

小学生坐在马桶上都可以读懂的“C语言编程”入门书

啊哈C语言



aha-c.com

后续内容还请关注

- [网站首页] <http://aha-c.com>
- [网站问答] <http://aha-c.com/qa>
- [新浪微博] <http://weibo.com/ahacpp>
- [人人主页] <http://page.renren.com/601196462>

如果您有任何建议

您，可以在 aha-c.com/qa 上留言，或者骚扰 [ahacpp @Gmail.com](mailto:ahacpp@gmail.com)

书写匆忙，欢迎批评纠错，感谢支持

目 录

第一章

第一节 让计算机开口说话	4
第二节 让颜色飞起来——让输出的内容带有颜色	18
第三节 计算机也会做加法	23
第四节 变量——用来存储数据的小房子	30
第五节 数据输出——我说咋地就咋地	38
第六节 从键盘输入数据——我说算啥，就算啥	42
第七节 究竟有多少种小房子呢	47
第八节 拨开云雾见月明——计算其实很简单	53
第九节 交换两个小房子中的数	56
第十节 让我们的代码变得更美	61

第一章

与计算机开始沟通吧

第一节

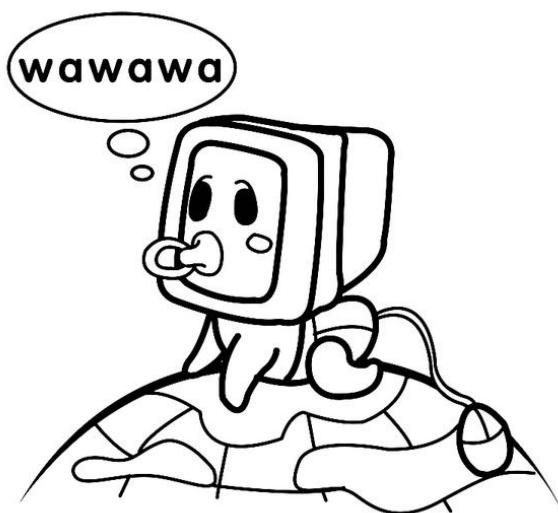
让计算机开口说话

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人类，改善人类的生活。计算机同样是用来帮助我们人类的工具。想一想，假如你现在希望让计算机来帮助你做一事情，首先你需要做什么？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、法语和德语等等。如果你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 语言以外，还有 C++、Java、C#语言等等。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，如果你能够熟练的掌握 C 语言，再学习其他语言就易如反掌啦。

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”所知道的东西告诉给我们人类是非常重要的。回想当年，我们

刚刚来到这个世界的时候，说到了第一句话是什么？应该不会是“你好！”，“吃了没？”……这样会把你的爸爸妈妈吓倒的，呵呵。

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”想说的告诉给人类，有两种方法，一种是显示在显示器屏幕上，一种是通过喇叭发出声音。就如同人类，一种是写在纸上，一种是用嘴巴说出来。我们目前让计算机用喇叭输出声音还比较麻烦，因此我们用另外一种方法，用屏幕输出“wa wa wa”。



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

这里有一个生疏单词叫做 `printf`，你不要被它吓到了，目前你不用搞清楚他的本质意义是什么，你只要记住它和中文里面“说”，英文里面的“say”是一个意思，就是让计算机说话的一个单词而已。在 `printf` 后面紧跟一对圆括号 `()`，是不是很像一个嘴巴，把要说的内容“放在”这个“嘴巴里”。这里还有一个需要注意的，在 `wa wa wa` 外边还有一对双引号 `"`，双引号里面的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，一句话的结束了要有一个结束的符号。汉语用句号“。”表示一句话的结束。英语用点号“.”表示一句话的结

束。在计算机语言中，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” 怎么办。

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

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

```
#include <stdio.h>

int main()
{
    printf("ni hao");
    return 0;
}
```

这里的

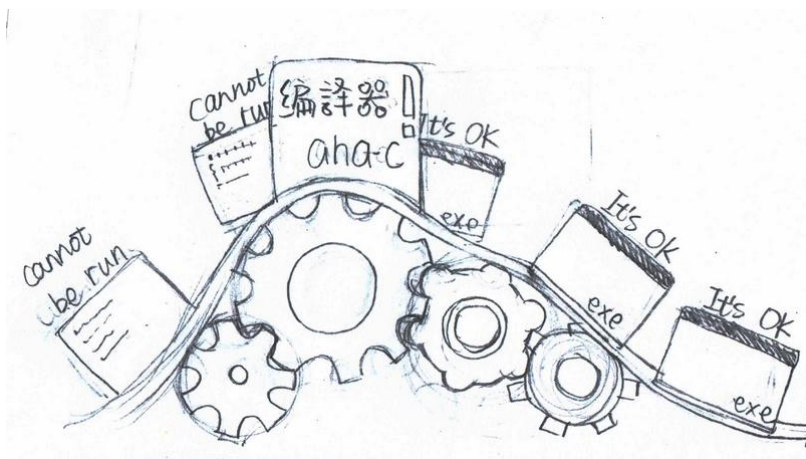
```
#include <stdio.h>

int main()
{
    return 0;
}
```

是所有 C 语言都必须要有框架，现在你暂时不要需要理解它为什么要有这个，反正要有这个就是了，以后再详细的讲这里的是做什么用的。但是有一点，我们今后写的所有类似 **printf** 这样的语句都要写在这一对 { } 之间才有效。

接下来我们需要让计算机运行一下我们刚才写的程序。

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，它叫做“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，叫做“啊哈 C”^②



首先你需要去 www.aha-c.com 上去下载“啊哈 C”。下面就要进入安装步骤啦，安装很简单，一共分 6 步，每一步我都截取了图片，你只需要一口气将 5 幅图片全部看完应该就 OK 啦。当然如果你觉得下面的步骤太多了，你可以直接观看配套光盘中的“啊哈 C 安装指导”视频。



图 1.1 - “啊哈 C” 安装 (此处只需双击图标)

^① “C 语言编译器”的作用是把把我们写的程序“变”成一个“exe”可以直接运行的程序。这个“变”的过程的专业术语叫做“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”直接运行你的程序了。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了。这里程序从某种意义上讲也可以叫做“软件”。

^② “啊哈 C”是一个 C 语言集成开发环境，使用的 gcc 的内核。

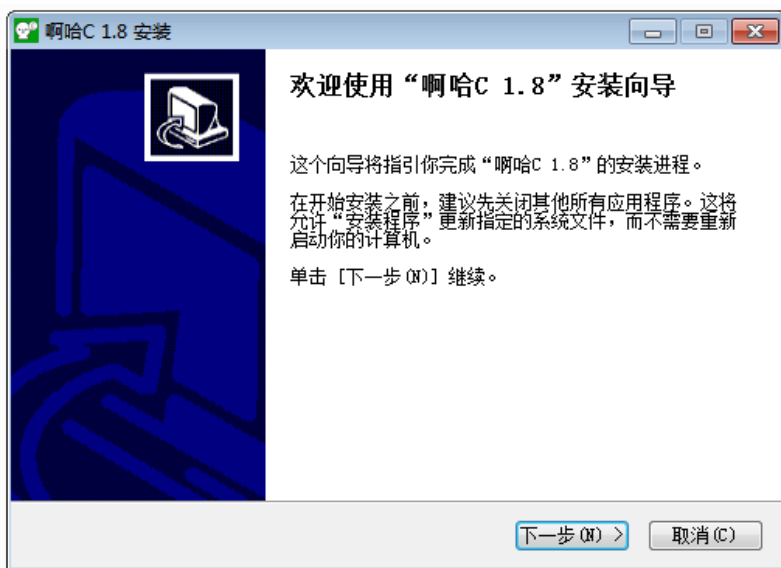


图 12 - “啊哈 C” 安装 (此处点击下一步)

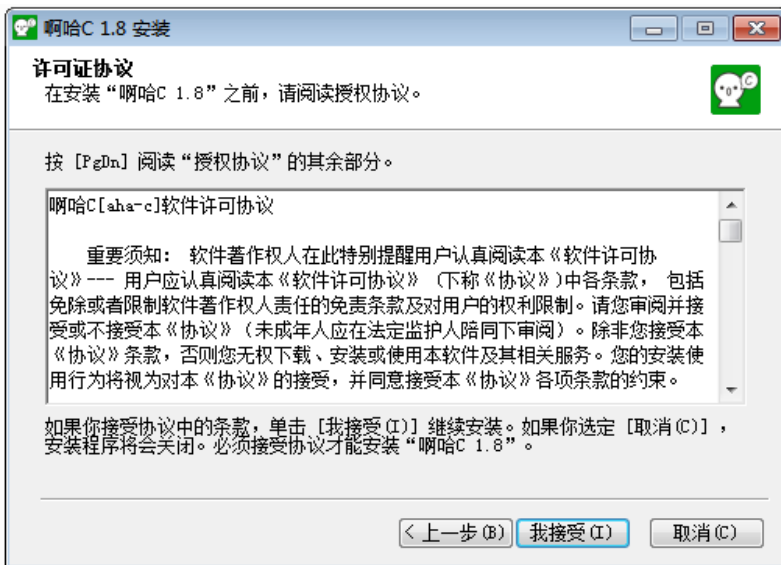


图 13 - “啊哈 C” 安装 (此处点击我接受)

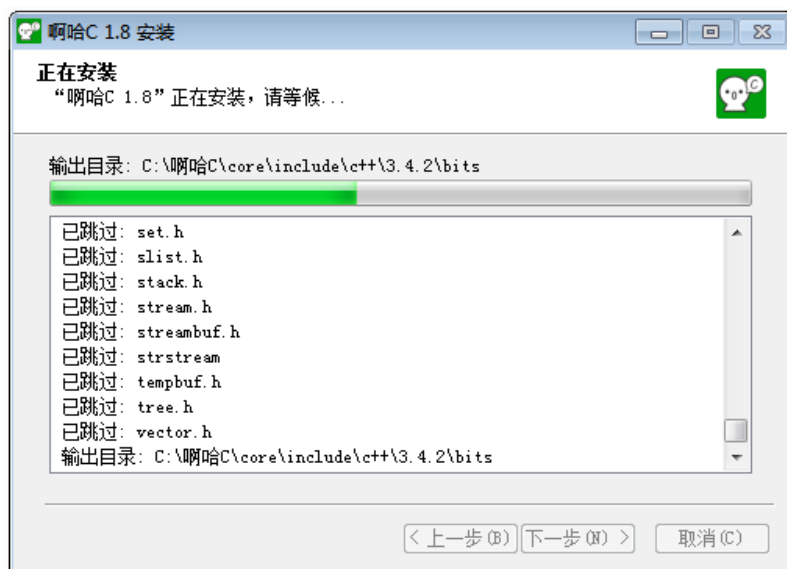


图 14 - “啊哈 C” 安装 (安装正在进行, 你只需要等待)



图 15 - “啊哈 C” 安装 (点击完成)

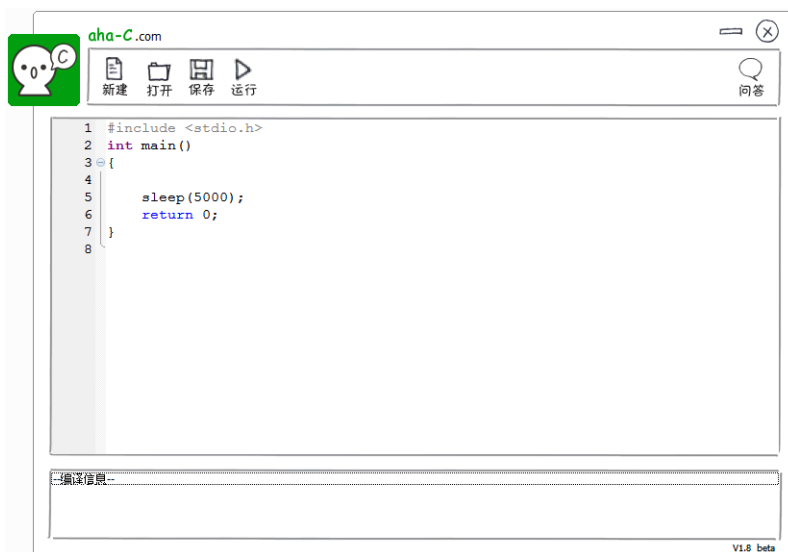


图 1.6 - “啊哈 C” 界面

“啊哈 C” 安装完毕后，我们便可以看到“啊哈 C”的界面如图 1.6，同时
在你的桌面上也会多一个“啊哈 C”的图标。

“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");
```

这条语句输入在“啊哈 C”中输入就好了，如下图：

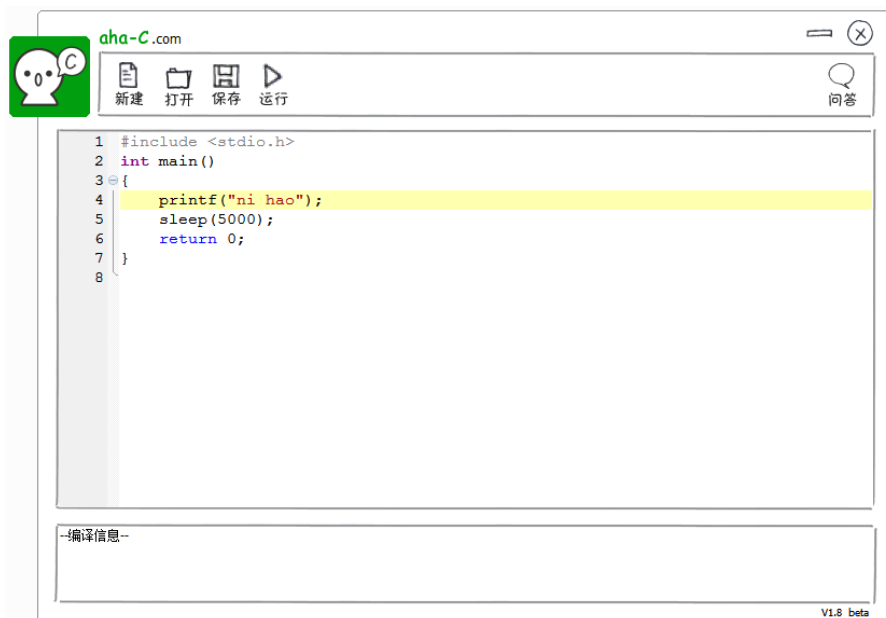


图 1.7 - 输出 “ni hao”

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

```
sleep(5000);
```

这句话是什么意思呢？稍后我们再揭晓，我们想将这句话删除，删除后，代码如下：

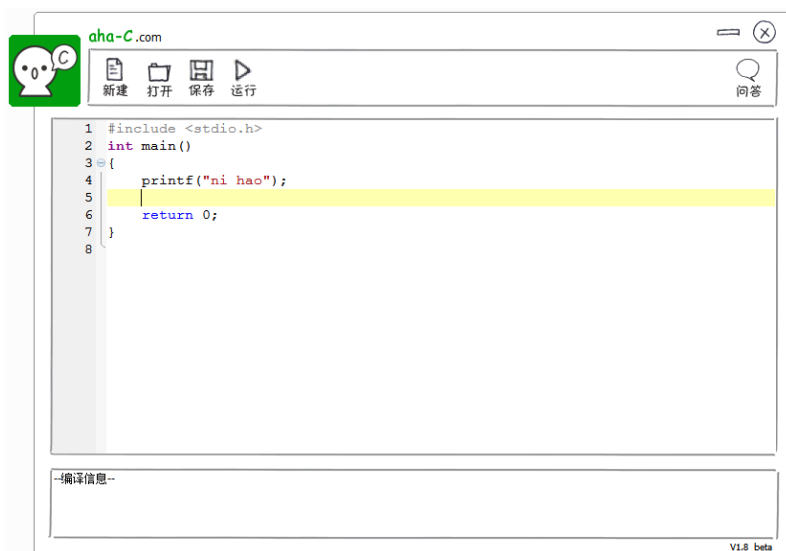



图 1.8 - 输出 “ni hao”

好了，童鞋们请注意，到了最后一步，我们需要让我们的代码运行起来。

现在你只需要点击一下“啊哈 C”上的“运行”按钮。计算机就会将你写的代码运行起来啦。

如果你的代码没有写错，那你“啊哈 C”将会弹出一个对话框，提示你“恭喜你编译成功”如图 1.9。请童鞋们注意在输入代码的时候，一定不要中文输入法，这里所有的符号都是英文的，一般也都是小写。

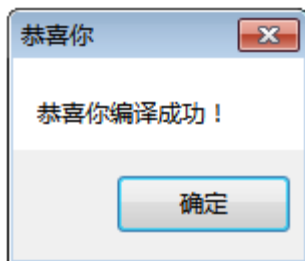


图 1.9 - “啊哈 C” 编译成功的提示

当然点击“确定”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”，请重新点击“运行”，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”？但是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，计算机在显示完“ni hao”之后，便又消失了。那应该怎么办呢？我们需要让计算机慢一点。

```
sleep(5000);
```

上面这句话是我们之前删除了的，其实他的作用就是让计算机“慢一点了”。好了，我们这句话放在 `printf("ni hao");` 的后面，完整的代码如下：

```
#include <stdio.h>

int main()
{
    printf("ni hao");
    sleep(5000);
    return 0;
}
```

代码：1-2.c

好了，再次点击“运行”吧。如果你的代码没有写错，你将看到图 1.10。

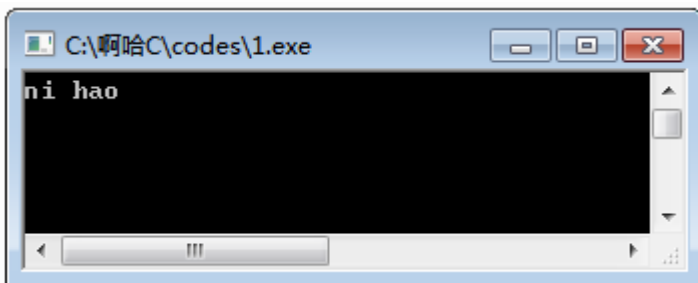


图 1.10 - 运行成功的结果

五秒钟之后，图 1.10 的窗口将会自动消失。如果你觉得 5 秒钟太短了，你

可以将代码 1-2.c 中的 `sleep(5000)` 改为 `sleep(10000)`，这样窗口 10 秒钟后才会消失。其实这里的 `sleep` 就是表示暂停的意思，圆括号内的数字就是表示需要暂停的时间，单位是毫秒，1000 毫秒等于 1 秒。

如果你想让 “ni hao” 分两行显示，你只需要将 `printf("ni hao");` 改为 `printf("ni\nhao");`；这里的 `\n` 表示的就是“换行”。注意这里的 `\` 是向右下角斜的，他在键盘上的位置，通常是在回车键的上面。代码如下，好赶快尝试一下吧。运行结果如图 1.11。

```
#include <stdio.h>

int main()
{
    printf("ni\nhao");
    sleep(5000);
    return 0;
}
```

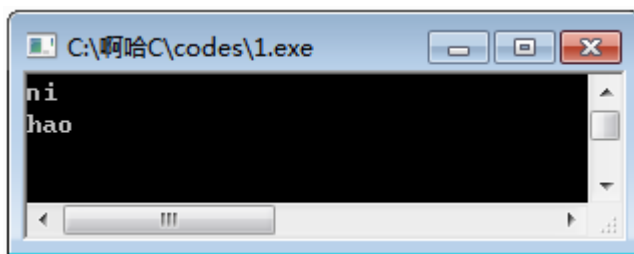
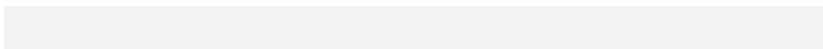


图 1.11 - 分行后运行成功的结果

➔ 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吧。

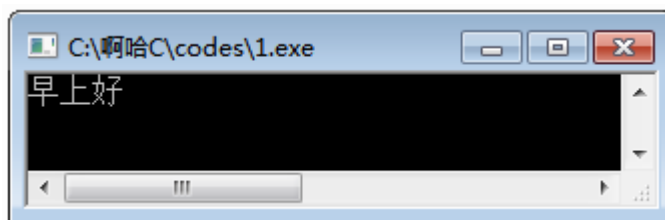


```
*  
**  
***
```

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

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

2. 那么如何让计算机说中文呢？，请让计算机说“早上好”，如下图，应该怎么办？



3. 再尝试一下让计算机显示下面这个图形吧。

```
A
BC
DEF
GHIJ
KLMNO
PRSTUV
W
X
Y
Z
```

✈ 这一节，你学到了什么

如何让计算机开口说话，让计算机开口说话的语句是：

第二节

让颜色飞起来——让输出的内容带有颜色

在上一节我们学习了让计算机开口说话是使用 `printf`。但是我们发现，计算机“说”出的话都是“黑底白字”的，其实计算机可以输出彩色的，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。如果你使用的本书推荐的编写 C 语言的软件“啊哈 C (aha-c)”的话，那么你一定可以编译运行的。OK，下面我们来看看，如何让颜色“飞”起来吧。

请尝试输入以下代码，并运行，看看会发生发什么？

```
#include <stdio.h>

int main()
```

```
{  
    system("color 2");  
    printf("wa wa wa");  
    sleep(5000);  
    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绿色的了。奥秘就在代码中。

```
system("color 2") ;
```

在这句话，2 代表绿色，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？来尝试下面这段代码

```
#include <stdio.h>  
int main()  
{  
    system("color 5f");  
    printf("wa wa wa");  
    sleep(5000);  
    return 0;  
}
```

运行结果如下：

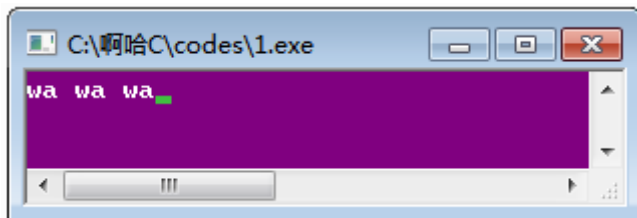


图 1.12 - 分行后运行成功的结果

上面这段代码在原来 2 前面加了一个 5，这里的 5 是代表的是背景色紫色。

那么设置背景色和文字颜色方法是，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 `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 16 进制的数，他只能是 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f 中某一个数。

[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。那么 9 之后的数字我们无法表示了，则使用“进位”来表示。例如数字“十”，阿拉伯数字只到 9，于是发现没有办法表示了，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”，则是使用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 来表示数字。“0”~“9”与“十进制”相同，但是“十”在“十六进制”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”中的最后一个字符，因此“十六”就表示不了。于是我们又采用了表示不了的时候就进一位的老办法，当前应该用“0”表示。“十六”在“十六进制”表示为“10”。同理“二十七”在“十六进制”中表示为“1B”。

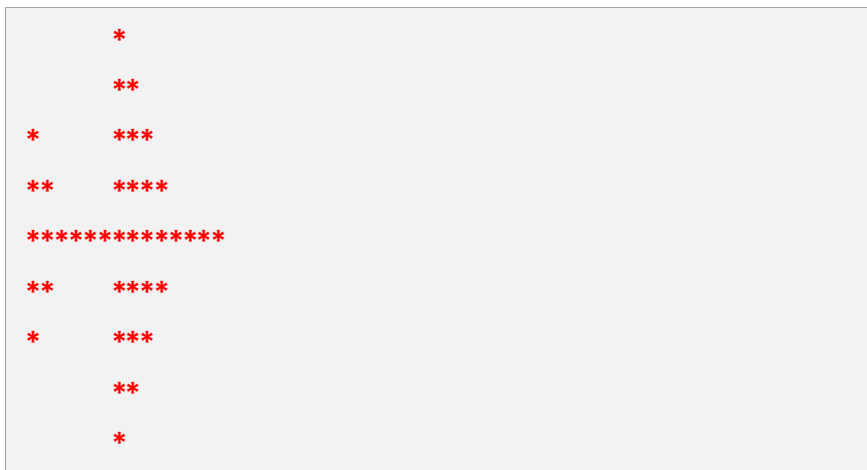
在中国古代，很多朝代都是用“十六进制”作为日常计数的，例如成语

“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 5，下面每颗珠子代表数 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。

其实我们现代日常生活中，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”。

✈ 更进一步，动手试一试

尝试一下下小飞机吧



尝试一下小队旗

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

➔ 这一节，你学到了什么

让计算机打印出来的字符有不同颜色的语句是：

第三节

计算机也会做加法

在上一节我们了解到让计算机说话是用“`printf`”这个单词，运用“`printf`”这个单词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，我们来看一个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。

回想一下我们人类小时候爸爸妈妈如何教我们算 $1+2$ 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呢？”我们在脑袋迅速的思考了一下，脱口而出“三个苹果”。没错！我们用大脑首先记住了左手有几个苹果，再用大脑记住了右手有几个苹果，此时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，大致分为以下几个步骤。

- 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) 最后将结果输出

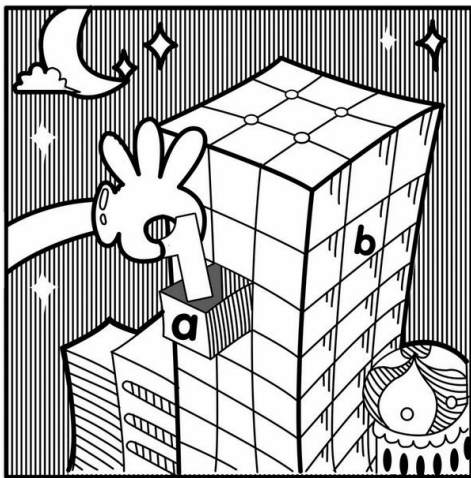
在这之中，我们大脑一共进行了：

- 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？同样也需要做以上四步。

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”只能放一个数，这样计算机就可以记住很多数了。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，把数字放到小盒子中。

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

这句话，就代表在计算机的“摩天大厦”中申请三个名字分别叫做 a, b 和 c 的三间小房子。（注意：int 和 a 之间有一个空格，a 与 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“=”号，这可不是“等于”号，他叫做“给予”号（也称作赋值号），他类似与一个箭头“ \leftarrow ”，意思是把“=”号右边的内容，给“=”号左边的。例如把 1 这个数给 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，“小房子 a”和“小房子 b”里面的数相加，将其结果再放到“小房子 c 中”。

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

这个式子计算机将会分两步执行。第一步现将 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，我们总结一下

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

很多童鞋是不是以为，现在就全部完成了？你忘记了一个最重要的一步，先别急着往下看，像想一想忘记了什么？

啊！你忘记了把答案输出。

你想想妈妈问你 $1+2$ 等于多少？你说：“我算出来了，但是我不想告诉你！”这个时候估计你少不了挨一顿了，o(>____<)o 不要啊！

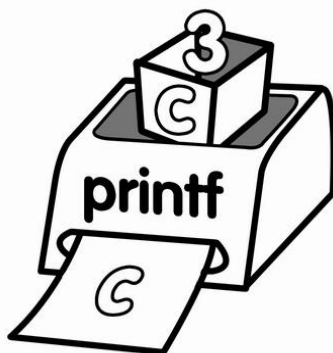
好那我们回忆一下，应该如何让计算机把结果输出呢。

对，使用 **printf()** 语句。那怎么把“小房子 c”里面存储的数输出呢？根据我们上一节学的知识，我们只要把要输出的内容，放在双引号里面就可以了，如下：

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

那你猜此时的计算机输出什么？

对，无情的输出一个 c。



那怎么样输出 c 里面存的值呢？

这时我们要另外一个人出场了。

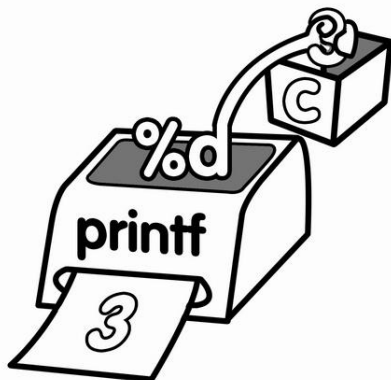
噚，噚，噚，噚

```
%d
```

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

```
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”，printf 就知道，此时需要输出一个具体的数值了，而不再是一个符号。printf 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 语言代码框架，计算机做加法的完整代码如下：

```
#include <stdio.h>
int main()
{
    int a,b,c;
    a=1;
    b=2;
    c=a+b;
    printf ("%d", c)
    sleep(5000);
    return 0;
```

```
}
```

现在赶紧去试一试吧。

➔ 更进一步，动手试一试

1. 那如果要进行三个数加法运算呢？例如 $5+3+1=?$

我们可以把上面的程序进行简单的改变，我们可以生申请 4 个小房子分别叫做 a, b, c 和 d。用 a, b, c 分别来存放三个加数，用 d 来存放他们的和。代码如下：

```
#include <stdio.h>

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

    sleep(5000);
    return 0;
}
```

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

2. 让计算机把下面三个算式算出来吧

```
123456789+43214321
```

```
70783*898712
```

```
4321*(123456+54321)
```

✈ 这一节，你学到了什么

如何申请一个小房子用来存储数字：

如何用 `printf` 输出小房子中数值：

第四节

变量——用来存储数据的小房子

上一节我们了解到计算机是使用一个一个小房子来记住数字。计算机有很多很多这样的小房子。



```
int a;
```

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

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

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

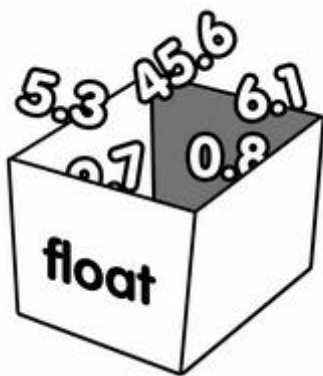
这里有一个小问题，就是给申请的“小房子”起名字，原则上来会说，你可以随便起，叫 `a` 可以，叫 `b` 也可以，叫 `x` 也可以，名字甚至是多个字母组成，例如可以叫做 `aaa`，也可以叫做 `abc`，也可以叫做 `book`。也可以是字母和数字的组合例如：叫做 `a1`，或者叫做 `abc123` 都是可以的。当然也有一些限制，如果你想知道请看看附录 3 吧。

到这里，可能还有很多童鞋想问，`int` 究竟是什么意思呢？

其实，`int` 是用来控制“小房子”是用来存放什么数的。`int` 表示你目前申请的小房子只能够存放整数。

`int` 是英文单词 `integer`（整数）的缩写。

如果要放小数该怎么办？



我们 `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;
```

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：小数在 C 语言中称作“浮点数”，在 C 语言中用 float 表示。

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

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

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];

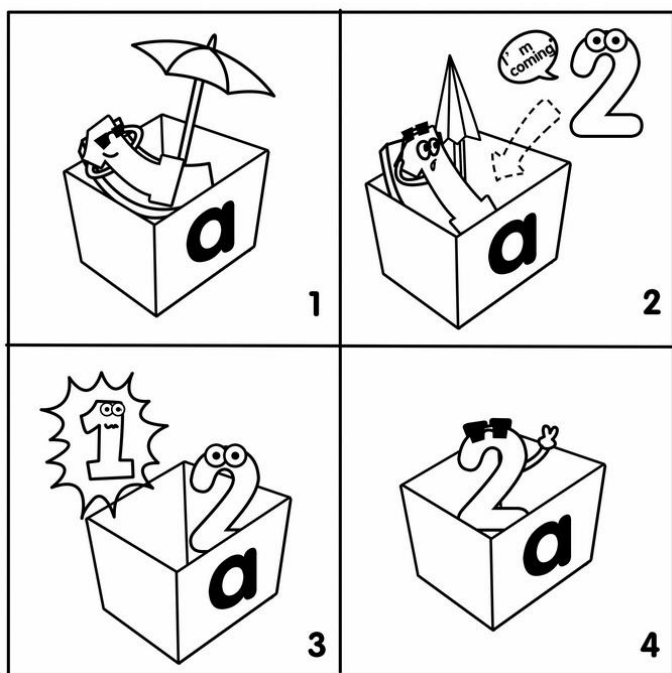
代码⇒ int a,b;

现在我们知道，int a 表示申请一个小房子 a 用来存放一个整数，即定义一个整型变量 a 来存放整数；而 float a 表示的则是申请一个小房子 b 用来存放一个整数，即定义一个浮点型（实型）变量 a 来存放浮点数（小数）。现在我有一个问题，请看如下代码：

```
#include <stdio.h>  
  
int main()  
{  
    int a;  
    a=1;
```

```
a=2;  
printf("%d",a);  
  
sleep(5000);  
return 0;  
}
```

请问计算机执行完上面的代码，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，紧接着我们又将 2 放入小房子 b 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 给覆盖掉了，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 中存放的始终是最后一次的值。例如：

```
#include <stdio.h>

int main()
{
    int a;

    a=1;
    a=2;
    a=3;
    a=4;
    a=5;
    a=6;

    printf("%d",a);

    sleep(5000);

    return 0;
}
```

计算机运行完上面这段代码最终将输出 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，请看继续看下面的代码：

```
#include <stdio.h>

int main()
{
    int a;

    a=7;

    a=a+1;

    printf("%d",a);
}
```

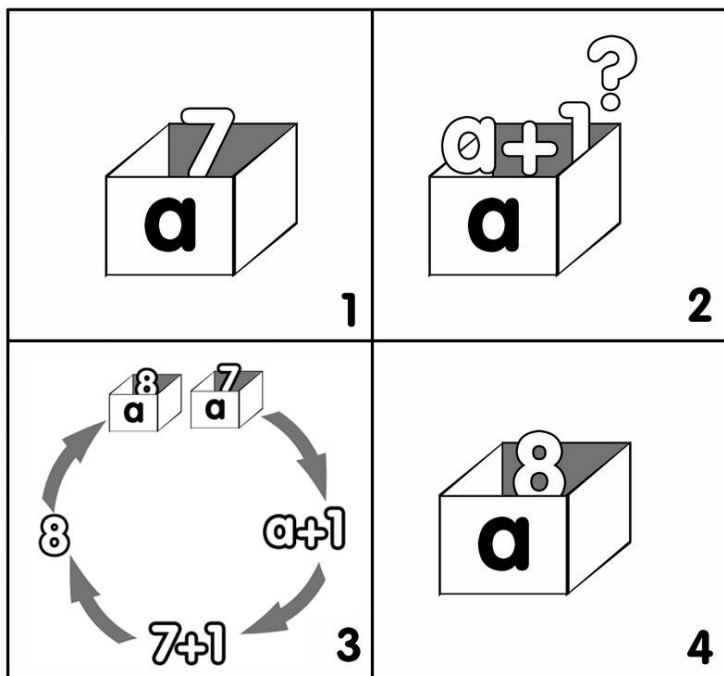
```

sleep(5000);
return 0;
}

```

计算机运行完上面这段代码最终将输出 8。也就是说小房子 a 中的值最终为 11。计算机在执行 `a=7` 这句话后，小房子 a 存储的值为 7，之后计算机又紧接着运行了 `a=a+1` 这句话。运行完 `a=a+1` 这句话后，小房子 a 中的值就变化为 8 了。也就是说 `a=a+1` 这句话的作用是把小房子 a 中的值在原本的基础上增加 1，我们来分析一下这句话。

`a=a+1` 这句话计算机分两步执行，这句话中两个操作符，第一个是“+”号，另一个是“=”（赋值号），因为+号的优先级别要比“=”要高，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。再计算完 `a+1` 的值后，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 了。



好啦猜猜下面的程序，计算机最终会输出多少？

```
#include <stdio.h>

int main()
{
    int a;
    a=10;
    a=a*a;
    printf("%d",a);

    sleep(5000);
    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 2。

➔ 更进一步，动手试一试

1. 如果要进行两个小数加法运算呢？例如 $5.2+3.1=?$ 代码如下：

```
#include <stdio.h>

int main()
{
    float a,b,c;
    a=5.2;
    b=3.1;
    c=a+b;
}
```

```
printf("%f",c);

sleep(5000);

return 0;

}
```

请注意，之前我们在 `printf` 语句中输出整型变量的值得时候，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. 让计算机把下面三个算式算出来吧

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

➔ 这一节，你学到了什么

如何定义一个用来存放小数的变量：

如何让一个小房子 `a`（变量 `a`）中的值增加 1：

第五节

数据输出——我说咋地就咋地

在第二节中我们已经学会如何让计算机做加减乘除运算，但是计算机在输出的时候，只显示一个结果，这样不够人性化。如果我们可以将整个算术等式输出就好了，形如：1+2=3。那我们应该怎么写呢？

新的代码：

```
#include <stdio.h>

int main()
{
    int a,b,c;
    a=1;
    b=2;
```

```

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

        sleep(5000);
        return 0;
    }

```

原来的代码

```

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

    sleep(5000);
    return 0;
}

```

仔细阅读代码你会发现的，新的代码和原来的代码只有最后一句 `printf` 不一样。好，那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` 执行，输出的形式必然是 `%d+%d=%d`。

当计算机遇到第一个 `%d`，知道“讨债的”来了，于是他便去双引号的后面

讨债，排在第一个的是 a，那么就像 a 讨。a 的值是 1，于是第一个%d 讨到的便是 1。

第二个是+，那么照样输出

第三个又是%d，同样到双引号的后面去讨债，因为排在第一个的 a 已经被讨过债了，因此向排在第二个的 b 讨。b 的值是 2，于是这个%d 讨到的便是 2。

第三个是=，依然照样输出。

第四个还是%d，同样到双引号的后面去讨债，因为排在第一个的 a 和排在第二个的 b 已经被讨过债了，因此向排在第三个的 c 讨。c 的值是 c，于是最后这个%d 讨到的便是 3。

最后输出的内容是 1+2=3

请注意通常，双引号内部%的个数，和后面变量个数是相等的，他们是“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，但是这不符合常理，请最好避免这样的情况出现。

➔ 更进一步，动手试一试

1. 指定两个数，输出这个两个数和、差、积与商。例如这两个数是 9 和 3，输出 9+3=11 9-3=8 9*3=27 9/3=3

```
#include <stdio.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);  
sleep(5000);  
return 0;  
}
```

➔ 这一节，你学到了什么

如何按照指定的要求输出：

第六节

从键盘输入数据——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候，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？注意啦，下面即将揭晓。

我们知道，让计算机说话是用 `printf`；那么让计算机学会听，则是用 `scanf`，计算机将会把听到的内容，告诉给你的程序。

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，计算机如何读入。

`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，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，只有两个地方不同：

第一个不同的是：读入是使用 `scanf` 这单词，而输出是使用 `printf`

第二个不同的是：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”。他的作用是得到“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（也就是变量）中，需要知道你指定的这个小房子的地址，才能把值成功的放进小房子中，但是在输出地时候，值已经在小房子中，就可以直接输出到屏幕。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，你走出这个教室的时候，因为此时你已经在教室中啦，因此这时候的你已经不再需要这个地址啦。

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。

那么从键盘读入两个数，输出这两个数的和，的完整代码如下。

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

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

    sleep(5000);
    return 0;
}
```

好了，总结一下：在 C 语言中 `printf` 是说出去，也就是计算机需要告诉你的；而 `scanf` 是听进来，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>
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",&a);
    c=a+b;
```

```
printf("他们的和是%d",c);

sleep(5000);
return 0;

}
```

➔ 更进一步，动手试一试

1. 从键盘读入两个数，输出这个两个数和、差、积与商。

```
#include <stdio.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);
    sleep(5000);
    return 0;
}
```

✈ 这一节，你学到了什么

如何从键盘读入一个数到小房子中：

第七节

究竟有多少种小房子呢

在之前的几节中，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 ;
```

就是申请三个小房子分别叫做 `a`、`b` 和 `c`。这三个小房子只能够用来存放整数（整型数据）。

再例如：

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


就是申请三个小房子 a, b 和 c。这三个小房子只能够用来存放小数（浮点型数据）。

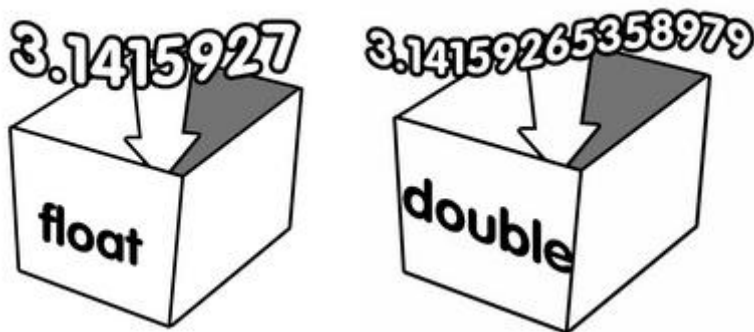
也就是说在计算机中，不同的类型数据需要相应类型的小房子来存储，那么计算机一共有多少种类型的小房子呢？我们来列举几个最常用的：

数据类型名称	用来存放哪种数据
int	用来存放整数
float	用来存放浮点数
double	用来存放极大和极小的浮点数
char	用来存放字符

表图 1.1 - C 语言数据类型

好了，目前为止

首先说明一下 float 和 double 的区别。



请观察下面两段代码

代码 1:

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

```
int main()
{
    float a;
    a=3.1415926535897932;
    printf("%.15f",a);

    sleep(5000);
    return 0;
}
```

代码 2:

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

    sleep(5000);
    return 0;
}
```

我们观察一下左边与右边的代码不同有两点。左边是用 `float` 来申请小房子 `a`，在输出时相对应的占位符是 `%f`，其中 “%” 和 “f” 之间的 “.15” 表示的保留小数点后 15（四舍五入）。右边是用 `double` 来申请小房子 `a`，在输出时相对应的是占位符 `%lf`，注意此处不是数字 “1” 而是字母 “l”，同样 “%” 和 “lf” 之间的 “.15” 表示的保留小数点后 15（四舍五入）。

他们的运行结果分别是如下：

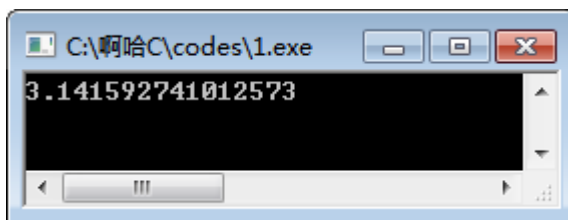


图 1.12 - 代码 1 运行的结果

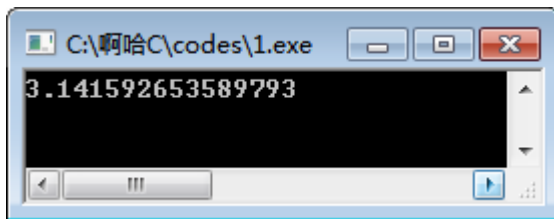


图 1.12 - 代码 2 运行的结果

怎么样，你发现问题了吧，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 可以表示的更精确。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 1-1 我们发现有一个新的数据类型“char”，用 char 申请出来的小房子是用来存放字符的，如下：

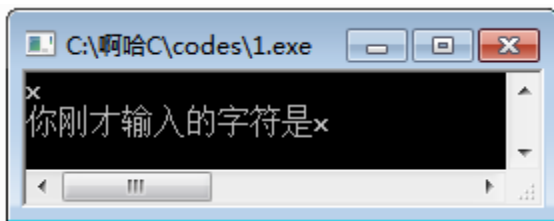


```
#include <stdio.h>
```

```
int main()
{
    char a;
    scanf("%c",&a);
    printf("你刚才输入的字符是%c",a);

    sleep(5000);
    return 0;
}
```

我们输入一个字符“X”后点击回车，结果如下图，当然你也可以尝试一下别的字符。



有的童鞋可能要问啦，如果想存储一大串字符该怎么办呢？不要着急，我们将在后续的章节中介绍如何存储一个字符串。

➔ 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是什么。例如输入字符 a，输出字符 b。

```
#include <stdio.h>

int main()
{
    char a;
```

```
scanf("%c",&a);  
printf("后面的一个字符是%c",a+1);  
  
sleep(5000);  
return 0;  
}
```

请思考一下，为什么一个字符后面的一个字符就是就是这个字符加 1 呢？

✈ 这一节，你学到了什么

double 是什么类型：

如何存储一个字符：

第八节

拨开云雾见月明——计算其实很简单

在之前的几节中，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。例如之前我们如果需要计算任意两个数的和，是这样的：

```
#include <stdio.h>

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

```
    sleep(5000);  
    return 0;  
}
```

其实 c 这个小房子（变量）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>  
int main()  
{  
    int a,b,c;  
    scanf("%d %d",&a,&b);  
    printf("%d+%d=%d",a,b,a+b);  
  
    sleep(5000);  
    return 0;  
}
```

当然了，如果你只想计算 4+5 的和，可以更简单

```
#include <stdio.h>  
int main()  
{  
    printf ("%d",4+5);  
    sleep(5000);  
}
```

```
    return 0;  
}
```

如果希望计算 $4+(6-3)*7$ 的值，可以直接这样写

```
#include <stdio.h>  
  
int main()  
{  
    printf("%d",4+(6-3)*7);  
    sleep(5000);  
    return 0;  
}
```


第九节

交换两个小房子中的数

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

先来看一段代码：

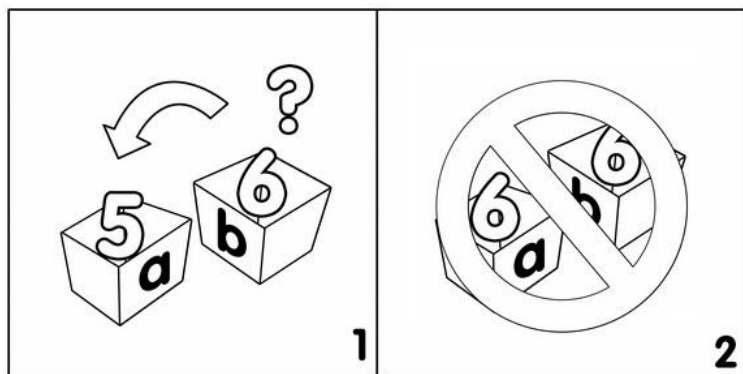
```
#include <stdio.h>

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

```
    sleep(5000);  
    return 0;  
}
```

上面这段代码是从键盘读入两个数，然后将这个两个数输出。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，那么输出的时候应该是 6 和 5。

```
#include <stdio.h>  
  
int main()  
{  
    int a,b;  
    scanf("%d %d",&a,&b);  
    a=b;  
    b=a;  
    printf("%d %d",a,b);  
  
    sleep(5000);  
    return 0;  
}
```



上面的代码企图通过 `a=b; b=a;` 这两行语句将变量 `a` 和变量 `b` 中的值交换，如果你已经运行过上面的代码，你会发现交换并没有成功，变量 `b` 的值没有变化，反而变量 `a` 的值也变成了变量 `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;` 计算机会申请两个小房子（变量），分别叫做 `a` 和 `b`。

`scanf("%d %d",&a,&b);` 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。

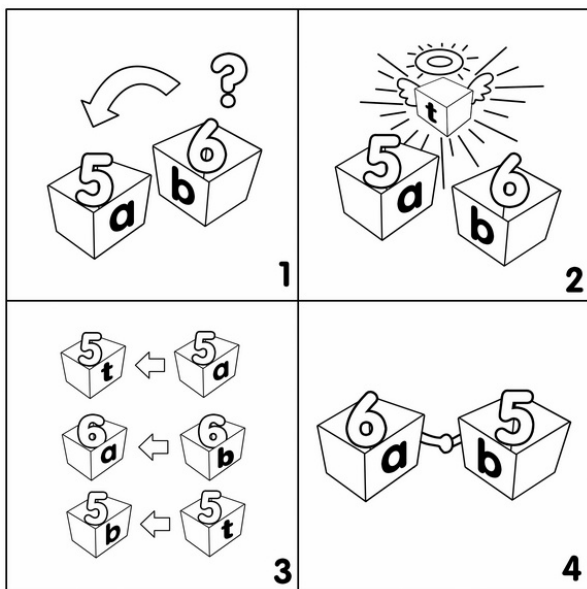
`a=b;` 计算机会将变量 `b` 中的值给变量 `a`，变量 `a` 中的值也变成了 6。变量 `a` 中原来的 5 被新来的 6 给覆盖了，也就是说原来变量 `a` 中的 5 丢失了。

`b=a;` 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，例如保存在变量 `t` 中，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，我们已经成功的将变量 *a* 和变量 *b* 中的值进行了交换。



完整代码入下：

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

```
    sleep(5000);  
    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交换，方法是增加一个临时变量来进行中转，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？来看看下面的代码把。

```
#include <stdio.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);  
  
    sleep(5000);  
    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？

第十节

让我们的代码变得更美

先来看一段代码：

```
#include <stdio.h>

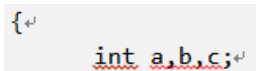
int main(){ int a,b,c; scanf("%d %d", &a, &b); c=a+b;
printf("%d",c); sleep(5000); return 0; }
```

怎么样你看懂了吗？这段代码其实就是从键盘读入两个整数并且输出他们的和。不错，上面的这段代码从语法角度来讲没有任何语法错误，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的“认识”这段代码，但是我们“正常”人类是不是看上去会比较吃力。一段优秀的代码，不仅仅要让计算机“看懂”，也要让我们“正常”的人类可以“看懂”。再来看看下面这段代码是不是更容易让人们理解。

```
#include <stdio.h>

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

    sleep(5000);
    return 0;
}
```

这里需要指出的是， 这里的 `int a,b,c;` 与上一行相比，往后面多空格了 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”键，养成使用“Tab”键来调整你的代码格式，是一名优秀的程序员所必须的，哈哈 :P

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

第二章

计算机可不是傻子

……敬请期待……

第一节 大于小与还是等于

第二节 如何判断正数呢

第三节 偶数怎么判断

第四节 用 `else` 来简化你的代码

第五节 计算机请告诉我谁大

第六节 三个数怎么办

第七节 我要排序——更复杂的判断来了

第八节 运算符总结

第九节 `1>2` 究竟对不对