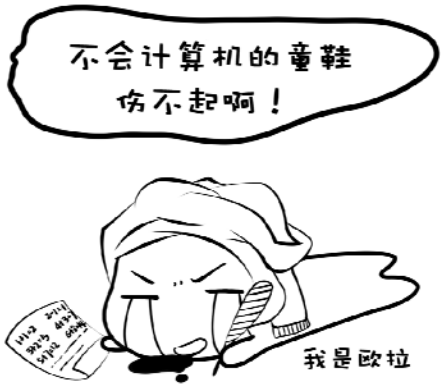
第 2 章

梦 想 起 航

第 1 节　编程的魔力

从一个神奇的数字说起——2 147 483 647。

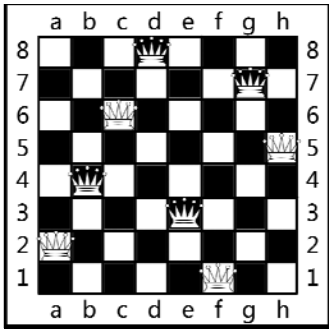
2 147 483 647 是一个质数（也称为素数，即只能被 1 和其本身整除的数）。发现这个质数的人是伟大的欧拉同学。1722 年，他在双目失明的情况下，以惊人的毅力靠心算证明了 2 147 483 647 是一个质数，堪称当时已知的世界上最大的质数，他也因此获得了“数学英雄”的美名。现在你通过计算机只需要 1 秒就可以证明 2 147 483 647 是一个质数。



再来看一个经典的问题——八皇后问题。

如何能够在 8×8 的国际象棋棋盘上放置 8 个“皇后”，使得任何一个“皇后”都无法直接吃

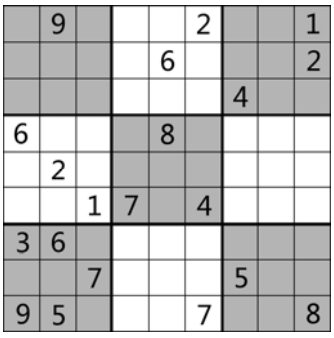
掉其他“皇后”？为了达到这个目的，任意两个“皇后”都不能处于同一条横行、纵行或斜线上。下面就是一种解决方案。没错，你可以自己拿出笔和纸画一画，看看还有没有其他方案。但是，如果我想知道所有的方案该怎么办？



又轮到计算机出马了，一共有 92 种不同的解决方案，很棒吧！计算机只需要 1 秒，就可以算出所有的解。

再来看一个很流行的益智游戏——数独。

在一个 9×9 格的大九宫格中有 9 个 3×3 的小九宫格，默认在其中填写了一些数字，现在请在其他空格上填入数字 1～9。每个数字在每个小九宫格内只能出现一次，每个数字在每行每列也只能出现一次。请看下面这个例子。



我想，你一定很快就找到了一种可行解，可是你知道上面这个数独一共有多少种不同解吗？99 410 种不同解！很难想象吧，计算机仍然只需 1 秒！怎么样，计算机编程是不是很神奇，你甚至可以轻而易举地在一定范围内去验证“哥德巴赫猜想”。

在接下来的内容里你将学会如何与计算机对话，如何让计算机进行数学计算和判断，如何让计算机永不停止地工作，以及做一些很有意思的程序和游戏。一场有趣的逻辑思维大战即将开始，不要走开，赶快进入第 2 节——让计算机开口说话！

第 2 节　让计算机开口说话

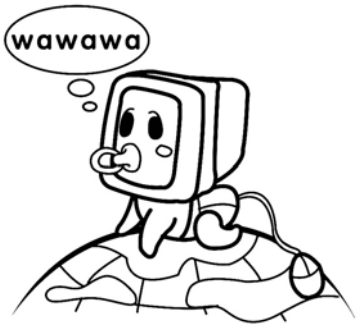
为什么会有计算机的出现呢？我们伟大的人类，发明的每一样东西都是为了帮助我们改善生活。计算机同样是用来帮助我们的工具。想一想，假如你现在希望让计算机帮助你做一件事情，你首先需要做什么？是不是要先与计算机进行沟通？那么沟通就需要依赖于一门语言。人与人的沟通，可以用肢体语言、汉语、英语、法语和德语等。你若要与计算机沟通，就需要使用计算机能够听懂的语言。我们学习的“C 语言”便是计算机语言的一种，计算机语言除了 C 语言外，还有 C++、Java、C#等。C 语言是一门比较简单的计算机语言，更加适合初学者。所有的计算机语言都是相通的，如果你能够熟练掌握 C 语言，那么再学习其他语言就会变得易如反掌。

既然计算机是人类制造出来的帮助人类的工具，显然让计算机开口说话，让计算机把“它”所知道的东西告诉我们是非常重要的。

下面我们就来解决第一个问题：如何让计算机开口说话？

回想当年，我们刚刚来到这个世界的时候，说的第一句话是什么？应该不会是“你好!”、“吃了没？”……这样会把你的爸爸妈妈吓坏的!

伴随着“wa wa wa”的一阵哭声，我们来到了这个精彩的世界。现在我们也让计算机来“哭一次”。这个地方特别说一下，计算机要把“它”想说的告诉我们，有两种方法，一种是显示在显示器屏幕上，另一种是通过喇叭发出声音。就如同我们，一种是写在纸上，另一种是用嘴巴说出来。目前我们让计算机用音箱输出声音还比较麻烦，因此我们采用另外一种方法，即用屏幕输出“wa wa wa”。




```
    printf("wa wa wa");
```

这里有一个生疏单词叫作 printf，你不要被它吓坏了，目前你不用搞清楚它的本质意义是什么，只要记住它和中文里面的“说”，以及英文里面的“say”是一个意思，就是控制计算机说话的一个单词而已。在 printf 后面紧跟的( )，是不是很像一个嘴巴，把要说的内容“放在”这个“嘴巴”里。这里还有一处需要注意，在"wa wa wa"的两边还有""，里面就是计算机需要“说”的内容，这一点是不是很像我们的汉语？最后，一句话结束时要有一个结束的符号。汉语中用句号表示一句话的结束；英语中用点号表示一句话的结束；计算机语言中用分号表示一个语句的结束。

注：计算机的每一句话，就是一个语句。

好了，现在如果让你写一个语句，让计算机说“ni hao”，该怎么办？

```
printf("ni hao");
```

我们现在让计算机来运行这个语句，这里需要说明一下，仅仅输入 printf("ni hao");，我们的计算机是识别不了的，需要加一个框架。完整的程序如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      printf("ni hao");
      return 0;
}
```

这里的

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      return 0;
}
```

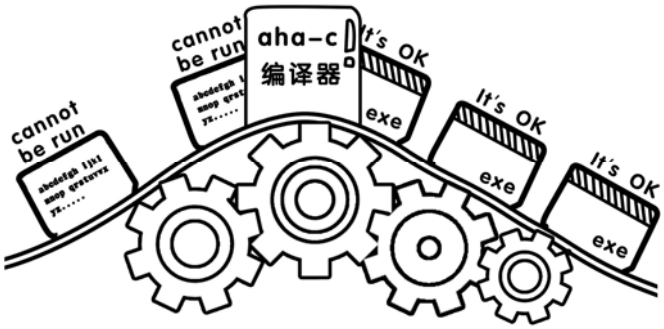
是所有 C 语言都必须要有的框架，现在你暂时不需要理解它，知道要有这个即可，以后再来详细地讲它的用途。但是有一点，我们今后写的所有类似 printf 的语句都要写在{ }里才有效。

接下来我们需要让计算机运行我们刚才写的程序。

如果让计算机运行我们写的东西（其实我们写的就是一个 C 语言程序），需要一个特殊的软件，它叫作“C 语言编译器”[1]，“C 语言编译器”有很多种，我们这里介绍一种比较简单的软件，

[1] “C 语言编译器”的作用是把我们写的程序“变”成一个“exe”，即可以让计算机直接运行的程序。这个“变”的专业术语称为“编译”。当你的程序“变”成一个“exe”后，你就可以脱离“C 语言编译器”直接运行你

叫作“啊哈 C”[2]。



首先你需要到 www.ahalei.com 下载“啊哈 C”。下面就要进入安装步骤啦，安装很简单，一共分 7 步（见图 2-1～图 2-7），每一步我都截取了图片，你只需一口气将这 7 幅图片全部看完应该就可以。

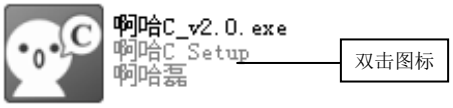


图 2-1　安装“啊哈 C”

的程序。此时你就可以把你写的“exe”发给你的朋友和同学，让他们一起来使用你编写的程序。这里的程序从某种意义上来讲也可以称为“软件”。

[2] “啊哈 C”是一款非常容易上手的 C 语言编程软件，使用的是 GCC 内核。界面简洁可爱，支持语法高亮、代码折叠、编译错误提示等。操作方便，上手快，特别适合 C 语言入门的初学者使用。

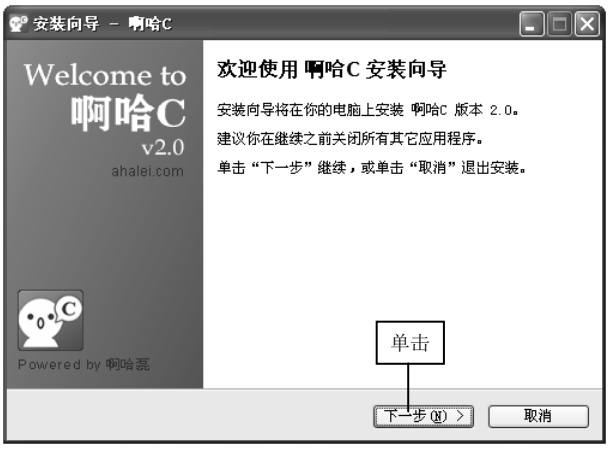


图 2-2　开始安装“啊哈 C”

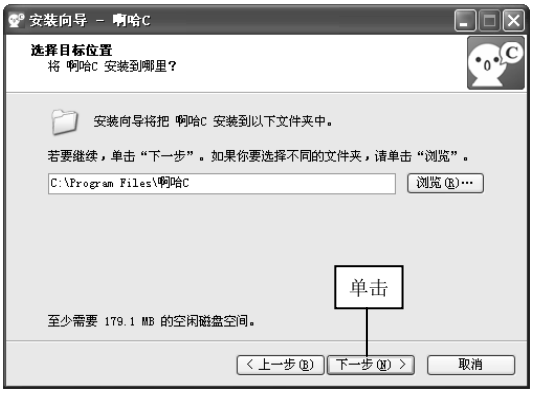


图 2-3　设置“啊哈 C”安装目录

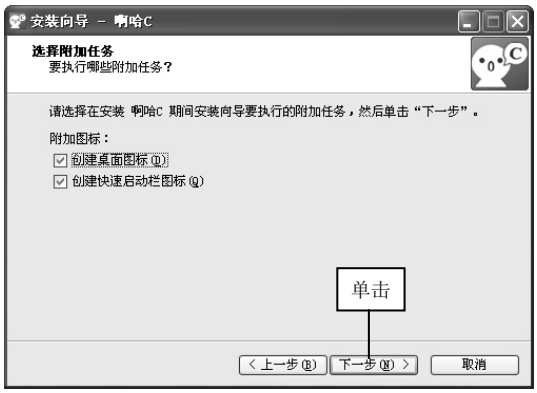


图 2-4　创建桌面图标和启动栏图标

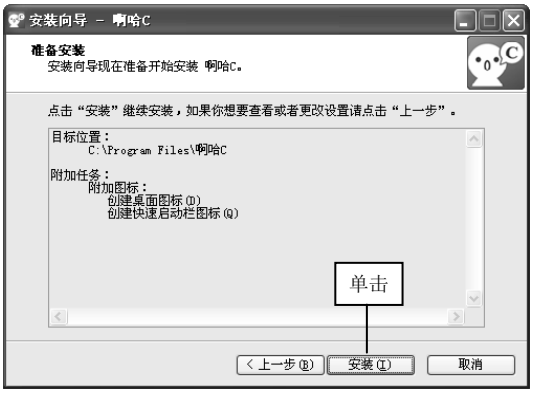


图 2-5　确认安装信息

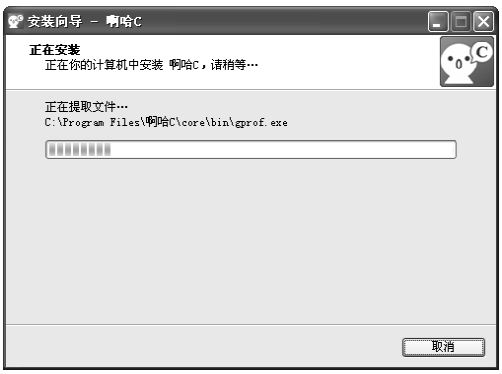


图 2-6　安装正在进行

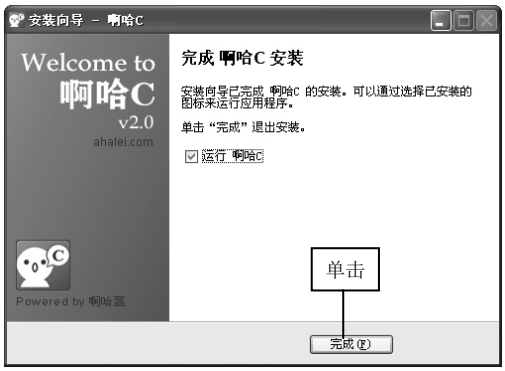


图 2-7 “啊哈 C”安装成功

“啊哈 C”安装完毕后，我们便可以看到如图 2-8 所示的“啊哈 C”的界面，同时在你的桌面上也会多一个“啊哈 C”图标。

图 2-8 “啊哈 C”的界面

“啊哈 C”是一个很人性化的软件，你将会发现“啊哈 C”已经帮你将 C 语言代码框架的那几行代码写好了。我们只需要将

```
printf("ni hao");
```

这条语句在“啊哈 C”中输入就好，如图 2-9 所示。

图 2-9　输入 printf("ni hao")

细心的同学可能会发现，“啊哈 C”默认的 C 语言框架，比我们之前说的 C 语言框架多了一句话：

```
system("pause");
```

这句话是什么意思呢？稍后我们再揭晓，我们先将这句话删除，删除后的界面如图 2-10 所示。

图 2-10　删除 system("pause")

好了，同学们请注意，到了最后一步，我们需要让我们的代码运行起来。现在你只需单击“啊哈 C”上的“运行”按钮 ▶ 。

接下来，你需要为所写的程序起一个名字，我为这个程序起的名字是“nihao”，当然你可以随便起名，中英文都可以。比如你可以称之为“abc”或“我的第一个程序”，或者叫“1”都行，但是你最好别写火星文或者特殊字符哦，也不能有英文的点号。将程序的名字输入在如图 2-11

所示的文本框中之后再单击“保存”按钮，接下来就是见证奇迹的时刻。

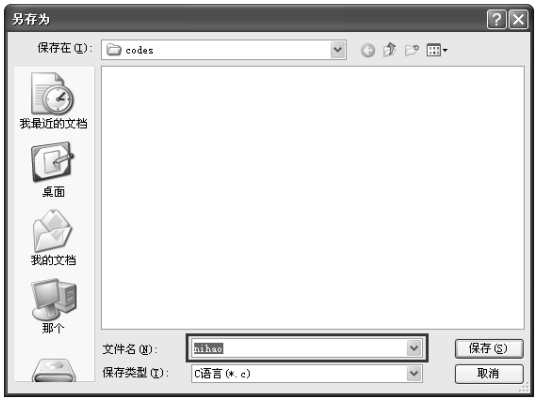


图 2-11　给程序起个名字

如果你的代码没有写错，那你的“啊哈 C”将会弹出一个对话框，提示“恭喜你编译成功”，如图 2-12 所示。请同学们注意，在输入代码的时候，一定不要用中文输入法，这里所有的符号都是英文的，一般也都是小写。

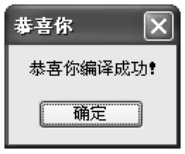


图 2-12　编译成功的提示

下面当然是单击“确定”按钮啦。接下来，请注意：请注视你的计算机屏幕，一秒也不要走开，数秒之后，你将会发现计算机的屏幕上有一个“黑影”闪过，如果你没有发现这个“黑影”，请重新单击“运行”按钮，并再次注视你的计算机屏幕。

此时，你可能想问，为什么屏幕上会出现这个“黑影”？但我们是要在屏幕上显示“ni hao”才对啊。其实刚才那个“黑影”就是“ni hao”，只不过计算机的运行速度太快了，在屏幕上显示之后，就立即消失了。那应该怎么办呢？我们需要让计算机暂停一下。

```
system("pause");
```

上面这句话是我们之前删除了的，其实它的作用就是让计算机“暂停一下”。好了，我们将这句话放在 printf("ni hao");后面，完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      printf("ni hao");
      system("pause");
      return 0;
}
```

好了，再次单击“运行”按钮吧。如果代码没有错误，你将看到如图 2-13 所示的界面。

图 2-13　运行成功的结果

“请按任意键继续…”是 system("pause");输出的一个提示，此时你只需按键盘上的任意一个键，这个小黑窗口就会关闭。

如果你想让“ni hao”分两行显示，则只需要将 printf("ni hao"); 改为 printf("ni \n hao"); 这里的“\n”表示的就是“换行”。注意，这里的“\”是向右下角斜的，它在键盘上的位置，通常是在回车键的上方。好，赶快尝试一下吧。运行结果如图 2-14 所示。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      printf("ni\nhao");
      system("pause");
      return 0;
}
```

图 2-14　分行后的运行结果

当然你也可以让“请按任意键继续...”在下一行显示，只需将 printf("ni\nhao"); 改为 printf("ni \n hao\n"); 即可，去试一试吧。

一起来找茬

1．下面这段代码是让计算机在屏幕上输出 hi。其中有 3 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
      print(hi)
      system("pause");
      return 0;
}
```

更进一步，动手试一试

1．尝试一下让计算机显示下面这些图形。

```
*
**
***
```

```
  *
 * *
*   *
 * *
  *
```

```
      *
     *
    *
*   *
 * *
  *
```

2．如何让计算机说中文呢？请让计算机像下面一样说“早上好”，应该怎么办？

3．再尝试一下让计算机显示下面这个图形。

```
A
BC
DEF
```

```
GHIJ
KLMNO
PRSTUV
W
X
Y
Z
```

✈ 这一节，你学到了什么

1．如何让计算机开口说话，以及让计算机开口说话的语句是什么？

第 3 节　多 彩 一 点

在本章第 2 节我们学习了让计算机开口说话应使用 printf 语句。我们发现，计算机“说”出的话都是黑底白字，其实计算机的输出可以是彩色的，我们一起来看看吧。

注意，此处代码只能在 windows 操作系统下编译运行。如果你使用的是本书推荐的 C 语言的软件“啊哈 C”，那么你的代码肯定可以运行成功。OK，下面我们来看看，如何让颜色出现。

请尝试输入以下代码并运行，看看会发生什么。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   system("color 5");
   printf("wa wa wa");
   system("pause");
   return 0 ;
}
```

运行之后你发现了什么？底色仍然是黑色。但是，文字的颜色已经变为“紫色”了，奥秘就在下面这行代码中。

```
system("color 5");
```

在这句话中，“5”代表“紫色”，你可以尝试一下其他数字，看看分别是什么颜色。

既然字的颜色可以变，那么背景色是否可以变呢？尝试一下下面这段代码：

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

```
    system("color f5");
    printf("wa wa wa");
    system("pause");
    return 0;
}
```

运行成功后的界面如图 2-15 所示。

图 2-15　运行成功后的界面（此时背景应该为白色，文字颜色应该为紫色）

上面这段代码在原来的 5 前面加了一个 f，这里的 f 代表背景色是白色。

那么设置背景色和文字颜色的方法是，在 color 后面加上两个一位数字，第一个数字表示背景色，第二个数字表示文字颜色。如果在 color 后面只加了一个一位数字，则表示只设置文字颜色，背景色仍然使用默认的颜色。

需要说明的是这里的一位数字其实是 16 进制的，它只能是 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f 中的某一个。

[题外话]“不看，也无伤大雅”

这里我们学习了一个新知识：进制。

在现代数学中，我们通常使用十进制，即使用数字 0、1、2、3、4、5、6、7、8、9。9 之后的数字便无法表示了，我们的解决方法是：使用“进位”来表示。例如，由于阿拉伯数字只到 9，于是我们便进一位，当前这位用 0 表示，便产生了用 10 来表示“十”。因为是“逢十进一”，所以称为十进制。

而十六进制是“逢十六进一”，即使用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 来表示。0～9 与在十进制时相同，但是“十”在十六进制时用大写字母 A 表示，以此类推，“十五”在十六进制中用大写字母 F 来表示。F 是“十六进制”中的最后一个，因此数字“十六”就表示不了。于是我们又采用刚才在十进制中表示不了就进一位的老办法，当前应该用 0 表示。“十六”在十六进制中表示为 10。同理，“二十七”在十六进制中表示为 1B。

在中国古代，很多朝代都是用十六进制作为日常计数的，例如，成语“半斤八两”的典故来源于十六进制；还有中国古代的算法是上面 2 颗珠子，下面 5 颗珠子。若上面每颗珠子代表

数字 5，下面每颗珠子代表数字 1，那么每位的最大计数值是 15，15 正是十六进制的最大基数。当使用算盘计数遇到大于 15 的时候，我们就需要在算盘上“进位”了。

其实在我们现代的日常生活中，也不都是“十进制”，例如，60 秒为 1 分钟，60 分钟为 1 小时，就是用的六十进制。

一起来找茬

1．下面这段代码是让计算机在屏幕上输出绿底白字的 hi。其中有 4 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
      system(color f2)
      print("hi");
      system("pause");
      return 0;
}
```

更进一步，动手试一试

1．尝试一下让计算机打印这个小飞机图案（绿底白字）。

```
     *
     **
*    ***
**   ****
**************
**   ****
*    ***
     **
     *
```

2．尝试一下让计算机打印这个小队旗图案（白底红字）。

```
A
I*
I**
I***
I****
I*****
I
I
I
I
```

✈ 这一节，你学到了什么

1．让计算机打印出来的字符有不同颜色的语句是什么？

第 4 节　让计算机做加法

通过之前的学习，我们了解到让计算机说话是用“printf”，运用“printf”我们就可以让计算机想说什么就说什么了。在学会了“说话”之后，我们来看如何让计算机做数学运算，首先我们先让计算机做“加法”，就先算 1+2 吧。

回想一下小时候爸爸妈妈是如何教我们算 1+2 的呢？

妈妈说：“左手给你一个苹果，右手给你两个苹果，现在一共有几个苹果？”我们迅速地思考了一下，脱口而出：“3 个苹果”。没错！我们首先用大脑记住左手有几个苹果，再用大脑记住右手有几个苹果，妈妈问我们一共有几个时，我们的大脑进行了非常快速的计算，将刚才记住的两个数进行相加，得到结果，最后将计算出的结果说出来。我们仔细分析一下，大致分为以下 5 个步骤。

（1）用大脑记住左手的苹果数量；

（2）用大脑记住右手的苹果数量；

（3）我们的大脑将两个数字进行相加；

（4）得到结果；

（5）将结果输出。

在这期间，我们大脑一共进行了以下 4 个动作。

（1）两次输入：分别是记录左手和右手中苹果的数量；

（2）存储了 3 个值：分别是记录左手和右手中苹果的数量和相加的结果；

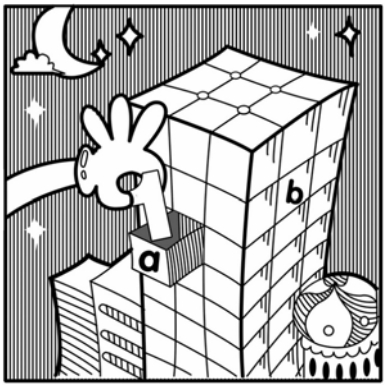
（3）进行了一次计算：相加；

（4）进行了一次输出：把相加的结果输出。

那我们如何让计算机做加法呢？同样也需要做以上几步。

首先我们来解决如何让计算机像我们的大脑一样记住一个数字。

其实计算机的大脑就像一个“摩天大厦”，有很多一间一间的“小房子”，计算机就把需要记住的数放在“小房子”里面，一个“小房子”里只能放一个数，这样计算机就可以记住很多数。好，我们来看一看，具体怎样操作。



“=”赋值符号的作用就相当于一只手，把数字放到小房子中。

```
int a,b,c;
```

这句话就代表在计算机的“摩天大厦”中申请三间分别叫作 a、b 和 c 的小房子（注意：int 和 a 之间有一个空格，a、b 与 c 之间分别用逗号隔开，末尾有一个分号表示结束）。

接下来，我们让小房子 a 和小房子 b 分别去记录两个数字 1 和 2，具体如下：

```
a=1;
b=2;
```

说明：此处有一个“=”，这可不是等于号，它叫作给予号（也称为赋值号），类似于一个箭头“←”，意思是把“=”右边的内容，给了“=”左边的。例如，把 1 这个数给小房子 a，这样一来计算机就知道小房子 a 里面存储的是数字 1 了。

然后，把小房子 a 和小房子 b 里面的数相加，再将其结果放到小房子 c 中。

```
c=a+b;
```

计算机会将这个式子分两步执行：第一步先将 a+b 算出来，第二步再将 a+b 的值给“=”右边的 c。

至此，就差不多完成了，我们总结一下：

```
int a,b,c;
a=1;
b=2;
c=a+b;
```

很多同学是不是以为，现在已经全部完成了？你忘记了最重要的一步，先别急着往下看，想一想忘记了什么？

啊！你忘记了把答案输出。

想一想妈妈问你一加二等于多少时，你说：“我算出来了，但是不想告诉你!”这个时候估计你少不了挨一顿揍了，不要啊!

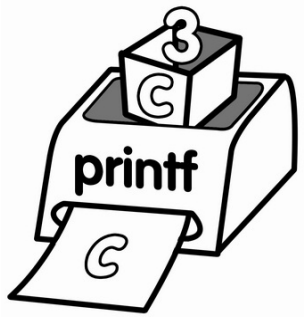
好，那我们回忆一下，应该如何让计算机把结果输出呢。

对，使用 printf 语句。那怎么把小房子 c 里面存储的数输出呢？根据我们在本章第 2 节学到的知识，只要把要输出的内容放在双引号里面就可以了，代码如下：

```
printf("c");
```

那你猜此时计算机会输出什么？

对，无情地输出了一个 c。



那怎样输出 c 里面的值呢？

这时我们要让另外一个角色出场了。

```
%d
```

“%d”其实一个“讨债的”，或者也可以说是“要饭的”。它的专职工作就是向别人“要钱”！那我们应该怎么使用它呢？

```
printf("%d",c)
```

将“%d”放在双引号之间，把小房子 c 放在双引号后面，并且用逗号隔开。

这时 printf 发现双引号里面是个“讨债的”，就知道此时需要输出一个具体的数值，而不是符号，就会向双引号后面的小房子 c 索取具体的数值了。

好了，最后加上 C 语言的代码框架，计算机进行加法运算的完整代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      int a,b,c;
      a=1;
      b=2;
      c=a+b;
      printf("%d",c);
      system("pause");
      return 0;
}
```

现在赶紧去试一试吧。

一起来找茬

1．下面这段代码是让计算机计算 321–123 的结果。其中有 6 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int mian( )
{
      int a,b,c;
      a=321
      b=123
      c=b-a
      print("%d",c)
      system("pause");
      return 0;
}
```

更进一步，动手试一试

1．如果要进行 3 个数相加的运算，该怎样做呢？例如：5+3+1=?

我们可以把上面的程序进行简单地改变，申请 4 个小房子分别叫作 a、b、c 和 d。用 a、b、c 分别存放 3 个加数，用 d 存放它们的和。代码如下：

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

```
        int a,b,c,d;
        a=5;
        b=3;
        c=1;
        d=a+b+c;
        printf("%d",d);

        system("pause");
        return 0;
}
```

如果要 10 个数相加岂不是得定义 11 个小房子？那也太麻烦了吧。对，目前我们只能这样，但是在后面的学习中，会有更为简单的方法。

2．用计算机算出下面 3 个算式。

```
123456789+43214321
7078*8712
321*(123456+54321)
```

✈ 这一节，你学到了什么

1．如何申请一个小房子来存储数字？

2．如何用 printf 输出小房子中的数值？

第 5 节　数字的家——变量

从本章第 4 节中，我们了解到计算机使用一个个的小房子来记住数字。计算机有很多不同种类的小房子。

```
int a;
```

代表向计算机申请一个小房子，用来存放数值，小房子的名字叫作 a。int 和 a 之间有一个空格，a 的末尾有一个分号，表示这句话结束。

如果要申请多个小房子，则要在 a 后面继续加上 b 和 c，用逗号分开。例如：

```
int a,b,c;
```

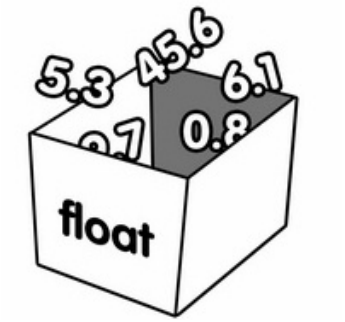
这里有一个小问题，就是给申请的“小房子”起名字。原则上，你可以随便起，叫作 a 可以，叫作 b 也可以，叫作 x 也可以，名字甚至可以是多个字母的组合，例如，可以叫作 aaa，也可以叫作 abc，也可以叫作 book。也可以是字母和数字的组合，例如，a1 或者 abc123 都是可以的。当然也有一些限制，如果你想知道，请看附录 A。

到这里，可能还有很多同学想问，int 究竟是什么意思呢？

其实，int 控制小房子用来存放的数的类型，表示你目前申请的小房子只能存放整数。

int 是英文单词 integer（整数）的缩写。

如果要放小数该怎么办？



我们用 float 来申请一个小房子，用来存放小数，形式如下：

```
float a;
```

这样，小房子 a 就可以用来存放小数了，例如：

```
float a;
a=1.5;
printf("%f",a);
```

就表示申请一个用来存放小数的小房子 a，里面存放了小数 1.5。

注意：在 C 语言中，小数称作浮点数，用 float 表示。

之前我们在用 printf 语句输出整数时，使用的是“%d”。此时需要输出小数，我们要用“%f”。

好了，我们来总结一下，这里的“小房子”在我们 C 语言的专业术语中称为变量。int 和 float 说明小房子是用来存放何种类型的数，我们这里将其称为“变量类型”或者“数据类型”。

类似 int a;或者 float a;的这种形式，我们称作“定义变量”，它们的语法格式如下：

```
口语 [小房子的类型] [小房子的名称] , [小房子的名称] ;
```

```
术语 [变量的类型] [变量的名称] , [变量的名称] ;
代码 int a, b ;
```

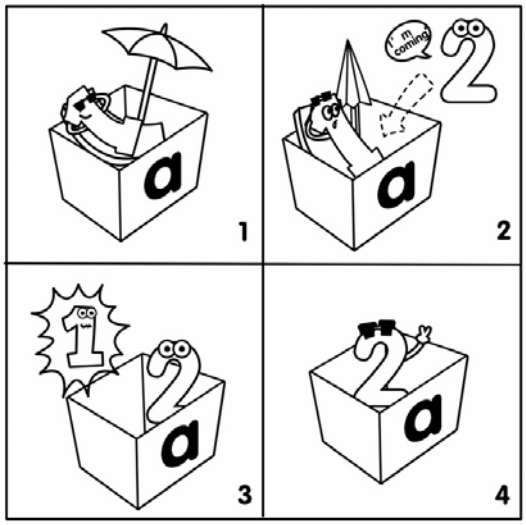
现在我们知道，int a;表示申请一个用来存放一个整数的小房子 a，即定义一个整型变量 a 来存放整数；而 float a;则表示申请一个用来存放一个小数的小房子 a，即定义一个浮点型（实型）变量 a 来存放浮点数（小数）。

再来看另外一个有趣的问题，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      int a;
      a=1;
      a=2;
      printf("%d",a);

      system("pause");
      return 0;
}
```

请问计算机执行完上面的代码后，将会输出 1 还是 2?



尝试过后你会发现，计算机显示的是 2，也就是说小房子 a 中的值最终为 2。通过观察代码我们可以发现，我们首先是将 1 放入小房子 a 中，紧接着又将 2 放入小房子 a 中，那么请问原来小房子中的 1 去哪里了呢？答案是被新来的 2 给覆盖了，原来的 1 已经消失了。也就是说，

小房子 a 中有且仅能存放一个值，如果多次给小房子 a 赋值的话，小房子 a 中存放的将始终是最后一次赋的值。例如：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      int a;
      a=1;
      a=2;
      a=3;
      a=4;
      a=5;
      a=6;
      printf("%d",a);

      system("pause");
      return 0;
}
```

计算机运行完上面这段代码后最终将输出 6。也就是说小房子 a 中的值最终为 6，前 5 次的赋值全部被覆盖了。

一个更有意思的问题来了，请继续看下面的代码：

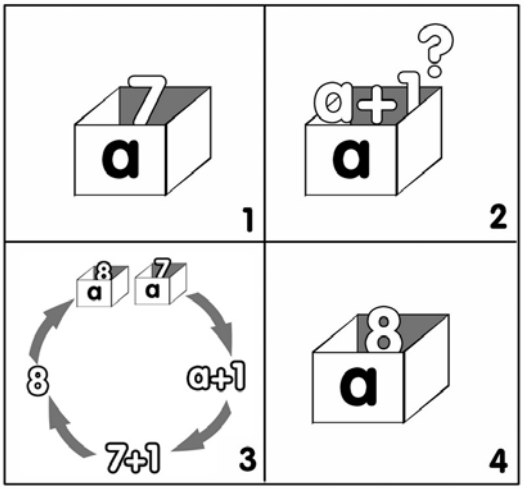
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      int a;
      a=7;
      a=a+1;
      printf("%d",a);

      system("pause");
      return 0;
}
```

计算机运行完上面这段代码后最终将输出 8。也就是说小房子 a 中的值最终为 8。计算机在执行完 a=7 这句话后，小房子 a 中存储的值为 7，之后计算机又紧接着运行了 a=a+1。运行完 a=a+1 后，小房子 a 中的值就变为 8 了。也就是说 a=a+1 的作用是把小房子 a 中的值在原来的基础上增加 1，我们来分析一下这句话。

对于 a=a+1 计算机分两步执行，这句话中有两个操作符，第一个是“+”，另一个是“=”（赋值号），因为“+”的优先级要比“=”高，因此计算机先执行 a+1，此时小房子 a 中的值仍然为

7，所以 a+1 的值为 8。紧接着计算机就会执行赋值语句，将计算出来的值 8 再赋值给 a，此时 a 的值就更新为 8。



好啦，猜猜下面的程序，计算机最终会输出多少？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=10;
    a=a*a;
    printf("%d",a);

    system("pause");
    return 0;
}
```

尝试过了吗？想一想为什么 a 最终的值为 100。

注：所有运算符的优先级详见附录 B。

一起来找茬

1．下面这段代码是让计算机计算 1.2×1.5 的值。其中有 5 个错误，快来改正吧！

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

```
    int a,b,c
    a=1.2;
    b=1.5;
    c=a*b;
    print("%d",c)
    system("pause");
    return 0;
}
```

✈ 更进一步，动手试一试

1．请进行两个小数的加法运算，例如：5.2+3.1=?代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float a,b,c;
    a=5.2;
    b=3.1;
    c=a+b;
    printf("%f",c);

    system("pause");
    return 0;
}
```

请注意，之前我们在 printf 语句中输出整型变量的值时，使用的是“%d”，此时需要输出的是实型变量的值，我们要用“%f”。

2．让计算机把下面 3 个式子算出来吧！

```
1.2+2.3+3.4+4.5
1.1*100
10.1*(10*10)
```

✈ 这一节，你学到了什么

1．如何定义一个用来存放小数的变量？

2．如何让一个小房子 a（变量 a）中的值增加 1？

第 6 节　数据输出——我说咋地就咋地

在本章第 4 节中我们已经学会了如何让计算机做加法运算，但是计算机在输出的时候，只

显示了一个结果，这样不够人性化。如果我们将整个算术等式输出就好了，例如：1+2=3。那应该怎么写呢？

新的代码：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    a=1;
    b=2;
    c=a+b;
    printf("%d+%d=%d",a,b,c);

    system("pause");
    return 0;
}
```

原来的代码：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    a=1;
    b=2;
    c=a+b;
    printf("%d",c);

    system("pause");
    return 0;
}
```

仔细阅读这些代码你会发现，新的代码和原来的代码只有一个 printf 语句不一样。好，我们现在来仔细分析一下 printf("%d+%d=%d",a,b,c);。

printf 语句只会输出双引号里面的部分，双引号之外的部分只是对双引号内的部分起到补充说明的作用。

例如，printf("%d+%d=%d",a,b,c);这行语句，双引号里面的部分是%d+%d=%d，那么计算机在输出的时候就严格按照%d+%d=%d 来执行，输出的形式必然是%d+%d=%d。

当计算机遇到第 1 个“%d”时，知道“讨债的”来了，于是它便向双引号后面的变量讨债，排在第 1 个的是 a，那么就向 a 讨债。a 的值是 1，于是第 1 个“%d”得到的便是 1。

第 2 个是“+”，那么照样输出。

第 3 个又是“%d”，同样到双引号的后面去讨债，因为排在第 1 个的 a 已经被讨过债了，因此向排在第 2 个的 b 讨债。b 的值是 2，于是这个“%d”得到的便是 2。

第 4 个是“=”，照样输出。

第 5 个还是“%d”，同样到双引号的后面去讨债，因为排在第 1 个的 a 和排在第 2 个的 b 已经被讨过债了，因此向排在第 3 个的 c 讨债。c 的值是 3，于是最后这个“%d”得到的便是 3。

最后输出的内容是 1+2=3。

请注意，通常双引号内部“%d”的个数，和后面变量的个数是相等的，它们是一一对应的。如果没有一一对应，从 C 语言的语法角度来讲是没有错误的，但这不合常理，最好避免这样的情况出现。

一起来找茬

1．下面这段代码是让计算机分别计算 10−5 的值与 10+5 的值，并分两行显示，第一行显示差，第二行显示和。其中有 3 个错误，快来改正吧！

```
#inlcude <stdio.h>
#include <stdlib.h>
int mian( )
{
    int a,b,c;
    a=10;
    b=5;
    c=a-b;
    printf("%d/n",c);
    c=a+b;
    printf("%d",c);
    system("pause");
    return 0;
}
```

更进一步，动手试一试

1．指定两个数，输出这两个数的和、差、积与商。例如，这两个数是 9 和 3，输出 9+3=12、9−3=6、9×3=27、9/3=3。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    a=9;
```

```
    b=3;
    c=a+b;
    printf("%d+%d=%d\n",a,b,c);
    c=a-b;
    printf("%d-%d=%d\n",a,b,c);
    c=a*b;
    printf("%d*%d=%d\n",a,b,c);
    c=a/b;
    printf("%d/%d=%d\n",a,b,c);
    system("pause");
    return 0;
}
```

第 7 节　数据输入——我说算啥就算啥

我们已经学会了如何做一个加法计算器，但是我们目前的加法计算器，不够人性化，每次计算两个数的和时，都需要修改我们的 C 语言代码，然后重新编译运行才能得到结果，很显然这样的加法计算器是没有人喜欢用的，那我们如何让使用者自己任意输入两个数，就可以直接得到结果呢？

我们知道，让计算机说话用 printf，那么让计算机学会听用什么呢？scanf 将会把听到的内容告诉你的程序。

计算机“说话”的过程，我们称为“输出”，那计算机“听”的过程，我们则称为“读入”。好，下面我们来看看，计算机是如何读入的。

scanf 的语法与 printf 语法类似，例如，我们要从键盘读入一个数，放在小房子 a 中，代码如下：

```
scanf ("%d", &a) ;
```

你瞧，与输出小房子 a 的语句 printf("%d",a); 是差不多的，只有以下两处不同。

第一处是：读入是使用 scanf 这个词，而输出是使用 printf 这个词。

第二处是：读入比输出在 a 前面多个一个“&”符号。

“&”符号我们称为“取地址符”，简称“取址符”。它的作用是得到小房子 a 的地址。那你可能要问为什么在读入的时候要得到小房子 a 的地址呢？而输出的时候却不需要呢？因为在读入数据的时候，计算机需要把读入的值存放小房子 a（也就是变量 a）中，需要知道你指定的这个小房子 a 的地址，才能把值成功地放进小房子 a 中，但是在输出的时候，值已经在小房子 a 中了，就可以直接输出到屏幕上。我们打一个比方：假如你要去一个教室上课，那么在上课之

前你需要知道这个教室的地址，这样你才能去，但是如果下课了，你需要走出这个教室，因为此时你已经在教室中啦，因此就不再需要这个教室的地址啦。

如果要从键盘读入两个数，分别给小房子 a 和小房子 b 呢？这里有以下两种写法。

第一种：

```
scanf("%d",&a);
scanf("%d",&b);
```

第二种：

```
scanf("%d %d",&a,&b);
```

第二种的写法较为简便，两个“%d”之间用一个空格隔开，“&a”和“&b”之间用逗号隔开。

从键盘读入两个数，输出这两个数的和的完整代码如下：

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

    system("pause");
    return 0;
}
```

好了，总结一下，在 C 语言中 printf 是说出去的，也就是计算机需要告诉你的；而 scanf 是听进来的，也就是你需要告诉给计算机的。

接下来，我们要让“加法计算器”更加人性化——带有提示的读入和输出。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    printf("这是一个加法计算器，欢迎您使用\n");
    printf("--------------------------------\n");
    printf("请输入第一个数（输入完毕后请按回车）\n");
    scanf("%d",&a);
    printf("请输入第二个数（输入完毕后请按回车）\n");
    scanf("%d",&b);
```

```
    c=a+b;
    printf("它们的和是%d",c);

    system("pause");
    return 0;
}
```

一起来找茬

1．下面这段代码是从键盘读入两个整数，并输出它们的和。其中有 6 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a,b,c;
    scanf("%d",a,b)
    c=a+b
    printf("%d/n",c);
    system("pause");
    return 0;
}
```

更进一步，动手试一试

1．从键盘读入两个数，并输出这个两个数的和、差、积与商。

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

请留意除法运算。在 C 语言中，当除号“/”左右两边都是整数时，商也只有整数部分。例如，5/3 的商是 1，2/3 的商是 0。

这一节，你学到了什么

1．如何从键盘读入一个数到小房子中？

第 8 节　究竟有多少种小房子

在之前的几节，我们已经知道计算机如果想“记住”某个值，就必须在计算机的大脑“摩天大厦”中，申请一个小房子。例如：

```
int  a, b, c ;
```

即申请 3 个小房子分别叫作 a、b 和 c。这 3 个小房子只能用来存放整数（整型数据）。

再例如：

```
float  a, b, c ;
```

即申请 3 个小房子 a、b 和 c。这三个小房子只能用来存放小数（浮点型数据）。

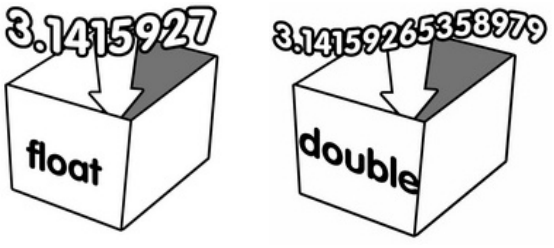
也就是说在计算机中，不同类型的数据需要相应类型的小房子来存储。

那么计算机一共有多少种类型的小房子呢？我们来列举几种最常用的，如表 2-1 所示。

表 2-1　C 语言常用的数据类型

数据类型名称	用来存放哪种数据	数 据 范 围
int	用来存放整数	–2147483648～2147483647
float	用来存放浮点数	$\pm1.18\times10^{-38}$～$\pm3.4\times10^{38}$
double	用来存放极大和极小的浮点数	$\pm2.23\times10^{-308}$～$\pm1.80\times10^{308}$
char	用来存放字符	256 种字符

double 也是用来存放小数的，那 float 和 double 有什么区别呢？



请观察下面两段代码。

代码 1：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float a;
    a=3.1415926535897932;
    printf("%.15f",a);

    system("pause");
    return 0;
}
```

代码 2：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    double a;
    a=3.1415926535897932;
    printf("%.15lf",a);

    system("pause");
    return 0;
}
```

通过观察，我们发现代码 1 和代码 2 的不同之处有两点。代码 1 中是用 float 来申请的小房子 a，在输出时相对应的占位符是“%f”，其中“%”和“f”之间的“.15”表示保留小数点后 15 位（四舍五入）。代码 2 中是用 double 来申请的小房子 a，在输出时相对应的占位符是“%lf”，注意此处不是数字 1 而是字母 l，同样“%”和“lf”之间的“.15”表示保留小数点后 15 位（四舍五入）。

它们的运行结果分别如图 2-16 和图 2-17 所示。

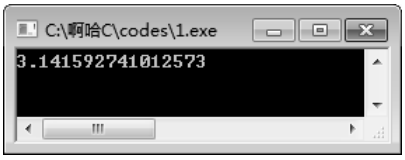


图 2-16　代码 1 运行的结果

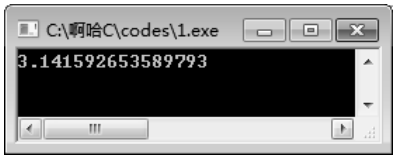
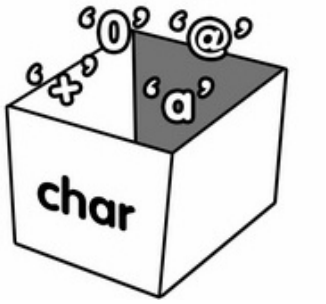


图 2-17　代码 2 运行的结果

怎么样，发现问题了吧，代码 1 运行后输出的是 3.141592741012573，显然从小数点后第 7 位开始就不对了，而代码 2 运行后输出的是 3.141592653589793，完全正确。因此我们可以发现 double 比 float 可以表示得更精确。另外 float 和 double 表示的数的大小范围也不同，请大家自己去尝试。

在表 2-1 中我们发现有一个新的数据类型 char，用 char 申请的小房子是用来存放字符的。




```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char a;
    scanf("%c",&a);
    printf("你刚才输入的字符是%c",a);

    system("pause");
    return 0;
}
```

我们输入一个字符 x 后按回车键，结果如图 2-18 所示，当然你也可以尝试一下别的字符。

图 2-18　输入一个字符并输出

想一想，对于上面这段代码，如果此时你输入的不是一个字母，而是一串字母，计算机会输出什么呢？很抱歉！计算机只会输出你输入的第一个字母。

有的同学可能要问，如果想存储一大串字符该怎么办呢？不要着急，我们将在后续章节中介绍如何存储一个字符串。

一起来找茬

1．下面这段代码是让计算机读入一个字符并把这个字符原样输出。其中有 3 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    char a;
    scanf("%c",c);
    printf("%d",c);
    system("pause");
    return 0;
}
```

更进一步，动手试一试

1．从键盘读入一个字符，输出这个字符后面的一个字符。例如，输入字符 a，输出字符 b。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char a;
    scanf("%c",&a);
    printf("后面的一个字符是%c",a+1);

    system("pause");
    return 0;
}
```

请思考一下，为什么这个字符后面的一个字符就是这个字符加 1 呢？

这一节，你学到了什么

1．double 是什么类型？

2．如何存储一个字符？

第 9 节　拨开云雾见月明

通过前面的学习，我们已经知道计算机如果想“记住”某个值，就必须在计算机的大脑“摩天大厦”中，申请一个小房子。例如，之前我们需要计算任意两个数的和，程序是这样写的：

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

    system("pause");
    return 0;
}
```

其实这个小房子 c 是多余的，可以直接写成：

```
printf("%d+%d=%d",a,b,a+b);
```

代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d+%d=%d",a,b,a+b);

    system("pause");
    return 0;
}
```

当然了，如果你只想计算 4+5 的值，可以更简单：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("%d",4+5);
    system("pause");
    return 0;
}
```

如果希望计算 4+(6−3)×7 的值，可以直接这样写：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("%d",4+(6-3)*7);
```

```
    system("pause");
    return 0;
}
```

第 10 节　逻辑挑战 1：交换小房子中的数

假如在计算机中我们已经有两个小房子（变量）分别叫作 a 和 b，并且它们都已经有了一个初始值，但是现在希望将变量 a 和变量 b 中的值交换，该怎么办呢？

先来看一段代码：

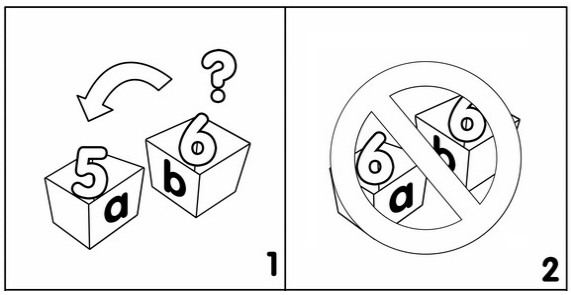
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d %d",a,b);

    system("pause");
    return 0;
}
```

上面这段代码是从键盘读入两个数，然后将这两个数输出。例如，如果你输入的是 5 和 6，那么输出的也是 5 和 6。可是，我们现在的需求是将变量 a 和 b 中的数交换后输出，也就是说如果读入的是 5 和 6，那么输出的应该是 6 和 5 才对。应该怎么办呢？来看一段代码：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    a=b;
    b=a;
    printf("%d %d",a,b);

    system("pause");
    return 0;
}
```



上面的代码企图通过 a=b;b=a;语句将变量 a 和变量 b 中的值交换，如果你已经运行过上面的代码，就会发现交换并没有成功，变量 b 的值没有变化，反而是变量 a 的值变成了变量 b 的值，这是为什么呢？

我们来模拟一下计算机运行的过程。

int a,b;指计算机会申请两个小房子（变量），分别叫作 a 和 b。

scanf("%d %d",&a,&b);指从键盘读入两个数，分别赋值给变量 a 和变量 b。假如我们从键盘读入的两个数分别是 5 和 6，那么变量 a 中的值就是 5，变量 b 中的值就是 6。

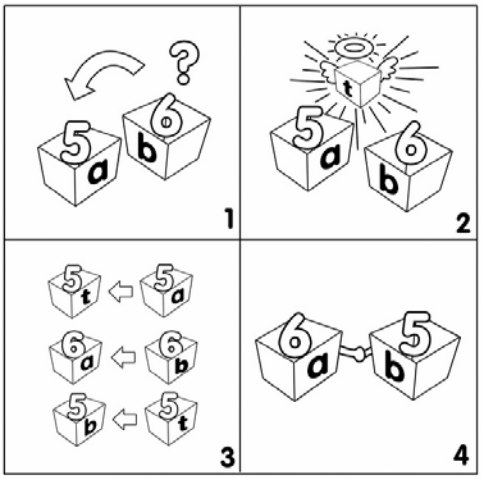
a=b;指计算机会将变量 b 中的值给变量 a，变量 a 中的值也变成了 6。变量 a 中原来的 5 被新来的 6 给覆盖了，也就是说原来变量 a 中的 5 丢失了。

b=a;指计算机会将此时变量 a 中的值给变量 b，此时变量 a 中的已经是 6 了，所以变量 b 的值其实还是 6。

最终，变量 a 和变量 b 中的值都为 6。那我们要怎么改呢？通过上面我们对计算执行过程的模拟，我们发现，主要问题是：计算机在执行完 a=b;这个语句后，原先变量 a 中的值被弄丢失了。那我们只要在执行 a=b;这个语句之前，先将变量 a 的值保存在另外一个临时变量中就可以了，例如，保存在变量 t 中，代码如下：

```
t=a;
a=b;
b=t;
```

我们先将变量 a 中的值给变量 t，变量 t 中值就变为 5（假如原来变量 a 中是 5，变量 b 中是 6），然后再将变量 b 中的值给变量 a，变量 a 中的值就变为 6，最后将变量 t 中的值给变量 b，此时变量 b 中的值就变为 5。成功！通过一个变量 t 作为中转站，我们已经成功地将变量 a 和变量 b 中的值进行了交换。



完整的代码入下：

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

    system("pause");
    return 0;
}
```

一起来找茬

1．下面这段代码是让计算机读入两个整数，分别放到变量 a 和变量 b 中，并将变量 a 和变量 b 中的数交换。其中有两个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a,b;
    scanf("%d %d",&a,&b);
    t=a;
    b=a;
```

```
        b=t;
        printf("%d %d",a,b);
        system("pause");
        return 0;
    }
```

✈ 更进一步，动手试一试

1. 在本节我们介绍了如何将两个变量的值进行交换，方法是增加一个临时变量来作为中转。你有没有想过，在不增加任何新的变量的情况下，也可以完成呢？来看看下面的代码吧。

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

        system("pause");
        return 0;
    }
```

请思考一下，为什么通过 a=b–a;b=b–a;a=b+a;也可以将变量 a 与变量 b 中的值交换呢？

第 11 节　天啊！这怎么能看懂

先来看一段代码：

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

怎么样，你看懂了吗？这段代码的意思其实就是从键盘读入两个整数并且输出它们的和。不错，上面的这段代码从语法角度来讲没有任何错误，编译器也可以对其编译运行，也就是说计算机可以准确无误地“认识”这段代码，但是我们会看得比较吃力。一段优秀的代码，不仅仅要让计算机“看懂”，也要让我们可以看懂。再来看看下面这段代码是不是更容易让人们理解呢。

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

```
int main()
{
    int a,b,c;
    scanf("%d %d", &a, &b);
    c=a+b;
    printf("%d",c);

    system("pause");
    return 0;
}
```

这里需要指出的是，这里的 int a,b,c;前面与上一行相比，多了 4 个空格。其实我在输入代码的时候，并不是输入 4 个空格，而是输入了一个 Tab[3]。使用 Tab 来调整你的代码格式，是一名优秀的程序员必须要养成的习惯。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1; //将变量 a 赋初始值
    printf("%d",a);
    system("pause");
    return 0;
}
```

在上面的代码中，“//”表示注释，它将告诉编译器从“//”开始一直到本行末尾的内容都是没有用的。注释的主要作用是给程序员看的，通常用来对一行代码进行解释说明或备注。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1; //将变量 a 赋初始值
    //printf("%d",a);
    system("pause");
    return 0;
}
```

[3] Tab 表示一个制表符，在编程中用 Tab 来代替空格是一个很好的习惯，可以让你的代码看起来更美。Tab 键在字母 Q 键的左边，赶快试一试吧。

上面的代码有两处注释，第 1 处注释我们已经讲过，主要是用来解释说明本行代码的作用。第 2 处的注释是将本来有用的代码 printf("%d",a);给注释掉，可以理解为“临时性删除”，就是告诉编译器 printf("%d",a);是没有用的。你可能要问那为什么不直接删除呢？因为有时我们并不希望真正删除，只是暂时不需要，以后说不定还要再用呢，这个如果删除了就找不回来了，如果我们合理地利用“//”进行注释，那么计算机就不会运行这句话，而是理解这句话是给程序员看的。如果我们以后又要使用这句话，只需将这句话前面的“//”去掉就可以了，这样是不是很方便呢。

有效地在代码中添加注释，可以让你的程序更具可读性。

“//”只能注释到本行末尾，如果要注释多行，就要在每行上写“//”。其实注释还有另外一种，以“/*”开始一直到“*/”结束，中间的内容编译器都不会理睬。使用“/* */”的好处就是它可以跨行。

例如，下面两段代码的效果是相同的：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    //a=2;
    //a=3;
    //a=4;
    //a=5;
    printf("%d",a);
    system("pause");
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    /*
    a=2;
    a=3;
    a=4;
```

```
    a=5;
    */
    printf("%d",a);
    system("pause");
    return 0;
}
```

上面两段代码中变量 a 的值最后还是 1。

再来看一段代码：

```
int a;
a=1;
```

上面这段代码是定义一个整型变量（小房子）a，并且给变量 a 赋一个初始值 1。我们以后会经常遇到在定义一个变量（小房子）之后，给其赋初始值的情况，我们可以简写如下：

```
int a=1;
```

多个变量也类似：

```
int a=1,b=2,c=3;
```

浮点型和字符型也类似：

```
float a=1.1;
char c='x';
```

需要注意的是，我们在给浮点型变量赋初始值的时候必须是一个小数，也就是说必须有小数点。在给字符型变量赋初始值的时候，字符两边需要加单引号，记住是单引号，不是双引号。在上面的代码中我们希望把字符 x 赋值给字符变量 c，所以我们在字符 x 的左右两边加上了单引号。

编程也是一门艺术，我们需要追求简洁、高效而且优美的代码，一名优秀的程序员往往也是一名艺术家。