第 3 章　顺序程序设计

顺序结构是最简单的一种程序结构形式，顺序结构指每一步操作或每一个操作块，在执行流中顺序逐个执行。其特点是，每一条语句按顺序执行，每一条语句只执行一遍，不重复执行，也没有语句不执行。所谓“块”是指作为一个整体对待的一组，一个操作块就是作为一个整体对待的一组操作。如图 3.1 所示，虚线框内是一个顺序结构，其中 A 和 B 两个框是顺序执行的。本章介绍为编写简单的程序所必需的一些内容。

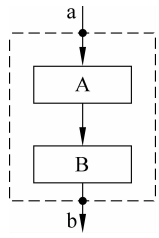

图 3.1　顺序结构

3.1　C 语句概述

算法是独立于具体的计算机语言的，而实现算法则需要用某一种计算机语言写出相应的程序。算法中的每一步操作对应程序的某一条执行语句，通过语句向计算机系统发出操作指令，完成相应的动作。和其他高级语言一样，C 语言的语句用来向计算机系统发出操作命令。一个语句经编译后产生若干条机器指令。一个实际的程序应当包含若干语句。一个函数包含声明部分和执行部分，执行部分即由语句组成。

C 程序结构可以用图 3.2 表示。即一个 C 程序可以由若干个源程序文件（分别进行编译的文件模块）组成，一个源文件可以由若干个函数和预处理命令以及全局变量声明部分组成，一个函数由数据定义部分和执行语句组成。

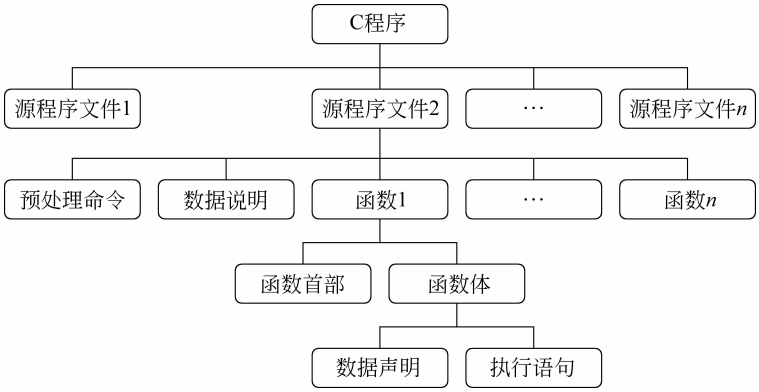

图 3.2　C 程序结构

程序一般包括数据描述和数据操作两大部分，数据描述由声明语句来实现，数据操作由执行语句来完成。数据描述主要定义数据结构和数据初值。声明语句主要包括数据类型的声明、函数和变量的声明、变量的初始化等(例如变量定义语句：int a,b;)。数据操作的任务是对已提供的数据进行加工。

在C语言中，语句就是能够独立完成一个简单功能的基本构成单位。程序是由一条条的语句组成和执行的，每一条语句完成一个基本的功能。最基础的不可以再分割的C语言语句称为简单语句。C的简单语句可以分为以下五类：

(1) 控制语句，用来控制程序中语句执行的次序。更确切地说，控制语句可以用来改变或打破程序中按语句的先后次序顺序执行的规律。

C语言只有九种控制语句，如表3-1所示。

表3-1 C语言的控制语句

序 号	语 句	备 注
1	if()～else～	条件语句
2	for()～	循环语句
3	while()～	循环语句
4	do～while()	循环语句
5	continue	结束本次循环语句
6	break	中止执行switch或循环语句
7	switch	多分支选择语句
8	goto	转向语句，可用于跳出多重循环
9	return	从函数返回语句，并可带回一个返回值

上面九种语句中的括号()表示其中是一个条件，～表示内嵌的语句。例如，if()-else-的具体语句可以写成：

```
if(x==y)
    printf("x与y相等");
else
    printf("x与y不相等");
```

(2) 函数调用语句。函数调用语句即由一次函数调用加上一个分号构成。例如：

```
max(x,y,z);        /*假设已定义了max函数，它有3个参数*/
```

(3) 表达式语句。由一个表达式在其后加上一个分号就成为一个表达式语句。例如，由赋值表达式加上一个分号构成一个赋值语句(编程时经常使用赋值语句)。

```
x=0;
```

注意：

① 一个C语句必须在最后出现分号，分号是语句中不可以缺少的一部分。例如：

```
x=0
```

是一个赋值表达式，而

```
x=0;
```

是一个赋值语句。二者意义截然不同。

② 任何表达式都可以加上分号而成为语句，例如

```
i++;
```

是一语句，作用是使 i 值加 1。又如

```
x+2*y;
```

也是一个语句，作用是完成 x＋2＊y 的操作，它是合法的，但是并不把 x＋2＊y 的值赋给其他变量，所以这条语句并无实际意义(应避免这样使用)。

③ 表达式能构成语句是 C 语言的一个重要特色。其实“函数调用语句”也属于表达式语句，因为函数调用(如 sin(x))也属于表达式的一种。只是为了便于理解和使用，我们把“函数调用语句”和“表达式语句”分开来说明。由于 C 程序中大多数语句是表达式语句(包括函数调用语句)，所以有人把 C 语言称作“表达式语言”。

(4) 空语句。空语句在 C 语言中很少用到，它的作用就是什么都不干，让计算机空转，浪费时间。其实是为了暂时等待其他东西先完成(有时可以结合使用循环和空语句来设计延时效果)。

空语句的形式为一个简单的；，其他什么都没有。形式如下：

```
;
```

一个单独的分号就是告诉计算机，在此暂留一步，然后再往下执行其他语句。

(5) 复合语句，有时也称为是语句块。之所以叫它复合语句，是因为它是由其他简单语句组成，看着像一块语句，所以也叫语句块。复合语句的使用条件通常是：条件语句和循环语句在语法上只允许带一条语句，但是此时在分支或循环中又需要进行多项操作。

以下是复合语句的常见形式：

① 语句块的形式。

C 语言中的语句块是由一对大括号和其中包含的若干条语句组成，其形式如下所示：

```
{
    语句 1;
    语句 2;
    ⋮
    语句 n;
}
```

② 语句块的嵌套。

C 语言的语句块中是可以包含语句块的，叫做语句块嵌套。语句块中的语句块是被当做一个简单的语句来处理的。

例如：

```
{
    s=0;
    ;
{
    i--;
    printf("%d",i);
}
}
```

这个程序中的语句块就稍微有点复杂了，语句 i－－；和 printf("％d",i)；被大括号括起来组成了一个语句块，这个语句块又和 s＝0；、；一起被另一个大括号括了起来，组成了一个更复杂的语句块。

注意：复合语句是用花括号{ }括起来的由多条语句组成的语句组。复合语句中虽然包含多条语句，但复合语句的意义就在于从逻辑上把它作为一条语句对待；复合语句中，可以出现变量定义，可以包含顺序、选择、循环语句。例如，

```
{
    int z;
    z=x+y;
    if(z>100) z=z-100;
    printf("z=%d",z);
}
```

3.2 赋值语句

由于在编程过程中，赋值语句应用十分普遍，所以在这里专门再介绍一下。

赋值语句的一般形式为：

```
变量=表达式;
```

例如：

```
x=1;
```

是一个合法的赋值语句。

注意：C 语言的赋值语句具有与其他高级语言的赋值语句不同的一些特点和功能：

(1) 注意赋值表达式和赋值语句的区别。赋值表达式是一种表达式，它可以出现在任何允许表达式出现的地方，而赋值语句则不能。例如：

```
if((a=b)>0) t=a;
```

按语法规定 if 后面的()内是一个条件，例如可以是 if(x＞0)…。现在在 x 的位置上换上一个赋值表达式 a＝b，其作用是：先进行赋值运算(将 b 的值赋给 a)，然后判断 a 是

否大于0，如大于0，执行t=a。在if语句中的a=b不是赋值语句而是赋值表达式，这样写是合法的。如果写成

```
if((a=b;)>0) t=a;
```

就错了。在if的条件中可以包含赋值表达式，但不能包含赋值语句。

(2) 由于在赋值号=右边的表达式也可以又是一个赋值表达式，因此，下述形式

```
变量=(变量=表达式);
```

是成立的，从而形成嵌套的情形。其展开之后的一般形式为：

```
变量=变量=…=表达式;
```

例如：

```
a=b=5;
```

按照赋值运算符的右接合性，因此实际上等效于：

```
b=5;
a=b;
```

(3) 在变量说明中，不允许连续给多个变量赋初值。

```
int a=b=c=5;          /*是错误的*/
int a=5,b=5,c=5;      /*是正确的*/
```

3.3　数据输入输出的概念及在C语言中的实现

在讨论输入输出时要注意以下几点：

(1) 什么是输入输出？

输入输出如图3.3所示。

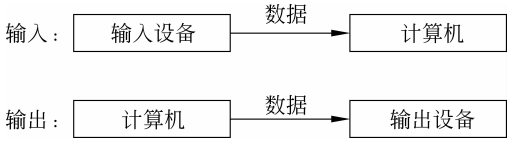

图3.3　输入输出

(2) C语言本身并不提供输入输出语句，所以C语言的输入和输出操作是由C函数实现的。不把输入输出作为C语言提供的语句的目的是使C语言编译系统简单，因为将语句翻译成二进制的指令是在编译阶段完成的，没有输入输出语句就可以避免在编译阶段处理与硬件有关的问题，可以使编译系统简化，而且通用性强，可移植性好，对各种型号的计算机都适用，便于在各种计算机上实现。

C语言函数库中有一批“标准输入输出函数”，它们是以标准的输入输出设备(一般为终端设备)为输入输出对象的。其中有putchar(输出字符)、getchar(输入字符)、printf

(格式输出)、scanf(格式输入)、puts(输出字符串)、gets(输入字符串)。

(3) 在使用C语言库函数时,要用预编译命令#include将有关的"头文件"包括到用户源文件中。在头文件中包含了与用到的函数有关的信息。例如使用标准输入输出库函数时,要用到stdio.h文件。文件后缀中的h是head的缩写,#include命令都是放在程序的开头,因此这类文件称为"头文件"。在调用标准输入输出库函数时,文件开头应有以下预编译命令:

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

或

```
#include "stdio.h"
```

stdio.h是standard input & output的缩写,它包含了与标准I/O库有关的变量定义和宏定义(相关内容在后续章节介绍)。考虑到printf和scanf函数使用频繁,系统允许在使用这两个函数时可不加#include<stdio.h>命令,但有的编译系统要求必须加上,所以希望养成在使用标准输入输出库函数时,一律加上#include<stdio.h>命令的习惯。

3.4 字符数据的输入输出

3.4.1 字符输出函数 putchar 函数

格式:putchar(ch)

功能:每调用putchar函数一次,就向终端输出一个字符。

其中,ch是字符变量、字符常量、整型变量、整型常量或转义字符等。请读者不要忘记,如果在一个函数中要调用putchar函数,应该在该函数的前面(或该文件开头)加上"包含命令":

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

使用putchar函数时经常有以下几种使用方式。

(1) 用putchar函数输出一个字符变量的值。

```
char ch='A';
putchar(ch);
```

将在显示器上输出字符A。

(2) 用putchar函数输出一个字符常量的值。

```
putchar('A');
```

将在显示器上输出字符A。

(3) 用putchar函数输出一个整型变量的值。

```
int x=65
```

```
putchar(x);
```

将在显示器上输出字符 A。

请读者思考一下：此时的整型变量的值是不是 C 语言中整型数据所能表示的所有值呢，比如很大、很小或负数时，输出结果会是什么呢？

(4) 用 putchar 函数输出一个整型常量的值。

```
putchar(65);
```

仍能在显示器上输出字符 A。

(5) 用 putchar 函数输出一个转义字符。

例 3.1　putchar 函数中转义字符的使用。

相应的程序代码如下：

```
#include<stdio.h>
main()
{
    char a,b,c,d;
    a='G';b='O';c='O';d='D';
    putchar(a);putchar(b);putchar(c); putchar(d);putchar('\n');
}
```

运行结果如下：

```
GOOD
```

程序中 putchar('\n')的作用是输出一个换行符，使输出的当前位置移到下一行的开头。如果将例 3.1 程序最后一行改为

```
putchar(a); putchar('\n');putchar(b); putchar('\n');putchar(c);putchar('\n');
    putchar(d);
```

则输出结果为：

```
G
O
O
D
```

3.4.2　字符输入函数 getchar 函数

格式：getchar()

功能：getchar 函数的作用是从终端(或系统隐含指定的输入设备)输入一个字符。每调用 getchar 函数一次，就从键盘接收一个字符，它的调用形式如下：

getchar 函数是一个无参函数，但调用 getchar 函数时，后面的括号不能省略。getchar()函数从键盘接收一个字符作为它的返回值。读者请不要忘记，如果在一个函数中要调用 getchar 函数，应该在该函数的前面(或本文件开头)加上“包含命令”：

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

或

```
#include "stdio.h"
```

注意：

(1) 在输入时，空格、回车键等都将作为字符读入，而且，只有在用户输入回车键时，读入才开始执行。

例 3.2 输入一个大写字母，输出其对应的小写字母。

相应的程序代码如下：

```
#include<stdio.h>
void main()
{
    char ch;
    int n;
    ch=getchar();          /*用 getchar 函数接受用户的字符输入*/
    n=ch+32;               /*将 ch 存放的字符隐式转换成 ASCII 码赋给 n*/
    putchar(n);            /*通过 putchar 函数输出其对应的小写字母*/
}
```

运行结果如下：

```
A↙      (输入 A 后，按 Enter 键，字符才送到内存)
a       (输出 A 字符对应的小写字母 a)
```

(2) getchar 函数只能接收一个字符。getchar 函数得到的字符可以赋给一个字符变量或整型变量(如 ch＝getchar();)，也可以不赋给任何变量，作为表达式的一部分。例如，例 3.2 第 6、7、8 行可以用下面一行代替：

```
putchar(32+getchar());
```

因为 getchar()的值为'A'，因此 putchar 函数输出'a'。也可以用 printf 函数输出：printf("%c", 32+getchar());

3.5 格式输入与输出

3.5.1 格式输出函数 printf

1. printf 函数的一般形式

格式：printf(格式控制，输出表列)

功能：printf 函数的作用是向终端(或系统隐含指定的输出设备)输出若干个任意类型的数据(putchar 只能输出字符，而且只能是一个字符，而 printf 可以输出多个数据，且为任意类型)。

例如：

```
printf("n=%3d,f=%5.1f\n,c=%c",n,f,c);
printf("\n %d+%f=%5.2f\n",a,b,sum);
```

但值得一提的是，printf 也经常像下面这样的方式使用，来输出相应的提示语。此时双引号里面不能有格式说明符。通过提示语的输出可以使程序具有人性化的界面，尤其当有多处需要输入时，或者在输出结果之前输出相应的提示语可以起到很好的提示作用。

```
printf("请输入变量 x 的值: \n");
printf("你成功了!");
```

说明：

(1)“格式控制”是用双引号括起来的字符串，也称“转换控制字符串”。可以包含下列三种字符。

① 格式说明符：由%和格式字符组成，例如，%3d、%5.1f、%d 和%c 等，这些字符用来控制数据的输出格式。

② 转义字符：这些字符常用来控制光标的位置，例如"\n"和"\t"等。

③ 普通字符：除格式说明符和转义字符之外的其他字符，这些字符原样输出，例如上面例子中的 n=、f=、+等。

(2)“输出表列”是需要输出的一些数据，可以是表达式，输出的各项之间用，间隔。

2. 格式说明符

格式说明符用于指定对应输出项的输出格式，其一般形式如下：

```
%[修饰符]格式字符
```

1) 格式字符

printf 格式字符如表 3-2 所示。

表 3-2　printf 中使用的格式字符

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数(不输出正号)
o	以八进制无符号形式输出整数(不输出前导符 0)
x 或 X	以十六进制无符号形式输出整数(不输出前导符 0x 或 0X)。对于 x 用 abcdef 输出；对于 X，用 ABCDEF 输出
u	按无符号的十进制形式输出整数
f	以[－]mmm.ddd 带小数点的形式输出单精度和双精度数，d 的个数由精度指定。隐含的精度为 6(即 6 位小数)，若指定的精度为 0，小数部分(包括小数点)都不输出(四舍五入)
e 或 E	以[－]m.dddddde±xx 或[－]m.ddddddE±xx 的指数形式输出单精度和双精度数。d 的个数由精度指定，隐含的精度为 6，若指定的精度为 0，小数部分(包括小数点)都不输出。用 E 时，指数以大写 E 表示
g 或 G	由系统决定采用%f 格式还是采用%e 格式，以使输出宽度最小，不输出无意义的 0。用 G 时，若以指数形式输出，则指数以大写表示
s	输出字符串中的字符，直到遇到'\0'(空字符)，或者输出由宽度指定的字符数

2）长整型修饰符

长整型修饰符"l"加在"%"和格式字符之间，用于输出长整型数据。

（1）%ld：以十进制输出长整型数据。

（2）%lo：以八进制输出长整型数据。

（3）%lx：以十六进制输出长整型数据。

（4）%lu：输出无符号长整型数据。

3）宽度和精度修饰

可以在%和格式字符之间加入形如 m 或 m.n（m、n 均为整数）的修饰。其中，m 为宽度修饰，n 为精度修饰。

宽度修饰用来指定数据的输出宽度。%md 表示输出的数据占 m 列。

精度修饰对不同的格式字符，作用不同：对于格式字符 f，用来指定输出小数位的位数；对于格式字符 e，用来指定输出有效数字的位数；对于格式字符 c，表示截取的字符个数。

有关宽度和精度修饰后面的格式符应用举例中有详细介绍。

4）左对齐修饰

在指定了宽度修饰时，如果指定宽度小于数据需要的实际宽度，则数据左边补空格，补够指定的宽度，这种对齐方式称为“右对齐”。也可以在数据的右边补空格来补够指定的宽度，这种对齐方式称为“左对齐”。指定左对齐的时候，使用左对齐修饰符－。

注意：

（1）printf 函数可以输出常量、变量和表达式的值。但格式控制字符串中的格式指示符必须按从左到右的顺序，与输出项表中的每个数据一一对应，否则出错。

（2）格式字符 x、e、g 可以用小写字母，也可以用大写字母。使用大写字母时，输出数据中包含的字母也大写。除了 x、e、g 格式字符外，其他格式字符必须用小写字母，例如，%f 不能写成%F。

（3）一个格式字符以%开头，在表 3-2 中的 9 个格式字符为结束，中间可以插入附加格式字符（也称修饰符）。但是下面语句中加下划线的字符 c 是普通字符。

```
printf("d=%dc=%c",d,c);
```

（4）如果只想输出字符%，则应该在“格式控制”字符串中用连续两个%表示。

3. 常用格式符应用举例

（1）d 格式符，用来输出十进制整数。d 格式符的应用如表 3-3 所示。

表 3-3　d 格式符应用举例

输 出 语 句	输出结果	说　　明
printf("%d",123);	123	按数据实际宽度进行输出
printf("%d",'A');	65	用%d 输出字符数据时，输出的是该字符的 ASCII 码值
printf("%5d",123);	␣␣123	输出数据占 5 位，右对齐方式，左边补空格
printf("%－5d",123);	123␣␣	输出数据占 5 位，左对齐方式，右边补空格

续表

输 出 语 句	输出结果	说　　明
printf("%2d",123);	123	输出数据位数超过指定的列宽，将按其实际长度输出
printf("%05d",123);	00123	%05d 是在输出一个小于 5 位的数值时，输出值前面补 0 使总宽度为 5 位
printf("%－05d",123);	123␣␣	在采用左对齐方式时右侧预补 0 时自动由空格替代。即%－05d 相当于%－5d
printf("%＋05d",123);	＋0123	输出正号
printf("%＋05d",－123);	－0123	输出负号

（2）o 格式符，以八进制数形式输出整数。使用 o 格式符输出的数值不带符号，即将符号位也一起作为八进制数的一部分输出。o 格式符应用如表 3-4 所示。

表 3-4　o 格式符应用举例

输 出 语 句	输 出 结 果	说　　明
printf("%o",－1);	37777777777	－1 在内存中的补码为 32 位 1，二进制的 32 个 1 转换为八进制即为 37777777777
printf("%13o",－1);	␣␣37777777777	用法类似于 d 格式符
printf("%－13o",－1);	37777777777␣␣	
printf("%3o",－1);	177777	
printf("%013o",－1);	0037777777777	
printf("%－013o",－1);	37777777777␣␣	

（3）x 格式符和 X 格式符，以十六进制数形式输出整数。和 o 格式符类似，使用 x 格式符或 X 格式符输出的数值不带符号，即将符号位也一起作为十六进制数的一部分输出。x 格式符和 X 格式符应用如表 3-5 所示。

表 3-5　x 格式符应用举例

输 出 语 句	输 出 结 果	说　　明
printf("%x",－1);	ffffffff	－1 在内存中的补码为 32 位 1，二进制的 32 个 1 转换为十六进制即为 ffffffff
printf("%X",－1);	－1,FFFFFFFF	x 格式符和 X 格式符的区别仅在于输出的字符是以小写字母还是大写字母形式。以下例子仅以 x 格式符为例
printf("%10x",－1);	␣␣ffffffff	用法类似于 d 格式符
printf("%－10x",－1);	ffffffff␣␣	
printf("%3x",－1);	ffffffff	
printf("%010x",－1);	00ffffffff	
printf("%－010x",－1);	ffffffff␣␣	

(4) u 格式符,用来以十进制形式输出 unsigned 型数据(即无符号数)。u 格式符应用如表 3-6 所示。

表 3-6 u 格式符应用举例

输出语句	输出结果	说明
printf("%u",-1);	4294967295	自己分析一下为什么是这样的结果
printf("%u",-2);	4294967294	
printf("%12u",-1);	␣␣4294967295	用法类似于 d 格式符
printf("%-12u",-1);	4294967295␣␣	
printf("%012u",-1);	004294967295	
printf("%-012u",-1);	4294967295␣␣	
printf("%2u",-1);	4294967295	

注意:也可以对长整型修饰符%ld、%lo、%lx(%lX)和%lu 指定字段宽度,方法类似于%d,请读者自行练习。

(5) c 格式符,用来输出一个字符。c 格式符应用如表 3-7 所示。

表 3-7 c 格式符应用举例

输出语句	输出结果	说明
printf("%c",'A');	A	用来输出字符 A,注意输出结果没有单引号
printf("%c",65);	A	用%c 输出一个合理整数(0~255)时,会输出该整数对应的字符
printf("%4c",'A');	␣␣␣A	输出数据占 4 位,右对齐方式,左边补空格
printf("%-4c",'A');	A␣␣␣	输出数据占 4 位,左对齐方式,右边补空格

(6) s 格式符,用来输出一个字符串。s 格式符应用如表 3-8 所示。

表 3-8 s 格式符应用举例

输出语句	输出结果	说明
printf("%s","HELLO");	HELLO	用来以实际宽度输出字符串 HELLO
printf("%7s","HELLO");	␣␣HELLO	用法类似于 d 格式符
printf("%3s","HELLO");	HELLO	
printf("%-7s","HELLO");	HELLO␣␣	
printf("%7.3s","HELLO");	␣␣␣␣HEL	输出数据占 7 列,但只取字符串中左端 3 个字符。这些字符输出在 7 列的右侧,左补空格
printf("%-7.3s","HELLO");	HEL␣␣␣␣	"-"表示左对齐,右补空格

(7) f 格式符,用来输出实数(单精度或双精度),以小数形式输出。%f 不指定字段宽度,由系统自动指定,使整数部分全部如数输出,并输出 6 位小数。而且并不是全部数字

都是有效数字。双精度数是可以使用%f格式进行输出的。f格式符应用如表3-9所示(lf格式符也可以指定输出的宽度,方法同f格式符,请读者自行练习)。

表3-9　f格式符应用举例

输出语句	输出结果	说　明
double x=111111.111; printf("%f",x*2);	222222.222000	很显然例子中的输出结果是正确的
double x=111111.111; printf("%lf",x*2);	222222.222000	双精度数是可以使用%lf格式进行输出
printf("%6.2f",12.6);	␣12.60	指定的数据占6列(包括1位小数点),小数部分占2列。数值长度共5列(12.60)小于6列,左补一个空格
printf("%-6.2f",12.6);	12.60␣	右补一个空格
printf("%3.2f",12.126);	12.13	小数部分占2位,必须先指定(数据小数位数大于指定小数宽度时四舍五入),总体数据宽度大于指定宽度,指定的总体的数据宽度无效,只有小数部分的指定生效

(8) e格式符,用来以指数形式输出实数。不指定输出数据所占的宽度和数字部分的小数位数,但有的系统自动指定。数值按规范化指数形式输出(即小数点前必须有而且只有1位非零数字)。e格式符应用如表3-10所示。

表3-10　e格式符应用举例

输出语句	输出结果	说　明
printf("%e",123.45);	1.234500e+002	输出的数据共占多少列,不同的系统略有不同
printf("%12.3e",123.4567);	␣␣1.235e+002	左边补2个空格
printf("%-12.3e",123.4567);	1.235e+002␣␣	右边补2个空格
printf("%.3e",123.4567);	1.235e+002	只指定尾数的小数部分,不指定整体数据宽度

(9) g格式符,用来输出实数,它根据数字的大小,自动选f格式或e格式(选择输出时占宽度较小的一种),且不输出无意义的零。g格式符应用如表3-11所示。

表3-11　g格式符应用举例

输出语句	输出结果	说　明
printf("%f",123.456);	123.456000	默认小数点后6位小数
printf("%e",123.456);	1.234560e+002	用指数形式表示(以规范化形式表示尾数),不同的系统略有不同
printf("%g",12.000);	12	选用%f格式,并去掉无意义的0
printf("%g",12000000.0);	1.2e+007	选用%e格式,并去掉无意义的0

3.5.2 格式输入函数 scanf

1. scanf 函数的一般形式

格式：scanf(格式控制,地址表列)

功能：scanf 函数的作用是从键盘(或系统隐含指定的输入设备)输入若干个任意类型的数据(getchar 只能输入字符,而且只能是一个字符,而 scanf 可以输入多个数据,且为任意类型)。

例如：

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

说明：

(1)“格式控制”可以包含以下三种类型的字符。

① 格式说明符：由%和格式字符组成,例如,%d、%f、%c 和%s 等,用来指定数据的输入格式。

② 转义字符：这些字符通常用来控制光标的位置,例如\n 和\t 等(但是 scanf 的格式控制中最好不使用转义字符,否则会制造很多麻烦)。

③ 普通字符：除格式说明符和转义字符之外的其他字符,这些字符原样输出,在输入有效数据时,必须原样一起输入。例如上面例子中的 n=、f=、+等,当然也包括空格、跳格键和回车键等空白字符(空白字符通常作为相邻两个输入数据的默认分隔符)。

(2)“地址表列”是由若干个地址组成的表列,可以是变量的地址,或字符串的首地址,各项之间用“,”间隔。

2. 格式说明符

格式说明符的一般形式如下：

```
%[修饰符]格式字符
```

1) scanf 中的格式字符(如表 3-12 所示)

表 3-12 scanf 中的格式字符

格式字符	说　明
c	输入单个字符
d,i	输入有符号的十进制整数
o	输入无符号的八进制整数
x,X	输入无符号的十六进制整数
u	输入无符号的十进制整数
f	输入实数,可以用小数形式或指数形式输入
e,E,g,G	与 f 的作用相同,e 与 f、g 可以互相替换(大小写作用相同)
s	输入字符串,将字符串送到一个字符数组中,在输入时以非空白字符开始,以第一个空白字符结束,字符串以串结束标志'\0'作为其最后一个字符

2）宽度修饰

宽度修饰用来指定输入数据所占列数，例如：

```
scanf("%2d%3d",&a,&b);
```

假设输入 123456↙，则系统将读取的 12（开头 2 个数字）赋给变量 a；将读取的 345（紧跟着的 3 个数字）赋给变量 b。

```
scanf("%3c%3c",&ch1,&ch2);
```

假设输入 abcdefg↙，则系统将读取的 abcdefg（开头 3 个字符）中的 abc 赋给变量 ch1；将读取的 abcdefg（紧跟着的 3 个字符）中的 def 赋给变量 ch2（注意这里的 ch1 和 ch2 不是普通的字符变量）。

3）抑制修饰符

抑制修饰符 * 表示对应的数据读入后，不赋给相应的变量，该变量由下一个格式指示符输入。例如：

```
scanf("%2d%*2d%3d",&a,&b);
```

假设输入 123456789↙，则系统将读取 12 并赋值给 a；读取 34，但舍弃掉（* 的作用）；读取 567 并赋值给 b。

4）长数据修饰符

长数据修饰符 l 加在%和格式字符之间，用于输入长型数据。

（1）%ld：以十进制输入长整型数据。

（2）%lo：以八进制输入长整型数据。

（3）%lx：以十六进制输入长整型数据。

（4）%lu：以无符号形式输入长整型数据。

（5）%lf：输入 double 型数据。

（6）%le：输入 double 型数据。

5）短整型数据修饰符

短整型数据修饰符 h 加在%和格式字符之间，即%hd、%ho、%hx 用于输入短整型数据。

3. 使用 scanf 函数时应注意的问题

（1）scanf 函数中“格式控制”和“输入表列”二者必不可少并且用逗号间隔起来。

（2）“地址表列”中的各项应是变量地址，而不应是变量名。例如 x 和 y 均是已定义的单精度变量，则 scanf("%f,%f",&x,&y);是正确的，千万别写成 scanf("%f,%f", x, y);或 scanf("%f,%f",&x, y);等其他形式。

（3）“格式控制”中若有除了格式说明符以外的其他字符，则输入数据时要在对应位置应输入与这些字符相同的字符。如果有：

```
scanf("%f,%f",&x,&y);
```

输入时应为下面的形式：

```
1.2,-3.14↙
```

注意 1.2 后面的逗号不可以忘记输入，也不可以用其他字符替代。

(4) 注意空白字符的使用。如果有：

```
scanf("%f␣␣%f",&x,&y);
```

即在两个%f 之间有两个空格(用␣表示空格)，此时输入数据时，两个数据之间应该有 1 个或更多的空白字符(即若干个空格、回车、Tab 来替代，或者是这些空白字符的任意多个组合)。例如：

```
1.2␣-3.14↙
```

或

```
1.2↙-3.14↙
```

此规则适用于多个%f，多个%d 或多个%d 和%f 的组合(但需要注意的是，%d 和%f 的多种组合之间都是只有空白字符，而没有其他普通字符)。

(5) 在用%c 格式输入字符时，空格字符和“转义字符”都作为有效字符输入，例如 a、b、c 都是已定义的字符变量，

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

若要将字符'x','y','z'分别赋给字符变量 a、b、c，则
正确输入为：

```
xyz↙
```

若输入：

```
x␣y␣z↙
```

则错误的将字符'x'送给 a，空格字符'␣'送给 b，字符'y'送给 c。

(6) 在输入数据时，遇以下情况时认为该数据结束：

① 遇空白字符(即空格，回车或 Tab)。

② 按指定的宽度结束，如“%2d”，只取 2 列。

③ 遇非法输入。

例如：

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

若输入

```
1234 a 123o.26↙
```

第一个数据对应%d 格式在输入 1234 之后遇字母 a，因此认为数值 1234 后已没有数字了，第一个数据到此结束，把 1234 送给变量 a。字符'a'送给变量 b，由于%c 只要求输入一个字符，因此输入字符 a 之后不需要加空格，后面的数值应送给变量 c。如果由于疏忽

把本来应为 1230.26 错打成 123o.26，由于 123 后面出现字母 o，就认为该数值数据到此结束，将 123 送给 c。

(7) 对 unsigned 型变量所需的数据，可以用%u，%d 或%o，%x 格式输入。

(8) 输入数据时不能规定精度，例如，

```
scanf("%7.2f",&a);
```

是不合法的，不能企图用这样的 scanf 函数输入数据。

3.6　编译预处理

编译预处理是在编译前对源程序进行的一些预加工。C 语言提供了三种预处理功能：宏定义、文件包含和条件编译。这些命令以符号＃开头，它不是 C 语句的组成部分。

3.6.1　宏定义

宏定义是将一个标识符定义为一串符号，被定义的标识符称为宏名，而这一串符号称为宏体。宏定义有两种形式：不带参数的宏定义和带参数的宏定义。

1. 不带参数的宏定义

定义形式：

```
#define 宏名　一串字符串(宏体)
```

宏名与宏体之间应以空格相隔。对程序中用双引号括起来的字符串内的字符，即使与宏名相同，也不进行置换。宏定义只作字符替换，不分配内存空间，不占编译时间。

注意：宏名中不能含有空格。宏名一般习惯用大写字母表示，以便与变量相区别。宏定义只是简单的字符串置换，不作正确性检查。宏替换不是 C 语句，不必在行末加“;”。

例 3.3　不带参数的宏定义应用。

相应的程序代码如下：

```
#include "stdio.h"
#define M 8
#define N M+M
#define N1(M+M)
main()
{  int k,k1;
   k=N*N*2;
   k1=N1*N1*2;
   printf("k=%d,k1=%d\n",k,k1);
}
```

该程序的运行结果为：

```
k=88,k1=512
```

在本例中，会发现使用宏名代替一个字符串，可以减少程序中重复书写这些字符串。宏替换时要宏体去替换宏名，往往使程序体积膨胀，加大了系统的存储开销。用宏名代替一个字符串，只做简单置换，不作正确性检查。进行宏定义时，可以引用已定义的宏名，可以层层置换。

2. 带参数的宏替换

定义形式：

```
#define  宏名(形参数表)  一串字符串
```

当程序中如有带实参的宏，则按＃define 命令行中指定的字符串从左到右进行置换。若字符串中包含有宏中的形参，则将程序语句中相应的实参代替形参，若宏定义中的字符串中的字符不是参数字符，则保留。

例 3.4 带参数的宏定义应用。

相应的程序代码如下：

```
#include "stdio.h"
#define N 8
#define M N+1
#define f(x)(x*M)
main()
{  int i1,i2;
   i1=f(2);
   i2=f(1+1);
   printf("%d  %d\n",i1,i2);
}
```

该程序的运行结果为：

```
17  10
```

字符串中的各个形参应该用圆括号括起来，以避免将实参代替形参时产生错误。如：

```
#define s(a,b) a*b
area=s(a+b,c+d);
```

则展开后：area＝a＋b＊c＋d；显然与原意不符。如改为＃define s(a,b)(a)＊(b)后，则展开后：area＝(a＋b)＊(c＋d)；正确。

3.6.2 “文件包含”处理

“文件包含”命令是很有用的，它可以节省程序设计人员的重复劳动。所谓“文件包含”处理是指一个源文件可以将另外一个源文件的全部内容包含进来，即将另外的文件包含到本文件之中。

C 语言提供了＃include 命令用来实现“文件包含”的操作。其一般形式为：

```
#include "文件名"
```

或

```
#include<文件名>
```

其中的“文件名”就是待包含的文件给定的函数库一般称为“头文件”，其扩展名为 h，用户自编的包含文件也可以用 c 或 cpp 为扩展名。

说明：

(1) 所包含的文件名必须是完整的文件名，扩展名不可以省略，也可以包含路径。例如 #include "D:\\c\\type.h"代表文件在 D 盘 C 目录下，文件名是 type.h。在含路径时，路径分隔符要双写。

(2) 在 #include 命令中，文件名可以用双撇号或尖括号括起来，并且都是合法的。二者的区别是尖括弧时，系统到存放 C 库函数头文件所在的目录中寻找要包含的文件，这称为标准方式。用双撇号时，系统先在用户当前目录中寻找要包含的文件，若找不到，再按标准方式查找(即再按尖括号的方式查找)。

(3) 在一个被包含文件中又可以包含另一个被包含文件，即文件包含是可以嵌套的。

(4) 一个 include 命令只能指定一个被包含文件，如果要包含 n 个文件，要用 n 个 include 命令。

例 3.5　“文件包含”的应用。

相应的程序代码如下：

```
/*powers.h*/
#define sqr(x)  ((x)*(x))
#define cube(x) ((x)*(x)*(x))
#define quad(x) ((x)*(x)*(x)*(x))
/*主文件代码 */
#include<stdio.h>
#include "power.h"
#define MAX_POWER 10
void main()
{  int n;
   printf("number\t exp2\t exp3\t exp4\n");
   printf("----\t----\t-----\t------\n");
   for(n=1;n<=MAX_POWER;n++)
     printf("%2d\t %3d\t %4d\t %5d\n",n,sqr(n),cube(n),quad(n));
}
```

该程序的运行结果为：

```
number    exp2     exp3      exp4
 ----      ----    -----    ------
  1          1        1         1
  2          4        8        16
  3          9       27        81
```

```
 4        16       64      256
 5        25      125      625
 6        36      216     1296
 7        49      343     2401
 8        64      512     4096
 9        81      729     6561
10       100     1000    10000
```

3.6.3 “条件编译”处理

一般情况下,源程序中所有的行都参加编译。但有时希望对其中一部分内容只在满足一定条件下才进行编译,即对一部分内容指定编译的条件,这就是“条件编译”。

格式 1:

```
#ifdef  标识符
  程序段 1
[#else
  程序段 2]
#endif
```

当前所指定的标识符已经被 # define 命令定义过,则在程序编译阶段只编译程序段 1,否则编译程序段 2。两个程序段必具其一。其中 # else 部分可以没有。

格式 2:

```
#ifndef  标识符
  程序段 1
#else
  程序段 2
#endif
```

当前所指定的标识符未被 # define 命令定义过,则在程序编译阶段只编译程序段 1,否则编译程序段 2。其中 # else 部分可以没有。

格式 3:

```
#if  表达式
  程序段 1
#else
  程序段 2
#endif
```

当指定的表达式值为真(非零)时就编译程序段 1,否则编译程序段 2。其中 # else 部分可以没有。

注意:

(1) 条件编译可以根据对标识符或表达式的判断来决定是否执行程序段 1 或程序段