计算机程序设计基础(1)
--- C语言程序设计(3)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193
2022. 9.

第3章 数据的输入与输出

3.1 格式输出函数

3.2 格式输入函数

3.3 字符输出函数

3.4 字符输入函数

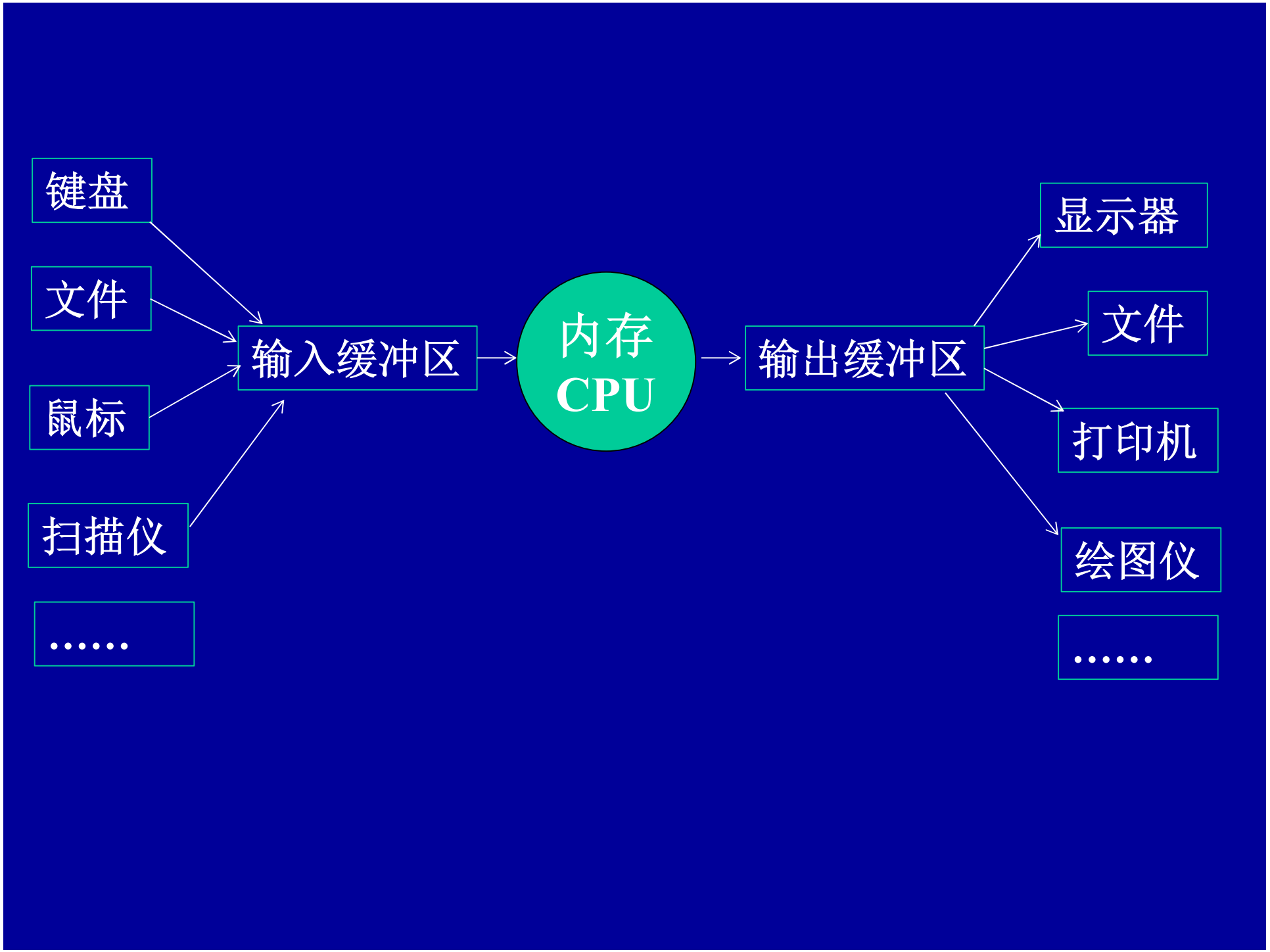
数据的输入输出函数

把数据从计算机内部送到计算机外部设备上的操作称为“输出”。例如把计算机运算结果显示在屏幕上或打印在纸上，或者送到磁盘上保留起来。从计算机外部设备（键盘、磁盘、固盘、鼠标、扫描仪、USB设备等等）将数据送入计算机内部的操作称为“输入”。

C语言本身并没有提供输入输出语句。但可以通过调用标准库函数提供的 *输入和输出函数* 来实现输入和输出。C语言提供了丰富的用于输入和输出的库函数。

输入输出过程往往伴随着数据格式转换过程：

整数 ⇌ 补码， 浮点数 ⇌ **IEEE 754 double/float**


键盘
文件
鼠标
扫描仪
……
输入缓冲区
内存
CPU
输出缓冲区
显示器
文件
打印机
绘图仪
……

数据的输入输出函数

数据的输入与输出应包括以下几项：

- 用于输入或输出的设备
- 输入或输出数据的格式
- 输入或输出的具体内容

在C语言中，提供了用于输入与输出的函数，在这些函数中，键盘是标准输入设备(stdin)，显示器是标准输出设备(stdout)。

另外要注意，如果在程序中要使用C语言所提供的输入函数或输出函数，则在使用前（即在每个文件的开头）应该使用包含(引用)命令，将C语言中进行输入与输出的库函数和常量变量等说明的(头)文件包含进来：

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

3.1 格式输出函数

3.1.1 基本的格式输出语句

- **printf ("格式控制", 输出列表);**

格式控制部分要用一对双引号括起来，它用于说明输出项目所采用的格式。输出列表中的各项目指出了所要输出的内容，格式是：输出项1, 输出项2, …输出项n 。在格式控制中，用于说明输出数据格式的格式说明符总是以%开头，后面紧跟的是具体的格式。

格式说明符与输出列表中的量是一一对应的，类型要一致，个数应该相同。

用于输出的常用格式说明符有以下几种：

1.整型格式说明符

- 十进制形式

%d 或 %md	用于int整型
%ld 或 %mld	用于long长整型
%hd 或 %mhd	用于short短整型
%u 或 %mu	用于无符号int整型
%lu 或 %mlu	用于无符号long长整型

- 八进制形式

%o 或 %mo	用于int整型
%lo 或 %mlo	用于long长整型

- 十六进制形式

%x 或 %mx	用于int整型
%lx 或 %mlx	用于long长整型

m表示输出的整型数据所占总宽度（即列数），当实际数据的位数不到m位时，数据前面将用空格补满。如果在格式说明符中没有用m来说明数据所占的宽度，则以输出数据的实际位数为准。如果在格式说明符中说明了宽度m，但实际输出的数据位数大于m，则以输出数据的实际位数进行输出，自动突破场宽限制。

另外：%lld，%llo，%llx 用于超长整数long long的十进制/八进制/十六进制输出。

2. 实型格式说明符

- 十进制数形式 %f 或 %m.nf
- 指数形式 %e 或 %m.ne

  %E 或 %m.nE

在输出实型数据时，格式说明符中的m表示整个数据所占的宽度，n表示小数点后面所占的位数。如果省略了m和n，那么%f, %e或%E都将输出6位小数（不包括小数点），不足补0。

如果在小数点后取n位后，所规定的数据宽度m不够输出数据前面的整数部分（包括小数点），则按实际的位数进行输出，自动突破场宽限制。

需要指出的是，在C语言中，用于输出单精度实型数据与双精度实型数据格式说明符是一样的， %f即可 。

3. 字符型格式说明符

- 格式说明符为%c 或 %mc

  其中m表示输出的宽度，即在这种情况下，在输出字符的前将会补m-1个空格。

下面对各种基本类型数据的格式输出作几点说明：

- 输出列表中可以有多个输出项目，但各输出项目之间要用“,”分隔。各输出项目可以是常量、变量以及表达式。

- 格式输出函数中的“格式控制”是一个字符串，其中每一个%后面的字符是格式说明符，用于说明相应输出数据的输出格式，而每一个格式说明符的结束符分别为d（整型）、f（实型）、c（字符型）、s（字符串）等。而格式控制中除格式说明符外的其他字符将按原样输出。

【例3-1】设有以下程序：

```
#include  <stdio.h>
main( )
{ int  a, b;
  float  x, y, s;
  a=34;   b= -56;
  x=2.5f;   y=4.51f;   s=x*x+y*y;
  printf("a=%d,b=%d\n",  a, b);
  printf("x=%6.2f,y=%6.2f,s=%6.2f\n",  x, y, s);
}
```

这个程序经编译链接后，运行输出的结果为(□表示一个空格)：

a=34,b=-56

x=□□2.50,y=□□4.51,s=□26.59

如把【例3-1】的程序改为：

```
#include  <stdio.h>
main( )
{ int  a, b;
  float  x,  y,  s;
  a=34;  b= -56;
  x=2.5;  y=4.51;  s=x*x+y*y;
  printf("a=%d,b=%d\n",  a,  b);
  printf("x=%6.2f,y=%6.2f,s=%6.2f\n",  x,  y,  s);
}
```

这个程序编译链接结果为:

ex3_1.c

ex3_1.c(6) : warning C4305: “=”: 从“double”到“float”截断

出现warning错误的原因是:

C语言默认所有浮点常数是double类型，以便以可能的最大精度存储数据。

因此2.5和4.51都会以double格式存储，当赋值给x, y这些float型变量时，需要把double类型数据截断（truncation）成float类型，从15位有效数字截断为7位有效数字，有精度损失。因此编译系统给出警告信息提醒(如果没有误差不出现警告信息)。

为了防止出现警告信息，语句改为:

```
x = 2.5f; y = 4.51f; /* 2.5，4.51以float格式存储 */
```

或： `x = (float)2.5; y = (float)4.51; /* 强制转换成float类型 */`

而将【例3-1】程序改为：

```
#include  <stdio.h>
main( )
{ int  a, b;
  double x, y, s;     /*  float 改为 double */
  a=34; b= -56;
  x=2.5; y=4.51; s=x*x+y*y;
  printf("a=%d,b=%d\n", a, b);
  printf("x=%6.2f,y=%6.2f,s=%6.2f\n", x, y, s);
}
```

程序编译链接将不会出现任何错误信息。

- 格式输出函数的执行过程如下：

首先，在计算机内存中开辟一个输出缓冲区，用于存放输出项目表中各项目数据。

然后，依次计算项目表中各项目（常量或变量或表达式）的值，并按各项目数据类型应占的字节数依次将它们存入输出缓冲区中。

最后，根据“格式控制”字符串中的各格式说明符依次从输出缓冲区中取出若干字节的数据（如果是非格式说明符，则将按原字符输出），转换成对应的十进制或指定的进制数据进行输出。其中从输出缓冲区中取多少个字节的数据是按照对应格式说明符说明的数据类型。

- 在“格式控制”的格式说明符中，如果带有宽度说明，则在左边没有数字的位置上用空格填满(即输出的数字是右对齐)。但如果在宽度说明前加一 个负号(-)，则输出为左对齐，即在右边补空格。

【例3-2】 设有如下C程序：

```
#include  <stdio.h>
main( )
{ int xx,  yy,  zz;
  xx=1;  yy=-65535;  zz=1;
  printf("xx=%ld, yy=%ld, zz=%ld\n", xx, yy, zz);
  printf("xx=%hd, yy=%hd, zz=%hd\n", xx, yy, zz);
  printf("xx=%d, yy=%d, zz=%d\n", xx, yy, zz);
}
```

该程序运行的结果如下：

```
xx=1, yy=-65535, zz=1
xx=1, yy=1, zz=1
xx=1, yy=-65535, zz=1
```

【例3-3】 设有如下C程序：

```
#include <stdio.h>
main( )
{ double  x=34.567;
  printf("x=%f\n", x);
  printf("x=%d\n", x);
  printf("x=%d\n", (int)x);
}
```

这个程序的实际运行结果为：

```
x=34.567000
x=1958505087
x=34
```

注意：强制转化时不四舍五入

显然，这个程序中的第二个格式输出语句输出的结果是错误的，这是因为在第二个格式输出语句中，格式说明符%d是基本整型格式说明符，而输出项目是双精度型的数据，它们是不匹配的。

如果将例3.1中的程序改成如下：

```
#include  <stdio.h>
main( )
{ int  a, b;
  float  x, y, s;
  a=34; b= -56;
  x=2.5f; y=4.51f; s=x*x+y*y;
  printf("a=%d, b=%d\n", a, b);
  printf("x=%-6.2f, y=%-6.2f, s=%-6.2f\n", x, y, s);
}
```

则这个程序经编译链接后，运行输出的结果为

```
a=34, b=-56
x=2.50□□, y=4.51□□, s=26.59□
```

即在输出的宽度范围内，因为左对齐，空格补在数据的后面。

3.1.2 printf函数中常用的格式说明

格式控制中，每个格式说明都必须用“%”开头，以一个格式字符作为结束，在此之间可以根据需要插入“宽度说明”、左对齐符号“-”、前导零符号“0”等。

1. 格式字符

%后允许使用的格式字符和它们的功能如表3.1所示。在某些系统中，可能不允许使用大写字母的格式字符。因此为了使程序具有通用性，在写程序时，尽量不用大写字母的格式字符。

2. 长度修饰符

在%和格式字符之间，可以加入长度修饰符，以保证数据输出格式的正确和对齐。对于长整型数（long）应该加l，如%ld。对于短整型数（short）可以加h，如%hd。对于超长整型数（long long）应该加ll，如%lld。

格式字符	说　　明
c	输出一个字符
d或i	输出带符号的十进制整型数，%ld为长整型，%hd为短整型，%lld为超长整型
o	以八进制格式输出整型数，%o不带先导0；%#o加先导0
x或X	以十六进制格式输出整型数，但不带先导0x或0X。%#x或%#X输出带先导0x或0X的十六进制数
u	以无符号十进制形式输出整型数
f	以带小数点的数学形式输出浮点数（单精度和双精度数）
e或E	以指数形式输出浮点数(单精度和双精度数)，格式是:[-]m.dddddd[e/E]±xxx。小数位数(d的个数)由输出精度决定，隐含是6。若指定的精度为0，则连小数点在内的小数部分都不输出
g或G	在满足输出精度的情况下，由系统决定用%f还是用%e(或%E)输出,未指定输出精度则默认6位有效数字
s	输出一个字符串，直到遇到"\0"。若字符串长度超过指定的精度则自动突破，不会截断字符串
p	输出变量的内存地址
%	也就是%%形式，输出一个%

3. 输出数据所占的宽度说明

当使用%d、%c、%f、%e、%s、...的格式说明时，输出数据所占的宽度（域宽）由系统决定，通常按照数据本身的实际宽度输出，前后不会自动加空格，并采用右对齐的形式。但可以用以下3种方法人为控制输出数据所占的宽度（域宽），按照使用者的意愿进行输出。

- （1）在%和格式字符之间插入一个整数常数来指定输出的宽度n（例如%4d，n代表整数4）。
  - 如果指定的宽度n不够，输出时将会自动突破，保证数据完整输出。
  - 如果指定的宽度n超过输出数据的实际宽度，输出时将会右对齐，左边补以空格，达到指定的宽度。
- （2）对于float和double类型的实数，可以用“n1.n2”的形式来指定输出宽度（n1和n2分别是一个常整数），其中n1指定输出数据的宽度（包括小数点），n2指定小数点后小数位的位数，n2也称为精度（例如%12.4f，n1代表整数12，n2代表整数4）。

- 对于f、e或E，当输出数据的小数位多于n2位时，截去右边多余的小数，并对截去部分的第一位小数做四舍五入处理。
- 当输出数据的小数位少于n2时，在小数的最右边补0，使得输出数据的小数部分宽度为n2。
- 若给出的总宽度n1小于n2加上整数位数和小数点（e或E格式还要加上指数的5位），则自动突破n1的限制；反之，数字右对齐，左边补空格。
- 也可以用“.n2”格式（例如%.6f），不指定总宽度，仅指定小数部分的输出位数，由系统自动突破，按照实际宽度输出。
- 如果指定“n1.0”或“.0”格式（例如%12.0f或%.0f），则不输出小数点和小数部分。

- 对于g或G，可以用%m.ng或%m.nG的形式来输出（m和n分别是一个常整数），其中m指定输出数据的宽度（包括小数点），n2指定输出的有效数字位数。
  - 若宽度超过数字的有效数字位数，则左边自动补空格；若宽度不足，则自动突破。
  - 若用%g、%G、%mg、%mG的形式不指定输出的有效数字位数，将自动按照最多6位有效数字输出，截去右边多余的小数，并对截去部分的第一位小数做四舍五入处理。
  - 若指定或默认的输出有效数字位数超过实际数字的有效位数，则把实际数字原样输出。
- （3）对于整型数，若输出格式是“0n1”或“.n2”格式（例如%05d或%.5d），如果指定的宽度超过输出数据的实际宽度，输出时将会右对齐，左边补以0。

- （4）对于字符串，格式“n1”指定字符串的输出宽度：
  - 若n1小于字符串的实际长度，则自动突破，输出整个字符串；
  - 若n1大于字符串的实际长度，则右对齐，左边补空格。
  - 若用“.n2”格式指定字符串的输出宽度，则若n2小于字符串的实际长度，将只输出字符串的前n2个字符(将字符串截断！)。
- 注意：输出数据的实际精度并不完全取决与格式控制中的域宽和小数的域宽，而是取决于数据在计算机内的存储精度。通常系统只能保证float类型有7位有效数字，double类型有15位有效数字。若你指定的域宽和小数的域宽超过相应类型数据的有效数字，输出的多余数字是没有意义的，只是系统用来填充域宽凑数而已。
- 4. 输出数据左对齐

由于输出数据都隐含右对齐，如果想左对齐，可以在格式控制中的“%”和宽度之间加一个“-”号来实现。

5. 使输出数据总带+号或-号

- 通常输出数据，如果是负数，前面有符号位“-”，但正数的“+”都省略了。如果要每一个数前面都带正负号，可以在%和格式字符间加一个“+”号来实现。

- 举例：若k为int型，值为1234，f为double型，值为123.456。下表列举了各种输出宽度和不指定宽度情况下的输出结果。

```
int k=1234;
double f =123.456;
```

输出语句	输出结果
printf("%d\n", k);	1234
printf("%6d\n", k);	□□1234
printf("%2d\n", k);	1234
printf("%f\n", f);	123.456000
printf("%12f\n", f);	□□123.456000
printf("%12.6f\n", f);	□□123.456000
printf("%2.6f\n", f);	123.456000
printf("%.6f\n", f);	123.456000
printf("%12.2f\n", f);	□□□□□123.46
printf("%12.0f\n", f);	□□□□□□□□123
printf("%.f\n", f);	123
printf("%e\n", f);	1.234560e+002

printf("%13e\n", f);	1.234560e+002
printf("%13.8e\n", f);	1.23456000e+002
printf("%3.8e\n", f);	1.23456000e+002
printf("%.8e\n", f);	1.23456000e+002
printf("%13.2e\n", f);	□□□1.23e+002
printf("%13.0e\n", f);	□□□□□□1e+002
printf("%.0e\n", f);	1e+002
printf("%06d\n", k);	001234
printf("%.6d\n", k);	001234
printf("%012.6f\n", f);	00123.456000
printf("%013.2e\n", f);	00001.23e+002
printf("%s\n", "abcdefg");	abcdefg
printf("%10s\n", "abcdefg");	□□□abcdefg
printf("%5s\n", "abcdefg");	abcdefg
printf("%.5s\n", "abcdefg");	abcde

printf("%g\n", 123.4);	123.4
printf("%g\n", 123.4567);	123.457
printf("%5g\n", 123.4567);	123.457
printf("%10g\n", 123.4567);	□□□123.457
printf("%g\n", 1234567.89);	1.23457e+006
printf("%5g\n", 1234567.89);	1.23457e+006
printf("%10.8g\n", 1234567.89);	□1234567.9
printf("%10.7G\n", 1234567.89);	□□□1234568
printf("%10.5G\n", 1234567.89);	1.2346E+006

printf("%-6d\n", k);	1234□□
printf("%-12.2f\n", f);	123.46□□□□□
printf("%-13.2e\n", f);	1.23e+002□□□
printf("%+-6d%+-12.2f\n", k, -f);	+1234□-123.46□□□□
printf("%4.1f%%\n", 12.5);	12.5%

3.1.3 使用printf函数时的注意事项

1．printf的输出格式为自由格式，是否在两个数之间留逗号、空格或回车，完全取决于你的格式控制，如果不注意，很容易造成数字连在一起，使得输出结果没有意义。例如：printf("%d%d%f\n", k, k, f); 语句的输出结果是：

12341234123.456000，无法分辨其中的数字含义。而如果改为printf("%d %d %f\n", k, k, f);输出结果是：

1234 1234 123.456000　　看起来就一目了然。

2．格式控制中必须含有与输出项一一相对应的输出格式说明，类型必须匹配。若格式说明与输出项的类型不一一对应匹配，则不能正确输出。而且编译时不会报错。若格式说明个数少于输出项个数，则多余的输出项不予输出；若格式说明个数多于输出项个数，则将输出一些毫无意义的数字乱码。

3．在格式控制中，除了前面要求的输出格式，还可以包含任意的合法字符（包括汉字和转义符），还可利用'\n'（回车）、'\r'（回行但不回车）、'\t'（制表）、'\a'（响铃）等控制格式。这些字符输出时将“原样照印”。

4．如果要输出%符号，可以在格式控制中用%%表示，将输出一个%符号。

5．printf函数有返回值，返回值是本次调用输出字符的个数，包括回车等控制符。

6．尽量不要在输出语句中改变输出变量的值，因为可能会造成输出结果的不确定性。这被称为C语言国际标准中的undefined behavior(未定义行为)，也就是说C语言国际标准中并没有规定一定要怎么做，由各家的C编译器自行确定。

例如：int k=8; printf("%d,%d\n", k, ++k); 输出结果不是你预想的8,9，而是9,9。这是因为调用函数printf时，其参数是从右至左进行传递的，将先进行++k运算。而实际上，VS2008编译器会把输出语句 printf("%d,%d\n", ++k, ++k); 的两个++k在传递参数之前一起先做完，因此输出结果是10,10 。

而printf("%d,%d\n", k++, k++); 输出结果是9,8 ，两个k++边传递参数边执行，并不是两个k++在传递参数之后再一起做，所以毫无规律可循。希望大家以后不要再抠此类问题！

7．输出数据时的域宽可以改变。

若整数变量m、n、i和浮点变量f都已正确定义并赋值，则语句printf("%*d", m, i); 将按照m指定的场宽输出i的值，并不输出m的值。

而语句 printf("%*.*f", m, n, f); 按照m和n指定的场宽输出浮点型变量f的值，并不输出m、n的值。这被称作 变场宽输出。

3.2 格式输入函数

3.2.1 基本的格式输入语句

- **scanf("格式控制", 内存地址表);**

其中**scanf( )**是C编译系统提供的格式输入函数。格式控制部分要用一对双引号括起来，它用于说明输入数据时应使用的格式。内存地址表中的各项目指出各输入数据所存放的内存地址。

与格式输出一样，在格式控制中，用于说明输入数据格式的格式说明符总是以%开头，后面紧跟的是具体的格式。用于数据输入的常用格式说明符有以下几种:

1. 整型格式说明符

- 十进制形式

%d 或 %md	用于一般int整型
%ld 或 %mld	用于long长整型
%lld 或 %mlld	用于long long超长整型
%u 或 %mu	用于无符号int整型
%lu 或 %mlu	用于无符号long长整型

- 八进制形式

%o 或 %mo	用于一般int整型
%lo 或 %mlo	用于long长整型
%llo 或 %mllo	用于long long超长整型

- 十六进制形式

%x 或 %mx	用于一般int整型
%lx 或 %mlx	用于long长整型
%llx 或 %mllx	用于long long超长整型

由此可以看出，用于输入与输出整型数据的格式说明符是完全一样的。**m**表示输入数据时的宽度（即列数）。

与输出情形一样，对于八进制形式与十六进制形式的输入格式，主要用于输入无符号整型的数据。

2. 实型格式说明符

- 单精度实型　　%f 或 %e
- 双精度实型　　%lf 或 %le

由此可以看出，与输出不同，在用于输入时，

无论是单精度实型还是双精度实型，

都不能用**m.n**来指定输入的宽度和小数点后的位数。

3. 字符型格式说明符

- 用于输入的字符型格式说明符为%c

下面是用到字符型格式输入的一个程序：

```
#include   <stdio.h>
main( )
{ int  a;
  float  b;
  char  c;
  scanf("%d%f%c", &a, &b, &c);
  printf("%d %f %c", a, b, c);
}
```

运行结果：

```
123 4.56
123 4.560000
请按任意键继续. . .
```

为字符变量c读入了什么？　回车符

下面对格式输入作几点说明：

- 在格式输入中，内存地址表中的各项目必须是变量地址，而不能是变量名，且彼此间用“,”分隔。为此，C语言专门提供了一个取地址运算符&。例如，&a表示变量a在内存中的地址。

- 当用于输入整型数据的格式说明符中没有宽度说明时，则在具体输入数据时分为以下两种情况：

① 如果各格式说明符之间没有其他字符，则在输入数据时，两个数据之间用"空格"、或"Tab"、或"回车"来分隔。

② 如果各格式说明符之间包含其他字符，则在输入数据时，应输入与这些字符相同的字符作为间隔。

例如，设有如下说明 **int a, b ;**

float c, d ;

现要利用格式输入函数输入数据，使得：

a=12, b=78, c=12.5, d=7.6。

采用不同的格式说明, 其输入数据的形式也是不同的。

输入语句为 **scanf("%d%d%f%f", &a, &b, &c, &d);**

(即格式说明符中没有宽度说明，各格式说明符之间也没有其他字符)

则输入数据的形式应为 **12 78 12.5 7.6**

(两个数据之间用空格来分隔，当然也可用“Tab”或“回车”来分隔)

输入语句为 **scanf("%d, %d, %f, %f", &a, &b, &c, &d);**

(格式说明符中没有宽度说明，但各格式说明符之间有其他字符:逗号)

则输入数据的形式应为 **12, 78, 12.5, 7.6**

（即在输入的两个数据之间同时要输入逗号)

输入语句 **scanf("a=%d, b=%d, c=%f, d=%f", &a, &b, &c, &d);**

（即格式说明符中没有宽度说明，但各格式说明符之间有其他字符）

输入数据的形式应为 **a=12, b=78, c=12.5, d=7.6**

（即在输入的两个数据之间同时要输入这些非格式说明符的字符）

- 当整型或字符型格式说明符中有宽度说明时，按宽度说明截取数据。

【例3-4】 设有以下程序：

```
#include   <stdio.h>
main( )
{ int  a, d;
  char  b, c;
  printf("input a, b, c, d：  ");
  scanf("%3d%3c%2c%2d", &a, &b, &c, &d);
  printf("a=%d, b=%c, c=%c, d=%d\n", a, b, c, d);
}
```

特别要注意的是：

一个字符型变量只能存放一个字符。

若从键盘输入如下(其中 "input a, b, c, d ： " 为输出的字符串)：

```
input a, b, c, d：  123456789012345 6
                    3d 3c 2c 2d
```

则它们与各格式说明符之间的对应关系如上，最后赋给各变量的值为 a=123, b='4', c='7', d=90

但运行时会出现致命错误，原因是，当按照3c格式把字符456给b赋值时，虽然字符'4'赋值给了b，但随后的字符'5'和'6'放到了b之后的内存中，造成了内存越界，引起致命错误。按照2c把字符78给c赋值存在同样的问题，因此以后一定要注意：不要再使用%mc(m>1)格式给char型变量读入字符值。

- 在用于输入的实型格式说明符中不能用**m.n**来指定输入的宽度和小数点后的位数(这是与输出的不同之处)。

例如，下列写法是错误的：　**scanf("%7.2f", &a);**

- 为了便于程序执行过程中从键盘输入数据，在一个C程序开始执行时，系统就在计算机内存中开辟了一个输入缓冲区，用于暂存从键盘输入的数据。开始时该输入缓冲区是空的。当执行到一个输入函数时，就检查输入缓冲区中是否有数据：

如果输入缓冲区中已经有数据（上一个输入函数读剩下的），则依次按照“格式控制”中的格式说明符从输入缓冲区中取出数据转换成计算机中的表示形式（二进制），最后存放到内存地址表中指出的对应地址的内存中。

如果输入缓冲区中没有数据（即输入缓冲区为空），则等待用户从键盘输入数据并依次存放到输入缓冲区中。当输入一个<回车>或<换行>符后，将依次按照“格式控制”中还未用过的格式说明符从输入缓冲区中取出数据转换成计算机中的表示形式（二进制），最后存放到内存地址表中指出的对应地址中。

在上述两种情况中的任一种情况下，从输入缓冲区中取数据，如果遇到<回车>或<换行>字符，则将输入缓冲区清空。此时如果“格式控制”中的格式说明符还未用完，则继续等待用户从键盘输入数据并依次存放到输入缓冲区中，直到输入一个<回车>或<换行>符后，再依次按照“格式控制“中还未用过的格式说明符从输入缓冲区中取出数据转换成计算机中的表示形式（二进制），最后存放到内存地址表中指出的对应地址的内存中。这个过程直到“格式控制”中的格式说明符用完为止。此时如果输入缓冲区中的数据还未取完，则将留给下一个输入函数使用。

从以上输入函数的执行过程可以看出，从键盘输入数据是以<回车>或<换行>作为结束的。当输入的数据一行不够时，可以在下一行继续输入; 当一行上的数据用不完时，可以留给下一个输入函数使用。

需要注意的是，由于<回车>或<换行>是作为键盘输入数据的结束符，因此，在输入函数的“格式控制”中，最后不能加换行符'\n'。 **scanf("%d\n", &a);** 是错误的，无法正确读入数据。

- 与格式输出一样，格式输入的格式控制中的各格式说明符与内存地 址表中的变量地址在个数、次序、类型方面必须一一对应。
- 尤其要注意：不存在用 宽输入格式 可以正确读入 窄变量 的可能！

double变量用%f不能正确读入

float变量用%lf也不能正确读入

【例3-5】设有C程序如下：

```
#include  <stdio.h>
main( )
{ double  x;
  printf("input x：  ");
  scanf("%f", &x);
  printf("x=%g\n", x);
}
```

这个程序的运行结果为：

（其中有下划线的部分为键盘输入）

```
input x：123.456
x=-9.25596e+061
```

显然，输出语句输出的x值是错误的。这是因为，x定义为双精度型的实型变量（占8个字节），但它使用的是单精度实型的输入格式说明符。当输入一个实型数123.456后，将按照单精度输入格式说明符将它转换成计算机中的表示形式（只占4个字节），最后存放到为双精度实型变量x所分配的存储空间的低4个字节中，而为双精度实型变量x所分配的存储空间的高4个字节中的各位均是未初始化的随机数，输出结果将是一个毫无意义的数。

将【例3-5】 scanf 中%f 改为 %lf , 则能正确读入 ：

```
#include  <stdio.h>
main( )
{ double  x;
   printf("input x：  ");
   scanf("%lf", &x);   /* 将 %f 改为 %lf , 则能正确读入  */
   printf("x=%g\n", x);
}
```

这个程序的运行结果为（其中有下划线的部分为键盘输入）

```
input x：123.456
x=123.456
```

3.2.2 scanf函数中常用的格式说明

每个格式说明都必须用%开头，以一个“格式字符”作为结束。

通常允许用于输入的格式字符和相应功能如下表所示。

格式字符	说　明
c	输入一个字符
d	输入带符号的十进制整型数
i	输入整型数，整型数可以是带先导 0 的八进制数，也可以是带先导0x(或0X)的十六进制数
o	以八进制格式输入整型数，可以带先导0，也可以不带
x	以十六进制格式输入整型数,可以带先导0x或0X,也可以不带
u	以无符号十进制形式输入整型数
f(lf)	以带小数点的数学形式或指数形式输入浮点数（单精度用f，双精度数用lf）
e(le)	同上
s	输入一个字符串，直到遇到"\0"。若字符串长度超过指定的精度则自动突破，不会截断字符串

- 说明：

（1）在格式串中，必须含有与输入项一一相对应的格式转换说明符。若格式说明与输入项的类型不一一对应匹配，则不能正确输入，而且编译时不会报错。若格式说明个数少于输入项个数，scanf函数结束输入，则多余的输入项将无法得到正确的输入值；若格式说明个数多于输入项个数，scanf函数也结束输入。多余的数据作废，不会作为下一个输入语句的数据。

（2）在VS2008编译环境下，输入short型整数，格式控制必须用%hd。要输入double型数据，格式控制必须用%lf(或 %le)，否则数据不能正确输入。

（3）在scanf函数的格式字符前可以加入一个正整数指定输入数据所占的宽度。但不可以对实数指定小数位的宽度。

（4）由于输入是一个字符流，scanf从这个流中按照格式控制指定的格式解析出相应数据，送到指定地址的变量中。因此当输入的数据少于输入项时，运行程序等待输入，直到满足要求为止。当输入的数据多于输入项时，多余的数据在输入流中没有作废，而是等待下一个输入操作语句继续从输入流读取数据。

（5）scanf函数有返回值，其值就是本次scanf调用正确输入的数据项的个数。

3.2.3 通过scanf函数从键盘输入数据

当用scanf函数从键盘输入数据时，每行数据在未按下回车键（Enter键）之前，可以任意修改。但按下回车键（Enter键）后，数据就送入了输入缓冲区，scanf函数即接受了这一行数据，不能再回去修改。

1．输入数值数据

在输入整数或实数这类数值型数据时，输入的数据之间必须用空格、回车符、 制表符（TAB键）等此类间隔符隔开，间隔符个数不限。即使在格式说明中人为指定了输入宽度，也可以用此方式输入。

例如：若已经有定义 int k; float a; double y; ， 以下输入语句：

scanf ("%d%f%le", &k, &a, &y);

若要给k赋值10， a赋值12.3， y赋值1234567.89，输入格式可以是（输入的第一个数据之前可有任意空格）：

10 12.3 1234567.89 <CR>

此处<CR>表示回车键。也可以是：

10<CR>

12.3<CR>

1234567.89<CR>

只要能把3个数据正确输入，可以按任何形式添加间隔符。

2．指定输入数据所占的宽度

可以在格式字符前加入一个正整数指定输入数据所占的宽度。例如上例改为：

```
scanf("%3d%5f%5le", &k, &a, &y);
```

若从键盘上从第1列开始输入：

123456.789.123

用 printf("%d %f %f\n", k, a, y); 打印的结果是：

123 456.700000 89.120000

- ✓ 可以看到，由于格式控制是%3d，因此把输入数字串的前三位123赋值给了k；由于对应于变量a的格式控制是%5f，因此把输入数字串中随后的5位数（包括小数点）456.7赋值给了a；由于格式控制是%5e，因此把数字串中随后的5位（包括小数点）89.12赋值给了y。
- ✓ 由以上示例可知，数字之间不需要间隔符，若插入了间隔符，系统也将按指定的宽度来读取数据，从而会引起输入混乱。除非数字一开始就是“粘联”在一起，否则不提倡指定输入数据所占的宽度。

3．跳过某个输入数据

可以在%和格式字符之间加入“*”号，作用是跳过对应的输入数据。例如：

```
int   x, y, z;
scanf("%d%*d%d%d", &x, &y, &z);
printf("%d  %d  %d\n", x, y, z);
```

若是输入：

```
12  34  56  78
```

则输出是：

```
12  56  78
```

系统将12赋给x，跳过34，把56赋给y，把78赋给z。

4．在格式控制字符串中插入其它字符

scanf函数中的格式控制字符串是为了输入数据用的，无论其中有什么字符，也不会输出到屏幕上，因此若想在屏幕上输出提示信息，应该首先使用printf函数输出。例如：

```
int  x, y, z;
scanf("Please input x,y,z : %d%d%d", &x, &y, &z);
```

屏幕上不会输出Please input x,y,z ：，而是要求输入数据时按照一一对应的位置原样输入这些字符。必须从第一列起以下面的形式进行输入：

```
Please input x,y,z: 12 34 56
```

包括Please input x,y,z:，字符的大小写、字符间的间格等必须与scanf中的完全一致。这些字符又被称为通配符。

但如果使用以下的形式：

```
int  x, y, z ;
printf("Please input x,y,z: ");
scanf("%d%d%d", &x, &y, &z);
```

由于printf语句的输出，屏幕上将出现提示：

Please input x,y,z:

只需按常规输入下面的数据即可：

12 34 56

如果在以上scanf中，在每个格式说明之间加一个逗号作为通配符：

```
scanf("%d,%d,%d", &x, &y, &z);
```

则输入数据时，必须在前两个数据后面紧跟一个逗号，以便与格式控制中的逗号一一匹配，否则就不能正确读入数据运行。

例如，输入：

12,34,56

能正确读入。输入：

12,　34,　56

也能正确读入。因为空格是间隔符，将全部被忽略掉。

但输入：

12　,34　,56

将不能正确读入，因为逗号没有紧跟在输入数据后面。

提醒：

为了减少不必要的麻烦，尽量不要使用通配符。

3.3 字符输出函数

- 字符输出函数的形式为:　**putchar(c)**

这个函数的功能是，在屏幕显示的当前光标位置处输出项目**c**所表示的一个字符。其中**c**可以是字符型常量、字符型变量、整型变量或整型表达式。

字符输出函数的执行过程与格式输出函数的执行过程完全相同。

字符输出函数putchar( )也可以输出转义字符。

【例3-6】 设有如下C程序：

```
#include  <stdio.h>
main( )
{ int x=68;
  char y='B';
  putchar('A');
  putchar(y);
  putchar(67);
  putchar(x);
  putchar(34+25);
}
```

该程序的输出结果为

ABCD;

【例3-7】 设有C程序如下：

```
#include  <stdio.h>
main( )
{ int x=68;
  char y='B';
  putchar('A'); putchar('\n');
  putchar(y); putchar('\n');
  putchar(67); putchar('\n');
  putchar(x); putchar('\n');
  putchar(34+25); putchar('\n');
}
```

在这个程序中，在输出每一个字符后，紧接着输出一个换行，最后输出结果为：

```
A
B
C
D
;
```

3.4 字符输入函数

在C语言中，字符输入函数的形式为：**getchar( )**

这个函数的功能是接收从键盘输入的一个字符。

例如，下面的程序执行过程中，将等待从键盘输入一个字符赋给字符型变量x：

```
#include   <stdio.h>
main( )
{    char  x;
     x=getchar( );
}
```

需要说明的是，在执行字符输入函数时，由键盘输入的字符（依次存放在输入缓冲区中）同时也在屏幕上显示，并且以<回车>结束，但一个字符输入函数只顺序接收一个字符，输入缓冲区中剩下的字符数据（包括回车符）将留给下面的字符输入函数或格式输入函数使用。

3.4 字符输入函数

在C语言中，另一类字符输入函数为：_getch( ) 或 _getche( )

这两个函数的功能是在按下相应键的同时接收从键盘输入的一个字符。但这两个函数不是C的标准字符输入函数。

例如，下面的程序执行过程中，将等待从键盘输入一个字符赋给字符型变量x：

```
#include   <conio.h>      /* 注意：这里引用的不是stdio.h */
main( )
{    char  x;
     x= _getch( );
 }
```

需要说明的是，使用字符输入函数_getch( )时，由键盘输入的字符不在屏幕上显示；但使用字符输入函数_getche( )时，由键盘输入的字符同时也在屏幕上显示。并且都不等待<回车>符，只要按下相应键的同时，此函数就读取(接收)了相应字符。

3.4 字符输入函数

将字符放回输入流的函数形式为：**_ungetch(c);**

这个函数的功能是把刚从键盘接收(输入)的一个字符c回写到输入流。使得输入流看起来未读过。

例如，下面的程序执行过程中，从键盘输入一个字符赋给字符型变量**x**：

```
#include   <conio.h>
main( )
{    char  x;
     x= _getch( );    /* 读入一个字符 */
     _ungetch(x);   /* 将读入的字符放回输入流中 */
 }
```

```
#include   <stdio.h>
#include   <conio.h>
main( )
{   char  x, y;
    x= _getche( );    /* 读入一个字符 */
    _ungetch(x);   /* 将读入的字符放回输入流中 */
    y= _getche( );
    putchar(y);
}
```

运行时，只需输入一个字符，就会出现结果，而且是输入的相应字符。

第3次作业：

教材 p61-62 2. 3. 4. 5. 6.

（注意：从现在开始，作业要求以电子版文档(.doc或.docx或.pdf) 网上提交，不再写纸版的作业，编程序的题目要求上机调试完成后，把源程序和结果one by one 顺序贴在电子版文档中）