第3章　程序控制语句

3.1　程序的三种基本结构

通常的计算机程序总是由若干条语句组成，从执行方式上看，从第一条语句到最后一条语句完全按顺序执行，是简单的顺序结构；若在程序执行过程当中，根据用户的输入或中间结果去执行若干不同的任务则为选择结构；如果在程序的某处，需要根据某项条件重复地执行某项任务若干次或直到满足或不满足某条件为止，这就构成循环结构。大多数情况下，程序都不会是简单的顺序结构，而是顺序、选择、循环三种结构的复杂组合。

三种基本结构的流程图、N-S图以及PAD图可以参看本书第1章1.4节“算法”相关内容。

C语言中，有一组相关的控制语句，用以实现选择结构与循环结构：

选择控制语句：　if；
　　switch、case

循环控制语句：　for、while、do…while

转移控制语句：　break、continue、goto

我们将在后面几节中详细介绍。

3.2　数据的输入与输出

在程序的运行过程中，往往需要由用户输入一些数据，而程序运算所得到的计算结果等又需要输出给用户，由此实现人与计算机之间的交互，所以在程序设计中，输入输出语句是一类必不可少的重要语句，在C语言中，没有专门的输入输出语句，所有的输入输出操作都是通过对标准I/O库函数的调用实现。最常用的输入输出函数有 scanf()、printf()、getchar()和putchar()，以下分别介绍。

3.2.1　scanf（）函数

格式化输入函数scanf()的功能是从键盘上输入数据，该输入数据按指定的输入格式被赋给相应的输入项。函数一般格式为：

scanf("控制字符串"，输入项列表）；

其中控制字符串规定数据的输入格式，必须用双引号括起，其内容是由格式说明和普通字符两部分组成。输入项列表则由一个或多个变量地址组成，当变量地址有多个时，各变量地址之间用逗号“，”分隔。

scanf()中各变量要加地址操作符，就是变量名前加“ &”，这是初学者容易忽略的一个问题。应注意输入类型与变量类型一致。

下面探讨控制字符串的两个组成部分：格式说明和普通字符。

1. 格式说明

格式说明规定了输入项中的变量以何种类型的数据格式被输入，形式是：

%[<修饰符>]<格式字>

各个格式字符及其意义见表3-1。

表3-1 输入格式字符

格式字符	意 义
d	输入一个十进制整数
o	输入一个八进制整数
x	输入一个十六进制整数
f	输入一个小数形式的浮点数
e	输入一个指数形式的浮点数
c	输入一个字符
s	输入一个字符串

各修饰符是可选的，可以没有，这些修饰符是：

(1) 字段宽度

例如：scanf("%3d"，&a)

按宽度3输入一个整数赋给变量a。

(2) l和h

可以和d、o、x一起使用，加l表示输入数据为长整数，加 h表示输入数据为短整数，例如：

```
scanf("%10ld%hd",&x,&i)
```

则x按宽度为10的长整型读入，而i按短整数读入。

(3) 字符*

*表示按规定格式输入但不赋予相应变量，作用是跳过相应的数据。

例如：

```
scanf("%4d%*d%4d",&x,&y,&z)
```

执行该语句，若输入为“ 1 2 3 ”

结果为x=1，z=3，y未赋值，2被跳过。

2. 普通字符

普通字符包括空格、转义字符和可打印字符。

(1) 空格

在有多个输入项时，一般用空格或回车作为分隔符，若以空格作分隔符，则当输入项中包含字符类型时，可能产生非预期的结果，例如：

```
scanf("%d%c",&a,&ch)
```

输入32␣q

期望a=32，ch=q，但实际上，分隔符空格被读入并赋给ch。

为避免这种情况，可使用如下语句：

```
scanf("%d ␣%c",&a,&ch)
```

此处%d后的空格，就可跳过字符“ q ”前的所有空格，保证非空格数据的正确录入。

(2) 转义字符：\n、\t

先看下面的例子：

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

输入为 1 2 3
4 5 6

结果为： a=1， b=2， x=3， y=4， z=5

若将上述语句改为：

```
scanf("%d%d\n",&a,&b);
scanf("%d%d%d",&x,&y,&z);
```

对同样的输入，其结果为 a=1，b=2，x=4，y=5，z=6，**由于在第一个**scanf**的最后有一个**\n，**所以第二个**scanf**语句将从第二个输入行获得数据。**

(3) **可打印字符**

例如：scanf("%d，%d，%c"，&a，&b，&ch);

当输入为：1, 2, q

即：a=1，b=2，ch=q

若输入为1␣2␣q

除a=1**正确赋值外，对**b**与**ch**的赋值都将以失败告终，也就是说，这些不打印字符应是输入数据分隔符，**scanf**在读入时自动去除与可打印字符相同的字符。**

[**例**3-1] **试编写求梯形面积的程序，数据由键盘输入。**

分析：设梯形上底为A，**下底为**B，**高为**H**面积为**S，**则**

S=（A+B）×H÷2

程序如下：

```
main()
 {
   float a,b,h,s;
   printf("please input a,b,h:");
   scanf("%f%f%f",&a,&b,&h);
   s=0.5*(a+b)*h;
   printf("a=%5.2f  b=%5.2f  h=%5.2f",a,b,h);
   printf("s=%7.4f",s);
 }
```

运行结果如下：

```
RUN
    please input a,b,h3.5 4.2 2.8
    a=3.50   b=4.20   h=2.80
    s=10.7800
```

3.2.2 printf( )函数

与格式化输入函数 scanf()**相对应的是格式化输出函数** printf()，**其功能为按控制字符串规定的格式，向缺省输出设备（一般为显示器）输出在输出项列表中列出的各输出项，其基本格式为：**

printf（**“控制字符串”，输出项列表**）

输出项可以是常量、变量、表达式，其类型与个数必须与控制字符串中格式字符的类型、个数一致、当有多个输出项时，各项之间用逗号分隔。

控制字符串必须用双引号括起，由格式说明和普通字符两部分组成。

1. 格式说明

一般格式为：

%[<修饰符>]<格式字符>

格式字符规定了对应输出项的输出格式，常用格式字符见表 3-2。

表3-2 输出格式字符

格式字符	意义	格式字符	意义
c	按字符型输出	o	按八进制整数输出
d	按十进制整数输出	x	按十六进制整数输出
u	按无符号整数输出	s	按字符串输出
f	按浮点型小数输出	g	按e和f格式中较短的一种输出
e	按科学计数法输出		

修饰符是可选的，用于确定数据输出的宽度、精度、小数位数、对齐方式等，用于产生更规范整齐的输出，当没有修饰符时，以上各项按系统缺省设定显示。

(1) 字段宽度修饰符

表3-3列出了字段宽度修饰符。

表3-3 字段宽度修饰符

修饰符	格式说明	意义
M	%md	以宽度m输出整型数，不足m时，左补空格
0m	%0md	以宽度m输出整型数，不足m时，左补零
m，n	%m .nf	以宽度m输出实型小数，小数位为n位

例如：设i=123，a=12.34567，

则：

```
printf("%4d+++%5.2f"，i，a);
```

输出：␣123+++12.35

```
printf("%2d+++%2.1f"，i，a);
```

输出：

123+++12.3

可以看出，当指定场宽小于数据的实际宽度时，对整数，按该数的实际场宽输出，对浮点数，相应小数位的数四舍五入。例如：12.34567按%5.2f 输出，输出12.35。若场宽小于等于浮点数整数部分的宽度，则该浮点数按实际位数输出，但小数位数仍遵守宽度修饰符给出的值。如上面的12.34567按%2.1f 输出，结果为：12.3。

在实际应用中，还有一种更灵活的场宽控制方法，用常量或变量的值作为输出场宽，方法是以一个"*"作为修饰符，插入到%之后。

例如：i=123;

```
printf("%*d",5,i);
```

此处，5为场宽，输出为

␣ ␣123

在程序中，可以用一个整形变量K来指示场宽：

```
printf("%*d",k,i);
```

可以根据k的值动态地决定i的显示场宽，这在解某些问题时是相当有用的。

(2) 对齐方式修饰符

负号“－”为“左对齐”控制符，一般所有输出数据为右对齐格式，加一个“ -”号，则变为“左对齐”方式。

例如： i＝123，a=12.34567

printf(“%4d%10.4f”，i，a);

输出为：␣123␣ ␣ ␣12.3457

printf(“%-4d%10.4f”，i，a);

输出为： 123␣ ␣ ␣ ␣12.3457

printf(“%4d%-10.4f”，i，a);

输出为：␣12312.3457␣ ␣ ␣

(3) l和h

可以与输出格式字符d、f、u等连用，以说明是用long型或short型格式输出数据，如：

%hd 短整型

%lf 精度型

%ld 长整型

%hu 无符号短整型

2. 普通字符

普通字符包括可打印字符和转义字符，可打印字符主要是一些说明字符，这些字符按原样显示在屏幕上，如果有汉字系统支持，也可以输出汉字。

转义字符是不可打印的字符，它们其实是一些控制字符，控制产生特殊的输出效果。

例如：i=123，n=456，a=12.34567，且i为整型，n为长整型。

```
printf("%4d\t%7.4f\n\t%lu\n",i,a,n);
```

输出为：

␣123␣ ␣ ␣ ␣12.3457

␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣456

其中\t为水平制表符，作用是跳到下一个水平制表位，在各个机器中，水平制表位的宽度是不一样的，这里设为８个字符宽度，那么“ \t”跳到下一个８的倍数的列上。

“\n”为回车换行符，遇到“ \n”，显示自动换到新的一行。

在c语言中，如果要输出%，则在控制字符中用两个%表示，即%%。

[例3-2] 输出格式控制符的使用。

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

```
    long int b;
    short int c;
    unsigned  int d;
    char  e;
    float f;
    double g;
    a=1023;
    b=2222;
    c=123;
    d=1234;
    e='x';
    f=3.1415926535898   ;
    g=3.1415926535898;
    printf("a=%d\n",a);
    printf("a=%0\n",a);
    printf("a=%x\n",a);
    printf("b=%ld\n",b);
    printf("c=%d\n",c);
    printf("d=%u\n",d);
    printf("e=%c\n",e);
    printf("f=%f\n",f);
    printf("g=%f\n",g);
    printf("\n");
}
```

执行程序，输出为：

```
RUN
a=1023
a=1777
a=3ff
b=2222
c=123
d=1234
e=x
f=3.141593
g=3.141593
```

3.2.3 getchar（）函数与putchar（）函数

putchar() 与getchar()是对单个字符进行输入输出的函数。

getchar()的功能是返回键盘输入的一个字符，它不带任何参数，其通常格式如下：

```
ch=getchar()
```

ch为字符型变量，上述语句接收从键盘输入的一个字符并将它赋给 ch。

putchar()的作用是向屏幕上输出一个字符，它的功能与 printf函数中的%c相当。putchar()必须带输出项，输出项可以是字符型常量、变量、表达式，但只能是单个字符而不能是字符串。

[例3-3] 输入一个字符，回显该字符并输出其 ASCII码值。

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

```
{
 char ch;
 ch=getchar();
 putchar(ch);
 printf("%d\n",ch);
}
```

运行程序：

```
RUN
 g
g   103
```

需要注意的是，字符输入输出函数定义在头文件 stdio.h**中，故当程序中使用** putchar()**或** getchar()**函数时，必须在** main()**之前用语句：**

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

将stdio.h**包含进来。**

3.2.4 程序应用举例

[例3-4] **下面的程序是一个复数加法的例子。**

```
  #include <stdio.h>
   main()
{
 float a1,b1,a2,b2;
 char  ch;
 printf("\t\t\tcomplexs  Addition\n");
 printf("please input the first complex:\n");
 printf("\t       realpart:");
 scanf("%f",&a1);
 printf("\t   virtualpart:");
 scanf("%f",&b1);
 printf("%5.2f +i %5.2f\n",a1,b1);
 printf("\n please input the second complex:\n");
 printf("\t       realpart:");
 scanf("%f",&a2);
 printf("\t   virtualpart :");
 scanf("%f",&b2);
 printf("%5.2f +i %5.2f\n",a2,b2);
 printf("\n The addition is :");
 printf("%6.3f +i %6.3f\n",a1+a2,b1+b2);
 printf(" program normal terminated,press enter...");
 ch=getchar();
 ch=getchar();
}
```

运行结果如下：

```
RUN
                complexs  addition
please input the  first complex :
     realpart  :1.2
  virtualpart  :3.4
```

```
  1.20 +i 3.40
please input the second complex :
     realpart  :5.6
  virtualpart  :7.8
  5.60 +i 7.80
  The  addition is: 6.800 +i 11.200
   program  normal terminated, press enter....
```

3.3 条件控制语句

在程序的三种基本结构中，第二种即为选择结构，其基本特点是 :程序的流程由多路分支组成，在程序的一次执行过程中，根据不同的情况，只有一条支路被选中执行，而其他分支上的语句被直接跳过。

C语言中，提供if语句和switch语句选择结构，if语句用于两者选一的情况，而 switch用于多分支选一的情形。

3.3.1 if语句

1. if语句的两种基本形式

首先，我们看一个例子，由此了解选择结构的意义及设计方法。

[例3-5] 输入三个数，找出并打印其最小数。

分析：设三个数为A、B、C，由键盘读入，我们用一个变量 MIN来标识最小数， A、B、C与MIN皆定义为int型变量。

每次比较两个数，首先比较 A和B，将小的一个赋给 MIN，再把第三个数 C与MIN比较，再将小的一个赋给MIN，则最后MIN即为A、B、C中最小数。

算法如下：

1) 输入A、B、C。

2) 将A与B中小的一个赋给MIN。

3) 将MIN与C中小的一个赋给MIN。

4) 输出MIN。

将第2）步细化为：若A<B，则MIN <==A，否则：MIN <==B；其流程图见图３-1。

第3）步细化为：若C<MIN，则MIN <==C；其流程图见图3-2。

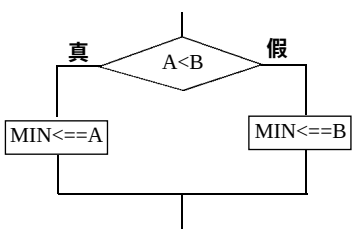

图3-1 例3-5中第2)步的流程图

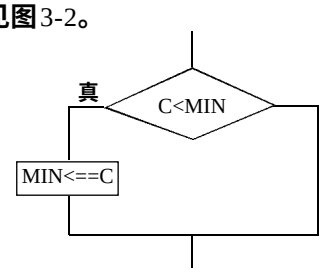

图3-2 例3-5中第3)步的流程图

对应图3-1和图3-2，正是if语句的两种基本形式，与图 3-2对应的if语句的格式为：

if <表达式> 语句

当表达式为真时，执行语句，表达式为假时跳过语句。

与图3-1对应的if语句的格式为：

```
if〈表达式〉
  语句1
else
  语句2
```

当表达式为真时，执行语句１，表达式为假时执行语句 2。无论如何，语句１与语句２每次只能有一个被执行。

要注意的是：if或if...else，包括后面要讲到的嵌套 if，即if...else if...被看成是一条语句，即使其中的语句是包含多条语句的复合语句，仍然如此。

下面是例3-5的源程序：

```
  main()
{
 int a,b,c,min;
 printf("  input a,b,c :");
 scanf("%d%d%d",&a,&b,&c);
 if (a<b)
   min = a;
 else
   min = b;
 if (c<min)
   min = c;
 printf("The result is %d\n",min);
}
```

执行情况如下：

```
  RUN
  input a,b,c: 3 5 2
  The result is :  2
```

这里顺便提一下程序书写的缩排问题，所谓缩排，就是下一行与上一行相比，行首向右缩进若字符，如上例的min = a 、min = b等。适当的缩排能使程序的结构、层次清晰、一目了然，增加程序的易读性。应该从一开始就养成一个比较好的书写习惯，包括必要的注释、适当的空行以及缩排。

2. 复合语句

if语句中，有时需要执行的语句不止一条，这就要用到复合语句。

复合语句，就是用一对花括号括起来的一条或多条语句，形式如下：

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

无论包括多少条语句，复合语句从逻辑上讲，被看成是一条语句。

复合语句在分支结构、循环结构中，使用十分广泛。

[例3-6] 读入两个数x、y，将大数存入x，小数存入y。

分析：x、y**从键盘读入，若**x>=y**，只需顺序打出，否则，应将**x，y**中的数进行交换，然后输出。两数交换必须使用一个中间变量** t，
定义三个浮点数x、y、t。

算法：

1) **读入**x、y**；**

2) **大数存入**x**，小数存入**y**；**

3) **输出**x、y。

第2**）步求精：**

若x<y**，则交换**x**与**y**；**

再求精，x**与**y**交换；**

① t <== x

② x <== y

③ y <== t

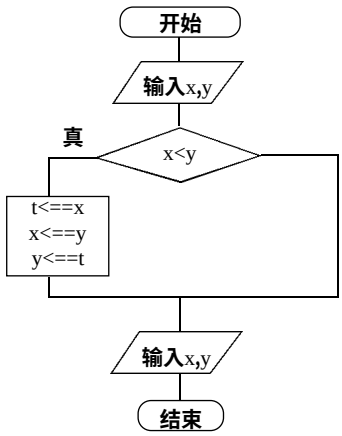

图3-3 例3-6算法流程图

算法的流程图见图3-3**，程序如下：**

```
# include <stdio.h>
main()
 {
    float x,y,t;
    printf("input x,y:");
    scanf("%f%f",&x,&y);
    if (x<y)
      {
        t=x;
        x=y;
        y=t;
      }
    printf ("result:%7.3f\t%7.3f\n",x,y);
  }
```

执行结果：

```
input x,y :43.2  56.7
result :  56.700   43.200
```

3. if...else if **语句**

实际应用中常常面对更多的选择，这时，将 if...else**扩展一下，就得到**if...else if**结构，其一般形式为：**

```
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 语句4
```

对应的流程图见图3-4。

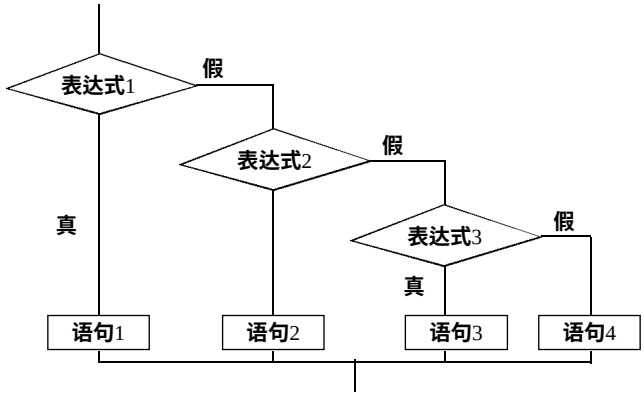

图3-4 if...else if 语句的流程图

[例3-7] 货物征税问题，价格在１万元以上的征 5%，5000元以上1万元以下的征 3%，1000元以上5000以下的征2%，1000元以下的免税，读入货物价格，计算并输出税金。

分析：读入price，计算tax，这是一个较复杂的分支结构程序设计(应注意避免重复征税)。

假定货物的价格在1万元以上，征税应分段累计，各段采用不同税率进行征收。

算法：若price>=10000

则 tax=0.05*（price-10000）; price=10000;

否则，若price>=5000

则 tax=0.03*(price-5000)+tax; price=5000;

否则，若price>=1000

则 tax=0.02*(price-1000)+tax; price=1000;

程序如下：

```
# include <stdio.h>
main()
 {
  float price,tax=0;
  printf("input price:");
  scanf("%f",&price);
  if(price>=10000.0)
 {
    tax=0.05*(price-10000)+tax; price=10000;
 }
  if (price>=5000.0)
 {
    tax=0.03*(price-5000)+tax;price=5000;
 }
  if(price>=1000.00)
 {
   tax=0.02*(price-1000)+tax;
 }
  printf("the tax=%10.3 f",tax);
 }
```

运行程序：

```
RUN
15000
the tax=480.000
```

4. if语句嵌套

在一个if 语句中可以又出现另一个if语句，这称为if语句的嵌套或多重if语句：

```
if <表达式1>
  if<表达式11>
  ……
 else
     语句2；
```

[例3-8] **计算函数**

$$y=\begin{cases}1 & x>0\\0 & x=0\\-1 & x<0\end{cases}$$

流程图见图3-5。

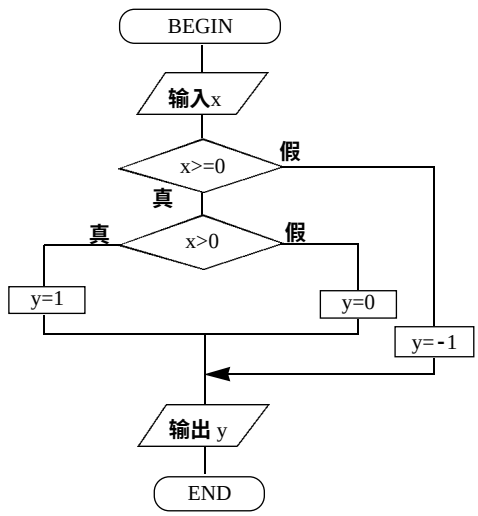

图3-5 例3-8的流程图

源程序如下：

```
main()
 {
  float x,y;
  printf("input x,y:");
  scanf("%f",&x);
  if (x>=0)
    if (x>0)
       y=1;
    else
```

```
        y=0;
    else
        y=-1;
printf("y=%4.0f\n",y);
 }
```

对多重if，最容易犯的错误是 if与else配对错误，例如，写成如下形式：

```
y=0;
if (x>=0)
  if (x>0)
   y=1;
 else
   y=-1;
```

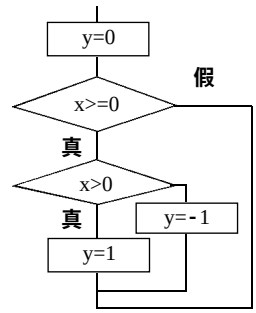

图3-6 错误的算法流程图

从缩排上可以看出，作者希望 else是与if x>=0配对，但是C语言规定else总是与离它最近的上一个if 配对，结果，上述算法的流程图变成图 3-6，完全违背了设计者的初衷。

改进的办法是使用复合语句，将上述程序段改写如下：

```
y=0;
if  (x>=0 )
  {
   if  (x>0)
      y=1;
  }
else
   y=-1;
```

3.3.2 switch 语句

if 语句只能处理从两者间选择之一，当要实现几种可能之一时，就要用 if...else if甚至多重的嵌套if来实现，当分支较多时，程序变得复杂冗长，可读性降低。 C语言提供了switch开关语句专门处理多路分支的情形，使程序变得简洁。

switch语句的一般格式为：

```
switch <表达式>
  case 常量表达式１：语句序列１；
                          break;
  case 常量表达式２：语句序列２；
                          break;
   ……
  case  常量表达式n: 语句n;
                          break;
  default:  语句n+1;
```

其中常量表达式的值必须是整型，字符型或者枚举类型，各语句序列允许有多条语句，不需要按复合语句处理，若语句序列 i为空，则对应的break语句可去掉。图3-7是switch语句的流程图。

特殊情况下，如果switch表达式的多个值都需要执行相同的语句，可以采用下面的格式：

```
switch (i)
 {
   case 1:
   case 2:
   case 3:  语句1；
            break;
   case 4:
   case 5:  语句2；
            break;
   default: 语句3；
 }
```

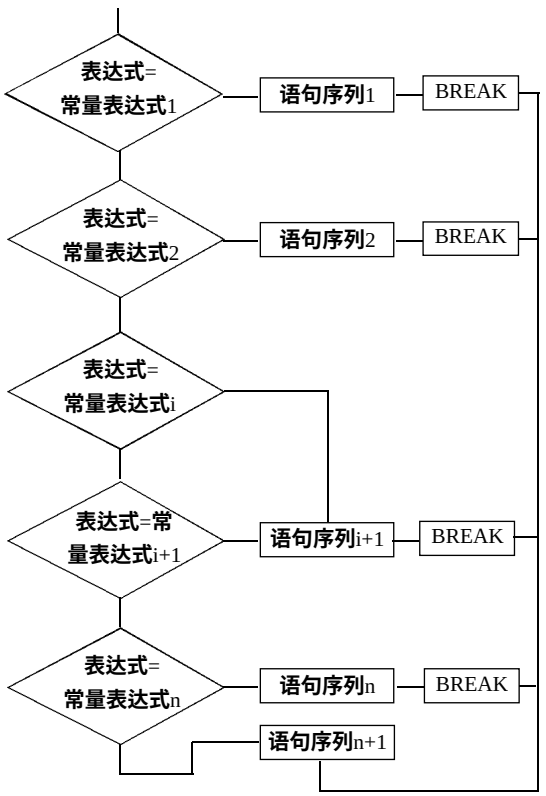

图3-7 switch 语句的流程图

当整型变量i的值为1、2或3时，执行语句1，当i的值为4或5时，执行语句2，否则，执行语句3。

[例3-9] 输入月份，打印1999年该月有几天。

程序如下：

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

```
    printf("please input the month number :");
    scanf("%d",&month);
    switch (month)
     {
       case 1:
       case 3:
       case 5:
       case 7:
       case 8:
       case 10:
       case 12:  day=31;
                 break;
         case 4:
         case 6:
         case 9:
         case 11:day=30;
                 break;
         case 2: day=28;
                 break;
       default : day=-1;
    }
      if day=-1
       printf("Invalid month input !\n");
    else
       printf("1999.%d has %d days \n",month,day);
}
```

3.3.3 程序应用举例

[例3-10] **解一元二次方程**$ax^2+bx+c=0$，a、b、c**由键盘输入。**

分析：对系数a、b、c**考虑以下情形**

1) **若**a=0：

① b<>0，**则**x=-c/b ；

② b=0， **则：**① c=0， **则**x**无定根；**

② c<>0，**则**x**无解。**

2) **若**a<>0;

① b^2-4ac>0，**有两个不等的实根；**

② b^2-4ac=0，**有两个相等的实根；**

③ b^2-4ac<0，**有两个共轭复根。**

用嵌套的if**语句完成。程序如下：**

```
 #include <math.h>
# include <stdio.h>
 main()
 {
   float a,b,c,s,x1,x2;
   double t;
```

```
    printf(" please input a,b,c:");
    scanf("%f%f%f",&a,&b,&c);
    if (a==0.0)
      if(b!=0.0)
         printf("the root is :%f\n",-c/b);
      else if (c==0.0)
              printf("x is inexactive\n ");
           else
              printf("no root!\n");
    else
     {
       s=b*b-4*a*c;
       if(s>=0.0)
          if(s>0.0)
           {
            t=sqrt(s);
            x1=-0.5*(b+t)/a;
            x2=-0.5*(b-t)/a;
          printf("There are two different roots:%f and%f\n",x1,x2);
            }
          else
            printf("There are two equal roots:%f\n",-0.5*b/a);
       else
       {
         t=sqrt(-s);
         x1=-0.5*b/a;        /*实部*/
         x2=abs(0.5*t/a);    /虚部的绝对值*/
         printf("There are two virtual roots:");
         printf("%f+i%f\t\t%f-i%f\n",x1,x2,x1,x2 );
       }
     }
   }
```

运行结果如下：

```
RUN
please input a,b,c :1 2 3
There are two virtual roots:
        -1.000000 + i 1.000000  -1.000000 - i 1.000000
RNU
please input a,b,c :2 5 3
There are two different roots :  -1.500000  and  -1.000000
RNU
please input a,b,c :0 0 3
No root!
```

3.4 循环控制语句

循环控制结构（又称重复结构）是程序中的另一个基本结构。在实际问题中，常常需要进行大量的重复处理，循环结构可以使我们只写很少的语句，而让计算机反复执行，从而完

成大量类同的计算。

C语言提供了while语句、do...while语句和for语句实现循环结构。

3.4.1 while语句

while语句是当型循环控制语句，一般形式为：

```
while <表达式> 语句;
```

语句部分称为循环体，当需要执行多条语句时，应使用复合语句。

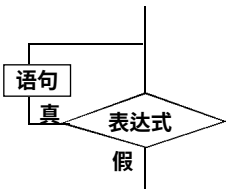

图3-8 while语句的流程图

while语句的流程图见图3-8，其特点是先判断，后执行，若条件不成立，有可能一次也不执行。

[例3-11] 求n！

分析：n！= n*（n-1）*（n-2）* .. 2*1，0!=1。即S_0=1，S_n=S_{n-1}*n。可以从S_0开始，依次求出S_1、S_2、...S_n。

统一令S等于阶乘值，S的初值为0！=1；变量i为计数器，i从1变到n，每一步令S=S*i，则最终S中的值就是n！。

流程图见图3-9，程序如下：

```
main()
{
  int n,i;
  long int s;
  printf("  please input n (n>=0) :");
  scanf("%d",&n);
  if (n>=0)
   {
      s=1;
      if (n>0)
      {
            i=1;
            while (i<=n)
            {
             s*=i;
             i=i+1;
            }
      }
    printf("%d! = %ld \n",n,s);
     }
    else
       printf("Invalid input! \n");
}
```

运行结果如下：

```
RUN
please input n(n>=0):0
0!= 1
```