第4章 C程序的流程设计

本章要点

(1) C 程序流程的三种基本结构。
(2) 输入输出函数的使用。
(3) 结构化程序设计方法与算法。
(4) if 语句的流程控制及 if 语句的嵌套使用。
(5) 条件运算符和条件表达式。
(6) switch 语句的使用。
(7) 三种循环语句的使用及其嵌套。

学习目标

(1) 掌握使用输入输出函数和其他语句进行顺序程序设计。
(2) 掌握 C 语言的逻辑表达式和关系表达式。
(3) 学会使用三种选择结构语句。
(4) 了解条件运算符。
(5) 掌握 switch 语句的使用。
(6) 掌握使用三种结构的循环控制语句。
(7) 学会使用循环的嵌套及相关语句。

从流程的角度，可以把程序分为三种基本结构：顺序结构、分支结构、循环结构。理论上已经证明，无论多么复杂的程序，都可以使用这三种基本结构来编制。C 语言提供了多种语句来实现这些程序结构。本章拟介绍这些基本语句及其在三种基本结构中的应用，使读者对 C 语言程序的基本结构有一个系统的认识，为后续各章的学习打下基础。

4.1 C 语句概述

C 语言程序的结构在第 1 章的 1.7 节已给出，图 4-1 给出的是更加详细的结构。

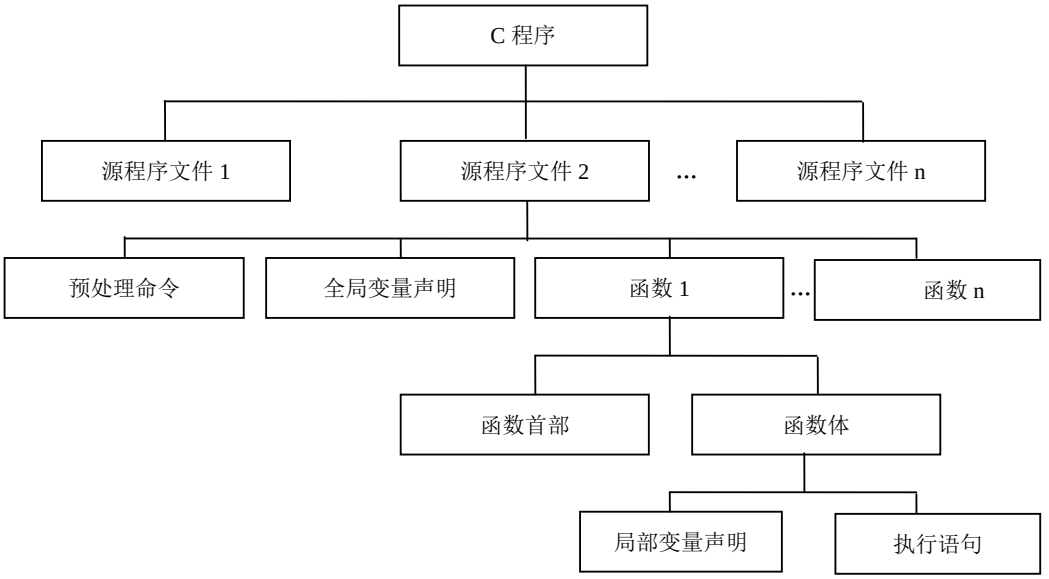


图 4-1　C 语言程序的详细结构

一个程序可以由多个源程序文件组成，一个源文件包括编译预处理命令、全局变量声明(二者均可以没有)以及一个或多个函数的定义。其中每个函数包括函数首部和函数体，函数体又包括局部变量声明和执行语句。

C 语言程序的实质性部分是其执行部分，一般是由多条执行语句组成的，程序的功能就是通过这些执行语句来实现的。

C 语言的语句可分为以下 5 类。

(1) 表达式语句。

(2) 函数调用语句。

(3) 控制语句。

(4) 复合语句。

(5) 空语句。

1. 表达式语句

前已提及，表达式语句是由表达式加上分号“;”组成的，其一般形式为：

```
表达式;
```

执行表达式语句，实际上就是计算表达式的值。例如：

```
x = y+z;     /*赋值语句*/
y+z;         /*加法运算语句(无实际意义，计算结果不能保留，编译虽无错，但给出警告)*/
3;           /*常数是表达式的特例，加分号也构成语句，虽无意义，但编译能通过*/
i++;         /*自增1语句，使i本身的值增1*/
```

2. 函数调用语句

函数调用语句由函数名、实参(实际参数)加上分号“;”组成，其一般形式为：

```
函数名(实参表);
```

执行函数语句就是调用该函数，并把实参赋给函数定义中的形参(形式参数)，然后执行被调函数中的语句，求得函数值(在第 6 章中详细介绍)。

例如：

```
printf("C Program");
```

该函数调用语句的功能，是调用库函数 printf，输出字符串“C Program”。

3. 控制语句

控制语句用于控制程序的执行流程，以实现程序的各种结构，它们由特定的语句定义符组成。

C 语言有 9 种控制语句，可以分成以下三类。

(1) 条件判断语句：if 语句、switch 语句。

(2) 循环执行语句：do while 语句、while 语句、for 语句。

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

4. 复合语句

把多个语句用括号{}括起来组成的一个语句组，称为复合语句。

C 语言把复合语句视为单条语句，而不是多条语句。

例如：

```
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

这就是一条复合语句，用于将 a、b 两个整型变量的内容交换。

复合语句内的各条语句都必须以分号“;”结尾，但在右括号}外不能加分号。

复合语句内可以定义变量。

5. 空语句

只由分号“;”组成的语句称为空语句。空语句什么也不执行，在程序中可用来作空循环体。

例如：

```
while(getchar() != '\n')
    ;
```

本语句的功能是，只要从键盘上输入的字符不是回车符，就重新输入。这里的循环体使用了空语句。

4.2 赋值语句

赋值语句是由赋值表达式加上分号而构成的表达式语句。其一般形式为：

```
变量 = 表达式;
```

赋值语句的功能和特点都与赋值表达式相同，它是程序中使用最多的语句。

使用赋值语句时，需要注意以下几点。

(1) 因为赋值运算符“=”右边的表达式也可以是另一个赋值表达式，所以：

```
变量1 = (变量2 = 表达式);
```

是合法的，从而形成赋值的嵌套。其展开之后的一般形式为：

```
变量1 = 变量2 = …… = 表达式;
```

例如：

```
x=y=z=6;
```

由赋值运算的右结合性可知，上面的这条赋值语句实际上等效于下面的三条语句：

```
z = 6;
y = z;
x = y;
```

事实上，这一点也体现了 C 语言的简洁性(很多其他语言并不支持此类赋值操作)。

(2) 注意变量初始化和赋值语句的区别。

变量初始化(定义变量时为变量赋初值)是变量说明的一部分，赋初值后的变量与其后的其他同类变量之间仍必须用逗号间隔，而整个赋值语句则必须用分号结束。例如：

```
int x=6, y, z;
```

(3) 在变量说明中，不允许连续给多个变量赋初值。

例如，下面的变量说明语句是错误的：

```
int x=y=z=6;      /* 错误! */
```

必须写为：

```
int x=6, y=6, z=6;
```

而赋值语句却允许连续赋值。

请读者注意，写成如下形式是可以的：

```
int y, z, x=y=z=5;
```

这是因为，x 被赋予了一个(赋值)表达式作为其初值，语法上并没有问题。

(4) 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，下面的语句是合法的：

```
if((x=y+5)>0) z=x;
```

语句的功能是，若表达式 x=y+5 大于 0，则将 x 值赋给 z。

但下面的语句是非法的：

```
if((x=y+5;)>0) z=x;    /* 错误! */
```

因为“x=y+5;”是语句，不能出现在表达式中。

4.3 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) 所谓输入/输出，是相对计算机主机而言的。

(2) C 语言无输入/输出语句，所有的数据输入/输出均由相应的库函数实现，因此都是函数语句。

(3) 本章介绍的输入函数是从标准输入设备(键盘)输入数据。

(4) 本章介绍的输出函数是向标准输出设备(显示器)输出数据。

(5) 在使用 C 语言库函数时，要用编译预处理命令“#include”将有关的“头文件”包含到源文件中。

使用标准输入/输出库函数时，要用到 stdio.h 文件，因此源文件开头应有以下编译预处理命令：

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

或者：

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

其中，stdio 是 standard input & output(标准输入/输出)的意思。

在大部分 C 编译系统下，考虑到 printf 和 scanf 两个函数使用频繁，所以系统允许在使用这两个函数时不加上述编译预处理命令。

4.4 字符数据的输入输出

4.4.1 putchar 函数(字符输出函数)

putchar 函数是单个字符输出函数，其功能是在屏幕上输出单个字符。

其一般形式为：

```
putchar(字符变量);
```

例如：

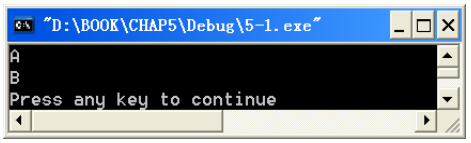
```
putchar('C');        /* 输出大写字母 C */
putchar(c);          /* 输出字符变量 c 的值 */
putchar('\103');     /* 输出字符 C */
```

对可视字符直接输出，对控制字符则直接执行控制功能，不在屏幕上显示。例如：

```
putchar('\n');       /* 换行 */
putchar('\a');       /* 响铃 */
```

【例 4.1】输出单个字符：

```
#include <stdio.h>
main()
{
    int c;
    char a;
    c=65;
    a='B';
    putchar(c);
    putchar('\n');
    putchar(a);
    putchar('\n');
}
```



正常输出时，函数返回值为显示的代码值；出错时返回 EOF(即−1)。

4.4.2 getchar 函数(键盘输入函数)

getchar 函数是字符输入函数，其功能是从键盘上输入一个字符，一般形式为：

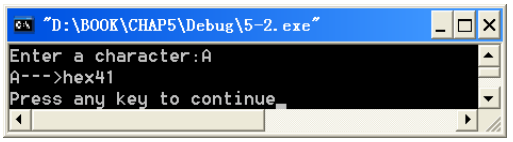
```
getchar();
```

通常把输入的字符赋给一个字符变量，构成赋值语句，例如：

```
char c;
c = getchar();
```

【例 4.2】输入单个字符并输出：

```
#include <stdio.h>
main()
{
    int c;
    printf("Enter a character:");
    c = getchar();
    printf("%c--->hex%x\n", c,c);
}
```



使用 getchar 函数还应注意几个问题。

(1) getchar 函数只接受单个字符，输入数字也按字符处理。输入多于一个字符时，只接收第一个字符。

(2) 正常输入时，本函数返回读取的代码值；出错时返回 EOF(−1)。

(3) 使用本函数前，应包含 stdio.h 文件。

(4) 在 C 语言集成编辑环境下运行本程序时，将临时退出 C 集成编辑环境，进入用户屏幕，等待用户输入。输入完毕后显示输出结果，按任意键后，返回集成编辑环境。

(5) 若仅仅要显示所输入的字符，则可用下面两行的任意一行语句：

```
putchar(getchar());
printf("%c", getchar());
```

4.5 格式输入与输出

4.5.1 printf 函数(格式输出函数)

printf 函数是 C 语言中使用最广泛的输出函数，称为格式输出函数。printf 的最末一个字母 f 即为“格式”(format)之意。该函数的功能是按用户指定的输出格式，将指定的数据显示到屏幕上。前面的例题中已多次使用过该函数。

1. printf 函数调用的一般形式

printf 函数是一个标准库函数，它的函数原型在头文件 stdio.h 中，一般编译系统要求在使用 printf 函数前，必须包含这一头文件。printf 函数调用的一般形式如下：

```
printf("格式控制字符串", 输出列表);
```

其中的“格式控制字符串”用于指定输出格式。格式控制串由格式字符串和非格式字符串两种元素组成。格式字符串是以%开头的一个字符串，在%后面跟有各种格式字符，用以说明输出数据的类型、形式、长度、小数位数等。

(1) %d：表示按十进制整型格式输出。

(2) %ld：表示按十进制长整型格式输出。

(3) %c：表示按字符型格式输出。

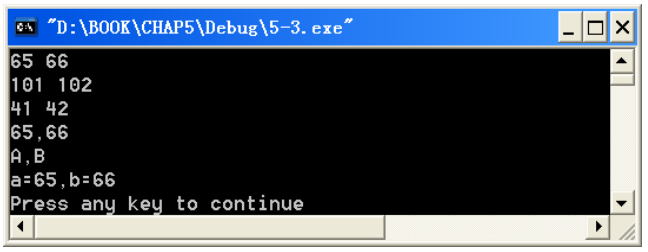
非格式字符串在输出时照原样显示，在显示中起提示作用。

在“输出列表”中，给出了各个输出项，格式字符串和各个输出项在数量和类型上要求一一对应。

【例 4.3】输出单个字符：

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

```
    int a=65, b=66;
    printf("%d %d\n", a,b);
    printf("%o %o\n", a,b);
    printf("%x %x\n", a,b);
    printf("%d,%d\n", a,b);
    printf("%c,%c\n", a,b);
    printf("a=%d,b=%d\n", a,b);
}
```



本例中 6 次输出了 a、b 的值，但由于格式控制串的不同，每次输出的结果也不尽相同。前 3 行分别以十进制、八进制、十六进制格式输出，第 5 行以字符格式输出。前 3 行的输出语句格式控制串中，两格式串%d 之间加了一个空格(非格式字符)，所以输出的 a、b 值之间有一个空格。第 4 行的 printf 语句格式控制串中加入了非格式字符——逗号，因此输出的 a、b 值之间加了一个逗号。第 6 行中，为了提示输出结果，又增加了非格式字符串，这些字符串均照原样输出。

2. 格式字符串

格式字符串的一般形式如下：

```
%[标志][输出最小宽度][.精度][长度]类型
```

其中，用方括号括起来的项为可选项。

各项的意义如下。

(1) 类型：类型用单个字母标识，用以表示输出数据的类型，其格式符和意义如表 4-1 所示。

表 4-1 printf 输出函数的格式字符及含义

格式字符	意 义
d, i	以十进制形式输出带符号整数(正数不输出符号)
o	以八进制形式输出无符号整数(不输出前缀 0)
x, X	以十六进制形式输出无符号整数(不输出前缀 0x)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
e, E	以指数形式输出单、双精度实数
g, G	以%f 或%e 中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

(2) 标志：标志字符为-、+、#和空格 4 种，其意义如表 4-2 所示。

表 4-2 printf 输出函数的标志字符及含义

标 志	意 义
-	结果左对齐，右补空格
+	输出符号(正号或负号)
空格	输出值为正时冠以空格，为负时冠以负号
#	对 c、s、d、u 类无影响；对 o 类，在输出时加前缀 o；对 x 类，在输出时加前缀 0x；对 e、g、f 类，当结果有小数时，才给出小数点

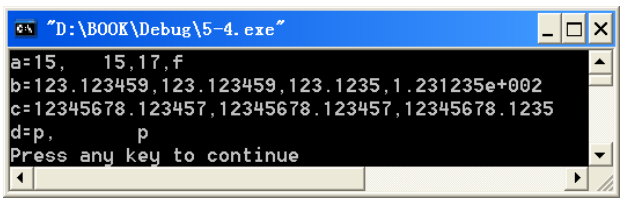
(3) 输出最小宽度：用十进制整数来表示输出所占的最少位数。若实际位数多于给定的宽度，则按实际位数输出，若实际位数少于给定的宽度，则补以空格或 0。

(4) 精度：精度格式符以“.”开头，后跟十进制整数。其意义是：如果输出数字，则表示小数的位数；如果输出字符，则表示输出字符的个数；若实际位数大于给定的精度，则截去超过的部分。

(5) 长度：长度格式符有 h、l 两种，h 表示按短整型量输出，l 表示按长整型量输出。

【例 4.4】格式输出：

```
#include <stdio.h>
main()
{
    int a = 15;
    float b = 123.1234567;
    double c = 12345678.1234567;
    char d = 'p';
    printf("a=%d,%5d,%o,%x\n", a,a,a,a);
    printf("b=%f,%lf,%5.4lf,%e\n", b,b,b,b);
    printf("c=%lf,%f,%8.4lf\n", c,c,c);
    printf("d=%c,%8c\n", d,d);
}
```



本例的程序从第 8 行开始，以 4 种格式输出整型变量 a 的值，其中“%5d”要求输出宽度占 5 位，而 a 值为 15，只有两位，故左补三个空格。第 9 行以 4 种格式输出实型量 b 的值，其中“%f”和“%lf ”格式的输出相同，说明“l”符对“f”类型无影响。“%5.4lf”指定输出宽度占 5 位，精度为 4，由于实际宽度超过 5，故按实际位数输出，小数位数超过 4 位部分被截去。第 10 行输出双精度实数，由于“%8.4lf”指定精度为 4 位，故截去了超过 4 位的部分。第 11 行输出字符量 d，其中“%8c”指定输出宽度为 8，故输出字符 p 时，左补 7 个空格。

使用 printf 函数时还要注意一个问题，那就是输出列表中的求值顺序。C 标准并未规定这一顺序，故不同的编译系统求值顺序不一定相同，可以从左到右，也可以从右到左。TC、VC 都是按从右到左的顺序进行求值的。

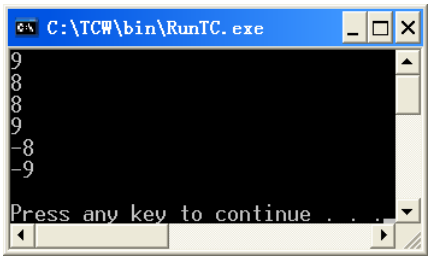
【例 4.5】函数调用时的参数传递顺序：

```
#include <stdio.h>
main()
{
    int i = 8;
    printf("%d\n%d\n%d\n%d\n%d\n%d\n", ++i,--i,i++,i--,-i++,-i--);
}
```

由于函数调用时参数的求值顺序(确切地说，是函数参数的传递顺序)并非 C 语言本身的内容，与具体的编译系统有关，故考试命题时，多回避此类题目。

【例 4.6】++，--应用举例：

```
#include <stdio.h>
main()
{
    int i = 8;
    printf("%d\n", ++i);
    printf("%d\n", --i);
    printf("%d\n", i++);
    printf("%d\n", i--);
    printf("%d\n", -i++);
    printf("%d\n", -i--);
}
```



4.5.2 scanf 函数(格式输入函数)

scanf 函数是 C 语言中使用最广泛的输入函数，称为格式输入函数，即按用户指定的格式从键盘上将数据输入到指定的变量中(更确切地说，是放在指定的地址处)。

1. scanf 函数的一般形式

scanf 函数是一个标准库函数，它的函数原型在头文件 stdio.h 中。与 printf 函数类似，一般的 C 编译系统也允许在使用 scanf 函数前不必包含 stdio.h 文件。

scanf 函数的一般形式为：

```
scanf("格式控制字符串", 地址列表);
```

其中，格式控制字符串的作用与 printf 函数相同，非格式字符串的作用与 printf 函数相对(printf 中的非格式字符串照原样输出，而 scanf 中的非格式字符串必须照原样输入)。地址列表中给出各变量的地址，地址是由变量名冠以地址运算符“&”组成的。例如，&a 和&b 分别表示变量 a 和 b 的地址。

&a 和&b 就是编译系统给 a、b 两个变量分配的内存地址。C 语言中使用了“地址”这个概念，这是与其他语言不同的。应该区分变量的值和变量的地址这两个不同的概念。变量的地址是 C 编译系统为变量分配的，用户不必关心具体的地址是多少。

变量的地址和变量值的关系说明如下。

如果在赋值表达式中给变量赋值，例如：

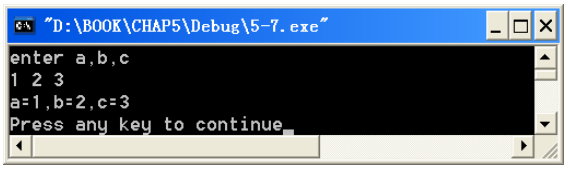
```
a = 123;
```

则 a 为变量名，123 是变量的值，&a 是变量 a 的地址(该地址可以使用 printf 函数输出)。

但在赋值号的左边只能写变量名，不能写地址。scanf 函数在本质上也是给变量赋值，但要求写变量的地址，如&a。这两者在形式上是不同的。&是一个取地址运算符，&a 是一个表达式，其功能是求变量的地址(第 8 章介绍指针时，将对此做详细讨论)。

【例 4.7】scanf 的应用：

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



本例中，由于 scanf 函数本身不能显示提示信息，故先用 printf 语句在屏幕上输出提示“enter a,b,c”，要求用户输入 a、b、c 的值。执行 scanf 语句后进入用户屏幕，等待用户输入数据。用户输入“1　2　3”后，按 Enter 键，此时，系统又返回到集成编译环境。在 scanf 语句的格式串中，由于没有非格式字符在“%d%d%d”之间作为输入时的间隔，因此在输入时要用空格、Tab 键或回车键作为相邻两个输入数据之间的间隔。例如：

```
1 2 3<Enter>
```

或者：

```
1<Tab>2<Tab>3<Enter>
```

或者：

```
1<Enter>
2<Enter>
3<Enter>
```

2. 格式字符串

(1) 格式字符串的一般形式为：

```
%[*][输入数据宽度][长度]类型
```

其中，用方括号括起来的项为可选项，各项的含义如下。

① 类型：表示输入数据的类型，其格式符和意义如表 4-3 所示。

表 4-3 格式符及其含义

格 式	字符意义
d, i	输入十进制整数
o	输入八进制整数
x	输入十六进制整数
u	输入无符号十进制整数
f 或 e	输入实型数(用小数形式或指数形式)
c	输入单个字符
s	输入字符串

② 星号“*”：称作抑制符，用以表示该输入项读入后不赋给相应的变量，即跳过该输入值。例如：

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

当输入“7 8 9”时，把 7 赋给 a，8 被跳过，9 赋给 b。

③ 宽度：用十进制整数指定输入的宽度(即字符数)。

例如：

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

输入 123456 后，只把 123 赋予变量 a，其余部分被截掉。

又如：

```
scanf("%4d%4d", &a,&b);
```

输入 12345678 后，将把 1234 赋给 a，而把 5678 赋给 b。

④ 长度：长度格式符为 l 和 h，l 用于输入长整型数据(如%ld)和双精度浮点数(如%lf)。h 用于输入短整型数据。

(2) 使用 scanf 函数时，须注意以下几点。

① scanf 函数中没有精度控制，如 scanf(“%6.2f”, &a);是非法的。不能企图用此语句输入小数位数为 2 的实数。

② scanf 中要求给出变量的地址，如给出变量名，则会出错。如 scanf(“%d”, a);是非法的，应将 a 改为&a 才合法。

③ 在输入多个数值数据时，若格式控制串中没有非格式字符作为输入数据之间的间隔，则可用空格、Tab 或回车符作为间隔。C 编译程序在遇到空格、Tab、回车符或非法数据(如对“%d”输入“12A”时，A 即为非法数据)时，即认为该数据结束。

④ 在输入字符数据时，若格式控制串中无非格式字符，则认为所有输入的字符均为有效字符。例如：

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

如果输入为：a b c(即以空格分隔)，则把‘a’赋给 a，‘ ’赋给 b，‘b’赋给 c。

只有当输入为：abc(即连续输入 abc)时，才能把‘a’赋给 a，‘b’赋给 b，‘c’赋给 c。

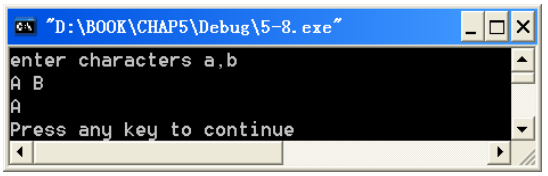
如果在格式控制中加入空格作为间隔，例如：

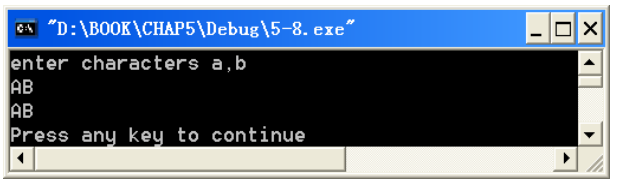
```
scanf("%c %c %c", &a,&b,&c);
```

则输入时，各数据之间必须加空格。

【例 4.8】用 scanf 函数输入字符数据：

```
#include <stdio.h>
main()
{
    char a,b;
    printf("enter characters a,b\n");
    scanf("%c%c", &a,&b);
    printf("%c%c\n", a,b);
}
```



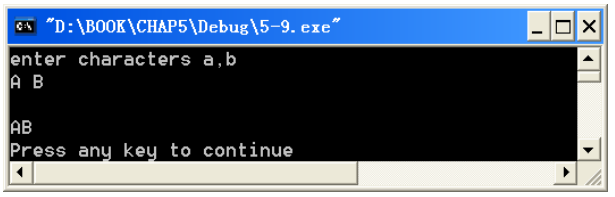


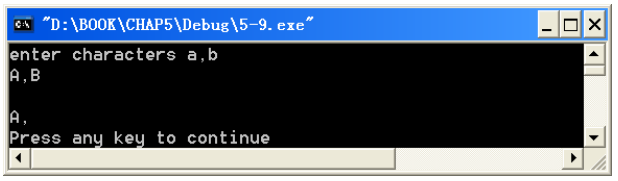
从两次运行的结果可以看出，由于 scanf 格式控制串“%c%c”中没有空格，输入 A B，结果输出只有 A。而输入改为 AB 时，则可输出 AB 两个字符。

【例 4.9】用 scanf 函数输入字符数据：

```
#include <stdio.h>
main()
{
    char a,b;
```

```
    printf("enter characters a,b\n");
    scanf("%c %c", &a,&b);
    printf("\n%c%c\n", a,b);
}
```





本例表明，因为 scanf 格式控制串“%c %c”之间有空格，所以输入的数据之间必须以空格间隔，以其他字符间隔是得不到正确结果的。

如果格式控制串中有非格式字符，则输入时也要输入该非格式字符。

例如：

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

其中，用非格式符“,”作为间隔符，若想给 a、b、c 分别输入 1，2，3，则输入应为：

```
1,2,3
```

又如：

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

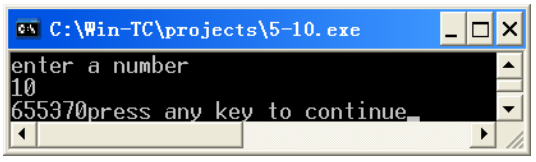
则输入应为：

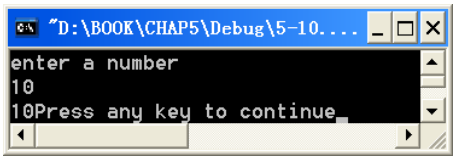
```
a=1,b=2,c=3
```

如输入的数据与输出的类型不一致时，虽然编译能够通过，但结果可能不正确。

【例 4.10】输入数据与输出数据的类型不一致：

```
#include <stdio.h>
main()
{
    int a;
    printf("enter a number\n");
    scanf("%d", &a);
    printf("%ld", a);
}
```





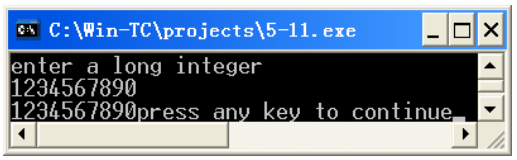
这是 TC 和 VC 各自的输出结果，可见二者是不同的。

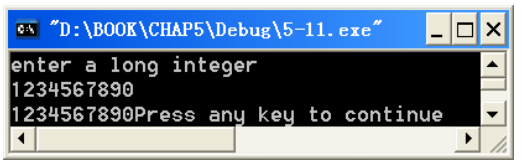
在 TC 中，由于输入数据类型为整型(2 字节)，而输出语句的格式串中说明为长整型(4 字节)，因此输出结果和输入数据不符。但在 VC 中，整型变量和长整型变量均占 4 个字节，因此输出结果没有问题。

如果输入数据与输出数据的类型一致(如例 4.11)，则两个编译系统下的运行结果相同。

【例 4.11】输入数据与输出数据的类型一致：

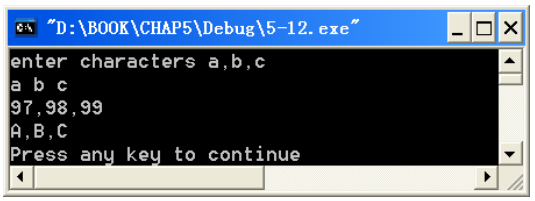
```
#include <stdio.h>
main()
{
    long a;
    printf("enter a long integer\n");
    scanf("%ld", &a);
    printf("%ld", a);
}
```





【例 4.12】输入小写字母，输出大写字母：

```
#include <stdio.h>
main()
{
    char a,b,c;
    printf("enter characters a,b,c\n");
    scanf("%c %c %c", &a,&b,&c);
    printf("%d,%d,%d\n%c,%c,%c\n", a,b,c, a-32, b-32, c-32);
}
```

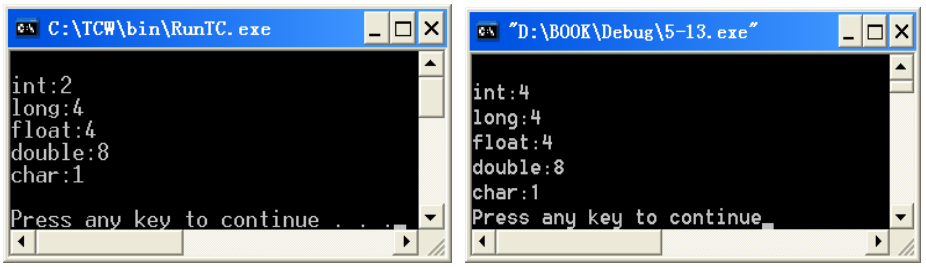


本例中输入的是三个小写字母，输出的是与其对应的 ASCII 码和相应的大写字母。

【例 4.13】输出各种数据类型所占的字节数：

```
#include <stdio.h>
main()
{
    int i;
    long l;
    float f;
    double d;
    char c;
    printf("\nint:%d\nlong:%d\nfloat:%d\ndouble:%d\nchar:%d\n",
```

```
    sizeof(i), sizeof(l), sizeof(f), sizeof(d), sizeof(c));
}
```



本例输出各种数据类型所占的字节数(左、右分别为 TC 和 VC 运行结果截图)。可以看出，TC 和 VC 的整型变量所占的字节数是不同的。

4.6 结构化程序设计的方法

结构化程序便于书写，便于阅读，便于修改和维护，因而减少了程序出错的机会，提高了程序的可靠性，保证了程序的质量。所以 C 语言中十分强调结构化程序设计。一般来讲，采取以下方法可以保证得到结构化的程序：

◎ 自顶向下。

◎ 逐步细化。

◎ 模块化设计。

◎ 结构化编码。

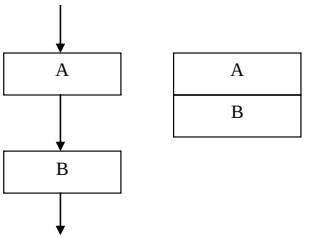
早在 1966 年，Bohra 和 Jacopini 就提出了结构化程序设计使用的三种基本结构：顺序结构、分支结构、循环结构。理论上已经证明，任何复杂的程序均可使用这三种基本结构来实现。

在程序设计中，经常提到“算法”这一概念。实际上，算法是一个广义的概念，不要认为只有程序设计才需要算法。为解决一个问题而采取的方法和步骤就是算法。当然，本书所关心的仅限于程序的算法，即计算机能执行的算法。例如，让程序完成“先输入一个小写字母，再转换成大写字母，最后输出该大写字母”这一功能，就是顺序算法；让程序完成“如果 a>b 则输出 a，否则输出 b”这一功能，就是分支算法，或称选择算法；让程序完成“求 100 以内自然数之和”这一功能，就是循环算法。

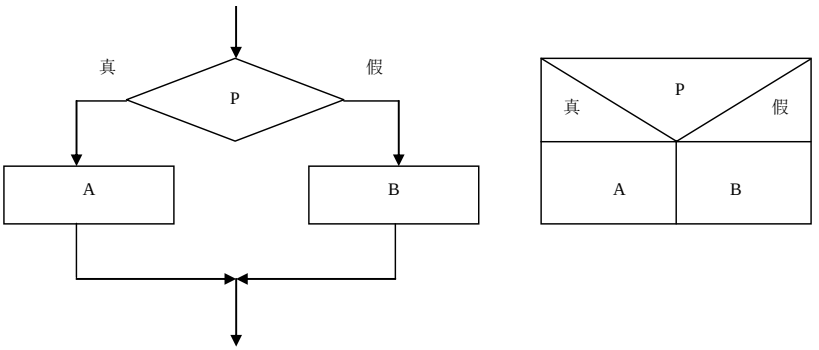
算法有不同的表示方法，例如自然语言、框图(程序流程图)、伪代码、PAD 图、N-S 图等。C 语言文献中多使用框图与 N-S 图。下面介绍一下三种基本结构的框图和 N-S 图。

(1) 顺序结构。就是一个程序从第一行一直运行到最后一行，也就是程序从头到尾顺序运行，其算法可以用图 4-2(a)所示的框图和 N-S 图来表示。

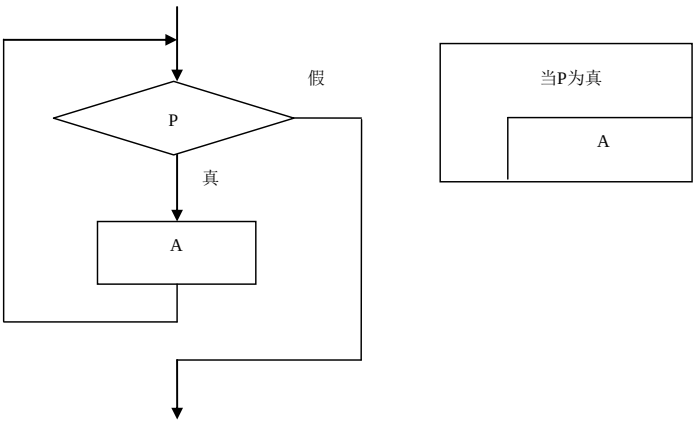
(2) 分支结构。亦称为选择结构，它依据一定的条件选择执行路径，而不是严格按照语句出现的物理顺序执行。分支结构程序设计方法的关键，在于构造合适的分支条件和分析程序流程，根据不同的程序流程选择适当的分支语句。分支结构适合于带有逻辑或关系比较等条件判断的计算，设计这类程序时，往往都要先绘制其程序流程图，然后根据程序流程写出源程序，这样做，把程序设计分析与所使用的编程语言分开，使得问题简单化，易于理解。分支结构的算法可以用图 4-2(b)所示的框图和 N-S 图来表示。



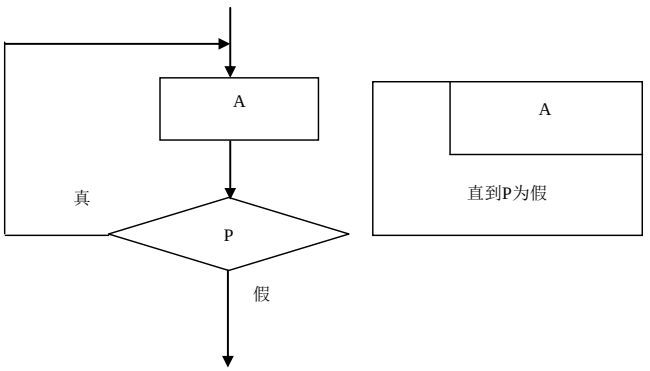
(a) 顺序结构的框图与 N-S 图



(b) 分支结构的框图与 N-S 图



(c) 循环结构(当型)的框图与 N-S 图



(d) 循环结构(直到型)的框图与 N-S 图

图 4-2 结构化程序设计中各种结构的流程图与 N-S 图

(3) 循环结构。亦称重复结构，程序根据给定的条件反复执行某个程序段。给定的条件称为循环条件，反复执行的程序段称为循环体。

有两类循环结构。

① 当型(while)循环结构。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。其算法可以用图 4-2(c)所示的框图和 N-S 图来表示。

② 直到型(until)循环结构。其特点是，先执行某程序段，然后判断给定的条件是否成立，如果成立，则继续执行该程序段，否则退出循环。其算法可以用图 4-2(d)所示的框图和 N-S 图来表示。

本章后续各节分别介绍这三种基本结构，并结合具体的例题讨论算法与流程。

4.7 顺序结构程序设计

4.7.1 顺序结构的程序

顺序结构是最简单的一种基本结构，程序中的语句自上而下执行，既无“回头”，又无“跳转”，是一种“直线式”的执行，其算法结构已示于前面的图 4-2(a)。

4.7.2 顺序结构程序的案例实训

【例 4.14】输入三角形的三边长，求三角形的面积。

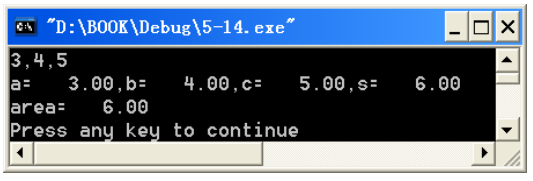
若已知三角形的三边长 a、b、c，则求三角形面积的公式(海伦公式)为：

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

其中 s = (a+b+c)/2。

源程序如下：

```
#include <stdio.h>
#include <math.h>
main()
{
    float a,b,c,s,area;
    scanf("%f,%f,%f", &a,&b,&c);
    s = 1.0/2*(a+b+c);
    area = sqrt(s*(s-a)*(s-b)*(s-c));
    printf("a=%7.2f,b=%7.2f,c=%7.2f,s=%7.2f\n", a,b,c,s);
    printf("area=%7.2f\n", area);
}
```



程序自上而下顺序执行，首先输入三角形的边长(当然要保证任意两边之和大于第三边，否则将出现运行错误)，然后计算边长和的一半，再利用海伦公式求三角形面积，最后顺序

输出三条边的长度、三边和的一半以及三角形的面积。

【例 4.15】求方程 $ax^2+bx+c=0$ 的根，a、b、c 由键盘输入，设 $b^2-4ac>=0$。

求根公式(韦达定理)为：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}\quad,\quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

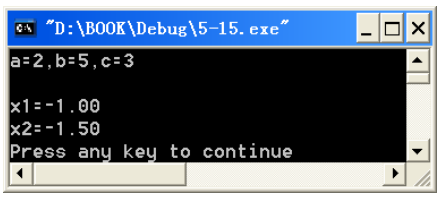
令：

$$p=\frac{-b}{2a}\quad,\qquad q=\frac{\sqrt{b^2-4ac}}{2a}$$

则 $x_1=p+q$，$x_2=p-q$。

求方程根的源程序如下：

```
#include <stdio.h>
#include <math.h>
main()
{
    float a,b,c,disc,x1,x2,p,q;
    scanf("a=%f,b=%f,c=%f", &a,&b,&c);
    disc = b*b - 4*a*c;
    p = -b/(2*a);
    q = sqrt(disc)/(2*a);
    x1=p+q; x2=p-q;
    printf("\nx1=%5.2f\nx2=%5.2f\n", x1,x2);
}
```



程序同样是按照自上而下的顺序执行的，首先输入系数 a、b、c 的值(当然要保证 $b^2-4ac>=0$，否则将出现对负数开方这一运行错误)，然后计算判别式的值，再利用韦达定理求两个实根，最后顺序输出两个实根。

上述两个例子虽然很简单，但说明了顺序程序设计算法的概念。

4.8 选择结构程序设计

4.8.1 关系运算符和关系表达式

在编程时，经常需要比较两个量的大小，以决定程序下一步的工作。比较两个量的运算符称为比较运算符，亦称关系运算符。

1. 关系运算符及其优先次序

在 C 语言中，存在下列关系运算符：

```
<    小于
```

```
<=   小于或等于
>    大于
>=   大于或等于
==   等于
!=   不等于
```

关系运算符均为双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，但高于赋值运算符。在 6 种关系运算符中，<、<=、>、>=的优先级相同(第 6 优先级)，高于==和!=。而==和!=的优先级相同(第 7 优先级)。

2. 关系表达式

关系表达式是由关系运算符连接运算对象组成的表达式，其一般形式为：

```
表达式  关系运算符  表达式
```

例如：

```
c > a+b
b != c
a == b+c
```

这些均为合法的关系表达式。因为表达式也可以又是关系表达式，所以也允许出现嵌套的情况。例如：

```
a>b != c
a == b<c
```

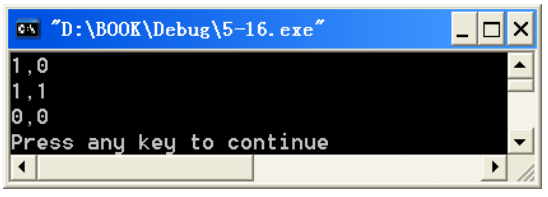
关系表达式的值只有两种——非“真”即“假”，分别用 1 和 0 表示。

例如，6>0 的值为“真”，即为 1。

又如，对于(a=2) > (b=6)，由于 2>6 不成立，故其值为假，即为 0。

【例 4.16】关系运算：

```
#include <stdio.h>
main()
{
    char c = 'k';
    int i=1, j=2, k=3;
    float x=3e+5, y=0.85;
    printf("%d,%d\n", 'a'+5<c, -i-2*j>=k+1);
    printf("%d,%d\n", 1<j<5, x-5.25<=x+y);
    printf("%d,%d\n", i+j+k==-2*j, k==j==i+5);
}
```



本例程序中求出了各种关系表达式的值。需要注意的是，字符变量是以它对应的 ASCII 码参与运算的。对于含有多个关系运算符的表达式，如“k==j==i+5”，根据运算符的左结

合性，应先计算“k==j”，该式不成立，其值为 0，再计算“0==i+5”，也不成立，故表达式的值为假，输出 0。

又如，有如下变量说明：

```
int a=3, b=2, c=1, d, f;
```

则：

```
a>b              值为 1
(a>b)==c         值为 1
b+c<a            值为 0
d=a>b            d 值为 1
f=a>b>c          f 值为 0
```

单纯从数学的角度讲，a>b>c(3>2>1)的确为真，但 f 值缘何为 0 呢？这是因为，根据运算符的左结合性，应先计算 a>b，值为 1；再计算 1>1，值为 0，故 f 值为 0。这里提醒读者注意，不能完全以数学的思维方式去理解 C 语言的关系表达式。

4.8.2 逻辑运算符和逻辑表达式

1．逻辑运算符及其优先次序

C 语言中有三种逻辑运算符：

◎ &&——与运算。

◎ ||——或运算。

◎ !——非运算。

与运算符&&和或运算符||均需要两个操作数，即为双目运算符，具有左结合性。而非运算符!为单目运算符，具有右结合性。逻辑运算符和其他运算符优先级的关系可表示如下。

(1) !(非) 高于 &&(与) 高于 ||(或)。

(2) &&和||低于关系运算符，而!高于算术运算符。

算术运算符、逻辑运算符、关系运算符、赋值运算符之间的优先关系如图 4-3 所示。

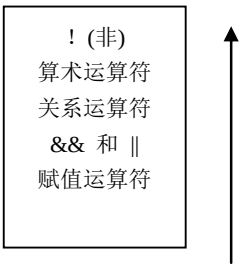


图 4-3 四类运算符之间的优先关系(下低上高)

根据运算符的优先顺序可知：

```
a<=x && x<=b     等价于   (a<=x) && (x<=b)
a>b&&x>y         等价于   (a>b)&&(x>y)
a==b||x==y       等价于   (a==b)||(x==y)
!a||a>b          等价于   (!a)||(a>b)
```

2．逻辑运算的值

逻辑运算的值也为“真”和“假”两种，分别用 1 和 0 来表示，其求值规则如下。

(1) 与运算规则：两个运算量均为真时，结果才为真，否则为假。

例如：

```
6>0 && 8>2
```

由于 6>0 为真，8>2 也为真，故逻辑与运算的结果也为真。

又如：

```
-1>0 && 3>8
```

由于−1>0 为假，不管后面的结果如何，逻辑与运算的结果一定为假(实际上，与运算时，若左边为假，则不管右边是真是假，结果都为假，所以右边的不再进行计算。此即逻辑与运算的短路)。

(2) 或运算规则：两个运算量只要有一个为真，结果就为真。两个量均为假时，结果才为假。

例如：

```
6>0||3>8
```

由于 6>0 为真，不管后面的结果如何，逻辑或运算的结果一定为真(实际上，或运算时，若左边为真，则不管右边是真是假，结果都为真，所以右边的不再进行计算。此即逻辑或运算的短路)。

(3) 非运算规则：运算量为真时，结果为假；运算量为假时，结果为真。

例如：

```
!(6>0)
```

的结果为假。

虽然 C 编译在给出逻辑运算值时，以 1 代表“真”，以 0 代表“假”，但在判断一个量是为“真”还是为“假”时，以 0 代表“假”，以非 0 的数值作为“真”。例如，由于 6 和 3 均为非 0，因此 6&&3 的值为“真”，即为 1。又如，6||0 的值为“真”，即为 1。

3．逻辑表达式

逻辑表达式的一般形式为：

```
表达式  逻辑运算符  表达式
```

其中的表达式可以又是逻辑表达式，从而形成了逻辑表达式的嵌套。

例如：

```
(a&&b)&&c
```

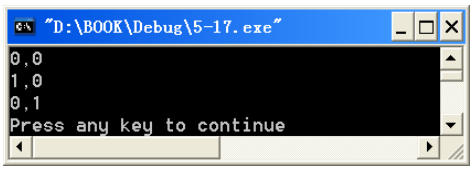
根据逻辑运算符的左结合性，上式也可以写为：

```
a&&b&&c
```

逻辑表达式的值是式中各种逻辑运算的最后值，以 1 和 0 分别表示“真”和“假”。

【例 4.17】逻辑运算的短路：

```
#include <stdio.h>
main()
{
    char c = 'k';
    int i=1, j=2, k=3;
    float x=3e+5, y=0.85;
    printf("%d,%d\n", !x*!y, !!!x);
    printf("%d,%d\n", x||i&&j-3, i<j&&x<y);
    printf("%d,%d\n", i==5&&c&&(j=8), x+y||i+j+k);
}
```



现对本程序最后三行的 6 项输出做如下分析。

(1) x 和 y 的值均为非 0，所以!x 和!y 的值均为 0，!x*!y 也为 0，故其输出值为 0。

(2) 由于 x 为非 0，故!!!x 的逻辑值为 0(相邻的两个!可视为互相抵消，故!!!x 实为!x)。

(3) 对 x || i && j-3 式，先计算 x 的值为 1，逻辑或运算短路，不再计算 i && j-3，故输出值为 1。

(4) 对 i<j&&x<y 式，由于 i<j 的值为 1，而 x<y 为 0，故表达式的值为 1 和 0 相与，最后为 0。

(5) 对 i==5&&c&&(j=8)式，由于 i==5 为假，即值为 0，逻辑与运算短路，不再计算后面的逻辑值，所以整个逻辑与表达式的值为 0。

(6) 对 x+y||i+j+k 式，由于 x+y 的值为非 0，逻辑或运算短路，不再计算后面的逻辑值，故整个逻辑或表达式的值为 1。

下面再举几个简单的例子。设有如下的变量说明：

```
int a=4, b=5;
```

则：

```
!a                  值为 0
a&&b                值为 1
a||b                值为 1
!a||b               值为 1
4&&0||2             值为 1
5>3&&2||8<4-!0      值为 1
'c'&&'d'            值为 1
```

4.8.3 if 语句

C 语言的 if 语句可以构成分支结构。它根据给定的条件进行判断，以决定执行程序的哪个分支。if 语句有三种基本形式。

1．if 语句的三种形式

(1) 基本形式：if。

if 语句的基本形式为：

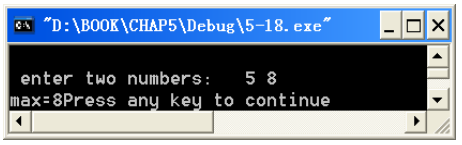
```
if(表达式) 语句
```

其含义是：若表达式的值为真，则执行其后的语句，否则不执行任何语句。其过程可表示为图 4-4。

这里需要特别指出，尽管图 4-2(b)给出了分支结构基本算法执行的流程，但具体到 C 语言，因为有各种形式的分支语句，所以又演化出了不同形式的算法流程。

【例 4.18】基本形式的 if 语句：

```
#include <stdio.h>
main()
{
    int a,b,max;
    printf("\n enter two numbers:    ");
    scanf("%d%d", &a,&b);
    max = a;
    if (max<b) max=b;
    printf("max=%d", max);
}
```



本例中，先输入两个整数 a、b，将 a 先赋给变量 max，再用 if 语句判别 max 和 b 谁大谁小，如 max 小于 b，则把 b 赋给 max。因此 max 中存放的总是较大的数，最后输出 max 的值。本例的核心算法流程如图 4-5 所示。

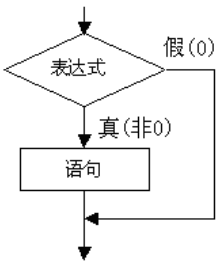


图 4-4　if 语句的执行过程

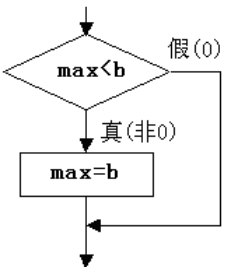


图 4-5　基本形式 if 语句的算法流程

(2) 双分支形式：if-else。

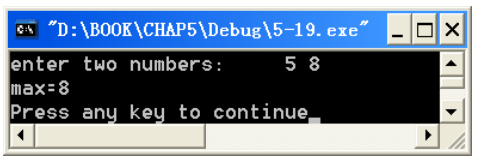
双分支 if 语句的基本形式为：

```
if(表达式)
    语句 1
else
    语句 2
```

其含义是：若表达式的值为真，则执行语句 1，否则执行语句 2。其执行过程如图 4-6 所示。

【例 4.19】双分支形式的 if 语句：

```
#include <stdio.h>
main()
{
    int a, b;
    printf("enter two numbers:     ");
    scanf("%d%d", &a,&b);
    if(a > b)
        printf("max=%d\n", a);
    else
        printf("max=%d\n", b);
}
```



本例程序完成的功能与上例相同，都是输入两个整数，输出其中的较大者。这里改用 if-else 语句判别 a 与 b 的大小，若 a 中的值较大，则输出 a，否则输出 b。本例的核心算法流程如图 4-7 所示。

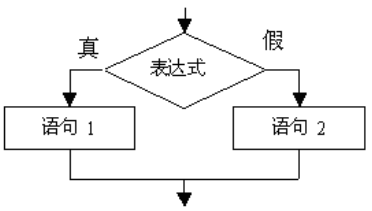


图 4-6　if-else 语句的执行过程

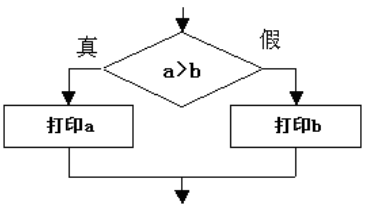


图 4-7　双分支形式 if 语句的算法流程

(3)　多分支形式：if-else-if。

前面两种形式的 if 语句一般用于两个分支的情况。当需要多分支选择时，可采用 if-else-if 语句，其一般形式如下：

```
if(表达式 1)
    语句 1
else if(表达式 2)
    语句 2
else if(表达式 3)
    语句 3
...
else if(表达式 4)
    语句 4
else
    语句 5
```

其含义是：依次判断各个表达式的值，当某个值为真时，则执行相应的语句，然后跳

到整个 if 语句之外继续执行程序。若所有的表达式均为假，则执行语句 n，然后继续执行后续的程序。if-else-if 语句的执行过程如图 4-8 所示。

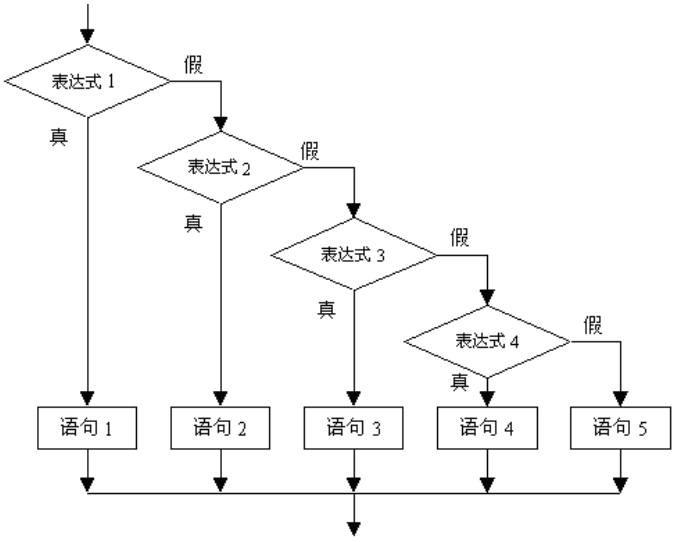
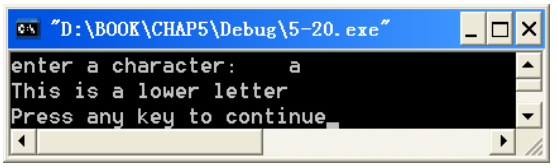


图 4-8　多分支条件语句的执行过程

【例 4.20】多分支形式的 if 语句：

```
#include <stdio.h>
main()
{
    char c;
    printf("enter a character:     ");
    c = getchar();
    if(c < 32)
       printf("This is a control character\n");
    else if(c>='0' && c<='9')
       printf("This is a digit\n");
    else if(c>='A' && c<='Z')
       printf("This is a capital letter\n");
    else if(c>='a' && c<='z')
       printf("This is a lower letter\n");
    else
       printf("This is an other character\n");
}
```



本例程序要求判断从键盘上输入字符的类别。其方法是根据输入字符的 ASCII 码范围来判别其类型。由 ASCII 码表(见附录)可知，ASCII 码小于 32 的字符为控制字符；ASCII 值介于‘0’和‘9’之间的字符为数字；在‘A’和‘Z’之间的为大写字母；在‘a’和‘z’之间的为小写

字母；其余的则统称为其他字符。显然，这是一个多分支选择的问题，可用上述的 if-else-if 语句编程实现，根据输入字符的 ASCII 码所在的范围，分别给出不同的输出。例如，输入为“a”时，输出显示它为小写字母；输入为“0”时，输出显示它为数字。本例的核心算法流程如图 4-9 所示。

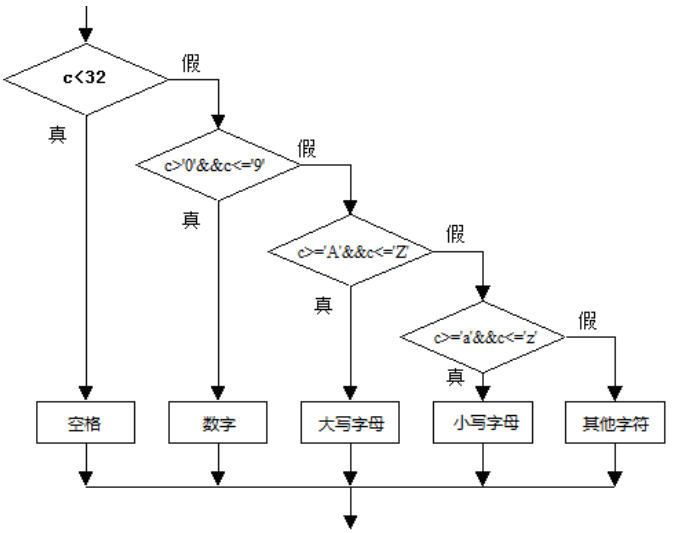


图 4-9　本例的核心算法流程

在使用 if 语句时还应注意以下问题。

(1)　无论哪种形式的 if 语句，在 if 关键字之后均为表达式。一般情况下，该表达式是一个关系表达式或逻辑表达式，但也可以是其他的表达式，如赋值表达式等，甚至可以是一个常量、变量、函数等。

例如：

```
if(x=1) 语句
if(a) 语句
if(3) 语句
```

这些都是允许的。只要其中表达式的值为非 0，就为“真”。

例如，对于：

```
if(a=5)...;
```

表达式的值永远为非 0，所以其后的语句总是要执行的。当然，这种情况在程序中不一定会出现，但在语法上还是合法的，只是编译时给出警告，读者应试时尤其需要注意。

又如，有如下程序段：

```
if(a=b)
    printf("%d", a);
else
    printf("a=0");
```

本语句的意义是，把 b 值赋给 a，如为非 0，则输出该值，否则输出字符串“a=0”。这种用法在程序中是常见的。

(2) if 语句的条件判断表达式必须用括号括起来，且语句之后必须加分号。

(3) 在 if 语句的三种形式中，所有的语句可为简单语句(单个语句)，亦可为复合语句。但要特别注意，在复合语句的“}”之后不能再加分号。

例如：

```
if(x > y)
{
    x++;
    y++;
}
else
{
    x = -x;
    y = -y;
}
```

2．if 语句的嵌套

如果 if 语句中的执行语句又是一个 if 语句，那么就构成了 if 语句的嵌套，其一般形式可表示为：

```
if(表达式)
    if 语句
```

或者表示为：

```
if(表达式)
    if 语句
else
    if 语句
```

在嵌套内的 if 语句可能又是 if-else 型的，这将会出现多个 if 和多个 else 重叠的情况，这时要特别注意某个 if 与哪一个 else 配对。

例如：

```
if(表达式 1)
if(表达式 2)
语句 1;
else
语句 2;
```

其中的 else 究竟与哪一个 if 配对呢？

换言之，应该理解为：

```
if(表达式 1)
    if(表达式 2)
        语句 1;
    else
        语句 2;
```

还是应该理解为：

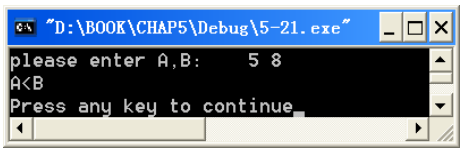
```
if(表达式 1)
```

```
    if(表达式2)
        语句1;
else
    语句2;
```

为了避免二义性，C 语言中规定，else 总是与它上面、离它最近的、尚未配对的 if 配对。因此，对上述例子，应该按前一种情况去理解。

【例 4.21】利用 if 嵌套结构比较两个数的大小：

```
#include <stdio.h>
main()
{
    int a,b;
    printf("please enter A,B:     ");
    scanf("%d%d", &a, &b);
    if(a != b)
        if(a > b) printf("A>B\n");
        else printf("A<B\n");
    else
        printf("A=B\n");
}
```

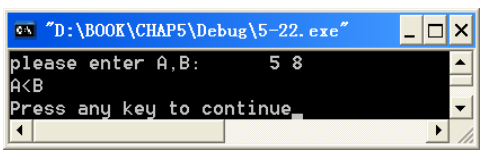


本例程序中使用了 if 语句的嵌套结构。采用嵌套结构是为了进行多分支选择。实际上有三种选择，即 A>B、A<B、A=B。此类问题用 if-else-if 语句也能够完成，而且程序更加清晰。

有鉴于此，在一般情况下，应尽量少用 if 语句的嵌套结构，以使程序更易于阅读理解。读者可以比较一下例 4.21 与例 4.22。

【例 4.22】利用多分支 if 语句比较两个数的大小：

```
#include <stdio.h>
main()
{
    int a,b;
    printf("please enter A,B:    ");
    scanf("%d%d", &a, &b);
    if(a==b) printf("A=B\n");
    else if(a>b) printf("A>B\n");
    else printf("A<B\n");
}
```



3．条件运算符和条件表达式

在双分支条件语句中，若每个语句都是单个的赋值语句，且给同一个变量赋值，则可以使用条件表达式来实现。这样做，不但使程序简洁，也提高了运行的效率。

条件运算符由“?”和“:”两个符号组成，它是 C 语言中唯一的一个三目运算符，也就是说，它有三个参与运算的量。

条件表达式的一般形式为：

```
表达式 1?  表达式 2 : 表达式 3
```

其求值规则为：若表达式 1 的值为真，则以表达式 2 的值作为整个条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值。

条件表达式通常用于赋值语句中。

例如，条件语句：

```
if(a>b) max=a;
else max=b;
```

可用条件表达式写为：

```
max = (a>b)? a : b;
```

该语句的语义是：若 a>b，则把 a 赋给 max，否则把 b 赋给 max。换言之，max 最终存放 a 和 b 中的较大者。

使用条件表达式时，应注意以下几点。

(1) 条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值运算符。

因此：

```
max=(a>b)?a:b
```

可以去掉括号，而写为：

```
max=a>b?a:b
```

(2) 条件运算符中的两个符号“?”和“:”要成对使用，不能分开单独使用。

(3) 条件运算符的结合方向是自右向左的。

例如：

```
a>b?a:c>d?c:d
```

应理解为：

```
a>b?a:(c>d?c:d)
```

实际上，这是条件表达式的嵌套，即其中的表达式 3 又是一个条件表达式。

【例 4.23】用条件表达式对上例重新编程，输出两个数中的较大者：

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