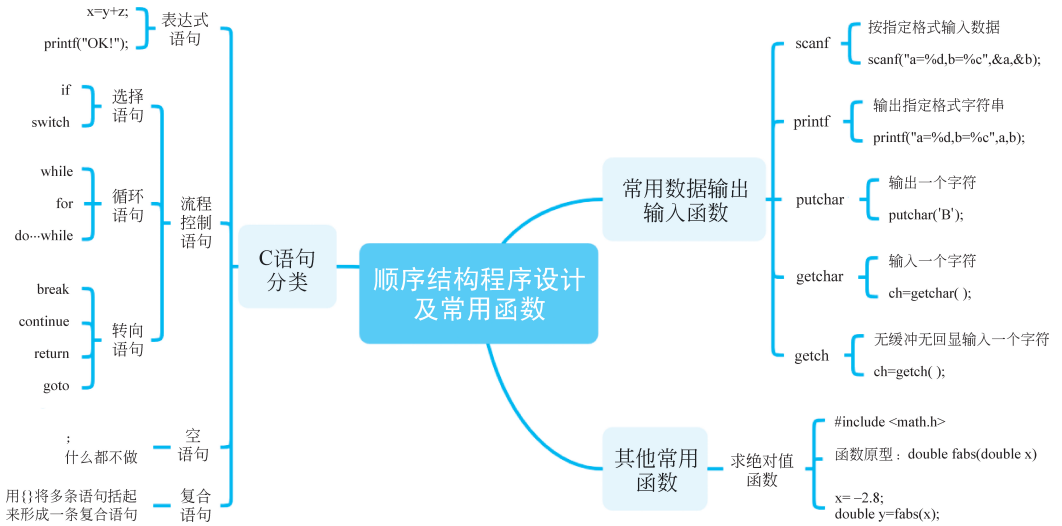
第3章 顺序结构程序设计及常用函数



用程序解决简单问题一般需要3步：①输入数据；②处理数据；③输出数据。例如，小程是一家销售瓷砖的网上商铺老板，需要经常根据用户提供的铺设瓷砖的面积和选择瓷砖的尺寸，计算需要瓷砖的块数。

这是一个小学生就能解决的问题，假设铺设面积用area表示，单位为平方米，瓷砖的长用length表示，宽用width表示，单位为厘米。则瓷砖块数tiles＝area×100÷(length×width)。编写程序解决这一实际需求可以分为3步。

(1) 输入数据。通过键盘输入area、length、width的值。

(2) 处理数据。根据上述公式求出瓷砖的块数。

(3) 输出数据。将瓷砖块数输出到屏幕。

确定总体思路后，还需要确定程序的若干细节问题。

(1) area等变量的值如何从键盘输入，tiles中的值如何输出到屏幕上呢？

(2) 根据公式计算出来的瓷砖块数一般带有小数点，如 32.6 块，而实际需要 33 块，如何实现向上取整呢？

本章主要介绍顺序结构、C 语句分类、输入以及常用函数的有关知识，运用本章的知识，读者可以编写出顺序结构的人机交互小程序。

扫一扫

3.1 顺序结构

如果用计算机程序解决某一个问题或完成某一任务，需要若干条语句配合完成，而反映语句之间的这种配合和执行顺序关系的就是程序流程控制问题。从程序流程控制视角看，程序结构可分为 3 种基本结构：顺序结构、选择结构和循环结构。无论功能多复杂的程序，皆是由这 3 种基本结构中的一种或几种灵活组合而成。

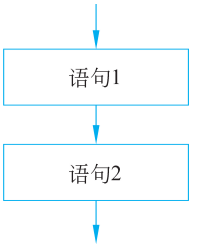


图 3-1 顺序结构流程图

本章介绍的顺序结构是其中最简单也是最常用的一种结构。顺序结构是指按照语句在程序中出现的先后顺序逐条执行，每条语句必须执行且只执行一次。其流程图如图 3-1 所示，即语句 1 执行完毕后再执行语句 2。

注意：顺序结构中的某些语句执行顺序影响程序执行结果，不可以交换。例如，计算瓷砖数的案例中，按(2)→(1)→(3) 顺序执行，会得到错误结果，因为计算 tiles 的值时 area、length、width 还没有输入数据，当然 tiles 无法得到正确结果。但某些语句交换后不影响程序结果，读者可以根据个人偏好调整，例如，area、length、width 3 个变量的赋值顺序可以调整，只要使用时保证已经赋值即可。

扫一扫

3.2 C 语句分类

一个 C 语言程序由一个或多个函数组成，而函数主要由语句组成。因此函数是 C 语言程序的基本组成单位，而语句是最小单位。根据语句的功能，C 语言中的语句可以分为以下 4 类。

(1) 表达式语句。表达式语句的突出特征是以分号“;”结尾。任何一个合法的表达式后面跟上一个分号就可以构成一条表达式语句，即其一般格式如下：

```
表达式;
```

表达式由运算符和运算数组合而成，其中运算符可以是 C 语言中任何的算术、关系、赋值、逗号、指针等运算符，运算数可以是变量、常量、宏定义、函数调用等形式，只要组成的表达式合法即可。其中，最典型、最常用的是赋值表达式和函数调用语句。

例如：赋值表达式语句

```
x=y+z;
```

是由算术运算符“+”和赋值运算符“=”以及 x、y、z 3 个变量组成的算术、赋值混合表达式语句，其作用为计算 $y+z$ 的值，然后赋给变量 x。因为最后进行的是赋值操作，可以简称为赋值表达式语句。

例如：函数调用语句

```
printf("Try my best!");
```

是由一个函数调用“printf("Try my best!")”和一个分号组成，其作用是调用 printf 函数，在屏幕上输出“Try my best!”。

当然，任何合法的表达式加分号都构成表达式语句，但不一定有实际意义，应避免使用。例如：

```
y+z;        //只计算出 y+z 的值，但计算结果没有保留，也没有输出，无实际意义
z=z;        //把 z 变量的值又赋给 z，对程序功能没起任何作用，无实际意义
```

(2) 流程控制语句。在 C 语言中，除了顺序结构通过出现顺序控制之外，其他流程均需要通过特定的语句定义符控制。C 语言中的流程控制语句共有 3 大类 9 小种。

① 选择语句：if 语句、switch 语句。

② 循环语句：while 语句、for 语句和 do…while 语句。

③ 转向语句：break 语句、continue 语句、return 语句、goto 语句。

(3) 空语句。空语句是一条只有一个分号“;”构成的语句。它其实是一个表达式语句，只不过表达式为空，所以什么也不做，一般用来占位、作被转向点或空循环体。例如：

```
for(i=0;(a[i]=getchar())!='\n';i++)
    ;       //注意循环体为空语句
```

该程序段的功能为从键盘上输入一串字符放到数组中，按回车键结束。因为循环必须有循环体，因此该空语句必不可少。当然，此处的空语句并不是空循环体的唯一表达方式，换成一条空复合语句，即一对花括号“{}”也可以。

虽然什么也不做，但空语句毕竟是一条语句，不能在程序中随便加，否则可能会造成程序语法或者逻辑错误。例如：

```
int count=0;
while((ch=getchar())!='\n')
    count++;
```

此程序段的功能为计算输入字符的个数。但如果写成：

```
int count=0;
while((ch=getchar())!='\n');
    count++;
```

则会发现无论输入多少个字符，count 中的值都为 1。这是因为 while 循环的循环体变成了空语句“;”，而“count++;”变成了循环结束后才能执行到的顺序结构语句，只执行一次。

(4) 复合语句。前面的一条表达式语句、空语句或流程控制语句均可统称为一条单语句,而将 $n(n\geqslant 0)$ 条语句用一对花括号“{}”括起来则称为复合语句。例如:

```
{
    temp=max;
    max=min;
    min=temp;
}
```

就是一条复合语句,功能为借助中间变量 temp 交换变量 max、min 的值。

如果去掉两端的花括号,则是三条单语句,但功能不变。既然花括号加与不加功能不变,为什么要画蛇添足地加呢?因为在某些场景中,比如选择或循环语句,要求分支体或者循环体只能是一条语句,所以,当分支体或者循环体是多条语句时,则可以加花括号将多条语句转换为一条复合语句。例如:

```
if (max<min)
{
    temp=max;
    max=min;
    min=temp;
}
```

功能为只有在满足 max 小于 min 的情况下才会交换 max、min 的值,也就是始终保持 max 大于或等于 min。此时若去掉花括号,则受 if 控制的分支体就只有单语句“temp=max;”,程序就会产生逻辑错误。

【例 3-1】 编程实现,将一个 24 小时制时间转换为 12 小时制时间。例如,输入:13:15,则输出:下午 1:15;输入:8:20,则输出:上午 8:20。

程序分析: 时间由时和分两部分组成,转换为 12 小时制时只有时变,而分不用变,因此将时和分放在两个整型变量中。

```
#include <stdio.h>                                  //文件包含命令,不是语句,所以没有分号
int main()
{
    int hour,minute;                                //变量声明语句
    printf("输入 24 小时制时间:(小时:分钟)\n");    //函数调用语句,输出提示信息
    scanf("%d:%d",&hour,&minute);                   //函数调用语句,从键盘输入时分,用:分隔
    if (hour<=12)
        printf("上午 %d:%d\n",hour,minute);
    else                                            <-- if 语句
        {                                           <-- 一条复合语句
            hour=hour-12;//表达式语句
            printf("下午 %d:%d\n",hour,minute);
        }
}
```

3.3 常用数据输出输入函数

输入与输出是程序与用户之间的一种交互方式，一个程序至少有 0 个输入和 1 个输出。C 语言通过调用标准函数库中的输入、输出函数来实现数据的输入、输出操作。因为输入设备可能是标准输入设备键盘，也可能是非标准输入设备，如文件、扫描仪等，输出设备可能是标准输出设备屏幕，也可能是非标准输出设备，如文件、打印机等，因此在标准函数库中提供了针对不同设备、不同读写方式的多个函数，这些函数的定义一般在头文件 stdio.h(stdio 是 standard input & output 的缩写)中。因此，如果程序中要使用它们，则需要在程序的开头加上预编译命令：

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

C 语言中的输出输入函数都是成对的，即有一个某功能的输入函数，就有一个相同功能的输出函数。例如，scanf 和 printf 是一对，都是格式化输入输出函数；getchar 和 putchar 是一对，都是字符输入输出函数。一对函数在格式、用法等方面有很多雷同点，因此可以对比着记忆、学习。

此章要介绍的输入函数皆为标准输入函数，即输入设备为键盘；输出函数皆为标准输出函数，即输出设备为屏幕。输入和输出设备为文件的若干函数将在文件一章中介绍。

3.3.1 格式输出函数 printf


扫一扫

1. printf 函数调用方法

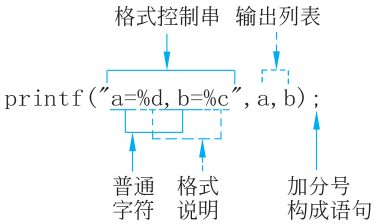
格式输出函数 printf 的功能是按指定格式将字符串输出到标准输出设备——屏幕上。其中，函数名中的 f 就是 format(格式)的缩写。printf 函数是应用最广、功能最强大的输出函数，因此使用语法也比较琐碎，初学者只需掌握基本输出方式即可，不必死抠细节，尤其是繁杂的格式说明项，用时查阅即可。

printf 函数的函数原型为：

```
int printf(const char * format[,argument…])
```

其中，参数 format 称为格式控制串，参数 argument 后面的“…”表示此参数的个数不确定，所以 argument 称为输出列表，“[]”表示参数 argument 为可选项，可有可无。

例如：



（1）格式控制串。format 前面的数据类型“const char *”说明此处应是一个字符型指针，常为一个用双引号引起来的常量字符串，功能为指定输出数据项的类型和格式，可以包含两种信息。

① 格式说明项，由%和格式说明符组成，作用是将输出列表中对应的数据按照指定格式输出。如例中的%d 说明按十进制整型格式输出，%c 说明按字符格式输出，但具体输出值需要在输出列表中指定。

② 普通字符，即除格式说明项以外的字符，会原样输出，在输出结果中起提示作用。如例中的“a=”和“,b=”，尤其注意“,b=”中的“,”也是普通字符。

普通字符一般采用字符常量形式，特殊情况也可以采用转义字符形式，例如：

```
printf("a\b*");
```

输出结果为：*。因为“\b”为退格键的转义字符形式，输出 a 以后会倒退一格，然后输出 *，所以 * 就会把 a 覆盖掉。

（2）输出列表。输出列表是需要在格式控制串指定位置输出的一些表达式集合，之间用逗号隔开。第 i 个表达式的值会按照第 i 个格式说明项所指定格式，代替第 i 个格式说明项输出。所以表达式的个数一定要和格式说明项的个数保持一致，否则程序会出现意想不到的逻辑错误。

在上例中，假如变量 a、b 均为整型变量，且值均为 65，则输出结果为：

```
a=65,b=A
```

即第一个表达式的值代替第一个格式说明项，以整数形式 65 输出；第二个表达式的值代替第二个格式说明项，以字符形式 A 输出（字符 A 对应的 ASCII 码为 65）。除此之外，其他均为普通字符，原样输出。

（3）函数返回值。printf 函数如果输出成功，则返回所输出字符的个数，否则返回一个负数。但最常用的是 printf 单独构成语句，因此极少使用其返回值。

2. 格式说明项

格式说明项的一般形式为：

```
%「标志」「最小宽度」「.精度」「长度」格式字符
```

由一般形式可以看出，格式说明项必须以%开头，以格式字符结尾，中间是若干可选项，根据需要选择输出宽度、精度等控制信息。

（1）格式字符：用于指定输出数据的类型，常用格式字符及其功能描述如表 3-1 所示。同一个功能可能会有多个格式字符，如输出十进制整数，d、i 均可，输出一个字符，c、C 均可。但为了提高程序可读性和减少记忆负担，建议初学者统一使用第一个小写字符。

（2）标志：标志字符有－、＋、＃、空格、0 共 5 种，其功能描述如表 3-2 所示。

（3）最小宽度：用十进制整数表示所输出数据占据的最少位数。如果实际位数多于定义宽度，则按实际位数输出，若实际位数少于定义宽度，则补以空格或 0。

（4）精度：如果输出数据为实数，则表示小数的位数；如果输出的是字符串，则表示输

出字符的个数。

表 3-1 常用格式字符及其功能描述

数据类型	格式字符	功 能 描 述
int	d 或 i	以十进制形式输出带符号整数(默认正数不输出符号)
	o	以八进制形式输出无符号整数(默认不输出前缀 0)
	x 或 X	以十六进制形式输出无符号整数(默认不输出前缀 0x 或 0X)
	u	以十进制形式输出无符号整数
char	c 或 C	输出一个字符
	s	输出一个字符串
float double	f	以小数形式输出单、双精度带符号实数,默认小数位数为 6 位
	e 或 E	以指数形式输出单、双精度实数
	g 或 G	以%f 和%e 中较短的输出宽度输出单、双精度实数
其他	%	输出符号%

表 3-2 标志字符功能描述

标志字符	功 能 描 述
-	表示输出结果左对齐,若结果不够指定宽度,则右边补充空格;默认为右对齐
+	输出数据前冠以符号(正号或负号);默认只有负数前有符号
#	加在格式字符 o(八进制)前,则输出结果加前缀 o;加在格式字符 x 或 X(十六进制)前,则输出结果加前缀 0x 或 0X
空格	输出数据为正时,前面冠以空格
0	若实际位数少于定义宽度,则左边补 0,但与'-'一起使用时不起作用,此时右边填充空格

(5) 长度:长度格式符有 h 和 l 两种,一般用于整型格式字符前,h 表示按短整型输出;l 表示按长整型输出,例如,ld 表示按长整型输出,lld 按长长整型输出。l 如果用于 f 前,即 lf 则表示按 double 型输出。

【例 3-2】 printf 格式输出综合测试程序。

```
#include <stdio.h>
int main()
{
   int datai=66;
   char datac='a';
   float dataf=138.3576278;        //精度超出 float 的 7 位,会有误差
   double datad=138.3576278;       //精度在 double 的范围内,没有误差
   //以不同进制形式输出 datai 的值 66
   printf("十进制: %d,八进制: %o,十六进制: %x\n",datai,datai,datai);
   printf("datai=%+d,-datai=%+d\n",datai,-datai); //输出带符号整数
   printf("datai=%-5d,%5d\n",datai,datai);        //按指定宽度 5,左、右对齐输出
   printf("datac=%c,ASCII=%d\n",datac,datac);     //按字符和整数输出字符型数据
   printf("dataf=%f,dataf=%10.4f\n",dataf,dataf); //按指定宽度与精度输出实数
   printf("datad=%lf\n",datad);                   //输出双精度实数
}
```

程序运行结果为：(//后面为对结果的解释说明)

```
十进制：66,八进制：102,十六进制：42
datai=+66,-datai=-66     //%+d是指定数据无论正负都输出符号,%d只是负数才输出符号
datai=66□□□,□□□66       //%-5d为左对齐,%5d为右对齐,不足用空格补齐
datac=a,ASCII=97         //字母a的ASCII码为97
dataf=138.357620,dataf=□□138.3576     //%10.4f指总体宽度10,精度4,小数点占1位
datad=138.357628         //%ld输出double型数据
```

学完此小节，可以解决计算瓷砖数案例中的问题“tiles中的值如何输出到屏幕上”，一种解决方案为：

```
printf("tiles=%d",tiles);
```

注意：语句中的两个tiles含义完全不同，第一个tiles是格式控制串中的普通字符，输出时原样输出；第二个tiles是变量名，表示其值代替%d输出。


扫一扫

3.3.2 格式输入函数 scanf

1. scanf 函数调用方法

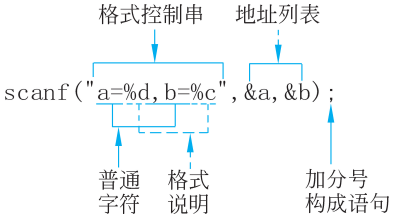
scanf函数是与printf函数相对应的一个标准输入库函数，其功能为按指定格式从键盘输入数据，并存放到指定地址的内存中。同样，scanf函数为了解决复杂的数据输入，其语法复杂、琐碎，稍不留神就出错，因此建议初学者使用最简单的输入方式，不需酷炫效果。

scanf函数的函数原型为：

```
int scanf(const char * format[,argument…])
```

scanf和printf的函数原型非常相似，参数format仍为格式控制串，多为常量字符串。参数argument同样可有可无，但此处为地址列表。

例如：



(1) 格式控制串。用于指定数据的接收类型和格式，同样包括格式说明项和普通字符两种信息。格式说明项用于指定接收类型，如上例中的%d指定按一个整数接收数据，%c指定按一个字符接收数据。普通字符用于指定接收格式，“a=”和“,b=”需原样输入。

(2) 地址列表。用于指定存放输入数据的内存地址。具体形式多种多样，如变量地址，数组地址或指针变量，甚至是三者组成的表达式，只要结果是一个地址就可以。如上例中，要把输入数据存放在变量 a 中，则需指定变量 a 的地址：&a。

上例中“scanf("a=%d,b=%c",&a,&b);”，格式控制串为字符串常量“a=%d,b=%c”，说明输入时“a=”“,b=”要原样输入，而%d和%c的位置分别用一个整数和一个

字符代替。假设输入信息如下(CR 表示回车键 Enter):

```
a=36,b=r<CR>
```

则运行结果为:36,被存放在变量 *a* 中,字符'r'被存放在变量 *b* 中。

特别提醒:使用 scanf 输入数据时,一定要和指定的格式保持一致,否则变量无法接收到正确数据,但又不会提示任何错误信息,因此会造成程序逻辑错误。例如,上例中如果输入"36r<CR>""36□r<CR>"或"a=36b=r<CR>"等,均不能正确接收。提醒初学者用最简单的输入语句即可。

函数返回值。scanf 函数的返回值为成功接收数据的项数,出错时则返回 EOF,即-1。

【例 3-3】 根据函数返回值判断正确接收数据的个数。

```
#include <stdio.h>
int main()
{
    int a,b,number;
    number=scanf("a=%d,b=%c",&a,&b);        //将 scanf 返回值赋给 number
    printf("正确接收数据数: %d\n",number);
}
```

程序运行结果为:

```
输入: a=36,b=r<CR>
输出: 正确接收数据数: 2
```

输入格式与指定格式完全一致,a、b 均正确接收。

```
输入: a=36b=r<CR>
输出: 正确接收数据数: 1
```

因为 a=36 与指定格式一致,*a* 能正确接收,但"b=r<CR>"因为 *b* 前少了一个",",与指定格式不一致,*b* 不能正确接收。

```
输入: 36r<CR>
输出: 正确接收数据数: 0
```

因为 *a*、*b* 均不能正确接收。

2. 格式说明

格式说明项的一般形式为:

```
%「*」「宽度」「长度」格式字符
```

scanf 函数和 printf 函数的格式说明项极其相似,仍然必须以%开头,以格式字符结尾,但中间控制信息比较简单,尤其注意没有精度信息,说明输入时不能指定输入数据的精度,例如,假设变量 *c* 为 float 类型:

```
scanf("3.1%f",&c);
```

这样的语句无论输入什么数据，变量 c 都无法正确接收到，而程序又不会提示任何错误信息，因此一定避免使用。

(1) 格式字符：用于指定输入数据的数据类型，常用的部分格式字符和功能描述如表 3-3 所示。建议初学者统一使用第一个小写字符。

表 3-3 常用格式字符功能描述表

数据类型	格式字符	功能描述
int	d 或 D 或 i 或 I	输入十进制整数
	o	输入八进制整数
	x 或 X	输入十六进制整数
	u 或 U	输入无符号十进制整数
char	c 或 C	输入一个字符
	s	输入一个字符串
float	f 或 e 或 E	输入实型数(用小数形式或指数形式)

【例 3-4】 输入一个字符，输出其 ASCII 码值。

```
#include <stdio.h>
int main()
{
    char c;
    printf("请输入一个字符: ");
    scanf("%c",&c);          //输入一个字符,存放到变量 c 中
    //输出列表中有两个 c,第一个按字符输出,第二个按整型输出
    printf("%c 的 ASCII 为: %d\n",c,c);
}
```

举一反三：

① 输入一个整数，输出对应的字符。

② 输入一个十或八进制数，输出其对应的十六进制数。

(2) “*”符：表示跳过对应的输入数据。例如：

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

当输入如下信息时：

```
30□15□6<CR>
```

系统会将 30 赋给 a，6 赋给 b，而 15 被跳过，不赋给任何变量。

【例 3-5】 用 * 忽略整数和字符之间的间隔符。

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

```
    int age;
    char sex;
    printf("请输入年龄和性别(M或F),空格分开: ");
    scanf("%d%*c%c",&age,&sex);        //忽略中间间隔符
    printf("年龄是: %d,性别是: %c\n",age,sex);
}
```

程序运行结果为:

```
输入: 18□F<CR>
输出: 年龄是: 18,性别是: F
```

输入数据时,我们习惯性地将两个数据用间隔符(空格、Tab键和回车键)分开,但用%c接收数据时,间隔符也会当正常字符一样被接收。例3-5中的输入语句如果写成:

```
scanf("%d%c",&age,&sex);
同样输入: 18□F<CR>
```

则sex会接收空格(□),而不是F。因此常常使用类似"%*c"的形式忽略间隔符。

(3) 宽度:用十进制正整数指定输入数据的宽度。例如:

```
scanf("%3d%2d",&a,&b);
输入: 123456<CR>
```

则123赋给变量a,45赋给变量b,最后的6会留在缓冲区中,用于下一个输入。

```
输入: 1234<CR>
```

则123赋给a,4赋给b。即使输入数据不够所需位数,遇到回车也终止接收。

```
输入: 1<CR>2345<CR>
```

则1赋给a,23赋给b,45留在缓冲区。

【例3-6】 日期格式转换:输入日期20220520,转换为2022年5月20日。

```
#include <stdio.h>
int main()
{
    int year,month,day;
    printf("请按样例格式输入一个日期: 20220520\n");
    scanf("%4d%2d%2d",&year,&month,&day);        //按位数截取年月日
    printf("%d年%d月%d日\n",year,month,day);
}
```

程序运行结果为:

```
输入: 20220520
输出: 2022年5月20日
```

(4) 长度：长度格式符有 l 和 h 两种，l 表示输入长整型数据(如%ld，%lo，%lx)或双精度浮点数(如%lf，%le)；h 表示输入短整型数据(如%hd，%ho，%hx)。

3. 使用 scanf 函数时需注意的问题

(1) 特别注意 scanf 函数中的第二个参数必须是地址列表。

【例 3-7】 地址列表错误示例。

```
#include <stdio.h>
int main()
{
    int a=0;
    scanf("%d",a);          //a 前缺少 &
}
```

则程序会出现运行错误，程序终止运行。Dev-C++ 中的错误提示如图 3-2 所示。

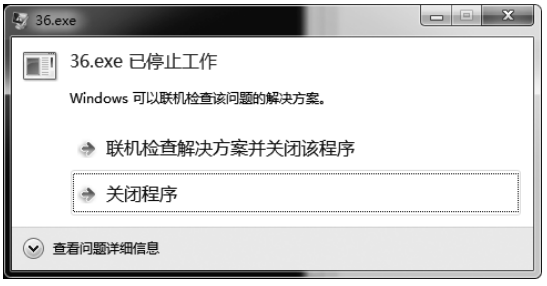


图 3-2 缺少 & 时的运行错误提示信息

单击“联机检查解决方案并关闭该程序”选项，则会发现程序的返回值为 3221225477，如图 3-3 所示。

图 3-3 返回错误编码图

此处的错误编码 3221225477(0xC0000005)表示访问越界。访问越界是指访问了不属于自己的内存空间。内存就像国土一样，绝不允许非法占用。应用程序运行时，操作系统要为其变量分配内存空间，只有分配给该应用程序的内存可以随意存取数据，但绝对不可以向其他内存中存放数据。而此例中，数据本来应该存放到首地址为 &a，即为 a 分配的内存中去，结果却存放到了首地址为 a，即地址为 0 的内存中去，而首地址为 a 的空间并不属于该应用程序，造成了越界访问。

(2) 当使用 scanf 函数从键盘输入数据完毕后，一定要按下回车键，函数才会接收到数据。其实 scanf 函数，以及后面要介绍的 getchar 函数，都是缓冲输入函数，即并不是直接接

收键盘数据，而是将用户输入的数据暂时存放在缓冲区中，当用户键入回车或缓冲区满了之后，scanf 函数才会从缓冲区中依次读取数据。如果缓冲区中的数据已经读取完，但仍需数据，则函数会等待输入，直到满足函数要求或遇到非法数据为止；如果输入的数据多于函数要求输入的数据，则多余的数据将留在缓冲区中，供下一个输入使用。

（3）当 scanf 中的两个相邻格式说明项紧密相连时，一般遇到以下三种情况之一，则认为当前数据结束：

① 遇到间隔符（空格、Tab 键和回车键）。

② 遇到宽度限制，如%3d，只取 3 列。

③ 遇到非法数据。

例如：

```
scanf("%d%c",&a,&b);
输入：12c34<CR>
```

则会把 12 赋给变量 a，把字符 c 赋给变量 b，而 34 留在缓冲区中。

```
输入：12□c34<CR>
```

则会把 12 赋给变量 a，把字符空格□赋给变量 b，而 c34 留在缓冲区中。因为间隔符也会被看作一个字符，正常接收。因此在 C 语言中不建议%c 与其他格式符连用，否则特别容易出现莫名其妙的错误。

例如：

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

输入：12□34＜CR＞，12＜Tab＞34＜CR＞，12＜CR＞34＜CR＞，三种方式均能正确接收，即 a 接收 12，b 接收 34。

```
输入：12c34<CR>
```

则 a 接收 12，因为字母 c 对于整数是非法数据，因此认为数据结束，这样 b 就无法接收到值，即保持原值不变。

（4）如果格式控制串中有普通字符串，即有多个紧密相连的普通字符，则输入数据时严格按照“**普通字符串前不加间隔符**”（即如果格式控制串为：普通字符串＋格式说明项，则普通字符串和数据之间可以加入若干间隔符；如果格式控制串为：格式说明项＋普通字符串，则数据和普通字符串之间不能加入任何间隔符）的原则输入即可，除非格式说明项中有宽度限制。

例如：

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

使用该语句把 46 接收到变量 a 中，把 18 接收到变量 b 中，则下面几种输入方式均合法（其中□代表空格字符）：

① a＝46，b＝18＜CR＞：不加任何间隔符，严格遵循输入格式。

② a=□46,b=18<CR>：普通字符串 a=和数据 46 间加入若干个空格间隔符。

③ a=<Tab>46,b=□18<CR>：普通字符串 a=和数据 46 间加入若干个 Tab 间隔符。

④ a=46,b=18□<CR>：数据 18 后面加入若干个空格、Tab 等间隔符。

而下面两种情况则不能正确接收：

① <Tab>a=46,b=18<CR>：普通字符串 a=前加入 Tab 间隔符，违背输入原则。

② a=46,□b=18<CR>：在普通字符串“,b=”中插入一个空格，导致与普通字符串不匹配。

(5) 用%c 输入字符时，间隔符都会作为有效字符输入，而不再起间隔数据的作用。如：

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

如果输入：c□d<CR>，则变量 *a* 接收字符 c，变量 *b* 接收字符空格，字符 d 留在缓冲区中。

如果输入：c<CR>，则变量 *a* 接收字符 c，变量 *b* 接收字符回车。

学完此小节，可以解决计算瓷砖数案例中的问题“area 等变量的值如何从键盘输入”。一种解决方案为：

```
scanf("%f%f%f",&area,&length,&width);
```

第二种解决方案为：

```
scanf("area=%f,length=%f,width=%f",&area,&length,&width);
```

注意：这两种解决方案在具体输入时一定严格按照规定格式输入，否则无法接收到正确数据。一定不要忘记各个变量前的 &，否则程序会出现运行错误。


扫一扫

3.3.3 单字符输出函数 putchar

putchar 是一个功能单一、使用简单的字符输出函数，其函数原型为：

```
int putchar(int ch)
```

功能：向标准输出设备(显示器)输出一个字符。

(1) 参数 ch 表示要输出的字符，字符应该为 char 类型，为什么此处为 int 呢？在计算机中，字符其实是以其对应 ASCII 码值形式存在的，char 类型数据的存储、运算都会隐式转换为 int 类型。因此，此处的实际参数具体形式可能千变万化，但本质是一个整数，只要表达式最终能转换为一个整数即可。

【例 3-8】 putchar 函数参数测试程序。

```
#include <stdio.h>          //把头文件 stdio.h 包括到文件中
int main()
{
    char a='A';             //定义字符型变量
    int b=65;               //定义整型变量,65 是字母 A 的 ASCII 码值
```

```
    putchar(a+1);          //参数为字符型表达式,可自动隐式转换为整型。输出字母 B
    putchar(b+1);          //参数为整型表达式,输出字母 B
    putchar('B');          //参数为字符型常量,可自动隐式转换为整型。输出字母 B
    putchar('\102');       //参数为转义字符,注意转义符中的 102 是八进制数
    putchar(7);            //有 ASCII 码的不可见字符也可输出,7 为一次响铃。
}
```

程序输出结果为：

```
BBBB<响铃>
```

本程序除响铃语句外，每条输出语句的输出结果相同，均为字母 B，但具体表达形式不同，以期为读者提供正确的使用参考。

特别提醒： 虽然参数 ch 为 int 类型，但 ASCII 码只有 256 个（除附录 A 中的 128 个外还有 128 个扩展字符）。因此，当 ch 的值超过 255 时，会自动进行取余运算：ch%256，使 ch 始终在正确范围内。因此 putchar(263)和 putchar(7)的功能是一样的。

(2) 函数返回值类型为整型，如果输出成功，则返回输出字符的 ASCII 码值，即参数 ch；若输出失败，则返回 EOF(−1)。

putchar(ch)与 printf("%c",ch)的功能等效，都是输出一个字符，经常替换使用，但返回值含义不同：putchar 返回的是输出字符的 ASCII 码值，而 printf 返回的是输出字符的个数，此处永远为 1。

3.3.4 单字符输入函数 getchar


扫一扫

getchar 函数与 putchar 函数是一对，同样功能单一，使用简单，其函数原型为：

```
int getchar()
```

功能：从标准输入设备(键盘)上输入一个字符。

getchar 函数是一个无参函数，如果正确接收，则函数的返回值为输入字符的 ASCII 码值，否则返回 EOF(−1)。

【例 3-9】 输入一个字符，输出它的后继字符。

程序分析： 一个字符的后继字符是指其在 ASCII 码表中的后一个字符，如字母 A 的后继字符是 B，d 的后继字符是 e。一个字符和其后继字符之间的 ASCII 码值差 1。

```
#include <stdio.h>
int main()
{
    char ch;
    ch=getchar();                              //接收一个字符
    printf("%c 的后继字符是: %c\n",ch,ch+1); //输出其后继字符
}
```

使用 getchar 函数时需要注意以下问题：

(1) getchar 函数是一个缓冲输入函数，输入字符后必须按回车键才能接收数据。

（2）如果接收的是一个数字，也会按照字符处理。例如程序输入：1<CR>，则程序运行结果为“1 的后继字符是：2”。即接收的是 1 这个数字的 ASCII 码值，即 49。

（3）空格、Tab 键以及回车键等都会作为一个 getchar 函数能接收的有效字符。例如程序输入：<CR>，则程序运行结果如图 3-4 所示。

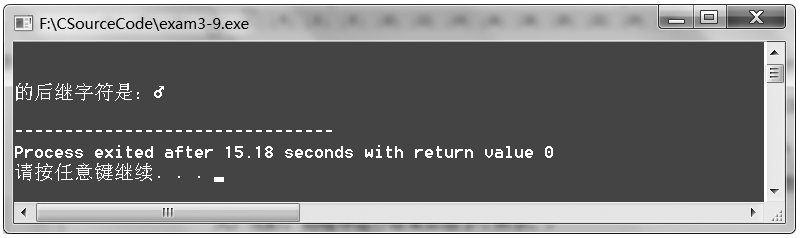


图 3-4　接收回车键时的程序结果

（4）如果输入多于一个字符，则只接收第一个字符。例如程序输入：32<CR>，则程序运行结果为“3 的后继字符是：4”，另一个字符 2 将留在缓冲区中。

例 3-9 中的 ch=getchar()与 scanf("%c",&ch)的功能基本等效，都是输入一个字符放在变量 ch 中，区别在于：scanf 的返回值为成功接收数据的项数，此处为 1 或 0，而 getchar 的返回值为接收字符的 ASCII 码值。在某些应用中，用 getchar 实现的程序代码更简洁，可读性更强，如 5.1 节中的例 5-2，6.4.2 节中的例 6-15 等，读者学习到相关内容时再深入体会即可。


扫一扫

3.3.5　不回显输入函数 getch

getch 不是一个标准库函数，它位于 conio.h 中。conio 是 Console Input/Output（控制台输入输出）的简写，其中定义了通过控制台进行数据输入和数据输出的一些函数。conio.h 是 Windows 系统特有的，其他操作系统中不一定提供，因此可移植性较差。

函数原型为：

```
int getch()
```

从函数原型看，getch 与 getchar 的调用形式完全相同，功能也类似，都是从标准输入设备（键盘）上输入一个字符，但主要有以下两点区别。

（1）getch 为不回显函数，即用户输入的字符只默默接收，但不显示在屏幕上。

（2）getch 为非缓冲输入函数，即有输入时立即接收，不需要按回车键。

getch 经常应用于以下两种场景中。

（1）实现暂停功能，解决闪退或闪过问题。在有些开发环境中运行 C 语言程序或者在操作系统中直接执行 C 语言生成的.exe 文件时，会出现一闪就退出的现象，无法看清执行结果；或者程序输出数据过多，一些有用信息一闪而过。这时都可以加一个 getch 语句，留住当前界面，实现“按任意键继续”功能。

（2）实现密码输入功能。用户输入密码时一般不显示真正的输入字符，而用特殊符号代替。

【例 3-10】　编程实现，利用 getch 实现一位密码输入功能。

```
#include <stdio.h>
#include <conio.h>          //包含相应头文件
int main()
{
    int ch;
    printf("请输入一位密码: ");
    ch=getch();             //接收字符,但屏幕不显示
    putchar('*');           //屏幕输出替代字符
    printf("\n您输入的密码是: %c",ch);
}
```

程序运行结果如图 3-5 所示。

图 3-5 程序运行结果图

3.4 其他常用函数

为更便捷、更高效地使用 C 语言,标准函数库中提供了众多函数,读者只需使用 #include 命令将其对应的头文件包含到程序中,然后按照函数原型说明使用即可。函数原型就像一份函数使用说明书,让读者清楚函数需要几个参数,参数的数据类型是什么,函数执行完毕后会返回一个什么数据类型的值。

本节内容初学时只需浏览框架,后期用到时再返回本节仔细阅读即可。

3.4.1 常用数学函数


扫一扫

数学函数对应的头文件是 math.h。

(1) 求双精度浮点数绝对值函数 fabs(求整数绝对值函数为 abs)。

函数原型: double fabs(double x)。

函数功能: 求 x 的绝对值。

通过函数原型可以看出,fabs 函数需要一个 double 类型的参数,实际应用时,参数可以是一个常量、变量或者表达式,只要最终结果值的数据类型为或者能隐式转换为 double 即可。函数返回值为 double 类型,说明其结果可以参与 double 类型数据能参与的一切运算。

【例 3-11】 编程实现: 求数轴上两点之间的距离。

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

```
int main()
{
    double x=2.3,y=8.9,distance;                    //x、y为数轴上的两个点
      /*将x-y的值,即-6.6传递给函数fabs;因为fabs函数的返回值类型为double,因此
        需要定义一个double类型的变量distance存放其返回值*/
    distance=fabs(x-y);
    printf("distance=%lf\n",distance);
}
```

等价于

```
    printf("distance=%lf\n",fabs(x-y));          //注意输出时一定用%lf
```

(2) 求平方根函数 sqrt。

函数原型：double sqrt(double x)。

函数功能：计算 x 的平方根。

【例 3-12】 编程实现：求平面中两点之间的距离。

```
#include <stdio.h>
#include <math.h>
int main()
{
    float x1,y1,x2,y2,distance;                          //点1(x1,y1) 点2(x2,y2)
    printf("请输入第一个点：空格分开");
    scanf("%f%f",&x1,&y1);
    printf("请输入第二个点：空格分开");
    scanf("%f%f",&x2,&y2);
    distance=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));       //距离公式
    printf("distance=%f\n",distance);
}
```

(3) 上取整函数 ceil。

函数原型：double ceil(double x)。

函数功能：求不小于 x 的最小整数，即上取整。

【例 3-13】 编程实现：根据铺设瓷砖的面积(平方米)，所选择瓷砖的尺寸(厘米×厘米)，计算需要瓷砖的块数。

```
#include <stdio.h>
#include <math.h>
int main()
{
    float area,length,width;
    int tiles;                  //瓷砖块数,整型变量
    printf("请输入铺设面积(平方米):") ;
    scanf("%f",&area);
    printf("请输入瓷砖的长度和宽度(厘米):");
    scanf("%f%f",&length,&width);
    //ceil返回值为double型,赋给整型变量tiles时,最好加一个强制类型转换
    tiles=(int)ceil(area*100/(length*width));
    printf("大约需要瓷砖块数: %d",tiles);
}
```

(4) 求方函数 pow。

函数原型：double pow(double x,double y)。

函数功能：计算 x 的 y 次方的值。

【例 3-14】 编程实现：计算 n 位二进制位能表示的最大整数。

```
#include <stdio.h>
#include <math.h>
int main()
{
    int n,maxint;
    printf("请输入二进制位的位数：");
    scanf("%d",&n);
    maxint=(int)pow(2,n-1)-1;        //n位二进制位能表示的最大整数 2^(n-1)-1
    printf("%d位能表示的最大整数为：%d",n,maxint);
}
```

例如，整型变量占 4B，32 位，程序执行结果为：

```
输入：32
输出：32位能表示的最大整数为：2147483647
```

3.4.2 常用字符函数


扫一扫

字符函数对应的头文件为 ctype.h。

(1) 小写字母转大写函数 toupper。

函数原型：int toupper(int c)。

函数功能：如果 c 有对应的大写字母，则返回大写字母，否则返回原值。

(2) 大写字母转小写函数 tolower。

函数原型：int tolower(int c)。

函数功能：如果 c 有对应的小写字母，则返回小写字母，否则返回原值。

【例 3-15】 编程实现：如果一个字符是小写字母，则转换为大写字母，否则保持不变。

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    char ch;
    printf("请输入一个小写字母：");
    ch=getchar();
    putchar(toupper(ch));
}
```

程序执行结果为：

```
输入：a      //输入是小写字母
输出：A      //则转换为大写字母
输入：3      //输入不是小写字母
输出：3      //则保持不变
```

(3) 检查字母函数 isalpha。

函数原型：int isalpha(int c)。

函数功能：判断字符 c 是否为英文字母，若是，则返回非 0(小写字母为 2，大写字母为 1)，否则返回 0。

例如：isalpha('*')的值为 0，isalpha('E')的值为 1，而 isalpha('e')的值为 0。


扫一扫

3.4.3 其他常用工具函数

(1) 获取系统时间函数 time，包含在头文件 time.h 中。

函数原型：time_t time(time_t *t)。

函数功能：获取计算机的当前日历时间。日历时间是指从一个时间点(一般是 1970 年 1 月 1 日 0 时 0 分 0 秒)到现在的秒数。

函数中用到的数据类型 time_t，实质是整型，为了提高程序的通用性，一般将 32 位编译器中的 long(长整型)，将 64 位编译器中的__int64(即长长整型)取别名为 time_t。

函数获取时间的方式既可以通过参数 t，也可以通过返回值，甚至参数和返回值同时获取也可以，但重复数据也没有实际意义。建议读者采用返回值形式，此时参数设置为空(NULL)即可。

【例 3-16】 编程实现：获取自时间纪元至今的秒数和年数。

```
#include <stdio.h>
#include <time.h>
int main()
{
    time_t curtime;          //定义接收当前时间的变量，也可以定义为 long 型
    curtime=time(NULL);      //用返回值形式返回日历时间，参数为 NULL
    printf("时间纪元已经飞逝了%ld 秒，约%ld 年",curtime,curtime/(24 * 60 * 60 *
365));
}
```

因为当前时间在不断变化，所以程序的运行结果中的秒数每次均不同。

(2) 产生一个伪随机数函数 rand，包含在头文件 stdlib.h 中。

函数原型：int rand()。

函数功能：产生一个范围在 0～RAND_MAX 的伪随机整数，其中 RAND_MAX 是一个符号常量，在 stdlib.h 中可以找到其定义“#define RAND_MAX 0x7fff”。0x7fff 就是 32767，即短整型的最大值，因此产生的就是一个短整型非负数。

(3) 随机数生成器初始化函数 srand，包含在头文件 stdlib.h 中。

函数原型：void srand(unsigned seed)。

函数功能：为随机数生成器提供一粒新的随机种子。

如果随机种子相同，rand 产生的随机数也是相同的。为真正达到每次产生的随机数不同，需要在调用 rand 之前用 srand 设置不同的随机种子。一般采用 time 函数产生的系统时间作为随机种子，因为系统时间精确到秒，变化很快，基本能满足程序多次执行时种子不同的需求。

【例 3-17】 编程实现：随机产生一个三位整数。