用它们，就好比使用 12、597 这样的常数。这样，表达式 a+b 就不难理解了。

提示 1-3：scanf 中的占位符和变量的数据类型应一一对应，且每个变量前需要加“&”符号。

可以暂时把变量理解成“存放值的场所”，或者形象地认为每个变量都是一个盒子、瓶子或箱子。在 C 语言中，变量有自己的数据类型，例如，int 型变量存放整数值，而 double 型变量存放浮点数值（专业的说法是“双精度”浮点数）。如果一定要把浮点数值存放在一个 int 型变量中，将会丢失部分信息——我们不推荐这样做。

下面来看一个复杂一点的例子。

例题 1-1　圆柱体的表面积

输入底面半径 r 和高 h，输出圆柱体的表面积，保留 3 位小数，格式见样例。

样例输入：

```
3.5  9
```

样例输出：

```
Area = 274.889
```

【分析】

圆柱体的表面积由 3 部分组成：上底面积、下底面积和侧面积。由于上下底面积相等，完整的公式可以写成：表面积=底面积×2+侧面积。根据几何知识，底面积=πr^2，侧面积=$2\pi rh$。不难写出完整程序：

[①] 在学习编程时，“明知故犯”是有益的：起码你知道了错误时的现象。这样，当真的不小心犯错时，可以通过现象猜测到可能的原因。

程序 1-5　圆柱体的表面积

```
#include<stdio.h>
#include<math.h>
int main()
{
  const double pi = acos(-1.0);
  double r, h, s1, s2, s;
  scanf("%lf%lf", &r, &h);
  s1 = pi*r*r;
  s2 = 2*pi*r*h;
  s = s1*2.0 + s2;
  printf("Area = %.3f\n", s)
  return 0;
}
```

这是本书中第一个完整的“竞赛题目”，因为和正规比赛一样，题目中包含着输入输出格式规定，还有样例数据。大多数的算法竞赛包含如下一些相同的“游戏规则”。

首先，选手程序的执行是自动完成的，没有人工干预。不要在用户输入之前打印提示信息（例如“Please input n:”），这不仅不会为程序赢得更高的“界面友好分”，反而会让程序丢掉大量的（甚至所有的）分数——这些提示信息会被当作输出数据的一部分。例如，刚才的程序如果加上了“友好提示”，输出信息将变成：

```
Please input n:
Area = 274.889
```

比标准答案多了整整一行！

其次，不要让程序“按任意键退出”（例如，调用 system("pause")，或者添加一个多余的 getchar()），因为不会有人来“按任意键”的。不少早期的 C 语言教材会建议在程序的最后添加这样一条语句来“观察输出结果”，但注意千万不要在算法竞赛中这样做。

提示 1-4：*在算法竞赛中，输入前不要打印提示信息。输出完毕后应立即终止程序，不要等待用户按键，因为输入输出过程都是自动的，没有人工干预。*

在一般情况下，你的程序不能直接读取键盘和控制屏幕：不要在算法竞赛中使用 getch()、getche()、gotoxy()和 clrscr()函数（早期的教材中可能会介绍这些函数）。

提示 1-5：*在算法竞赛中不要使用头文件 conio.h，包括 getch()、clrscr()等函数。*

最后，最容易忽略的是输出的格式：在很多情况下，输出格式是非常严格的，多一个或者少一个字符都是不可以的！

提示 1-6：*在算法竞赛中，每行输出均应以回车符结束，包括最后一行。除非特别说明，每行的行首不应有空格，但行末通常可以有多余空格。另外，输出的每两个数或者字符串之间应以单个空格隔开。*

总结一下，算法竞赛的程序应当只做 3 件事情：读入数据、计算结果、打印输出。不要打印提示信息，不要在打印输出后“暂停程序”，更不要尝试画图、访问网络等与算法无关的任务。

回到刚才的程序，它多了几个新内容。首先是“const double pi = acos(-1.0);”。这里也声明了一个叫 pi 的“符号”，但是 const 关键字表明它的值是不可以改变的——pi 是一个真正的数学常数[①]。

提示 1-7：*尽量用 const 关键字声明常数。*

接下来是 s1 = pi * r * r。这条语句应该如何理解呢？“s1 等于 pi*r*r”吗？并不是这样的。若把它换成“pi * r * r = s1”，编译器会给出错误信息：invalid value in assignment。如果这条语句真的是“二者相等”的意思，为何不允许反着写呢？

事实上，这条语句的学术说法是赋值（assignment），它不是一个描述，而是一个动作。其确切含义是：先把“等号”右边的值算出来，然后赋于左边的变量中。注意，变量是“喜新厌旧”的，即新的值将覆盖原来的值，一旦被赋了新的值，变量中原来的值就丢失了。

提示 1-8：*赋值是个动作，先计算右边的值，再赋给左边的变量，覆盖它原来的值。*

最后是“Area = %.3f\n”，该语句的用法很容易被猜到：只有以“%”开头的部分才会被后面的值替换掉，其他部分原样输出。

提示 1-9：*printf 的格式字符串中可以包含其他可打印符号，打印时原样输出。*

这里还有一个非常容易忽略的细节：输入采用的是"%lf"而不是"%f"。关于这一点，本章的末尾会继续讨论，现在先跳过。

1.3 顺序结构程序设计

例题 1-2 三位数反转

输入一个三位数，分离出它的百位、十位和个位，反转后输出。

样例输入：

```
127
```

样例输出：

```
721
```

【分析】

首先将三位数读入变量 n，然后进行分离。百位等于 $n/100$（注意这里取的是商的整数部分），十位等于 $n/10\%10$（这里的%是取余数操作），个位等于 $n\%10$。程序如下：

[①] 有的读者可能会用 math.h 中定义的常量 M_PI，但其实这个常数不是 ANSI C 标准的。不信可以用 gcc-ansi 编译试试。

程序 1-6　三位数反转（1）

```
#include<stdio.h>
int main()
{
  int n;
  scanf("%d", &n);
  printf("%d%d%d\n", n%10, n/10%10, n/100);
  return 0;
}
```

此题有一个没有说清楚的细节，即：如果个位是 0，反转后应该输出吗？例如，输入是 520，输出是 025 还是 25？如果在算法竞赛中遇到这样的问题，可向监考人员询问[①]。但是在这里，两种情况的处理方法都应学会。

提示 1-10：*算法竞赛的题目应当是严密的，各种情况下的输出均应有严格规定。如果在比赛中发现题目有漏洞，应向相关人员询问，尽量不要自己随意假定。*

上面的程序输出 025，但要改成输出 25 似乎会比较麻烦——必须判断 n%10 是不是 0，但目前还没有学到“根据不同情况执行不同指令”（分支结构程序设计是 1.4 节的主题）。

一个解决方法是在输出前把结果存储在变量 m 中。这样，直接用%d 格式输出 m，将输出 25。要输出 025 也很容易，把输出格式变为%03d 即可。

程序 1-7　三位数反转（2）

```
#include<stdio.h>
int main()
{
  int n, m;
  scanf("%d", &n);
  m = (n%10)*100 + (n/10%10)*10 + (n/100);
  printf("%03d\n", m);
  return 0;
}
```

例题 1-3　交换变量

输入两个整数 *a* 和 *b*，交换二者的值，然后输出。

样例输入：

```
824  16
```

样例输出：

```
16  824
```

[①] 如果是网络竞赛，还可以向组织者发信，在论坛中提问或者拨打热线电话。

【分析】

按照题目所说，先把输入存入变量 a 和 b，然后交换。如何交换两个变量呢？最经典的方法是三变量法：

程序 1-8　变量交换（1）

```
#include<stdio.h>
int main()
{
  int a, b, t;
  scanf("%d%d", &a, &b);
  t = a;
  a = b;
  b = t;
  printf("%d %d\n", a, b);
  return 0;
}
```

可以将这种方法形象地比喻成将一瓶酱油和一瓶醋借助一个空瓶子进行交换：先把酱油倒入空瓶，然后将醋倒进原来的酱油瓶中，最后把酱油从辅助的瓶子中倒入原来的醋瓶子里。这样的比喻虽然形象，但是初学者应当注意它和真正的变量交换的区别。

借助一个空瓶子的目的是：避免把醋直接倒入酱油瓶子——直接倒进去，二者混合以后，将很难分开。在 C 语言中，如果直接进行赋值 $a=b$，则原来 a 的值（酱油）将会被新值（醋）覆盖，而不是混合在一起。

当酱油被倒入空瓶以后，原来的酱油瓶就变空了，这样才能装醋。但在 C 语言中，进行赋值 $t=a$ 后，a 的值不变，只是把值复制给了变量 t 而已，自身并不会变化。尽管 a 的值马上就会被改写，但是从原理上看，$t=a$ 的过程和“倒酱油”的过程有着本质区别。

提示 1-11：赋值 $a=b$ 之后，变量 a 原来的值被覆盖，而 b 的值不变。

另一个方法没有借助任何变量，但是较难理解：

程序 1-9　变量交换（2）

```
#include<stdio.h>
int main()
{
  int a, b;
  scanf("%d%d", &a, &b);
  a = a + b;
  b = a - b;
  a = a - b;
  printf("%d %d\n", a, b);
  return 0;
}
```

这次就不太方便用倒酱油做比喻了：硬着头皮把醋倒在酱油瓶子里，然后分离出酱油倒回醋瓶子？比较理性的方法是手工模拟这段程序，看看每条语句执行后的情况。

在顺序结构程序中，程序一条一条依次执行。为了避免值和变量名混淆，假定用户输入的是 a_0 和 b_0，因此 scanf 语句执行完后 $a=a_0$，$b=b_0$。

执行完 $a=a+b$ 后：$a=a_0+b_0$，$b=b_0$。

执行完 $b=a-b$ 后：$a=a_0+b_0$，$b=a_0$。

执行完 $a=a-b$ 后：$a=b_0$，$b=a_0$。

这样，就不难理解两个变量是如何交换的了。

提示 1-12：可以通过手工模拟的方法理解程序的执行方式，重点在于记录每条语句执行之后各个变量的值。

这个方法看起来很好（少用一个变量），但实际上很少使用，因为它的适用范围很窄：只有定义了加减法的数据类型才能采用此方法[①]。事实上，笔者并不推荐读者采用这样的技巧实现变量交换：三变量法已经足够好，这个例子只是帮助读者提高程序阅读能力。

提示 1-13：交换两个变量的三变量法适用范围广，推荐使用。

那么是不是说，三变量法是解决本题的最佳途径呢？答案是否定的。多数算法竞赛采用黑盒测试，即只考查程序解决问题的能力，而不关心采用了什么方法。对于本题而言，最合适的程序如下：

程序 1-10　变量交换（3）

```
#include<stdio.h>
int main()
{
  int a, b;
  scanf("%d%d", &a, &b);
  printf("%d %d\n", b, a);
  return 0;
}
```

换句话说，我们的目标是解决问题，而不是为了写程序而写程序，同时应保持简单（Keep It Simple and Stupid，KISS），而不是自己创造条件去展示编程技巧。

提示 1-14：算法竞赛是在比谁能更好地解决问题，而不是在比谁写的程序看上去更高级。

1.4　分支结构程序设计

例题 1-4　鸡兔同笼

已知鸡和兔的总数量为 n，总腿数为 m。输入 n 和 m，依次输出鸡的数目和兔的数目。

[①] 这个方法还有一个“变种”：用异或运算“^”代替加法和减法，还可以进一步简写成 a^=b^=a^=b，但不建议使用。

如果无解，则输出 No answer。

样例输入：

```
14  32
```

样例输出：

```
12  2
```

样例输入：

```
10  16
```

样例输出：

```
No answer
```

【分析】

设鸡有 a 只，兔有 b 只，则 $a+b=n$，$2a+4b=m$，联立解得 $a=(4n-m)/2$，$b=n-a$。在什么情况下此解“不算数”呢？首先，a 和 b 都是整数；其次，a 和 b 必须是非负的。可以通过下面的程序判断：

程序 1-11　鸡兔同笼

```
#include<stdio.h>
int main()
{
  int a, b, n, m;
  scanf("%d%d", &n, &m);
  a = (4*n-m)/2;
  b = n-a;
  if(m % 2 == 1 || a < 0 || b < 0)
    printf("No answer\n");
  else
    printf("%d %d\n", a, b);
  return 0;
}
```

上面的程序用到了 if 语句，其一般格式是：

```
if(条件)
    语句1;
else
    语句2;
```

注意语句 1 和语句 2 后面的分号，以及 if 后面的括号。“条件”是一个表达式，当该表达式的值为“真”时执行语句 1，否则执行语句 2。另外，“else 语句 2”是可以省略的。

语句 1 和语句 2 前面的空行是为了让程序更加美观，并不是必需的，但强烈推荐读者使用。

提示 1-15：if 语句的基本格式为：if(条件) 语句 1; else 语句 2。

换句话说，“m%2==1 || a < 0 || b < 0”是一个表达式，其字面意思是“m 是奇数，或者 a 小于 0，或者 b 小于 0”。这句话可能正确，也可能错误。因此这个表达式的值可能为真，也可能为假，取决于 m、a 和 b 的具体数值。

这样的表达式称为逻辑表达式。和算术表达式类似，逻辑表达式也由运算符和值构成，例如“||”运算符称为“逻辑或”，a || b 表示 a 为真，或者 b 为真。换句话说，a 和 b 只要有一个为真，a || b 就为真；如果 a 和 b 都为真，则 a || b 也为真。和其他语言不同的是，在 C 语言中单个整数也可以表示真假，其中 0 为假，其他值为真。

提示 1-16：if 语句的条件是一个逻辑表达式，它的值可能为真，也可能为假。单个整数值也可以表示真假，其中 0 为假，其他值为真。

细心的读者也许发现了，如果 a 为真，则无论 b 的值如何，a || b 均为真。换句话说，一旦发现 a 为真，就不必计算 b 的值。C 语言正是采取了这样的策略，称为短路(short-circuit)。也许读者会觉得，用短路的方法计算逻辑表达式的唯一优点是速度更快，但其实并不是这样，稍后将通过几个例子予以证实。

提示 1-17：C 语言中的逻辑运算符都是短路运算符。一旦能够确定整个表达式的值，就不再继续计算。

例题 1-5　三整数排序

输入 3 个整数，从小到大排序后输出。

样例输入：

```
20 7 33
```

样例输出：

```
7 20 33
```

【分析】

a、*b*、*c* 这 3 个数一共只有 6 种可能的顺序：*abc*、*acb*、*bac*、*bca*、*cab*、*cba*，所以最简单的思路是使用 6 条 if 语句。

程序 1-12　三整数排序（1）（错误）

```
#include<stdio.h>
int main()
{
  int a, b, c;
  scanf("%d%d%d", &a, &b, &c);
  if(a < b && b < c)printf("%d %d %d\n", a, b, c);
  if(a < c && c < b)printf("%d %d %d\n", a, c, b);
```

```
  if(b < a && a < c)printf("%d %d %d\n", b, a, c);
  if(b < c && c < a)printf("%d %d %d\n", b, c, a);
  if(c < a && a < b)printf("%d %d %d\n", c, a, b);
  if(c < b && b < a)printf("%d %d %d\n", c, b, a);
  return 0;
}
```

上述程序看上去没有错误，而且能通过题目中给出的样例，但可惜有缺陷：输入“1 1 1”将得不到任何输出！这个例子说明：即使通过了题目中给出的样例，程序仍然可能存在问题。

提示 1-18：*算法竞赛的目标是编程对任意输入均得到正确的结果，而不仅是样例数据。*

将程序稍作修改：把所有的小于符号“<”改成小于等于符号“<=”（在一个小于号后添加一个等号）。这下总可以了吧？很遗憾，还是不行。对于“1 1 1”，6种情况全部符合，程序一共输出了6次“1 1 1”。

一种解决方案是人为地让6种情况没有交叉：把所有的if改成else if。

程序 1-13　三整数排序（2）

```
#include<stdio.h>
int main()
{
  int a, b, c;
  scanf("%d%d%d", &a, &b, &c);
  if(a <= b && b <= c) printf("%d %d %d\n", a, b, c);
  else if(a <= c && c <= b) printf("%d %d %d\n", a, c, b);
  else if(b <= a && a <= c) printf("%d %d %d\n", b, a, c);
  else if(b <= c && c <= a) printf("%d %d %d\n", b, c, a);
  else if(c <= a && a <= b) printf("%d %d %d\n", c, a, b);
  else if(c <= b && b <= a) printf("%d %d %d\n", c, b, a);
  return 0;
}
```

最后一条语句还可以简化成单独的else（想一想，为什么），不过，幸好程序正确了。

提示 1-19：*如果有多个并列、情况不交叉的条件需要一一处理，可以用else if语句。*

另一种思路是把a、b、c这3个变量本身改成$a \leqslant b \leqslant c$的形式。首先检查$a$和$b$的值，如果$a>b$，则交换$a$和$b$（利用前面讲过的三变量交换法）；接下来检查$a$和$c$，最后检查$b$和$c$，程序如下：

程序 1-14　三整数排序（3）

```
#include<stdio.h>
int main()
{
```

```
    int a, b, c, t;
    scanf("%d%d%d", &a, &b, &c);
    if(a > b) { t = a; a = b; b = t; } //执行完毕之后 a≤b
    if(a > c) { t = a; a = c; c = t; } //执行完毕之后 a≤c，且 a≤b 依然成立
    if(b > c) { t = b; b = c; c = t; }
    printf("%d %d %d\n", a, b, c);
    return 0;
}
```

为什么这样做是对的呢？因为经过第一次检查以后，必然有 $a \leqslant b$，而第二次检查以后 $a \leqslant c$。由于第二次检查以后 a 的值不会变大，所以 $a \leqslant b$ 依然成立。换句话说，a 已经是 3 个数中的最小值。接下来只需检查 b 和 c 的顺序即可。值得一提的是，上面的代码把上述推理写入注释，成为程序的一部分。这不仅可以让其他用户更快地搞懂你的程序，还能帮你自己理清思路。在 C 语言中，单行注释从“//”开始直到行末为止；多行注释用“/*”和“*/”包围起来[①]。

提示 1-20：适当在程序中编写注释不仅能让其他用户更快地搞懂你的程序，还能帮你自己理清思路。

注意上面程序中的花括号。前面讲过，if 语句中有一个“语句 1”和可选的“语句 2”，且都要以分号结尾。有一种特殊的“语句”是由花括号括起来的多条语句。这多条语句可以作为一个整体，充当 if 语句中的“语句 1”或“语句 2”，且后面不需要加分号。当然，当 if 语句的条件满足时，这些语句依然会按顺序逐条执行，和普通的顺序结构一样。

提示 1-21：可以用花括号把若干条语句组合成一个整体。这些语句仍然按顺序执行。

1.5　注解与习题

经过前几个小节的学习，相信读者已经初步了解顺序结构程序设计和分支结构程序设计的核心概念和方法，然而对这些知识进行总结，并且完成适当的练习是很必要的。

为了突出实践的重要性，本章从一开始就不加解释地给出了一段程序，并鼓励读者暂时忽略不理解的细节，把注意力集中在变量、表达式、赋值等核心内容。然而，实践的步伐也不是越快越好，因此笔者在每章的最后加入一些理论知识，供读者在实践之余稍加注意。也可以直接跳到第 2 章继续阅读，以后再阅读（并且实践）这些文字。

1.5.1　C 语言、C99、C11 及其他

本书的前 4 章介绍 C 语言，更具体地说是介绍 C99 标准中对算法竞赛而言最核心的部分。C 语言的历史和特点不难在网上以及其他书籍中找到，并且本书的前言中也详细叙述了为什么要介绍 C 语言，因此这里唯一想讲的是 C99 和编译器。

[①] 单行注释原先只有 C++支持，后来已成为 C99 的标准的一部分。

什么是编译器？简单地说，编译器的任务就是把人类可以看懂的源代码变成机器可以直接执行的指令。“机器可以直接执行的指令”很抽象，并且笔者也无意在这里进行进一步的解释——但有一点可以说明，那就是这里的“机器”有很多种，甚至还可以是非物理的虚拟机器。诚然，让同一段程序完美地运行在千差万别的机器上并不是容易的事情，但编译器仍然大大减轻了工作量。

C语言并不是只有一种编译器[①]，例如gcc和微软的Visual C++系列[②]。为了避免同一段程序被不同的编译器编译成截然不同的机器指令，C语言标准诞生了。目前最新的是C11，其次是C99。考虑到C11的新特性未影响算法竞赛[③]，因此这里仍然讨论C99。正如前言中所说，本书介绍C语言只是为学习C++做铺垫。C99中最常用的特性已经基本包含在了C++中（例如64位整数、随处声明变量、单行注释），所以在前4章中无须过多地关注哪些特性是C99新增的，哪些是ANSI C（即C89）中已经包含的特性，把更多的注意力放在代码和算法本身。

本书介绍C语言的目的是为C++语言铺垫（因为后面章节的代码用了很多C++特性），但是**有读者仍然希望先学习到“纯粹的C”，所以在写作本书时确保了前4章中的代码全部能使用gcc -std=c99编译通过**[④]。“与C99兼容”是要付出代价的。例如，在C99中，double的输出必须用%f，而输入需要用%lf，但是在C89和C++中都不必如此——输入输出可以都用%lf。为了保持与C99兼容，不得不向这种不一致性妥协。如果一开始就使用C++，则不必拘泥于C99，把所有代码以.cpp而不是.c为扩展名保存，用C++编译器编译即可。**本书前4章中的代码均可以直接用C++编译器编译**。不仅如此，多数比赛中的C语言都是指ANSI C，即C89而不是C99，**在参加比赛时也需要把C语言程序当作C++程序提交**。

是不是很晕？没关系，只要你不是一个纯粹主义者，作者最推荐的方式就是：从现在开始直接认为你学的不是C语言，而是C++语言中与C兼容的部分。这样一来，ANSI C、C99之类的名词都和你无关了。

1.5.2 数据类型与输入格式

在继续学习之前，强烈建议读者完成以下两个实验。它们不仅能帮助你搞清楚数据类型以及输入输出的一些细节，还能培养你的实践习惯，锻炼实践能力。

数据类型实验。本章中涉及的int和double并不能保存任意的整数和浮点数。它们究竟有着怎样的限制呢？不必解释背后的原因，但需注意现象。

实验A1：表达式11111*11111的值是多少？把5个1改成6个1呢？9个1呢？

实验A2：把实验A1中的所有数换成浮点数，结果如何？

实验A3：表达式sqrt(-10)的值是多少？尝试用各种方式输出。在计算的过程中系统会报错吗？

① 事实上，它甚至有多种解释器——无须编译直接执行的C语言解释器，例如Ch和TCC。
② Visual C++不仅包含IDE，也包含C和C++编译器。
③ 有一个例外：gets在C11中被移除了。详见第3章。
④ 如果使用其他编译器，请自行查阅相关文档，确保代码按照C99标准编译，否则可能会出现编译错误。

实验 A4：表达式 1.0/0.0、0.0/0.0 的值是多少？尝试用各种方式输出。在计算的过程中系统会报错吗？

实验 A5：表达式 1/0 的值是多少？在计算的过程中系统会报错吗？

输入格式实验。本章介绍了 scanf 和 printf 这两个最常见的输入输出函数。考虑下面的函数段，可以从实验结果总结出什么样的规律？

程序 1-15　输入输出实验

```
#include<stdio.h>
int main()
{
  int a, b;
  scanf("%d%d", &a, &b);
  printf("%d %d\n", a, b);
  return 0;
}
```

实验 B1：在同一行中输入 12 和 2，并以空格分隔，是否得到了预期的结果？

实验 B2：在不同的两行中输入 12 和 2，是否得到了预期的结果？

实验 B3：在实验 B1 和 B2 中，在 12 和 2 的前面和后面加入大量的空格或水平制表符（TAB），甚至插入一些空行。

实验 B4：把 2 换成字符 s，重复实验 B1~B3。

输出技巧。读者有没有注意到在本章中所有的 printf 中，双引号中的内容总是以\n 结尾？\n 是一个特殊字符，叫做“换行符”，其中 n 是英文单词 newline（换行）的首字母。换句话说，在输出的最后加一个\n 会在输出结束后换行。既然“换行”只是一个特殊字符，完全可以用 printf("1\n2\n")分两行输出 1 和 2，并且用“printf("1\n\n2\n");”分三行输出 1 和 2，并且在 1 和 2 中间换一行。更多的特殊字符将在第 3 章中介绍。但是这样一来，问题出现了：如果真的要输出斜线“\”和字符 n，怎么办？方法是“printf("\\n");”，编译器会把双斜线“\\”理解成单个字符“\”[①]。

最后请读者思考这样一个问题：如何连续输出“%”和 d 两个字符？不难发现使用“printf("%d\n");”是不行的，那么应该怎样办呢？读者可以自行尝试，也可以查阅 printf 的资料[②]。从一开始就养成查文档的好习惯是有益的。

1.5.3　习题

程序设计是一门实践性很强的学科，读者应在继续学习之前确保下面的题目都能做出来。请先独立完成，如果有困难可以翻阅本书代码仓库中的答案，但一定要再次独立完成。

习题 1-1　平均数（average）

输入 3 个整数，输出它们的平均值，保留 3 位小数。

[①] 这是一个很有意思的设计，建议读者花时间琢磨一下这样做的用意。

[②] 例如 http://en.wikipedia.org/wiki/Printf。

习题 1-2　温度（temperature）

输入华氏温度 f，输出对应的摄氏温度 c，保留 3 位小数。提示：$c=5(f-32)/9$。

习题 1-3　连续和（sum）

输入正整数 n，输出 $1+2+\cdots+n$ 的值。提示：目标是解决问题，而不是练习编程。

习题 1-4　正弦和余弦（sin 和 cos）

输入正整数 n（$n<360$），输出 n 度的正弦、余弦函数值。提示：使用数学函数。

习题 1-5　打折（discount）

一件衣服 95 元，若消费满 300 元，可打八五折。输入购买衣服件数，输出需要支付的金额（单位：元），保留两位小数。

习题 1-6　三角形（triangle）

输入三角形 3 条边的长度值（均为正整数），判断是否能为直角三角形的 3 个边长。如果可以，则输出 yes，如果不能，则输出 no。如果根本无法构成三角形，则输出 not a triangle。

习题 1-7　年份（year）

输入年份，判断是否为闰年。如果是，则输出 yes，否则输出 no。

提示：简单地判断除以 4 的余数是不够的。

接下来的题目需要更多的思考：如何用实验方法确定以下问题的答案？注意，不要查书，也不要在网上搜索答案，必须亲手尝试——实践精神是极其重要的。

问题 1：int 型整数的最小值和最大值是多少（需要精确值）？

问题 2：double 型浮点数能精确到多少位小数？或者，这个问题本身值得商榷？

问题 3：double 型浮点数最大正数值和最小正数值分别是多少（不必特别精确）？

问题 4：逻辑运算符号“&&”、“||”和“!”（表示逻辑非）的相对优先级是怎样的？也就是说，a&&b||c 应理解成(a&&b)||c 还是 a&&(b||c)，或者随便怎么理解都可以？

问题 5：if(a) if(b) x++; else y++的确切含义是什么？这个 else 应和哪个 if 配套？有没有办法明确表达出配套方法？

1.5.4　小结

对于不少读者来说，本章的内容都是直观、容易理解的，但这并不意味着所有人都能很快地掌握所有内容。相反，一些勤于思考的人反而更容易对一些常人没有注意到的细节问题产生疑惑。对此，笔者提出如下两条建议。

一是重视实验。哪怕不理解背后的道理，至少要清楚现象。例如，读者若亲自完成了本章的探索性实验和上机练习，一定会对整数范围、浮点数范围和精度、特殊的浮点值、scanf、空格、TAB 和回车符的过滤、三角函数使用弧度而非角度等知识点有一定的了解。这些内容都没有必要死记硬背，但一定要学会实验的方法。这样即使编程时忘记了一些细节，手边又没有参考资料，也能轻松得出正确的结论。

二是学会模仿。本章始终没有介绍“#include<stdio.h>”语句的作用，但这丝毫不影响读者编写简单的程序。这看似是在鼓励读者“不求甚解”，但实为考虑到学习规律而作出的决策：初学者自学和理解能力不够，自信心也不够，不适合在动手之前被灌输大量的理

论。如果初学者在一开始就被告知“stdio 是 standard I/O 的缩写，stdio.h 是一个头文件，它在 XXX 位置，包含了 XXX、XXX、XXX 等类型的函数，可以方便地完成 XXX、XXX、XXX 的任务；但其实这个头文件只是包含了这些函数的声明，还有一些宏定义，而真正的函数定义是在库中，编译时用不上，而在连接时……”多数读者会茫然不知所云，甚至自信心会受到打击，对学习 C 语言失去兴趣。正确的处理方法是“抓住主要矛盾”——始终把学习、实验的焦点集中在最有趣的部分。如果直观地解决方案行得通，就不必追究其背后的原理。如果对一个东西不理解，就不要对其进行修改；如果非改不可，则应根据自己的直觉和猜测尝试各种改法，而不必过多地思考“为什么要这样”。

当然，这样的策略并不一定持续很久。当学生有一定的自学、研究能力之后，本书会在适当的时候解释一些重要的概念和原理，并引导学生寻找更多的资料进一步学习。要想把事情做好，必须学得透彻，但没有必要操之过急。