

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



www.ahalei.com



目 录

最新版 2013.9.1

第 1 章 编程改变思维	1
第 1 节 为什么要学习编程	1
第 2 节 本书是讲什么的、写给谁看的	4
第 2 章 梦想起航	7
第 1 节 编程的魔力	7
第 2 节 让计算机开口说话	9
第 3 节 多彩一点	18
第 4 节 让计算机做加法	21
第 5 节 数字的家——变量	25
第 6 节 数据输出——我说咋地就咋地	30
第 7 节 数据输入——我说算啥就算啥	33
第 8 节 究竟有多少种小房子	36
第 9 节 拨开云雾见月明	39
第 10 节 逻辑挑战 1：交换小房子中的数	41
第 11 节 天啊！这怎么能看懂	44
第 3 章 较量才刚刚开始	48
第 1 节 大于、小于还是等于	48
第 2 节 判断正数	49
第 3 节 偶数判断	52
第 4 节 神器 else	53
第 5 节 请告诉我谁大	55
第 6 节 逻辑挑战 2：3 个数怎么办	58
第 7 节 逻辑挑战 3：我要排序	62
第 8 节 运算符总结	66
第 9 节 $1 > 2$ 究竟对不对	67

第 10 节	讨厌的嵌套	69
第 11 节	if-else 语法总结	72
第 4 章	重量级选手登场	74
第 1 节	永不停止的哭声	74
第 2 节	我说几遍就几遍	78
第 3 节	if 对 while 说：我对你很重要	82
第 4 节	求和！求和！！求和！！！	85
第 5 节	逻辑挑战 4：60 秒倒计时开始	89
第 6 节	这个有点晕——循环嵌套来了	93
第 7 节	逻辑挑战 5：奔跑的字母	97
第 8 节	究竟循环了多少次	101
第 9 节	逻辑挑战 6：奔跑的小人	103
第 10 节	for 隆重登场	107
第 5 章	好戏在后面	111
第 1 节	程序的 3 种结构	111
第 2 节	啰嗦一下	112
第 3 节	逻辑挑战 7：判读质数很简单	113
第 4 节	更快一点：break	117
第 5 节	continue	119
第 6 节	逻辑挑战 8：验证哥德巴赫猜想	120
第 7 节	逻辑挑战 9：水仙花数	123
第 8 节	逻辑挑战 10：解决奥数难题	127
第 9 节	逻辑挑战 11：猜数游戏	130
第 10 节	逻辑挑战 12：你好坏，关机啦	134
第 6 章	天啊！一大串数正在接近	136
第 1 节	逆序输出	136
第 2 节	申请 100 个小房子怎么办	137
第 3 节	100 个数的逆序	139
第 4 节	逻辑挑战 13：陶陶摘苹果	140
第 5 节	逻辑挑战 14：一个萝卜一个坑	143
第 6 节	逻辑挑战 15：选择排序	148

第 7 节	二维数组	151
第 8 节	剩下的一些东西	153
第 7 章	有了它你能做更多的事	157
第 1 节	字符的妙用	157
第 2 节	多余的回车键	159
第 3 节	字符的本质	162
第 4 节	人名怎么存储呢	163
第 5 节	逻辑挑战 16: 字母的排序	167
第 6 节	逻辑挑战 17: 字典序	169
第 7 节	多行字符	170
第 8 节	存储一个迷宫	173
第 8 章	游戏时间到了	176
第 1 节	走迷宫	176
第 2 节	推箱子	186
附录 A	标识符命名规则	188
附录 B	运算符的优先级和结合性	190

—

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。

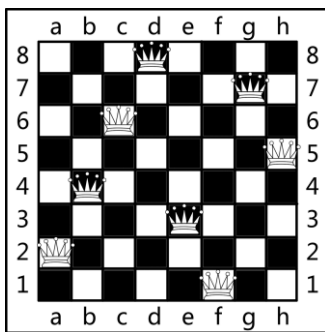


再来看一个经典的问题——八皇后问题。

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

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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。

	9			2				1
				6				2
						4		
6				8				
	2							
		1	7	4				
3	6							
		7				5		
9	5			7				8

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

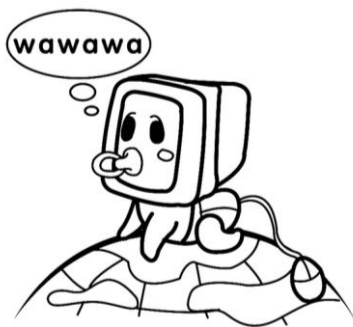
为什么会有计算机的出现呢？我们伟大的人类，发明的每一样东西都是为了帮助我们改善生活。计算机同样是用来帮助我们的工具。想一想，假如你现在希望让计算机帮助你做一件事情，你首先需要做什么？是不是要先与计算机进行沟通？那么沟通就需要依赖于一种语言。人与人的沟通，可以用肢体语言、汉语、英语、法语和德语等。你若要与计算机沟通，就需要使用计算机能够听懂的语言。我们学习的“C语言”便是计算机语言的一种，计算机语言除了C语言外，还有C++、Java、C#等。C语言是一门比较简单的计算机语言，更加适合初学者。所有的计算机语言都是相通的，如果你能够熟练掌握C语言，那么再学习其他语言就会变得易如反掌。

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十分重要的。

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

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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 `()`，是不是很像是一个嘴巴，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;
}
```

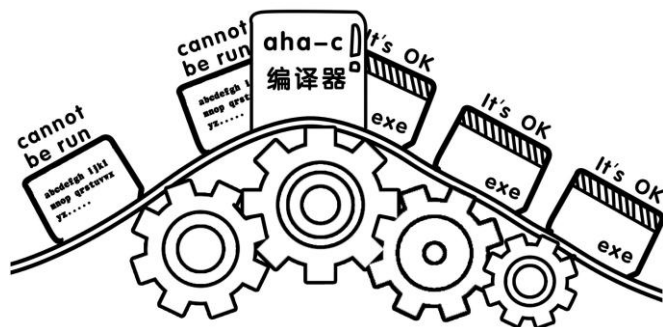
是所有 C 语言都必须要有框架，现在你暂时不需要理解它，知道要有这个即可，以后再详细地讲它的用途。但是有一点，我们今后写的所有类似 `printf` 的语句都要写在 `{ }` 里才有效。

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讲也可以称为“软件”。

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”

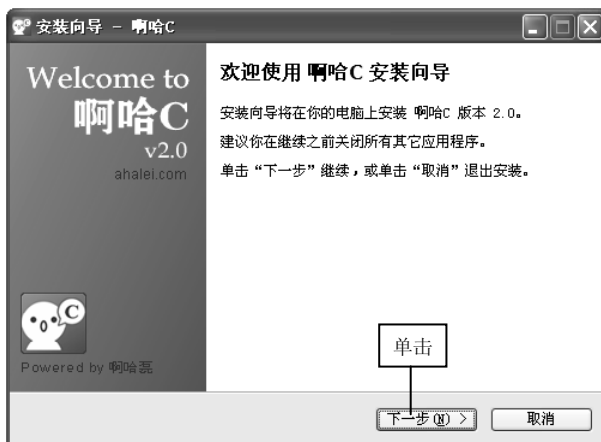


图 2-2 开始安装“啊哈 C”

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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

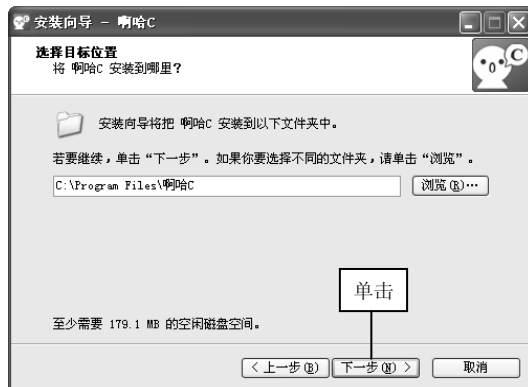


图 2-3 设置“啊哈 C”安装目录

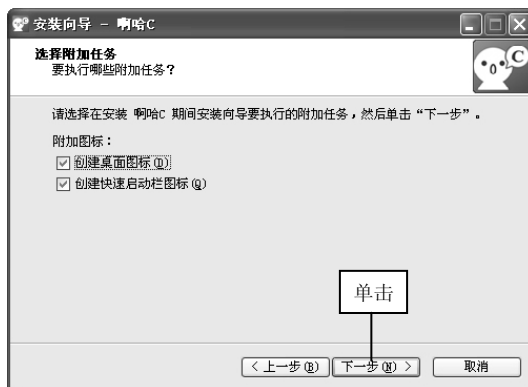


图 2-4 创建桌面图标和启动栏图标

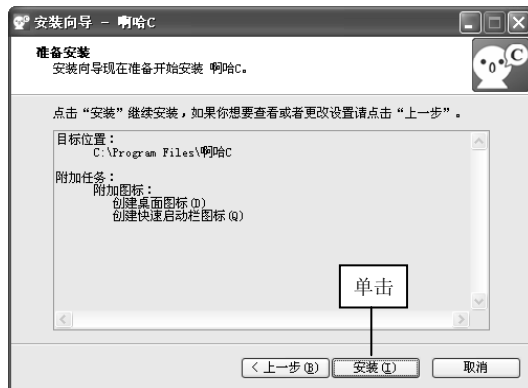


图 2-5 确认安装信息

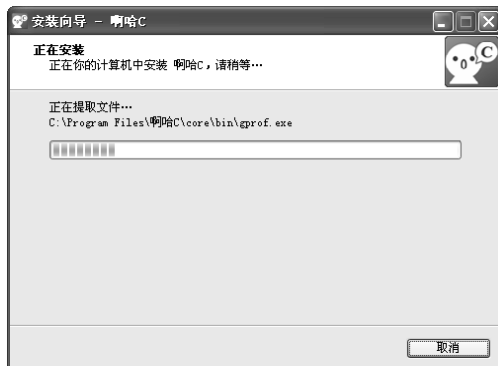


图 2-6 安装正在进行

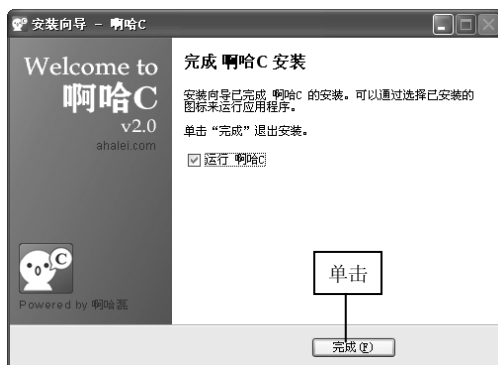


图 2-7 “啊哈 C” 安装成功

“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，同时也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！思考快你一步——用编程轻松提升逻辑力

“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。

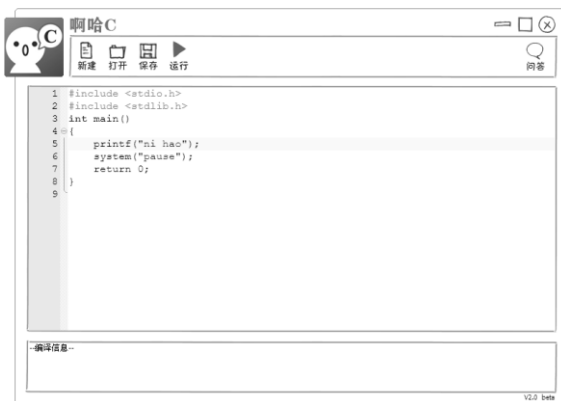


图 2-9 输入 printf("ni hao")

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。

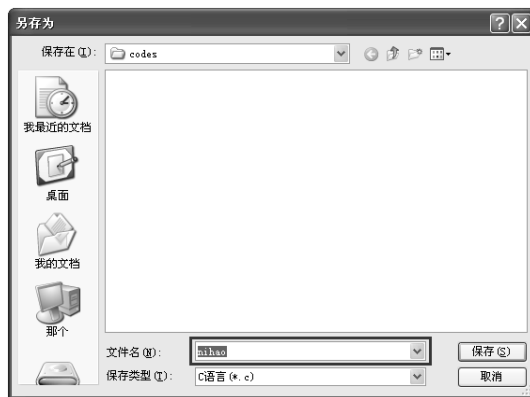


图 2-11 给程序起个名字

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。

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”？但我们是想要在屏幕上显示“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>
```



图 2-12 编译成功的提示

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
#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\nhao");`；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\nhao\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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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
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。

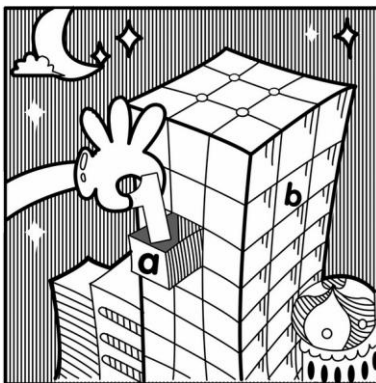
- (1) 两次输入：分别是记录左手和右手中苹果的数量；
- (2) 存储了 3 个值：分别是记录左手和右手中苹果的数量和相加的结果；
- (3) 进行了一次计算：相加；
- (4) 进行了一次输出：把相加的结果输出。

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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力



“=” 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！

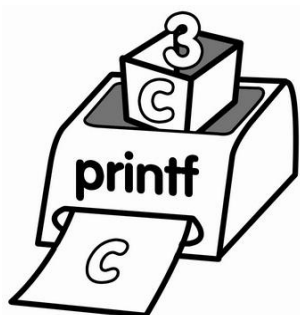
好，那我们回忆一下，应该如何让计算机把结果输出呢。

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？

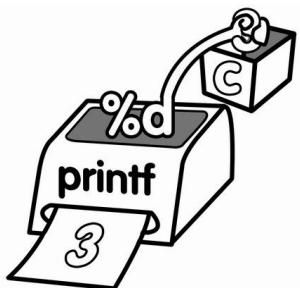
这时我们要让另外一个角色出场了。

```
%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。



啊哈 C！思考快你一步——用编程轻松提升逻辑力

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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

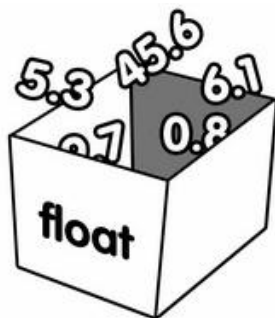


到这里，可能还有很多同学想问，int 究竟是什么意思呢？

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 ;`

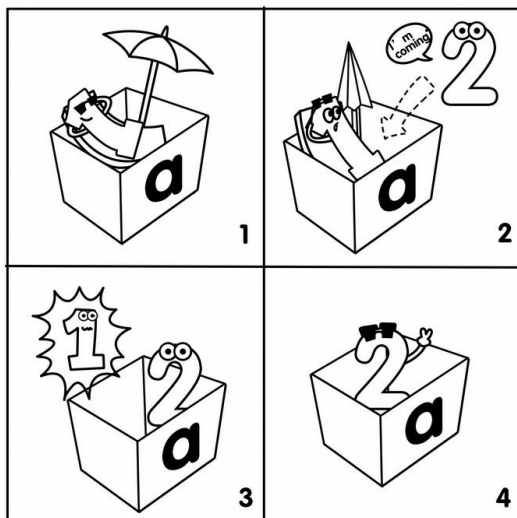
现在我们知道, `int a;`表示申请一个用来存放一个整数的小房子 `a`, 即定义一个整型变量 `a` 来存放整数; 而 `float a;`则表示申请一个用来存放一个小数的小房子 `a`, 即定义一个浮点型(实型)变量 `a` 来存放浮点数(小数)。

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,

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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：

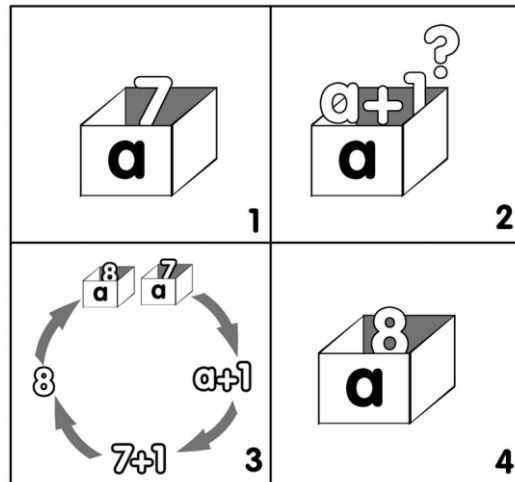
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=7;
    a=a+1;
    printf("%d",a);

    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( )
{
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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！

```
#include <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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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。

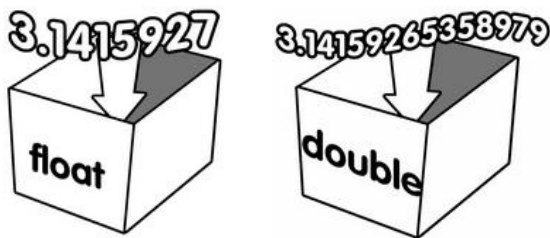


那么计算机一共有多少种类型的小房子呢？我们来列举几种最常用的，如表 2-1 所示。

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	$\pm 1.18 \times 10^{-38} \sim \pm 3.4 \times 10^{38}$
double	用来存放极大和极小的浮点数	$\pm 2.23 \times 10^{-308} \sim \pm 1.80 \times 10^{308}$
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("%.15f",a);

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

通过观察，我们发现代码 1 和代码 2 的不同之处只有一点。代码 1 中是用 `float` 来申请小房子 `a`，代码 2 中却是用 `double` 来申请小房子 `a`。在输出时，两段代码中 `printf` 里面所用的占位符都是 `“%f”`。代码中 `“%”` 和 `“f”` 之间的 `“.15”` 表示保留小数点后 15 位（四舍五入）。这里特别说明一下，在用 `scanf` 读入 `double` 类型数据时所用的占位符是 `“%lf”`（注意此处不是数字 1 是字母 l）而不是 `“%f”`。

它们的运行结果分别如图 2-16 和图 2-17 所示。

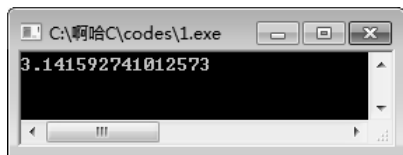


图 2-16 代码 1 运行的结果

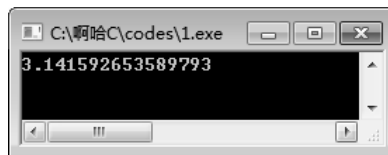


图 2-17 代码 2 运行的结果

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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>
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
#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 $4+(6-3)\times 7$ 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

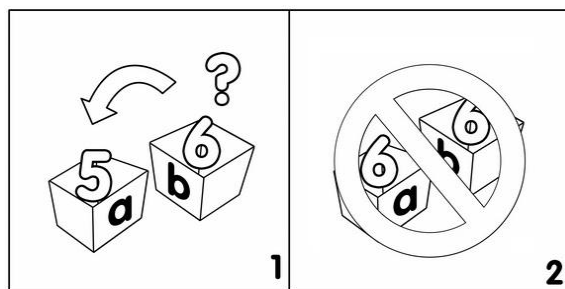
假如在计算机中我们已经有两个小房子（变量）分别叫作 a 和 b，并且它们都已经有了一个初始值，但是现在希望将变量 a 和变量 b 中的值交换，该怎么办呢？

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。

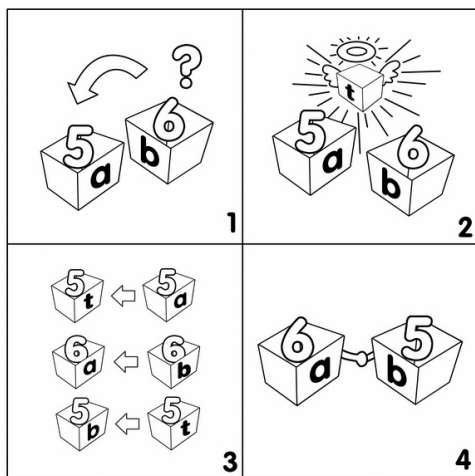
`a=b;`指计算机将变量 `b` 中的值给变量 `a`,变量 `a` 中的值也变成了 6。变量 `a` 中原来的 5 被新来的 6 给覆盖了,也就是说原来变量 `a` 中的 5 丢失了。

`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;
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
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;
}
```

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。

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。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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：

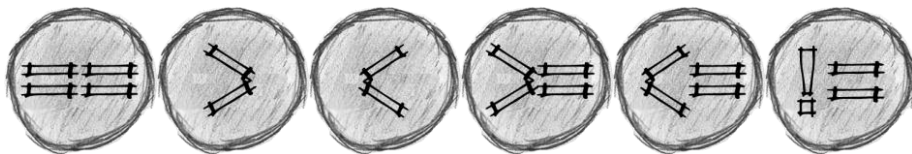
```
float a=1.1;  
char c='x';
```

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。

较量才刚刚开始

第 1 节 大于、小于还是等于



和我们一样，计算机也可以判断大小。假如你告诉计算机有 a 和 b 两个数，计算机除了可以告诉你这两个数的和、差、积和商，还可以告诉你谁大、谁小。现在我们就来瞧瞧计算机是如何判断谁大谁小的。

在此之前，我们需要先说明一下在计算机中用来判断两个数的关系的符号，即关系运算符，一共有如下 6 个：

== 等于
> 大于
< 小于
>= 大于等于
<= 小于等于
!= 不等于

需要特别注意的是，在计算机中，一个等于号“=”表示赋值，两个等于号“==”表示判断是否相等，同学们在编写代码的时候千万不要写错。一个感叹号加一个等于号“!=”表示“不等于”。此外计算机只有大于等于号“>=”和小于等于号“<=”，没有等于大于号和等于小于号，

即不存在“>=”和“=”，这一点请一定要注意。

例如，以下写法是正确的：

```
5>=4
7!=8
a<b
c==d
```

以下写法是错误的：

```
4=<7
8=>3
```

第2节 判断正数

假如你现在想让计算机判断一个整数是否为正数，如果是则显示 `yes`，不是则什么都不显示，应该怎么办呢？

下面方框中的内容，就是让计算机判断一个数是否为正数的“算法”。

首先，计算机需要有一个小房子（即变量）来存储这个数。
然后，你需要告诉计算机这个数是什么？
接下来，计算机需要判断这个数是否为正数。
最后输出计算机的判断结果。

算法：其实就是解决问题的方法（千万要被这个专业名词给吓住了）。

我们每遇到一个问题，首先需要思考的是解决这个问题的算法，也就是解决这个问题的方法和步骤。像上面一样一步一步地列出来，然后再将算法的每一步通过 C 语言来实现。

下面，我们就用 C 语言来实现上面的算法。

首先，计算机需要有一个小房子（即变量）来存储这个数。

我们可以用 `int a;`来申请一个名字叫作 `a` 的小房子（即变量），来存储需要判断的数。

然后，你需要告诉计算机这个数是什么？

我们可以用 `scanf("%d",&a);`来读入一个数并将这个数存储在小房子 `a` 中。

接下来，计算机需要判断这个数是否为正数。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

这可怎么办？不要紧，我们待会儿再来分析。

最后输出计算机的判断结果。

如果是正数则显示 `yes`，我们使用 `printf("yes");`。

好，我们现在集中精力来解决刚才的第 3 步——判断存放在小房子 `a` 中的数是否为正数。

想一想，我们是如何判断一个数是否为正数的？那就要从正数的定义出发，如果一个数大于 0，就是正数。好，那计算机也是这么想的，哈哈。

如果 `a` 大于 0，则显示 `yes`。

接下来，我们尝试用 C 语言来实现。

其中“如果”在 C 语言中用 `if` 来表示。代码如下：

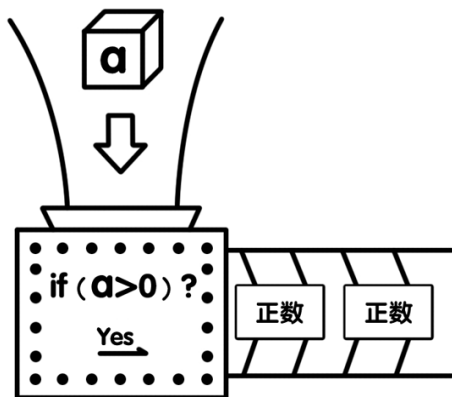
```
if (a>0) printf("yes");
```

当然，如果你觉得写在同一行很不爽，写成两行也是可以的：

```
if (a>0)
printf("yes");
```

更好的写法应该是在 `printf("yes");` 前面空 4 个空格或者空 1 个 Tab，表示 `printf("yes");` 是 `if (a>0)` 的一部分，当 `a>0` 条件成立时才执行 `printf("yes");` 这条语句。

```
if (a>0)
    printf("yes");
```



完整的代码如下：

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



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

好了，赶快试一试吧。

假如我希望输入正数时显示 yes，输入负数或 0 时显示 no，应该怎么办呢？

那么，第 3 部分则应改为：

如果 (a 大于 0)	显示 yes
如果 (a 小于等于 0)	显示 no

对应的 C 语言代码是：

```
if (a>0)    printf("yes");
if (a<=0)  printf("no");
```

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a;
    scanf("%d",&a) ;
    if (a>0)    printf("yes");
    if (a<=0)  printf("no");
    system("pause");
    return 0;
}
```

强调一下语法，if 后面的一对圆括号中，是一个关系表达式。

if 语句的语法格式为：

```
if (条件)    语句;
```

在条件为真时执行后面的语句。

好了，赶快试一试吧。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

🌀 一起来找茬

1. 下面这段代码用来判断一个数是否小于或等于 100，如果是则输出 yes。其中有 3 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a;
    scanf("%d",a) ;
    if (a<100) ; printf("yes");
    system("pause");
    return 0;
}
```

➔ 更进一步，动手试一试

1. 假如我希望输入正数时显示 yes，输入负数时显示 no，输入 0 时显示 0，应该怎么办呢？

➔ 这一节，你学到了什么

1. if 语句的基本格式是什么？

第 3 节 偶数判断

通过对本章第 2 节的学习，我们知道计算机是通过 if 语句来进行判断的。现在我们来尝试一下判断一个数是否为偶数。首先，我们先写出算法。

- (1) 计算机需要有一个小房子（即变量）来存储这个数。
- (2) 你需要告诉计算机这个数是什么？
- (3) 计算机需要判断这个数是否为偶数。
- (4) 计算机输出判断结果。

其中，在第 3 步你可能遇到一点小麻烦。我们想一下，什么是偶数呢？偶数就是能够被 2 整除的数，也就是说如果一个数除以 2 的余数为 0，那么这个数就是偶数。

那么我们现在只需要判断这个数除以 2 的余数是不是 0，即：

如果 a 除以 2 的余数与 0 相等，则输出 yes; 如果 a 除以 2 的余数与 0 不相等，则输出 no。
--

C 语言中求余数的运算符是“%”，所以判断一个数是否为偶数的 C 语言代码就是：

```
if (a % 2 == 0) printf("yes");
if (a % 2 != 0) printf("no");
```

完整的 C 语言代码如下：

请注意：在 C 语言中用两个等号“==”表示判断是否相等，一个等号“=”表示赋值。

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a;
    scanf("%d",&a) ;
    if (a%2==0) printf("yes");
    if (a%2!=0) printf("no");
    system("pause");
    return 0;
}
```

好了，应该尝试一下了。

🔗 一起来找茬

1. 下面这段代码用来判断一个数是否是 7 的倍数。其中有 4 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    scanf("%d %d",&a) ;
    if a%7=0 printf(yes);
    system("pause");
    return 0;
}
```

➔ 更进一步，动手试一试

1. 如何判断一个数的末尾是不是 0 呢？如果是则输出 yes（比如 120），不是则输出 no（比如 1 234）。

第 4 节 神器 else

在本章第 3 节我们使用了两个 if 语句来让计算机判断一个数是否为偶数，如果不出意外的话，我想你已经成功了。本节我们将学习另外一个语句来简化之前的代码，那就是神奇的 else。else 表示否则，只能和 if 配合使用。

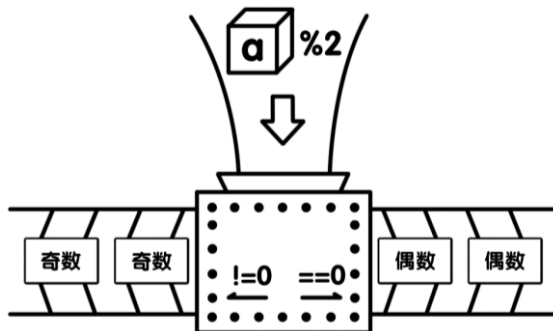
啊哈 C！思考快你一步——用编程轻松提升逻辑力

现在让我们回到如何让计算机来判断一个数是否为偶数这个问题，回顾一下本章第 3 节的算法：

如果 a 除以 2 的余数与 0 相等，则输出 yes；
如果 a 除以 2 的余数与 0 不相等，则输出 no。

其实上面第 2 个“如果”中的条件“a 除以 2 的余数和 0 不相等”正好就是第 1 个“如果”中的条件，即“a 除以 2 的余数 和 0 相等”的相反情况，因此我们用“否则”来代替，从而简化我们的代码。

如果 a 除以 2 的余数与 0 相等，则输出 yes；
否则，输出 no。



这里的“如果”在 C 语言中仍然用 if 来表示，这里的“否则”就用 else 来表示。好，转换为如下代码：

```
if (a % 2==0) printf("yes");  
else printf("no");
```

其实，更漂亮的写法是下面这样的：

```
if (a % 2==0)  
    printf("yes");  
else  
    printf("no");
```

从键盘读入一个整数判断它是否为偶数的完整代码如下：

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

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

if(a%2==0) printf("yes");
else printf("no");

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

if-else 语句的语法格式为:

```
if (条件) 语句;
else 语句;
```

当条件为真的时候执行 if 后面的语句; 当条件为假的时候执行 else 后面的语句。

🌀 一起来找茬

1. 下面这段代码用来判断一个数的末尾是否为 7, 例如: 7、17、127……如果是则打印 yes, 不是则打印 no。其中有 5 个错误, 快来改正吧。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    scanf("%d",&a) ;
    if (a%10=7) printf("yes")
    else ; printf("no")
    system("pause");
    return 0;
}
```

➔ 更进一步, 动手试一试

1. 从键盘输入一个正整数, 让计算机判断这个数是否为一位数 (1~9)。如果是则输出 yes, 不是则输出 no。

➔ 这一节, 你学到了什么

1. if-else 语句的基本格式是什么?

第 5 节 请告诉我谁大

在本章第 4 节我们学习了使用 if-else 语句来判断一个整数是否为偶数。本节我们将学习如

啊哈 C！思考快你一步——用编程轻松提升逻辑力

何从键盘输入两个整数，让计算机来判断哪一个整数较大，把较大的那个整数输出来。例如，如果我们输入的是 5 和 8，那么计算机输出 8。

在学习“如何让计算机判断两个数中，谁更大”这个问题之前，我们先回顾一下第 2 章中如何从键盘读入两个整数并且算出它们的和的问题。

```
#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);
}
```

在上面这段代码中，我们输出的是“和”。那如何让计算机输出较大的一个数呢？我们仍然使用“如果”的方法。

首先还是定义 3 个变量：a 和 b 用来存放输入的两个数，c 用来存放 a 和 b 中较大的那个。

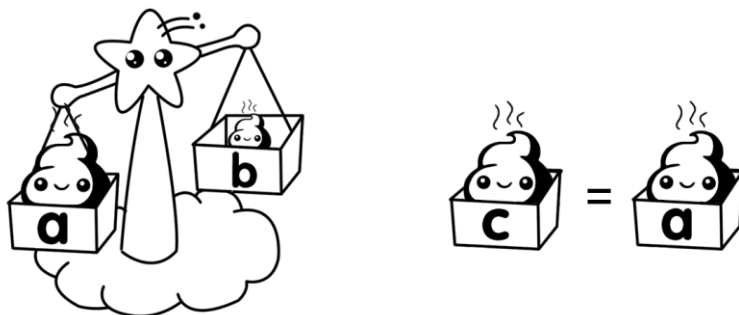
```
int a,b,c;
```

然后读入从键盘输入的两个数，分别存放到变量 a 和 b 中。

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

接下来要注意了，我们将通过之前学过的“如果”和“否则”的方法，来分情况讨论并判断谁更大。

```
如果 (a>b)    c=a;
```



上面这两行代码是说明在 $a > b$ 条件成立的情况下，我们将 a 的值给 c。但是 $a > b$ 条件并不一定成立，所以我们还要告诉计算机在条件不成立的情况下，应该怎么办。

所以还要写：

```
否则 c=b;
```

那么完整的代码如下：

```
如果 (a>b)    c=a;
否则         c=b;
```

总结一下，如果 $a > b$ 成立，就将 a 的值给 c ，执行 $c=a$ 。如果不成立，就执行“否则”部分，将 b 的值给 c ，执行 $c=b$ 。

计算机通过“如果”和“否则”方法来分情况讨论。当 $a > b$ 成立时，给出一种解决方案即执行某一个语句，这里是 $c=a$ ；当假设不成立的时候，给出另外一种解决方案即执行另外一个语句，这里是 $c=b$ 。

完整代码如下，赶快尝试一下吧。

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

一起来找茬

1. 下面程序的功能是从键盘读入两个整数，判断它们是否相等，如果相等则输出 `yes`，不相等则输出 `no`。其中有 6 个错误，快来改正吧。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    scanf("%d",&a,&b) ;
    if (a=b) ;
        printf("yes") ;
    else ;
        prinf("no") ;
    system("pause");
    return 0;
}
```

➔ 更进一步，动手试一试

1. 从键盘输入两个正整数，让计算机判断第 2 个数是不是第 1 个数的约数。如果是则输出 yes，不是则输出 no。

第 6 节 逻辑挑战 2：3 个数怎么办

在本章第 5 节我们学习了如何从两个数中找出较大的一个数，那么 3 个数该怎么办呢？

在解决这个问题之前，先回忆一下，我们是如何在 3 个数中找出最大的一个数呢，例如，1322、4534、1201 这 3 个数中哪个数最大？

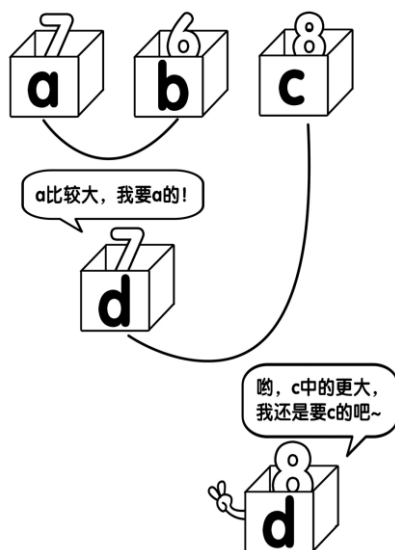
好了，赶快回忆一下，如果我给你任意 3 个数，你是怎么知道哪个数最大的呢？

怎么样，想出来了没有？千万不要告诉我，你是“一眼”就看出来的，如果是这样的话，请在下列数中找出最大的那个来，并且请“一眼”就看出来并告诉我。

123 971 141 723 813 743 60 402 592 742 737 834 656 814 562 951 20 352 8 1117 315 123 746
1064 1030 26 619 588 1061 361 1134 232 416 373 353 902 46 1223 790 66 406 141 911 631 512
908 528 802 601 392 474 474 813 1097 833 694 386 977 553 227 476 1 121 710 420 566 291 1094
13 1012 149 1010 356 362 132 558 373 921 128 681 165 252 60 4 934 41 603 70 280 20 357 1205
843 397 673 756 1107 809 630 615 1088 1152 608 448 949 268 669 1033 449 314 1088 604 134 17
269 1119 696 974 307 331 553 752 870 563 309 1179 853 816 155 361 546 252 197 820 330 975
679 1052 829 1205 1074 121 543 481 749 720 1106 157 1058 436 407 39 1232 181 198 1061 1114
532 303 264 633 530 741 475 1223 505 1127 275 4 339 305 594 907 615 377 800 234 108 263
1040 1174 795 497 256 60 248 441 213 1222 135 816 152 39 703 419 760 392 749 506 182 669
821 1131 874 235 1176 637 160 1115 578 924 832 452 1186 933 16 446 694 417 17 773 87 141
326 990 1084 988 266 981 1202 1122 770 1034 935 9 119 286 291 348 203 1221 275 258 1145 747
406 915 303 503 572 330 927 983 1231 230 393 804 911 446 722 934 621 507 777 742 1169 918
1160 480 756 817 1100 812 358 106 943 187 1223 143 73 308 437 260 612 575 645 644 669 681
1065 778 528 357 20 1210 615 991 92 585 757 629 636 410 166 798 470 264 526 860 679 313 648
806 706 572 546 34 1218 341 818 1190 781 691 101 985 336 922 728 7 790 1097 295 1205 390
1145 643 550 801 683 580 465 496 432 169 541 6 651 685 566 145 741 225 361 124 412 641 314 5
1115 27 873 597 1041 1222 280 137 789 102 293 694 944 199 789 962 913 276 246 299 351 1005
310 1124 711 335 132 182 185 496 438 634 462 696 910 801 576 916 525 1130 268 397 755 820
633 696 451 107 702 332 817 212 398 113 1185 862 991 542 786 461 889 564 913 4 656 41 861 15

怎么样，你“一眼”就看出来了吗？最大的数是多少？如果你可以在1秒内看出来，那你一定不是地球人，最大的数是1232。

现在回归正题，我们从一个数列中寻找最大的一个数的时候，大致是从左到右、从上到下快速地扫描（当然，古代的中国人可能是从上到下、从右到左），在快速扫描的过程中，我们首先会记住第1个数，然后继续往下看，一直看到一个数比之前记住的最大的数还要大时，就转为记住这个更大的数，然后一直快速扫描完毕，找出最大的一个。下面我们来模拟这个过程。



同理（我上学的时候最怕看到这个词，没有办法，这里我也借用一下，因为我一时半会儿也想不到更好的词语了），我们来找出3个数中最大的数也是相同的原理。

计算机要想找出3个数中最大的数，其实就是模仿我们的思维过程。

首先，用3个变量 a、b、c 分别存放从键盘读入的3个数。

然后，先比较变量 a 和 b 的值，将较大的值赋给变量 d。

再比较变量 d 和 c 的值，如果变量 c 的值大于变量 d 的值，则把变量 c 的值赋给变量 d。
最后输出变量 d 的值。

完整的代码如下：

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

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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

if(a>b) d=a;
else    d=b;

if(d<c) d=c;

printf("%d",d);

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

当然还有另外一种方法，就是分别比较变量 a 和 b，以及变量 a 和 c 的关系……思路如下：

如果 $a \geq b$ 并且 $a \geq c$ ，则输出 a；
如果 $b > a$ 并且 $b \geq c$ ，则输出 b；
如果 $c > a$ 并且 $c > b$ ，则输出 c。

其中“并且”在 C 语言中用“&&”来表示，顺便说一下在 C 语言中“或”用“||”表示。“||”这个符号可能在键盘上不太好找，它通常在“Enter”键的上面。在英文输入法状态下，按住“Shift”键不要松手，再按下“Enter”键上方的那个键，就会出现 1 个“|”，重复两次就可以啦。

“&&”表示逻辑“并且”
“||”表示逻辑“或”

想一想为什么不能像下面这样写？这样写会有什么问题？自己去探索吧！⁴

如果 $a \geq b$ 并且 $a \geq c$ ，输出 a；
如果 $b \geq a$ 并且 $b \geq c$ ，输出 b；
如果 $c \geq a$ 并且 $c \geq b$ ，输出 c。

完整的代码如下：

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

⁴ 在写本书草稿的时候，我也没有注意到这个问题，感谢@滚雪球 snow 的提醒。

```

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

    system("pause");
    return 0;
}

```

使用这种方法虽然代码比较简洁，但是在 10 个数中找出最大的数就很麻烦了。而从介绍的第 1 种方法则可以很容易地扩展出在 10 个数中找出最大的数的方法。

🌀 一起来找茬

1. 下面这个程序的功能是从键盘读入一个整数，判断这个数是否为 7 的倍数或者为末尾含 7 的数，例如：7、14、17、21、27、28……如果是则输出 yes，不是则输出 no。其中有 5 个错误，快来改正吧。

```

#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    scanf("%d",&a) ;
    if (a%7=0 | a%10=7) ;
        printf("yes") ;
    else
        printf("no")
    system("pause");
    return 0;
}

```

➔ 更进一步，动手试一试

1. 从键盘任意读入 3 个整数，如何从中找出最小的一个？
2. 从键盘任意读入 4 个整数，如何从中找出最大的一个？
3. 从键盘输入一个年份（整数），判断这个年份是否为闰年，是则输出 yes，不是则输出 no。

第 7 节 逻辑挑战 3：我要排序

在本章第 6 节我们学习了如何从 3 个数中找出最大的一个，现在有一个新的问题：如何从键盘输入任意 3 个数，并将这 3 个数从大到小排序呢？例如，无论你输入 2 1 3、3 2 1、1 2 3 还是 3 1 2，计算机都能够输出 3 2 1，这该怎么办呢？此时你先不要急着往下看，思考一下，通过我们之前学习的内容，你应该可以独立完成，赶快打开“啊哈 C”去尝试一下吧！

怎么样？我想你应该已经做出来了，即使不是完全正确也应该有了大概的思路。如果你还没有尝试过，请赶快再去尝试一下吧，这样会让你更容易理解下面的内容，同时也可以比较一下你想的和我所讲的是否一样。顺便说一下，要想学好编程，最重要的就是要多尝试。

要想把 3 个数从大到小排序，其实有很多种方法，我们这里主要讲解两种方法。

下面来讲第 1 种方法，这是一种最直接的方法。

如果 $a \geq b$ 并且 $b \geq c$ ，打印 a b c；

如果 $a \geq c$ 并且 $c > b$ ，打印 a c b；

如果 $b > a$ 并且 $a \geq c$ ，打印 b a c；

如果 $b \geq c$ 并且 $c > a$ ，打印 b c a；

如果 $c > a$ 并且 $a \geq b$ ，打印 c a b；

如果 $c > b$ 并且 $b > a$ ，打印 c b a。

完整的代码如下：

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

第 2 种方法，我称之为“换位法”。一共有 3 个变量，也就是说分别有 3 个小房子 a、b 和 c。我们的目标是在小房子 a 中存储最大的，在小房子 b 中存储次大的，在小房子 c 中存储最小的。

首先，我们先将小房子 a 中的数与小房子 b 中的数作比较，如果小房子 a 中的数小于小房子 b 中的数，则将小房子 a 和小房子 b 中的数交换。这样我们就可以确定，在小房子 a 和小房子 b 中，一定是小房子 a 中存的是比较大的数。关于如何交换两个变量的值，我们在第 2 章的第 10 节已经讨论过了，需要借助另外一个小房子 t 作为中转，代码如下：

```
if (a<b) {t=a; a=b; b=t;}
```

在上面这行代码中，当 $a < b$ 这个条件成立时我们需要连续执行 3 条语句，此时我们需要将这 3 条语句放在一对 { } 括号中形成一个语句块，这样当条件 $a < b$ 成立时，计算机才会依次执行 $t=a$; $a=b$; $b=t$; 这 3 条语句。如果不加 { }，例如：

```
if (a<b) t=a; a=b; b=t;
```

则当条件 $a < b$ 成立时计算机会执行 $t=a$;，而 $a=b$; 和 $b=t$; 这 2 条语句计算机无论如何都会执行。因为 if 语句后面只能跟随一条语句或者一个语句块，使得 $a=b$; 和 $b=t$; 与 $if(a < b)$ 这个条件没有任何关系。或许如下写法更容易让你理解：

```
if (a<b) t=a;
a=b;
b=t;
```

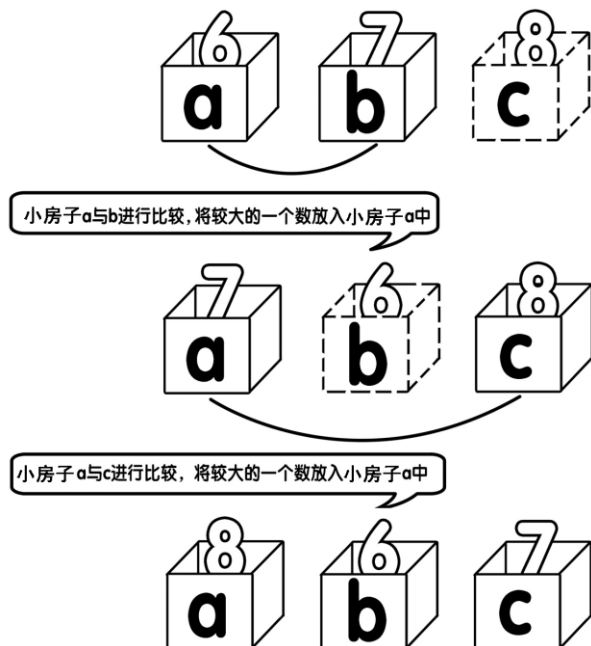
所以当我们需要在 if 语句后面执行多条语句的时候，需要用 { } 把所有需要执行的语句括起来，形成一个语句块，这样，计算机就知道它们是一起的了，要执行就一起执行，要么就都不执行。

接下来，我们需要再比较小房子 a 和小房子 c 中的数。如果小房子 a 中的数小于小房子 c 中的数，则将小房子 a 和小房子 c 中的数交换。这样我们就可以确定，在小房子 a 和小房子 c 中，一定是小房子 a 中存的数的值比较大。

```
if (a<c) {t=a; a=c; c=t;}
```

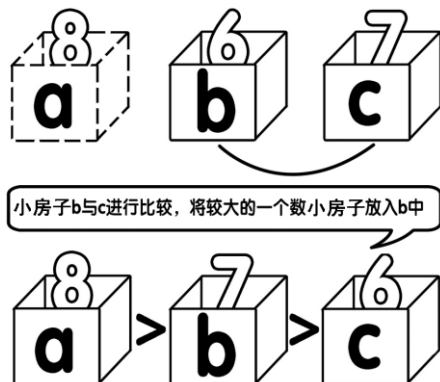
经过将小房子 a 中的数分别与小房子 b、小房子 c 中的数进行比较和交换，我们可以确定小房子 a 中存储的数一定是 3 个数中最大的。至于目前小房子 b 和小房子 c 目前存的是什么值不重要，因为我们待会儿还要继续比较小房子 b 和小房子 c 中的值。重要的是我们已经确定小房子 a 中存储的已经是最大的数了。

啊哈 C！思考快你一步——用编程轻松提升逻辑力



下面继续比较小房子 b 和小房子 c 中的值，将较大的值放在小房子 b 中。

```
if (b<c) {t=b; b=c; c=t;}
```



经过 3 轮比较，我们终于排序完毕，最大的数放在小房子 a 中，次大的数放在小房子 b 中，最小的数放在小房子 c 中。

下面是完整的代码，赶快来试一试吧。

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

```
int main( )
{
    int a,b,c,t;
    scanf("%d %d %d",&a,&b,&c);
    if (a<b) {t=a; a=b; b=t;}
    if (a<c) {t=a; a=c; c=t;}
    if (b<c) {t=b; b=c; c=t;}
    printf("%d %d %d",a,b,c);
    system("pause");
    return 0;
}
```

我们在第 6 章将会学习选择排序，它就是基于这种方法的扩展。
题外话：有时像这样的写法，显得过于紧凑。

```
if (a<b) {t=a; a=b; b=t;}
```

我们可以改为如下较为宽松的写法：

```
if (a<b)
{
    t=a;
    a=b;
    b=t;
}
```

其完整的代码如下：

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

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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

🌀 一起来找茬

1. 下面程序的功能是从键盘读入 1 个整数，如果这个数是奇数就输出这个数后面的 3 个数，如果这个数是偶数，就输出这个数前面的 3 个数。例如，如果输入的整数是 5，就输出 678；如果输入的整数是 4，就输出 321。其中有两个错误，快来改正吧。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    scanf("%d",&a) ;
    if (a%2==1)
        printf("%d ",a+1) ;
        printf("%d ",a+2) ;
        printf("%d ",a+3) ;
    else
        printf("%d ",a-1) ;
        printf("%d ",a-2) ;
        printf("%d ",a-3) ;
    system("pause");
    return 0;
}
```

➔ 更进一步，动手试一试

1. 从键盘读入任意 4 个整数，将其从小到大输出。

第 8 节 运算符总结

通过前面的学习，我们了解了 C 语言中的许多运算符，有算术运算符如“+”、关系运算符如“==”和逻辑运算符如“&&”、“||”。下面我们来总结一下，如表 3-1 所示。

表 3-1 运算符总结

名 称	作 用	名 称	作 用
+	加	>=	大于等于
-	减	<=	小于等于
*	乘	!=	不等于
/	除	&&	与
>	大于		或
<	小于	!	非
==	等于		

第9节 1 > 2 究竟对不对

幼儿园的小朋友都知道 $1 > 2$ 这个关系表达式是不成立的。对 C 语言来讲， $1 > 2$ 这个关系表达式是假的，但是这个表达式的写法并没有任何错误，只不过它是假的。如果喜欢，你也可以写 $11 < 10$ ，相信你还可以写出很多这样假的关系表达式。可是你千万不要以为类似于 $11 < 10$ 这样的假的表达式没有任何意义，在第 4 章你会发现它大有用途。

此外， $2 >= 2$ 这个关系表达式是真的，因为它表示的是 2 大于 2 或者 2 等于 2，只需满足其中任意一个条件就是真的关系表达式。类似于 $1 <= 2$ 这样的表达式也是真的。

请看下面这段代码：

```
if (1>2)
    printf("yes");
else
    printf("no");
```

上面这段代码表示，如果 $1 > 2$ 成立，也就是说如果 $1 > 2$ 这个关系表达式是真的，则输出 yes，否则输出 no。很显然 $1 > 2$ 是假的，计算机会输出 no。这个应该很容易理解。但是看到下面这段代码你肯定会晕：

```
if (1)
    printf("yes");
else
    printf("no");
```

你猜计算机会输出什么？去试一试！

如果是下面这样呢？

```
if (-5)
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
printf("yes");  
else  
printf("no");
```

你猜计算机输出了什么，去试一试吧。

如果是这样呢？代码如下：

```
if (0)  
printf("yes");  
else  
printf("no");
```

计算机又输出了什么呢？

如果上面的 3 段代码你都尝试过，你会发现前两段代码都是输出 yes，也就是说，计算机认为前两个代码中 if 后面圆括号内的关系表达式都是成立的，是真的。第 3 段代码输出的是 no，即认为第 3 段 if 后面一对圆括号内的关系表达式不成立，是假的。

这时你可能会觉得奇怪了，关系表达式不应该是一个式子吗，至少也应该有一个“>”、“<”或“==”之类的运算符才对啊。为什么单独一个数字也有真假呢？

这个确实很奇怪，我们计算机就是认为 1 和-5 是真的，0 是假的。其实在 C 语言中，对于某一个数讨论真假时，只有 0 是假的，其余都被认为是真的。很显然，如下 3 个程序都是打印 yes：

```
if (8)  
printf("yes");  
else  
printf("no");
```

```
if (1000)  
printf("yes");  
else  
printf("no");
```

```
if (-123)  
printf("yes");  
else  
printf("no");
```

只有下面这个程序才会打印 no：

```
if (0)  
printf("yes");  
else
```

```
printf("no");
```

第10节 讨厌的嵌套

if-else 语句的“嵌套”就是在一个 if-else 语句中再“嵌套”另外一个 if-else 语句。在讲“嵌套”之前我们先回忆一下本章第6节中的一个例子：如何在三个数中找出最大的一个数。

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

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

在上面的代码中，我们使用了“&&”这个逻辑关系运算符来解决两个条件同时“满足”的需求。其实我们还有另外一种方法来解决这个问题。

```
if(a>=b && a>=c)
    printf("%d",a);
```

例如，上面这段代码，我们可以用“嵌套”的方式写成：

```
if(a>=b)
{
    if(a>=c)
    {
        printf("%d",a);
    }
}
```

其意思是：当 $a \geq b$ 条件满足时，再进一步讨论 a 与 c 的关系（如果 $a \geq c$ 也成立的话，就打印 a ）。

我们接着往下想，如果此时 $a \geq b$ 已经成立，但是 $a \geq c$ 不成立的话，是不是就意味着在 a 、 b 、 c 中， c 是最大值呢？答案是肯定的。代码如下：

```
if(a>=b)
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
{
    if(a>=c)
    {
        printf("%d",a);
    }
    else
    {
        printf("%d",c);
    }
}
```

那如果第一个条件 $a \geq b$ 不成立呢？完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    scanf("%d %d %d",&a,&b,&c);
    if(a>=b) //a>=b 成立的情况
    {
        if(a>=c) //进一步讨论 a 与 c 的关系
        {
            printf("%d",a);
        }
        else
        {
            printf("%d",c);
        }
    }
    else //a>=b 不成立的情况
    {
        if(b>=c) //进一步讨论 b 与 c 的关系
        {
            printf("%d",b);
        }
        else
        {
            printf("%d",c);
        }
    }

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

在上面的代码中所有的 if-else 语句都加了 { }，这样看起来很臃肿，我们之前说过如果 if 和 else 后面只有一条语句的话，我们是省略 { } 的。代码如下：

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

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

上面的代码其实还可以更简洁，如下：

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

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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

你发现没有，在上面的代码中，我们把最外层 if-else 语句的 { } 也去掉了。有的同学可能就有问题要问了。

```
if (a>=b)  
{  
    if (a>=c)  
        printf("%d",a);  
    else  
        printf("%d",c);  
}  
else  
    if (b>=c)  
        printf("%d",b);  
    else  
        printf("%d",c);
```

问题 1：这里可以看成是一个语句

问题 2：这个 else 是哪个 if 的呢

我们先来解决问题 2。问题 2 比较简单，else 的匹配采用就近原则，离上方哪个 if 最近，就属于哪个 if。所以问题 2 所指向的那个 else 是属于 if(b>=c) 这个 if。这样你就可以把 if 和 else 分别一一配对了，是不是有点连连看的感觉。

问题 1 说起来比较麻烦，但是原理很简单，请听我慢慢道来。其实上面虚线框中的代码是一个 if-else 语句，并且是一个很完整的 if-else 语句，因为它不但有 if 部分还有 else 部分。只不过很特别的是，if 部分和 else 部分都分别有自己的子语句 printf 部分，所以看起来就显得很多了。其本质上就是一条 if-else 语句，一个“复合语句”。不过在外层的 if(a>=b) 看来，它就是一条 if-else 语句，至于这条语句内部长什么样子，if(a>=b) 并不关心，其实就像大盒子里面套小盒子一样，if(a>=b) 是搞不清楚 if(a>=c) 里面是什么样子的。

第 11 节 if-else 语法总结

其实说起来很简单，当 if() 括号内的关系表达式成立的时候，就执行 if() 后面的 { } 中的内容，不成立的时候执行 else 后面 { } 中的内容。当 { } 内的语句只有一条的时候，{ } 可以省略。

```
if (关系表达式)  
{  
    语句;  
    语句;  
    .....  
}
```

```
else
{
    语句;
    语句;
    .....
}
```

当{ }内的语句只有一条的时候，可以省略{ }。

```
if(关系表达式)
    语句;
else
    语句;
```

重量级选手登场

第 1 节 永不停止的哭声

在第 2 章我们就知道，如果让计算机开口说话则使用 `printf` 语句，例如，让计算机说“wa”，则使用 `printf("wa");`。那如果让计算机说 10 遍“wa”呢？你可以尝试这样写：

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

或者你也可以这样写：

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

如果让计算机说 10000 遍“wa”呢？该怎么办呢？我想你肯定要疯了吧。在本节我们就是要学习如何让计算机做重复的事情。

好，首先我们学习如何让计算机“永无止境”地说“wa”。这个很简单，代码如下：

```
while(1>0)  
{  
    printf("wa");
```



```
}

```

赶快尝试一下，你是不是发现计算机开始无止境地说“wa”了呢。

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    while(1>0)
    {
        printf("wa");
    }
    system("pause");
    return 0;
}
```

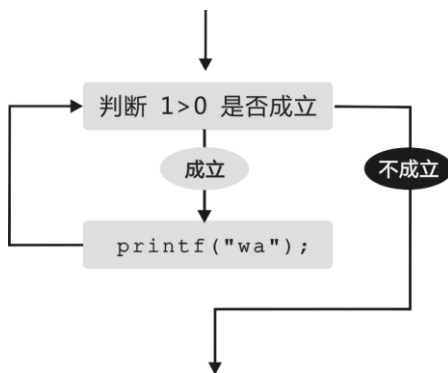
一定要尝试啊，尝试后再往下看。

当然，你也可以让计算机无止境地说任何内容。例如：

```
while(1>0)
{
    printf("bie wa");
}
```

回到正题。在上面的代码中有两部分组成，一部分是 `while()`，另一部分是 `{ }` 中的内容。它表示的意思是，当 `while` 后面 `()` 中的关系表达式为真时，也就是关系表达式成立时才执行 `{ }` 中的内容。

那么很显然，`1>0` 这个关系是永远成立的，所以计算机会一直执行 `{ }` 中的内容，而上面的例子 `{ }` 中只是输出 `wa`，所以计算机就会不停地输出 `wa`。



啊哈 C！思考快你一步——用编程轻松提升逻辑力

这里顺便说一下，如果{ }中只有一条语句，那么{ }便可以省略。也就是说可以简写成：

```
while(1>0)
    printf("wa");
```

这时你可能要问了：**while** 后面()中的 $1>0$ 是否可以写成 $2>1$ 或者 $3>0$ 等？

当然可以，如果想让计算机永不停止地执行，你可以写任何一个关系表达式为真的式子。甚至，你可以写 **while(1)**，还记得吗？我们在第 3 章说过，如果对某个数字来判断真假，只要这个数不为 0 就是真的，例如，下面这段代码也可以永不停止地在屏幕上打印 **wa**：

```
while(1)
    printf("wa");
```

下面我来实现一个很炫的效果，“黑客帝国”来啦！

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    while(1>0)
    {
        printf("0 1");
    }
    system("pause");
    return 0;
}
```

执行上面的代码后计算机就会不停地在屏幕上打印 0 和 1。

当然，你可以改变一下打印的背景与字的颜色，例如，改为黑色的背景，绿色的字。还记得吗？是使用 **system("color 0a");**这个语句。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    system("color 0a");
    while(1>0)
    {
        printf("0 1");
    }
    system("pause");
    return 0;
}
```

是不是很像黑客帝国？

猜一猜，运行下面这段代码，计算机会有什么反应？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    while(1<0)
    {
        printf("wa");
    }
    system("pause");
    return 0;
}
```

我想你应该猜到了，计算机一句“wa”都没有说，这是为什么呢？因为 $1 < 0$ 这个关系表达式不成立，所以计算机没有执行后面 { } 中的内容。

此时你会发现计算机要么打印无数遍都不停止，要么就一次都不打印。如果想打印指定的次数该怎么办？例如，我们之前遗留下来的问题：打印 1000 遍 wa，一次也不多、一次也不少该怎么办？不要着急，让我们一起进入第 2 节。

一起来找茬

1. 下面这段代码是让计算机“永无止境”地打印 hello。其中有两个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    while(1>0);
    print("hello");
    system("pause");
    return 0;
}
```

更进一步，动手试一试

1. 让计算机“永无止境”地在屏幕上显示中文汉字“你好”。

这一节，你学到了什么

1. 在 C 语言中我们用什么语句来实现循环的功能？

第 2 节 我说几遍就几遍



在本章第 1 节，我们学习了如何使用 `while` 循环来让计算机做重复的事情，本节我们将揭晓如何让计算机重复指定的次数。

我们知道如果 `while` 后面 `()` 中的关系表达式成立，计算机就会运行 `{ }` 中的内容。如果 `()` 中的关系表达式永远成立，那么计算机会“永无止境”地去重复执行 `{ }` 中的内容。

假如让计算机打印 100 次“wa”，我们需要解决的就是如何让 `while()` 中的关系表达式在前 100 次是成立的，然后在第 101 次的时候就不成立了。想一想，根据我们之前学过的知识，你有没有什么方法？

很显然 `while` 后面的 `()` 中的关系表达式，像 `2>1` 或者 `1<=100` 等都是不适合的，因为像这样用固定的数字组合的关系表达式，一旦被写出，那么这个式子是否成立就已经是板上钉钉啦，非真即假，而且永远不会改变。例如，`1<=100` 这个关系表达式是永远成立的，这样并不符合我们的要求。我们需要创造怎样一个新的关系表达式，才能让这个式子有时成立，有时不成立？该怎么办呢？

我猜你已经想到了！那就是……伟大的“变量”。

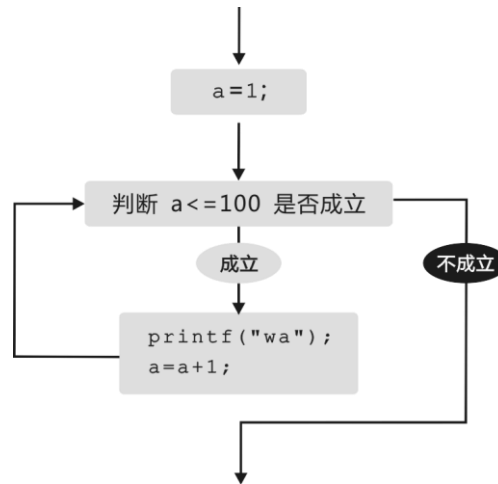
我们可以尝试一下带有变量的关系表达式，例如，`a<=100`。因为 `a` 是一个变量，`a` 这个小房子里面所装的数是可以变化的。当小房子 `a` 中的数是 1 的时候，`a<=100` 是成立的；当小房子 `a` 中的数是 101 时，`a<=100` 就不成立了，这正好满足了我们对于表达式 `a<=100` 有时成立有时不成立的要求。对于 `a<=100` 这个关系表达式是否成立的关键就在于 `a` 这个变量所装的数是多少，也就是变量 `a` 的值。

如果我们想让 `a<=100` 在前 100 次成立，在第 101 次不成立的话，只需将变量 `a` 的值从 1

变化到 101 就可以了。那么如何让变量 a 的值从 1 变化到 101 呢？我们只需在最开始的时候将变量 a 的值赋为 1，然后 `while` 循环每循环一次，就将变量 a 的值在原来的基础上再加 1 就可以了。当变量 a 的值加到 101 时， $a \leq 100$ 就不成立了，就会结束循环。代码如下：

```
int a;
a=1;
while(a<=100)
{
    printf("wa");
    a=a+1;
}
```

再来分析一下上面的代码，`a=a+1`；这条语句的作用是把小房子 a 中的值在原本的基础上增加 1（在第 2 章第 5 节有详细解释，如果还没有搞懂还是回去看看吧）。变量 a 最开始时值为 1，每执行一次 `while` 循环，变量 a 的值就会在原来的基础上增加 1，依次变成 2、3、4、5、6、7、8、9、 \dots ，100、101。直到变量 a 的值为 101 时， $a \leq 100$ 的条件不成立，退出循环。



完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        printf("wa");
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
        a=a+1;
    }
    system("pause");
    return 0;
}
```

赶快尝试一下吧。如果要输出 10000 个 wa 该怎么办？又或者输出 10 个 hello 呢？赶快去尝试一下吧。

接下来一个问题：如果要打印 1~100 该怎么办？

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        printf("wa");
        a=a+1;
    }
    system("pause");
    return 0;
}
```

在这段代码中，我们打印了 100 个 wa，那么打印 1~100 也很简单，只需修改 printf 语句就可以了。那么如何修改 printf 语句呢？之前 printf 语句的作用是输出 wa，现在需要输出 1~100，正巧变量 a 的值就是从 1 一步步增加到 100 的。我们只需在每次循环时把 a 的值输出就好了，即把 printf("wa");改为 printf("%d ",a);。完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        printf("%d ",a);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

赶快尝试一下吧。

如果我们想倒序输出呢？就是让计算机先输出 100 再输出 99 接着输出 98、97、96、95、…、1，该怎么办？

我们刚刚学过的“让计算机输出 1~100”，就是让变量 a 从 1 开始，通过 while 循环把变量 a 的值每次都输出来，并且每次循环的时候将变量 a 的值增加 1，这样就会打印出 1~100。而此时的要求是从 100 打印到 1。很显然我们需要让变量 a 从 100 开始，通过 while 循环把变量 a 的值每次都输出来，不过每次需要递减 1，一直递减到 1 为止。代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=100;        //初始值从 100 开始
    while(a>=1)  //请注意这里的循环条件变为 a>=1
    {
        printf("%d ",a);
        a=a-1;   //每循环一次将 a 的值递减 1
    }
    system("pause");
    return 0;
}
```

问题又来啦，如果希望打印的是 1~100 中的偶数呢？自己想一想吧。

怎么样，有没有思路？其实要输出 1~100 中的偶数，有很多种方法。比如之前变量 a 是从 1 开始的，之后每次在原有的基础上增加 1，那么 a 就会从 1 到 2 再到 3 再到 4……一直到 101，当变量 a 的值增加到 101 时，不满足条件 $a \leq 100$ ，就会退出 while 循环。现在我们可以改变一下思路，让变量 a 的值从 2 开始，每次增加 2，这样变量 a 就会从 2 增加到 4 再增加到 6……以此类推。代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=2;
    while(a<=100)
    {
        printf("%d ",a);
        a=a+2;
    }
    system("pause");
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
    return 0;
}
```

好了，又到了尝试的时候了。

如何让计算机打印 1~100 中能被 3 整除的数，你应该也会了，就是先将变量 a 的初始值赋为 3，然后每次增加 3。赶快再去尝试一下吧。

上面的写法虽然好，却不是万能的，假如不是输出 1~100 中能被 3 整除的数，而是输出 1~100 中所有不能被 3 整除的数呢？例如：1、2、4、5、7、8、10，…，97、98、100，又该怎么办呢？不要着急，我们将在本章第 3 节彻底解决这个问题。

一起来找茬

1. 下面这段代码是让计算机从 100 打印到 200。其中有 3 个错误，快来改正吧！

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=100;
    while(a<200) ;
    {
        printf("%d ",a);
    }
    system("pause");
    return 0;
}
```

更进一步，动手试一试

1. 让计算机从 1 打印到 100 再打印到 1，例如：1 2 3 …… 98 99 100 99 98 …… 3 2 1。

第 3 节 if 对 while 说：我对你很重要

在本章第 2 节，我们学习了 while 循环的基本使用方法，但是遗留了一个问题，即如何让计算机输出 1~100 中所有不能被 3 整除的数，例如：1、2、4、5、7，…，97、98、100。

通过本章第 2 节的内容我们可以很容易地让计算机打印 1、2、3，…，100，只需让变量 a 从 1 开始每次增加 1 就可以了。如果想每次遇到 3 的倍数就不打印的话，我们只需在每次打印之前对变量 a 的值进行判断就好了，即当变量 a 的值是 3 的倍数时就不输出，否则就输出。那怎么判断变量 a 的值是否是为 3 的倍数呢？这就需要我们在第 3 章学习的 if 语句。我们只需要通过 if 语句来判断变量 a 的值除以 3 的余数是否为 0 就可以了。如果余数不为 0，说明变量 a 中的值不是 3

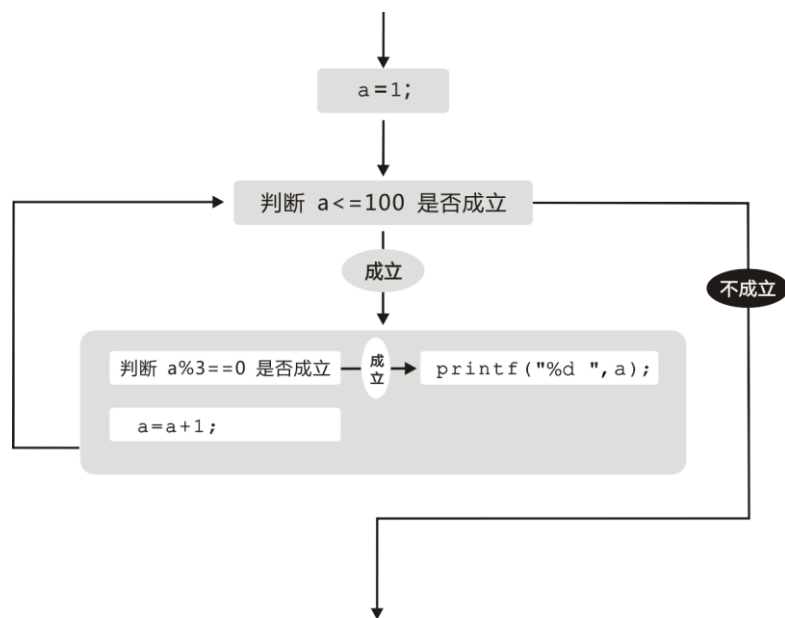
的倍数，就将变量 `a` 中的值打印出来；否则就说明变量 `a` 中的值是 3 的倍数，不能打印。

那么怎么判断变量 `a` 中的值除以 3 的余数是否为 0 呢，需要使用 “%” 这个运算符。在第 3 章中我们介绍过，“%” 读作 `mod`，也可以称之为“取模”，作用就是获取余数。另外说一下 “%” 这个运算符的左右两边必须为整数。而 “/” 这个符号表示除号，作用是获取商，“/” 这个运算符的左右两边既可以是整数也可以是小数。代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        if(a%3!=0)
            printf("%d ",a);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

赶快尝试一下吧。

啊哈 C！思考快你一步——用编程轻松提升逻辑力



如果要输出 1~100 中能被 3 整除但是不能被 5 整除的所有数，又该怎么办？

这个数是 3 的倍数但不是 5 的倍数，也就是需要变量 `a` 除以 3 的余数为 0 并且除以 5 的余数不为 0。这里的逻辑关系“并且”在 C 语言中的表示方法我们在第 3 章已经学习过，用“`&&`”表示，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        if(a%3==0 && a%5!=0)
            printf("%d ",a);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

更复杂的来啦！

你有没有和同学玩过一个游戏：大家围成一圈，从 1 开始报数，但是每逢遇到 7 的倍数或

者末尾含 7 的数，例如 7、14、17、21、27、28 等，就要拍手并且不能报出，谁出错了，谁就要受到惩罚。

现在我想知道 1~100 中有多少这样的数，请你写这样一个程序，输出 1~100 中所有 7 的倍数和末尾含 7 的数。

很简单，我们先参照以往的程序，利用 `while` 循环，让变量 `a` 从 1 递增到 100，不过我们每次在输出变量 `a` 的值之前需要对变量 `a` 进行判断。根据题目的要求，如果变量 `a` 的值是 7 的倍数或者末尾含有 7 就打印出来。判断一个变量是否为 7 的倍数我们已经很熟悉了，只需要判断变量 `a` 除以 7 的余数是否为 0 就可以，即如果 `a%7==0` 这个关系表达式成立就输出。那怎么判断变量 `a` 的值末尾是否含 7 呢？我们仔细想一想就会发现末尾含 7 的数其实就是这个数的个位为 7，也就是这个数除以 10 的余数为 7。发现这个规律就好办了，即在 `a%10==7` 这个关系表达式成立时输出就可以啦。

好了，现在有两个关系表达式 `a%7==0` 和 `a%10==7`，分别表示这个数是否为 7 的倍数以及末尾是否含 7。这两个式子是“或者”的关系，只要有一个成立，就将这个数输出。这里的逻辑关系“或者”在 C 语言中的表示方法我们在第 3 章也学习过，用“`||`”表示，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=100)
    {
        if(a%7==0 || a%10==7)
            printf("%d ",a);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

第 4 节 求和！求和！！求和!!!

在本章第 2 节我们已经学习了如何让计算机打印 1~100，那如何让计算机求 1~100 中所有数的和？

你可能会说，首项加尾项的和乘以项数然后再除以 2，就可以了。没错，你可以这样做，但

啊哈 C！思考快你一步——用编程轻松提升逻辑力

是如果求 1~100 中所有 7 的倍数或者末尾含 7 的数的总和，又该怎么办呢？

在求 1~100 的和之前，我们先来解决如何求 1+2+3 的值。

没错，你可以这样写：

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

但是如果计算 1~100 中所有数的和也这样写的话，岂不是太麻烦了。我们可以尝试另一种写法，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    a=0; //想一想为什么 a 的初始值要为 0 呢？
    a=a+1;
    a=a+2;
    a=a+3;
    printf("%d",a);
    system("pause");
    return 0;
}
```

你可能会说这样写岂不是更麻烦……但是我们发现在上面的这段代码中，`a=a+1`；`a=a+2`；`a=a+3`；这三句话，基本思想是相同的，第一次加 1，第二次加 2，第三次加 3。我们可以把这三个语句用 `a=a+i`；来表示。然后让变量 `i` 从 1 到 3 循环就可以了。代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,i;
    a=0;
```

```
i=1;
while(i<=3)
{
    a=a+i;
    i=i+1;
}
printf("%d",a);
system("pause");
return 0;
}
```

如果需要计算 1~100 的和，我们只需将上面代码中 $i \leq 3$ 修改为 $i \leq 100$ 就可以了，赶快去尝试一下吧。

如果求 1~100 中所有 7 的倍数或者末尾含 7 的数的总和，又该怎么办呢？先来回顾一下刚刚才学会的求 1~100 中所有数的和的方法，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,i;
    a=0;
    i=1;
    while(i<=100)
    {
        a=a+i;
        i=i+1;
    }
    printf("%d",a);
    system("pause");
    return 0;
}
```

在上面的代码中，变量 i 会从 1 到 100 每次递增 1，然后每次将变量 i 的值累加到变量 a 上。这个变量 i 就像是一个搬运苹果的工人，刚开始只拿 1 个苹果，之后拿 2 个苹果，再之后又拿 3 个苹果……最后一次一下拿了 100 个苹果。变量 a 就像是一个很大很大的水果篮子，用来装这些苹果。每次拿来的苹果统统被装进篮子里面，第一次放 1 个苹果进去，第二次放 2 个苹果，第三次放 3 个苹果进去……最后一次放 100 个苹果进去。最后篮子 a 中苹果的总数就是 1~100 的和。所以我们最后输出了变量 a 的值，这就是答案。

啊哈 C！思考快你一步——用编程轻松提升逻辑力

那么求 1~100 中所有 7 的倍数或者末尾含 7 的数的总和，又该怎么办呢？

此时我们不再是每次都把苹果扔进篮子里面。只有当苹果的个数是 7 的倍数或者末尾含 7 的时候，这堆苹果才能被扔进篮子里面，所以就不能每次都执行 `a=a+i`。此时我们需要借助 `if` 语句来完成我们的目标。其中变量 `i` 就是每次拿的苹果的数量，代码如下：

```
if(i%7==0 || i%10==7)
{
    a=a+i;
}
i=i+1;
```

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,i;
```

```

a=0;
i=1;
while(i<=100)
{
    if(i%7==0 || i%10==7)
    {
        a=a+i;
    }
    i=i+1;
}
printf("%d",a);
system("pause");
return 0;
}

```

🌀 一起来找茬

1. 下面这段代码是求 $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10$ 的值。其中有 3 个错误，快来改正吧！

```

#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int a,i;
    a=0;
    i=1;
    while(i<10)
    {
        a=a*i;
    }
    printf("%d",a);
    system("pause");
    return 0;
}

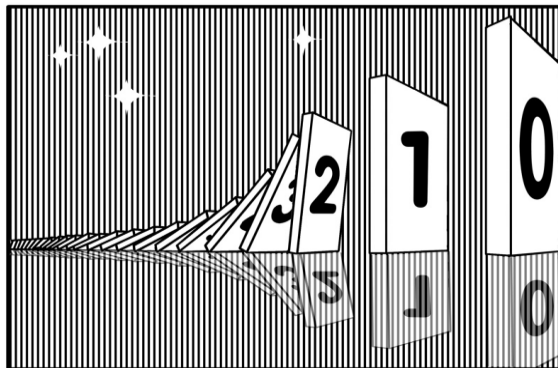
```

➔ 更进一步，动手试一试

1. 求 1~100 所有偶数的和。
2. 输入一个整数 $n(1 \leq n \leq 9)$ ，求 n 的阶乘⁵。

⁵ 正整数阶乘指从 1 乘以 2 乘以 3 乘以 4 一直乘到所要求的数。例如，所要求的数是 4，则阶乘式是 $1 \times 2 \times 3 \times 4$ ，得到的积是 24，24 就是 4 的阶乘。如果所要求的数是 6，则阶乘式是 $1 \times 2 \times 3 \times \cdots \times 6$ ，得到的积是 720，720 就是 6 的阶乘。如果所要求的数是 n ，则阶乘式为 $1 \times 2 \times 3 \times \cdots \times n$ ，设得到的积是 x ， x 就是 n 的阶乘。

第 5 节 逻辑挑战 4：60 秒倒计时开始



你是否曾看过 60 秒倒计时，就是从 60 到 59、58、57、56……然后一直到 0，如果我们现在也能做出这种效果是不是很帅，不要走开，精彩马上开始。

在尝试做 60 秒倒计时之前，我们先学习如何做 3 秒倒计时，就是让计算机输出 3、2、1、0。这个很简单，使用 4 次 `printf` 语句就可以了。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("3");
    printf("2");
    printf("1");
    printf("0");
    system("pause");
    return 0;
}
```

但是计算机一下子就显示了 3210，丝毫没有倒计时的感觉，我们希望计算机先打印 3，1 秒后打印 2，再过 1 秒打印 1，再过 1 秒打印 0。如果要实现每过 1 秒打印一个数，我们就需要用到“等待”这个语句，这个语句就是 `Sleep()`，注意第一个字母 S 是大写，例如，`Sleep(1000)` 就表示等待 1 秒。其实这里的 `Sleep` 就是“等待”的意思，圆括号内的数字就是表示需要“等待”的时间，单位是毫秒，1000 毫秒等于 1 秒。还有很重要的一点，如果需要用 `Sleep()`，就必须在代码的开头加上 `#include <windows.h>` 才行⁶。

我们现在让计算机每打印一个数就等待一秒，也就是每执行 `printf()` 一次，就 `Sleep(1000)`。

⁶ 此方法只在 Windows 系统下有效。

修改之后的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    printf("3");
    Sleep(1000);
    printf("2");
    Sleep(1000);
    printf("1");
    Sleep(1000);
    printf("0");

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

尝试过后，你是不是发现计算机开始每过一秒打印一个数了呢，但是计算机每次打印新的数之前，并没有把之前打印出来的数清除，离我们所希望的倒计时还差那么一点点。这里介绍一个“清屏”语句，就是把现在屏幕上所有的内容清除干净，这个语句是 `system("cls");`。好了，我们现在就把 `system("cls");` 加在每一个 `printf()` 语句的前面。这样就可以起到在每次打印新的内容之前先把屏幕清除干净。代码如下，赶快尝试一下吧。

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    system("cls");
    printf("3");
    Sleep(1000);

    system("cls");
    printf("2");
    Sleep(1000);

    system("cls");
    printf("1");
    Sleep(1000);

    system("cls");
    printf("0");
    Sleep(1000);
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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

怎么样，是不是有点儿意思啊。通过这种方法我们就可以做出 60~0 的倒计时，不过像上面这样写的话，10 以内的倒计时还可以接受，60~0 的倒计时写起来就太麻烦了。我们仔细分析一下上面这段代码，就会发现它由 4 个小部分组成（在代码中已经用空行隔开），这 4 个小部分，除了 `printf()` 语句中的数字不一样之外，其余都是一样的，而且数字也是有规律的，即从 3 到 0。我们很自然就想到利用之前学习的 `while` 循环来代替这 4 个 `printf()` 语句。

我们之前学习过如何从 100 到 1 依次输出，即让变量 `a` 从 100 开始，通过 `while` 循环每次把变量 `a` 的值输出来，同时每次循环时还需要将变量 `a` 的值减少 1。这样就会打印出 100~1。显然，让计算机从 3 打印到 0 也是一样的，只不过是让变量 `a` 从 3 开始，然后也是通过 `while` 循环每次把变量 `a` 的值输出，同时每次递减 1，一直递减到 0 为止。代码如下：

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

然后在这个代码的基础上，在 `printf()` 语句前加上清屏语句 `system("cls")`，在 `printf()` 语句之后加上暂停语句 `Sleep(1000)` 就可以了。完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a;
    a=3;
    while(a>=0)
```

```

    {
        system("cls");
        printf("%d",a);
        Sleep(1000);
        a=a-1;
    }
    system("pause");
    return 0;
}

```

如果要从 60 秒开始倒计时，只需将变量 a 的初始值改为 60 就可以。另外，你可以让这个倒计时看起来更好看一些，我们可以修改一下输出屏幕的背景以及字的颜色，例如，将下面这段代码改成了黑底绿字，看起来是不是更酷呢。

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a;
    a=60;
    system("color 0a");
    while(a>=0)
    {
        system("cls");
        printf("%d",a);
        Sleep(1000);
        a=a-1;
    }
    system("pause");
    return 0;
}

```

好了，现在你可以做 100 秒甚至 1000 秒的倒计时了，尝试将 Sleep 括号内的数值改小一点，例如，改为 Sleep(50)，你会发现不同的效果，赶快尝试一下吧。

➔ 更进一步，动手试一试

1. 请尝试编写一个两分钟的倒计时。形如：2:00 1:59 1:58 …… 1:00 0:59
0:58 …… 0:02 0:01 0:00

➔ 这一节，你学到了什么

1. 清屏的命令是什么？

第 6 节 这个有点晕——循环嵌套来了

首先，我们尝试这样一个图形：

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

上面这个图形由 3 行星号组成，每行有 5 个星号，也就是说一共有 $3 \times 5 = 15$ 个星号。如果我们想打印这个图形，有很多种方法。最简单的方法如下：

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

上面的写法当然可以，但是如果输出 100 行并且每行 100 个星号的话就太麻烦了。利用我们学过的 `while` 循环，可以改进一下：

```
a=1;  
while(a<=15)  
{  
    printf("*");  
    if(a%5==0)  
        printf("\n");  
    a=a+1;  
}
```

上面这段代码的思想是：我们知道一共需要输出 15 个星号，所以只需要循环 15 次（每循环一次就输出一个星号）。但是每行只能有 5 个星号，也就意味着，每打印 5 个星号就需要换一行，我们可以通过 `if` 语句来控制打印换行。如何控制呢？我们知道每循环一次就会打印一个星号，变量 `a` 的值也会递增 1，也就是说目前变量 `a` 的值其实就是已经打印的星号的个数。如果变量 `a` 的值恰好是 5 的倍数，就说明此时需要换行了，`if(a%5==0) printf("\n");` 正是起到了这个作用。完整的代码如下：

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

```

#include <stdlib.h>
int main()
{
    int a;
    a=1;
    while(a<=15)
    {
        printf("*");
        if(a%5==0)
            printf("\n");
        a=a+1;
    }
    system("pause");
    return 0;
}

```

当然还有别的方法，就是使用嵌套循环。我们再来仔细观察一下这个图。

```

*****
*****
*****

```

该图一共有 3 行，可以用 3 次 `while` 循环，每行只需要打印一个 “\n”，就能解决打印行的问题，代码如下：

```

a=1;
while(a<=3)
{
    printf("\n");
    a=a+1;
}

```

然后，每行需要打印 5 个星号，我们在刚才写好的 `while` 循环中再嵌套一个 `while` 循环来打印 5 个星号，代码如下：

```

a=1;
while(a<=3) //while a 循环用来控制换行
{
    b=1;
    while(b<=5) //while b 循环用来控制输出每行 5 个星号
    {
        printf("*");
        b=b+1;
    }
    printf("\n");
}

```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
    a=a+1;
}
```

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    a=1;
    while (a<=3)
    {
        b=1;
        while (b<=5)
        {
            printf("*");
            b=b+1;
        }
        printf("\n");
        a=a+1;
    }
    system("pause");
    return 0;
}
```

在上面的代码中，有两个 **while** 循环，一个是外循环，另一个是内循环，内循环嵌套在外循环中。其实内循环是外循环的一部分。外循环每循环一次，内循环就会从头到尾循环一遍。其中用来控制外循环的循环次数的变量是 **a**，因此我们称这个外循环为 **while a** 循环。用来控制内循环的循环次数的变量是 **b**，因此我们称这个内循环为 **while b** 循环。

想一想，如果想要完成这样的图形该怎么办？

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

经过分析，我们发现，这个图形有 5 行，仍然先用 **while a** 循环来解决 5 行的问题，代码如下：

```
a=1;
while(a<=5)
{
    printf("\n");
    a=a+1;
}
```

但是如何使每行星号的个数不同呢？回想一下我们之前打印过 3 行且每行有 5 个星号的代码，其中 `while b` 循环的作用是在每一行上面打印 5 个星号，所以变量 `b` 是从 1 递增到 5 同时每次都打印 5 个星号。可是现在的要求变了，每行不都是 5 个星号，而是第一行 1 个星号，第二行 2 个星号……我们这里只需将 `while b` 循环的条件改一下，不再是 `b<=5`，改为 `b<=a` 就可以了（`b` 的初始值不变，仍然是 1）。`while a` 循环中的变量 `a` 是用来控制每一行的，变量 `a` 等于 1 时就是第一行，打印 1 个星号；变量 `a` 等于 2 时就是第二行，打印 2 个星号，所以变量 `a` 的值恰好就是这行所需要的星号数，代码如下：

```
a=1;
while(a<=5)
{
    b=1;
    while(b<=a)
    {
        printf("*");
        b=b+1;
    }
    printf("\n");
    a=a+1;
}
```

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    a=1;
    while(a<=5)
    {
        b=1;
        while(b<=a)
        {
            printf("*");
            b=b+1;
        }
        printf("\n");
        a=a+1;
    }
    system("pause");
    return 0;
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

➔ 更进一步，动手试一试

1. 请尝试用 `while` 循环打印下面的图形：

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

2. 请尝试用 `while` 循环打印下面的图形：

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

第 7 节 逻辑挑战 5：奔跑的字母



之前我们已经学习了如何通过 `while` 循环，并结合暂停命令 `Sleep` 和清屏幕命令 `system("cls")` 来做“倒计时”程序，本节我们将通过这些命令写一个“奔跑的字母”的程序。

首先我们想一下，如果希望一个字母（假设这个字母是 `H`）从屏幕的左边往右边跑，即第一秒时字母 `H` 在屏幕的第一行的最左边（也就是第一行第一列），第二秒时字母 `H` 在屏幕第一行的第二列，第三秒时字母 `H` 在屏幕第一行的第三列，以此类推，该怎么实现呢？

我们知道，如果直接使用 `printf("H");`，字母 `H` 就会出现在屏幕的第一行第一列，即最靠近右上角的位置。那如何让字母 `H` 在屏幕的第一行第二列呢？我们可以用“空格”来占位。也就是说，在输出时先输出一个空格，再输出字母 `H`，即 `printf(" H");`（在 `H` 左边加一个空格来填充第一列，这样 `H` 就会出现在第二列）。同样，如果希望字母 `H` 在第一行第三列的话，只需在输出时，在 `H` 左边多加两个空格就可以了，即 `printf(" H");`，好了，我们来尝试一下。

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



```

#include <stdlib.h>
#include <windows.h>
int main()
{
    system("cls");
    printf("H");
    Sleep(1000);

    system("cls");
    printf(" H");
    Sleep(1000);

    system("cls");
    printf("  H");
    system("pause");
    return 0;
}

```

怎么样？字母 H 是不是从左边向右边移动了 3 步。用这种方法，我们也可以让字母移动 50 步，但是如果像上面这样写，是不是太麻烦了，我们需要复制粘贴 50 次，然后每一次都要修改 `printf` 语句中字母 H 前面空格的个数，真是太麻烦了。

我们仔细分析一下上面这段代码，有 3 个部分基本上相同，只有 `printf` 语句中字母 H 前面的“空格”的个数不同，在第 1 部分字母 H 前面有 0 个空格，在第 2 部分字母 H 前面有 1 个空格，在第 3 部分字母 H 前面有 2 个空格。我们便想到了用 `while` 循环解决这个问题。

首先，仔细观察之前的代码你就会发现，其中有 3 段代码是差不多的。我们可以用 `while` 循环 3 次来解决重复的问题，代码如下：

```

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a;
    a=0;
    while(a<=2)
    {
        system("cls");
        printf("H");
        Sleep(1000);
        a=a+1;
    }
    return 0;
}

```

```
}
```

运行一下你会发现，字母 H 并没有向右移动。这是为什么呢？因为在上面的 while 循环中，虽然循环了 3 遍，但是每次循环输出的都是 `printf("H");`，字母 H 的左边并没有空格，所以字母 H 并没有向右边跑。把 `printf("H");`改为 `printf(" H");`也不行，那样每次输出的都是字母 H 在第一行第二列的位置，字母 H 会一直停留在第一行第二列，不会往右边跑。我们需要解决的是，在循环第一次时 H 在第一列，即 H 前面有 0 个空格；循环第二次时 H 在第二列，即 H 前面有 1 个空格；循环第三次时 H 在第三列，即 H 前面有 2 个空格。

我们发现每次循环空格的变换规律是 0、1、2，这恰好和变量 a 的变化规律是一样的。第 1 次循环时变量 a 的值为 0，第 2 次循环时变量 a 的值为 1，第 3 次循环时变量 a 的值为 2。也就是说每次循环时，在打印字母“H”前，打印 a 个空格就可以了。可是如何使每次循环输出 a 个空格呢？这里我们需要用到 while 循环的嵌套。代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a,b;
    a=0;
    while(a<=2)
    {
        system("cls");
        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }

        printf("H");
        Sleep(1000);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

在上面这段代码中，我们利用 while a 循环来控制字母 H 一共需要走多少步，利用 while b 循环来控制字母 H 每走一步需要在字母 H 前面打印多少个空格。

下面我们来仔细分析一下上面这段代码。

计算机自顶向下一步步执行：

```
首先 a 的初始值为 0
a<=2 成立，进入外循环
    清屏
    b 的初始值被赋为 1
    b<=a 不成立（此时 a 为 0，b 为 1），不进入内循环，不会打印空格
    打印字母 H
    暂停 1 秒
    a=a+1（a 从 0 变为 1）
外循环末尾，跳转到外循环的开始部分，重新判断 a<=2 是否成立
a<=2 成立（此时 a 为 1），继续进入外循环
    清屏
    b 的初始值被赋为 1
    b<=a 成立（此时 a 为 1，b 为 1），进入内循环
        打印空格
        b=b+1（b 从 1 变为 2）
    内循环末尾，跳转到内循环的开始部分，重新判断 b<=a 是否成立
    发现此时 b<=a 不成立，（此时 a 为 1，b 为 2），退出内循环
    打印字母 H
    暂停 1 秒
    a=a+1（a 从 1 变为 2）
外循环末尾，跳转到外循环的开始部分，重新判断 a<=2 是否成立
a<=2 成立（此时 a 为 2），继续进入外循环：
    清屏幕
    b 的初始值被赋为 1
    b<=a 成立（此时 a 为 2，b 为 1），进入内循环
        打印空格
        b=b+1（b 从 1 变为 2）
    内循环末尾，跳转到内循环的开始部分，重新判断 b<=a 是否成立
    此时 b<=a 成立，（此时 a 为 2，b 为 2），再次进入内循环
        打印空格
        b=b+1（b 从 2 变为 3）
    内循环末尾，跳转到内循环的开始部分，重新判断 b<=a 是否成立
    发现此时 b<=a 不成立，（此时 a 为 2，b 为 3），退出内循环
    打印字母 H
    暂停 1 秒
    a=a+1（a 从 2 变为 3）
外循环末尾，跳转到外循环的开始部分，重新判断 a<=2 是否成立
此时 a<=2 不成立（此时 a 为 3），退出外循环
```

第 8 节 究竟循环了多少次

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    a=1;
    while(a<=2)
    {
        b=1;
        while(b<=3)
        {
            printf("ok ");
            b=b+1;
        }
        a=a+1;
    }
    system("pause");
    return 0;
}
```

猜猜看，计算机执行上面这段代码后，会打印多少次“OK”？

6次！为什么计算机打印6次“OK”？

我们仔细分析一下上面的代码，会发现有两个 while 循环，即 while a 循环和 while b 循环，并且 while b 循环嵌套在 while a 循环里面。

这里 while a 循环每循环一次，while b 循环就会被完整地从头到尾执行一遍（循环3次，打印3个“OK”）。这里的 while a 循环会循环2次，所以 while b 循环就会被完整地执行两遍（每遍打印3个“OK”）。所以，一共会打印出6个“OK”，我们可以这样计算循环的次数： $2 \times 3 = 6$ 。

我们再来看看下面这段代码循环了多少次：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b;
    a=1;
    while(a<=4)
    {
        b=1;
        while(b<=2)
        {
```

```
        printf("ok ");
        b=b+1;
    }
    a=a+1;
}
system("pause");
return 0;
}
```

实验过后你会发现计算机一共打印出了 8 个“OK”，我们可以这样计算循环次数 $4 \times 2 = 8$ 。再来看看更复杂的，代码如下：

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,b,c;
    a=1;
    while(a<=2)
    {
        b=1;
        while(b<=4)
        {
            c=1;
            while(c<=3)
            {
                printf("ok ");
                c=c+1;
            }
            b=b+1;
        }
        a=a+1;
    }
    system("pause");
    return 0;
}
```

上面这段代码，有 3 层循环嵌套，while a 循环中嵌套了 while b 循环，while b 循环中又嵌套了 while c 循环。while a 循环会循环 2 次，while b 循环会循环 4 次，while c 循环会循环 3 次。也就是说，while a 循环每循环 1 次，while b 循环就会循环 4 次，while b 循环每循环 1 次，while c 循环就会循环 3 次，所以一共循环了 $2 \times 4 \times 3 = 24$ 次，打印了 24 个“OK”。

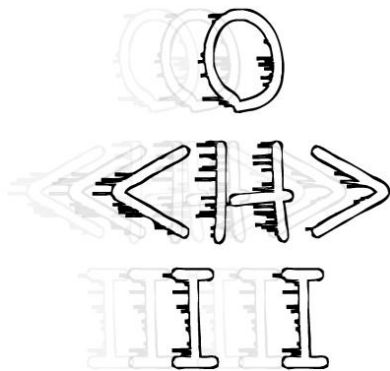
啊哈 C！思考快你一步——用编程轻松提升逻辑力

➔ 更进一步，动手试一试

1. 请问下面这段代码会打印多少个“OK”？

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    int i,j;
    i=1;
    while(i<=10)
    {
        j=1;
        while(j<=i)
        {
            printf("OK ");
            j=j+1;
        }
        i=i+1;
    }
    system("pause");
    return 0;
}
```

第 9 节 逻辑挑战 6：奔跑的小人



在本章第 7 节我们学会了如何让字母奔跑起来，本节我们将在“奔跑的字母”基础上，让一个小人奔跑起来。而且我们还可以控制小人奔跑的速度。

首先我们来设计这个小入：

```
O
<H>
I I
```

将这个小人分为 3 行来分别表示 3 个部分：

第 1 行用一个大写字母 **O** 表示小人的脑袋；

第 2 行用左尖括号 **<** 表示小人的左手，用大写字母 **H** 表示小人的身体，用右尖括号 **>** 表示小人的右手；

第 3 行用两个大写字母 **I** 表示小人的两条腿，为了对称，两个大写字母 **I** 之间用一个空格隔开。

代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    printf(" O\n");
    printf("<H>\n");
    printf("I I\n");
    system("pause");
    return 0;
}
```

现在我们让小人动起来。首先回顾一下让字母奔跑起来的代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a,b;
    a=0;
    while(a<=2)
    {
        system("cls");

        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }
    }
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
        printf("H");
        Sleep(1000);
        a=a+1;
    }
    system("pause");
    return 0;
}
```

我们把上面代码中的

```
printf("H");
```

改为：

```
printf(" O\n");
printf("<H>\n");
printf("I I\n");
```

完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a,b;
    a=0;
    while(a<=2)
    {
        system("cls");

        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }

        printf(" O\n");
        printf("<H>\n");
        printf("I I\n");

        Sleep(1000);
        a=a+1;
    }
    return 0;
}
```


运行后你会发现，只有小人的脑袋往右边移动，身体和腿呆在原地，这是为什么？

分析后我们发现，让小人往右移动主要通过在小人的左边不停地打印空格来实现。但是我们只在第 1 行的左边打印了空格，在第 2 行和第 3 行都没有打印空格的语句。因此我们要将打印空格的 while 循环再复制一遍分别放在 `printf("<H>\n");`和 `printf("I I\n");`前面，完整的代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
int main()
{
    int a,b;
    a=0;
    while(a<=2)
    {
        system("cls");

        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }
        printf(" O\n");

        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }
        printf("<H>\n");

        b=1;
        while(b<=a)
        {
            printf(" ");
            b=b+1;
        }
        printf("I I\n");

        Sleep(1000);
        a=a+1;
    }
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

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

怎么样，小人是不是奔跑起来啦！

如果希望小人跑得更远，我们只需把 `while(a<=2)` 改为 `while(a<=80)`。如果让小人跑得更快一点，我们之前已经学习过，只需把 `Sleep(1000)` 改为较小的值就可以了，越小越快，例如，改为 `Sleep(100)`。赶快试一试吧。

➔ 更进一步，动手试一试

1. 你可以设计一个“小人”并让它从右边向左边跑吗？

第 10 节 for 隆重登场

通过之前的学习，如果要让计算机做重复的事情，我们可以使用 `while` 循环。本节介绍另一种循环——`for` 循环。有时它要比 `while` 循环使用起来更加方便。

首先我们来回顾一下，如果让计算机从 1 循环到 10，并且把 1~10 都打印出来，用 `while` 循环的写法是这样的：

```
#include <stdio.h>  
#include <stdlib.h>  
int main()  
{  
    int a;  
    a=1;  
    while(a<=10)  
    {  
        printf("%d ",a);  
        a=a+1;  
    }  
    system("pause");  
    return 0;  
}
```

上面代码中的 `while` 循环，分为 3 部分，控制 `while` 循环从 1 到 10 执行 10 次。

第 1 部分：设置 `a` 的初始值为 1，即 `a=1`；

第 2 部分：设置循环条件，即 `a<=10`；

第 3 部分：`a` 每次增加 1，即 `a=a+1`。

上面 3 部分的共同作用使得让 `while` 循环从 1 到 10 循环了 10 次。如果上面 3 部分忘记写任

一部分,根据我的经验,很多同学都会忘记写 `a=1;`或 `a=a+1;`。这样 `while` 循环就不能正常运行了。因为这 3 部分在 3 个不同的地方,确实容易让人漏写。不要紧,粗心的我们可以使用 `for` 循环来解决这个问题。我们来看一看,用 `for` 循环如何解决让计算机从 1 循环到 10,并且把 1 到 10 都打印出来的问题。代码如下:

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

    for(a=1;a<=10;a=a+1)
    {
        printf("%d ",a);
    }
    system("pause");
    return 0;
}
```

你发现了没有, `for` 循环后面的括号,把 `while` 循环的 3 部分统统写了进去,并且用分号隔开,请注意只有两个分号,最后的 `a=a+1` 后面没有分号,它表达的意思仍然是从 1 循环到 10。这样写起来是不是方便很多。现在我们只需看 `for` 后面括号内的 3 个式子就可以知道,这个循环是从几开始,到几结束,每次增加几。

另外说一下, `a=a+1` 可以简写为 `a++`。其他简写方法我们将在第 5 章进一步说明。因此上面的 `for` 循环:

```
for(a=1;a<=10;a=a+1)
```

可以简写为:

```
for(a=1;a<=10;a++)
```

同样,我们也可以利用 `for` 循环来实现 1~100 中所有数的求和,代码如下:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a,sum;
    sum=0;
    for(a=1;a<=100;a++)
    {
        sum=sum+a;
    }
}
```

啊哈 C！思考快你一步——用编程轻松提升逻辑力

```
printf("%d",sum);
system("pause");
return 0;
}
```

打印 1~100 中所有偶数的代码如下：

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

    for(a=2;a<=100;a=a+2)
    {
        printf("%d ",a);
    }

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

用 for 循环输出 1~100 中所有 7 的倍数或者末尾含 7 的数，代码如下：

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

    for(a=1;a<=100;a++)
    {
        if(a%7==0 || a%10==7)
            printf("%d ",a);
    }
    system("pause");
    return 0;
}
```

怎么样，for 循环是不是比 while 循环要简洁很多。很多同学可能要问，既然 for 循环要比 while 循环好，为什么还要学习 while 循环呢？其实，在控制已知循环次数时，例如，需要循环 10 次或者 1000 次，for 循环确实要比 while 循环好使。但是，并不是在任何情况下 for 循环都要优于 while 循环，还要看我们对循环的需求。随着编程学习的慢慢深入，你会了解什么时候该用 for 循环，什么时候该用 while 循环。其实还有一种 do-while 循环，这里不再做介绍，有兴趣的

后续内容还请关注

[网站首页] <http://www.ahalei.com>

[网站问答] <http://www.ahalei.com/qa>

[新浪微博] <http://weibo.com/ahalei>

[人人主页] <http://page.renren.com/601196462>

如果您有任何建议，可以在 bbs.ahalei.com 上留言

书写匆忙，欢迎批评纠错，感谢支持