第 3 章

顺序结构

程序设计有 3 种基本结构，分别是顺序结构、分支结构和循环结构。使用这 3 种结构可以编写任何复杂的程序。本章介绍顺序结构程序设计。顺序结构程序设计是指在程序的执行过程中，各条语句按照出现的先后顺序依次执行，并且只执行一次，中间没有中断、分支和重复。它是程序设计 3 种基本结构中最简单的一种。

3.1 C语言语句

根据 C 语言语句的构造和功能的不同，可以分为函数调用语句、表达式语句、控制语句、空语句和复合语句 5 种。

1. 函数调用语句

C 语言的基本单位是函数，C 语言利用函数体中的语句向计算机系统发出执行命令。函数调用语句由函数名、实际参数加上分号“;”组成。其一般形式如下。

```
函数名(实际参数表);
```

例如：

```
printf("Hello world!");      //调用输出函数，输出字符串"Hello world!"
pow(2,3);                    //调用幂函数，计算2的3次方
```

函数调用语句的具体内容将在第 6 章中进行详细介绍。

2. 表达式语句

表达式语句由表达式加上“;”组成，其一般形式如下。

```
表达式;
```

例如，“s=1+2;”是一条赋值语句；“1+2;”是加法运算语句，但计算结果不能保留，这样的语句对程序而言没有任何意义。

3. 控制语句

控制语句用于完成一定的控制功能，可以控制程序的流程。C 语言有 9 种控制语句，可

分为以下 3 类，具体将在后面的章节中进行详细介绍。

（1）分支结构控制语句：if 语句及其变形、switch 语句。

（2）循环结构控制语句：while 语句、do…while 语句、for 语句。

（3）其他控制语句：return 语句等。

4. 空语句

空语句只由分号构成，表示什么操作都不执行。

程序中出现空语句时，起占位作用，如可以用来作为空循环体。需要注意的是，空语句虽然什么都不做，但是在程序（尤其是分支结构和循环结构）中随意增加空语句，会改变程序的流程。对于初学者来说，尤其要注意这一点。

5. 复合语句

复合语句是使用大括号把许多语句和声明组合到一起形成单条语句。

例如：

```
{
    s=s+i;
    i=i+1;
}
```

这里的“{}”括起来的是一条复合语句。

关于复合语句，需要注意以下几点。

（1）在语法上，复合语句被看作单条语句，而不是多条语句。

（2）复合语句内的各条语句都必须以分号结尾，在“}”外不能加分号。

（3）复合语句可以嵌套，即复合语句中也可以包含一个或多个复合语句。

3.2　数据的输出和输入

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，把计算机的运算结果显示在屏幕上或打印在纸上。从计算机外部设备将数据送入计算机内部的操作称为“输入”。例如，通过键盘、鼠标等输入设备把数据输入到计算机中。

C 语言本身不提供用于输入和输出的语句，在程序中，可通过调用标准库函数提供的输入和输出函数来实现数据的输入和输出。调用标准库函数时要用到“stdio.h”头文件，因此源文件开头应该包含以下预处理命令。

```
#include< stdio.h >   //或者#include"stdio.h"
```

这里的 stdio 是 standard input & output 的意思。

3.2.1　printf( )函数和 scanf( )函数

C 标准函数库中包含了多个输出、输入函数。其中使用最多、最灵活的函数是 printf( )函数和 scanf( )函数。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不加#include<stdio.h>命令行。

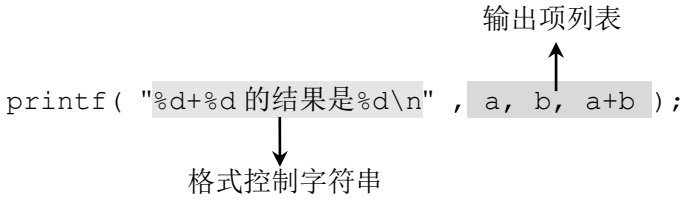
1. printf( )函数

其一般形式如下。

```
printf("格式控制字符串",输出项列表);
```

作用：程序严格按照格式控制字符串中的说明将输出项列表逐一输出。

示例 1：



（1）格式控制字符串：必须用双引号括起来，用于指定输出的内容及其格式，如示例 1 中的第一个逗号之前的字符串。格式控制字符串可以包含下列 3 种字符。

① 格式控制符：以%开头，以规定的某个字母结束的字符。格式控制符本身并不输出，它的作用是指定其所在位置的输出项的输出形式，例如，“%d”表示按十进制整型输出，“%c”表示按字符型输出等。示例 1 中有 3 个格式控制符，都是%d。表 3-1 所示为常用的格式控制符。

② 转义字符：以\开头，以规定的某个字母结束的字符，详见第 2 章中关于转义字符的介绍，转义字符通常用来控制光标的位置，使输出结果更清晰。示例 1 中有 1 个转义字符“\n”。

③ 普通字符：除格式控制符和转义字符之外的其他字符。普通字符输出时是原样输出的，起到了提示作用，使程序更清晰易懂。示例 1 中的“+”“的结果是”都是普通字符，需要注意的是，此处的“+”是普通字符，而不是算术运算符，不做加法运算，是需要原样输出的。

表 3-1 常用的格式控制符

格式控制符	说　明	举　例	输出结果(_代表空格)

%d	以十进制形式输入/输出一个整数	printf("%d",10);
		printf("%d",-10);
		printf("%d ",'a');
			10
			−10
			97

%u	按无符号十进制形式输出	printf(" %u",10); printf("%u ",'a');	10 97
%md	m 表示指定输出数据的宽度	printf("%5d",10); printf("%-5d ",'a');	_ _ _10 97_ _ _

%o	按无符号八进制形式输出	printf("%o",10);
		printf("%#o",10);
		printf("%o",'a');
			12
			012
			141

%x	按无符号十六进制形式输出	printf("%x",10);
		printf("%#x",10);
		printf("%x ",'a');
			a
			0xa
			61

%c	输出/输入一个字符	printf(" %c",10); printf("%c ",'a');	换行 a
%f %lf	以小数形式输出/输入实数（单精度） 以小数形式输出/输入实数（双精度）	printf("%f",3.1415926); printf("%lf",3.1415926);	3.141593 3.141593

续表

格式控制符	说　　明	举　　例	输出结果(_代表空格)
%m.nf	m 用于指定输出数据总的宽度，n 用于指定输出数据的小数位数	printf("%5.2f",3.1415926); printf(" %.2f",3.1415926);	_3.14 3．14
%e	以指数形式输出，小数位数由精度决定	printf(" %e",314.15); printf(" %e",0.314);	3.141500e+002 3.140000e-001
%s	输出/输入一个字符串	printf("%s","Hello");	Hello
%%	输出/输入一个%	printf("%%");	%

（2）输出项列表：即需要输出的数据，它的形式可以是常量、变量或表达式等。输出项的个数与格式控制符的个数是对应的，多个输出项之间用逗号隔开。示例 1 中有 3 个输出项，分别是变量 a、变量 b 和表达式 a+b。有时如果只是用来输出一些提示信息，则可以没有输出项，如 printf("Hello world!");。

示例 1 中，由于有 3 个输出项，所以在格式控制字符串中出现了 3 个格式控制符，它们一一对应，即第 1 个%d 修饰 a，第 2 个%d 修饰 b，第 3 个%d 修饰 a+b。

如果格式控制符的个数多于输出项的个数，那么会有一些随机值输出。例如：

```
int x=10,y=5;执行printf("%d,%d,%d",x,y);
```

格式控制符有 3 个“%d”，而输出项只有 x、y 两项，则输出结果为 10,5,0023342。其中，0023342 为随机值。

反之，如果格式控制符的个数少于输出项的个数，则多余的输出项不予输出。例如，执行“printf("%d",x,y);”时，由于只有一个%d，而输出项有 x、y 两项，因此输出结果为 10，多余项 y 不输出。

假设有 a=3，b=6；则执行 printf("%d+%d 的结果是%d\n", a, b, a+b)，其输出结果是“3+6 的结果是9”。

如果改成 printf("%d,%d,%d",a,b,a+b)，则其输出结果是“3,6,9”。

如果改成 printf("%d%d%d",a,b,a+b)，则其输出结果是“369”。

显然，后面两种的输出结果不如第一种的输出结果所要表达的意思清晰。因此，在使用 printf( )函数时，可以适当地加上一些普通字符，使得结果的输出形式更加丰富、更加人性化。

下面通过具体的实例进一步介绍 printf( )函数使用时的注意事项。

【例 3.1】格式控制符的使用。

1）源程序代码(demo3_1.c)

```
#include<stdio.h>
main( )
{
    int i=2;
    char c='H';
    float x=3.14;
    printf("i=%d,c=%c,b=%s,x=%f\n", i,c,"ABCD",x);
}
```

2）程序运行结果

```
i=2,c=H,b=ABCD,x=3.140000
```

3）程序说明

%f 修饰输出项时，默认保留 6 位小数，因此 x 的值为 3.140000。

【例 3.2】格式控制符的使用。

1）源程序代码（demo3_2.c）

```
#include<stdio.h>
main( )
{
    int i=12,j=32;
    char c='H';
    float x=3.14;
    printf("i=%o,j=%x,c=%u,x=%e\n", i,j,c,x);
}
```

2）程序运行结果

```
i=14,j=20,c=72,x=3.140000e+000
```

3）程序说明

在程序中，通过%o、%x 使得 i、j 的值（12 和 32）在输出时分别以八进制、十六进制格式（14，20）输出。但从运行结果来看，很容易误认为 14、20 是十进制数。因此，可以分别在%o 和%x 中间加上“#”，使得输出结果分别加上数字 0 和符号 0x 来标识八进制和十六进制。例如：

```
printf("%o,%#o,%x,%#x\n",10,10,10,10)
```

其运行结果如下。

```
12,012,a,0xa
```

【例 3.3】格式控制符的使用。

1）源程序代码(demo3_3.c)

```
#include<stdio.h>
main( )
{
    float  x=1234.567;
    double y=1234.5678;
    printf("x=%f,y=%f\n", x,y);
    printf("x=%6.3f,y=%10.3f\n", x,y);
}
```

2）程序运行结果

```
x=1234.567017,y=1234.567800
x=1234.567,y=  1234.568
```

3）程序说明

%6.3f 对应输出项 x，表示输出 x 的值时，在屏幕上占据的宽度是 6 位，其中小数部分占 3 位。因此，%m.nf 表示所修饰的输出项在输出时总共在屏幕上占 m 位，其中小数部分占

n 位。如果指定的输出宽度不够，则按数据的实际宽度输出，如果指定的输出宽度多于数据实际宽度，则数据默认右对齐，左边补空格。例如：

```
printf("%5f",123.54); 123.540000      //指定宽度不够输出宽度，按实际宽度输出
printf("%12f",123.54); _ _123.540000  //指定总宽度为12，前面输出2个空格
printf("%8.1f",123.54);_ _ _123.5     //指定总宽度为8，小数位为1位
printf("%8.3f",123.54);_123.540       //指定总宽度为8，小数位为3位
printf("%8.0f",123.54);_ _ _ _ _124   //指定总宽度为8，小数位为0位
```

另外，可在 m 前加“-”来使输出数据左对齐，例如：

```
printf("%6d##\n",123)            _ _ _123##
printf("%-6d##\n",123)           123_ _ _##
printf("%14.8f##\n",1.3455)      _ _ _ _1.34550000##
printf("%-14.8f##\n",1.3455)     1.34550000_ _ _ _##
```

格式修饰符%md 也是如此。

【例 3.4】错误的格式化输出。

1）源程序代码（demo3_4.c）

```
#include<stdio.h>
main( )
{
    int a=10,b=100;
    printf("a=%d,b=%d\n",a*1.0,b);
    printf("a=%f,b=%d\n",101,b);
}
```

2）期望的程序运行结果

```
a=10.000000,b=100
a=101,b=100
```

3）实际的程序运行结果

```
a=0,b=1076101120
a=0.000000,b=4198912
```

4）程序说明

在第 1 条输出语句中，输出 double 型数据 a*1.0，却使用了%d，因此，不会正常输出 10.000000，并会影响到下一个表达式的输出。

在第 2 条输出语句中，输出 int 型数据 101，却使用了%f，因此，不会正常输出 101，同时会影响到下一个表达式的输出。

修改方法：将第一条输出语句中的 a=%d 改为 a=%f，将第二条输出语句中的 101 改为 101.0 或者将 a=%f 改为 a=%d。

因此，printf( )函数中的输出项列表中的各项要与格式控制符相适应，且格式控制项与输出列表中的项数要一致，否则会出现不可预见的错误。

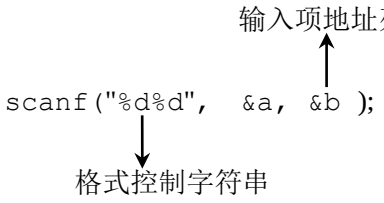
2. scanf( )函数

scanf( )函数用来接收从键盘上输入的数据，并可将其转换为各种形式，如整型、字符型、实型等。其一般形式如下。

```
scanf("格式控制字符串",输入项地址列表);
```

作用：scanf( )函数是格式化输入函数，要求程序的执行者必须严格按照“格式控制字符串”中规定的格式从键盘上输入数据。scanf( )函数会将这些数据依次存入对应变量的地址。

例如：



（1）格式控制字符串主要包含两种字符。

① 格式控制符：指定数据的输入格式，如%d，它和 printf( )函数中使用的几乎一样。其主要区别在于%f 和%lf。在输出时，%f 可同时用于 float 类型和 double 类型的数据，而在输入时，%f 只能用于 float 类型，double 类型则应用%lf。

例如：

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

此时，应该输入一个十进制整数，如果输入的数据是 5，则 5 会被存入变量 a 中。

如果输入的数据是实数 5.6，则只接收整数部分，舍弃小数部分。因此，变量 a 中的值为 5。

② 普通字符：如果格式控制字符串中有普通字符，那么输入数据时，必须原样输入，否则会出错。

例如：

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

如果要使变量 a 正确收到数据 10，则在输入数据时，只能按照下列形式输入。

```
a=10 ↙
```

其他任何形式的输出都会使得变量 a 得不到正确的数据。

因此，在 scanf( )函数中要慎重使用普通字符，以免造成麻烦。如果想增强程序的可读性，则可以在 scanf( )函数前调用 printf( )函数以对输入进行说明。

例如：

```
int a;
printf("请输入变量a的值：\n");
scanf("%d",&a);
```

另外，在 scanf( )函数中，对于格式控制字符串内的转义字符，系统并不把它当作转义字符来解释，而是将其视为普通字符，也会原样输入，因此应避免在 scanf( )函数中使用转义字符。

例如：

```
int a;
scanf(“%d\n”,&a);
```

如果要使变量 a 正确收到数据 10，则在输入数据时只能按照下列形式输入。

```
10\n↙
```

（2）输入项地址列表：用来指定输入数据的存储地址。如果有多个输入数据，则应在输入项地址列表中指定多个相应的变量地址，且地址之间用逗号隔开。输入项地址列表中的地址可以是变量的地址，也可以是字符数组名或者指针变量（将在后面章节中介绍）。变量地址的表示方法是&变量名，如“&a”，其中“&”是取地址运算符，“a”是普通变量。

下面的程序段表示接收两个数据并存储在变量 a 和 b 中。

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

如果改成下面的形式，则程序不会有语法错误，但运行结果是不可预知的。

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

因为调用 scanf( )函数时，变量 a 和 b 前面都漏掉了取地址运算符“&”，所以初学者要牢记一点：使用 scanf( )函数时，输入项地址列表的变量前面一定要加上运算符“&”。

【例 3.5】scanf( )函数的使用。

1）源程序代码(demo3_5.c)

```
#include<stdio.h>
main( )
{
    int i,j,k;
    printf("请输入数据: ");
    scanf("%d,%d,%d",&i,&j,&k);
    printf("i=%d,j=%d,k=%d\n",i,j,k);
}
```

2）程序运行结果

① 第 1 次运行（正确的输入）：

```
请输入数据: 3,4,5↙
i=3,j=4,k=5
```

② 第 2 次运行（错误的输入）：

```
请输入数据: 3 4 5↙
i=3,j=-858993460,k=-858993460
```

3）程序说明

第 2 次运行时，除 i=3 被正确赋值外，对 j 和 k 的赋值都将以失败告终。因为 scanf( )中含有普通字符“,”，必须原样输入。

下面把例 3.5 中的 scanf( )改为如下形式。

```
scanf("%d%d%d",&i,&j,&k);
```

由于格式控制字符串中没有普通字符，因此在输入时数据之间没有任何间隔。例如，345↙，如果是这种形式的输入，则计算机会把 345 当作一个整体接收，并赋值给变量 i。而 j、k 的值会等待下一次数据的输入。