# 第 3 章

# 基本控制结构及其应用

C 语言的最大特点之一是结构化，结构化使程序结构简洁、明了。结构化程序设计包含了程序的控制结构和程序设计的思想。

## 3.1 算法及结构化程序设计

### 3.1.1 算法及其特征

目前的软件开发已经步入工程化阶段。一个大型软件的开发通常需要经历“规划”、“需求分析”、“设计”、“编码”、“测试”和“运行维护”等几个阶段。它就像是一个大型的工程。其中，正确的“需求分析”对程序设计是至关重要的，而“编码”就是对需求的具体实现。在“编码”过程中，算法是非常重要的，良好的算法，往往会得到事半功倍的效果，算法思想决定了程序的质量和性能。

而目前许多软件的开发需要处理大量的关系复杂的数据，这些数据通常都具有一定的结构性。因此，软件设计的实质就是设计合适的数据结构和基于这个数据结构的算法。于是著名的计算机专家沃恩教授提出了一个著名的公式：

程序＝数据结构＋算法

算法是程序设计中一个非常重要的概念，所谓算法就是处理问题的方法和步骤。一个完整的算法应具有如下特征。

- 有穷性：一个合理的算法要求能在执行有限步之后结束。

例如：

$$N!=1*2*3*\cdots*(N-1)*N$$

其中 N 是一个特定数，例如 50，那么上面的描述就可以称为一个算法。而下面的式子：

$$sum=1+2+3+\cdots+N+\cdots$$

就不能称之为算法，因为该描述在执行有限步后仍不能结束，不满足“有穷性”的特点，它只能称为一个计算方法。

- 确定性：算法的每一步执行，其顺序和内容都必须有确切的规定，不能有二义性。
- 可执行性：算法的所有操作都是能通过计算机程序代码实现的，即可操作性。
- 0 个或多个输入；1 个或多个输出。

算法既然是为解决特定问题而设计的，那么无任何输出信息的算法就是无意义的。

至于输入,有时可能已经把初始化数据直接写在代码中的某个合理的位置,不必要再从键盘输入数据,但这样程序是不灵活的,它只能求解某一特定的问题。更多情况下,程序的运行是要求有数据输入的,这样可以满足不断变化的数据对求解的要求。建议编程时要充分考虑程序在应用上的灵活性。

为了更深入地理解算法,下面介绍几个有关算法的实例。

**【例 3-1】** 求解一元二次方程的算法。

求解一元二次方程是大家非常熟悉的,利用大家熟悉的问题说明一种概念,更有利于对问题的理解。设有一元二次方程 $ax^2+bx+c=0$,其中 a、b 和 c 是不等于 0 的实数。在用计算机求解此方程时,应首先给出求解方程的算法,然后再根据此算法和相关的编码规则编写求解该方程的程序,具体参考步骤如下。

(1) 首先计算 p = b * b−4 * a * c。

(2) 判断 p 是否≥0,若结果为“真”,说明它只有实根,则执行步骤(3),若结果为“假”,则说明有虚根,执行步骤(4)。

(3) 求二实根。

$$x1=(-b+\mathrm{sqrt}(p))/(2*a)$$
$$x2=(-b-\mathrm{sqrt}(p))/(2*a)$$

然后输出计算结果。

(4) 求虚根。

$$x1=-b/(2*a)+i\ \mathrm{sqrt}(-p)/(2*a)$$
$$x2=-b/(2*a)-i\ \mathrm{sqrt}(-p)/(2*a)$$

然后输出计算结果,结束计算。

以上步骤就是解此一元二次方程的一个算法。其中 sqrt()是求平方根函数,具体代码如下:

```
#include "stdio.h"
#include "math.h"
void main()
{   float a,b,c,p,q,x1,x2,tmp;
    printf("Please input a/b/c \n");
    scanf("%f%f%f",&a,&b,&c);
    p=-b/(2.0*a);
    tmp=b*b-4.0*a*c;
    if(tmp>=0.0)                                    //求实根
    {
        q=sqrt(tmp)/(2.0*a);
        x1=p+q;
        x2=p-q;
        printf("x1=%f\nx2=%f\n",x1,x2);
    }
    else                                            //求虚根
    {   q=sqrt(-tmp)/(2.0*a);
```

```
            printf("x1=%f+i%f\n",p,q);
            printf("x2=%f-i%f\n",p,q);
        }
    }
```

### 3.1.2　算法的类型与结构

计算机通常可处理“数值算法”和“非数值算法”。“数值算法”常用于科学计算，而“非数值算法”则广泛用于各类数据的处理，它常常要涉及大数据量和复杂的数据结构。

不管是何种算法，都是由“结构”和“原操作”所构成。最基本的结构有顺序、分支和循环三种，原操作包括输入、输出、表达式求值、变量赋值、比较两个变量等。下面分别描述这三种基本结构。

**1. 顺序结构**

顺序结构是由一组顺序执行的程序块所组成的，每一个程序块可以是一个非转移语句、分支语句、循环语句或这些语句的排列、嵌套，它是任何一个算法都离不开的一个基本主体结构。例如有两个处理块所构成的顺序结构如图 3-1 所示。

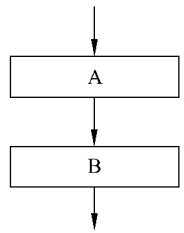



图 3-1　顺序结构示意图

**2. 分支结构**

分支结构根据分支条件的取值来决定程序执行的走向，分支结构的示意图如图 3-2 所示。

分支结构也是一种基本和常用的结构，它提供了根据条件取值来选择不同处理块的方法。如图 3-2 所示，如果判断结果是“真”(T)，就执行 A 程序块的操作，如果判断结果是“假”(F)，就执行 B 程序块的操作。

**3. 循环结构**

循环结构是一种对某一处理块反复执行指定次数的结构，图 3-3 是循环结构的示意图，循环结构又分为“当循环”和“直到型循环”。“当循环”的特点是先判断，后执行循环体，如图 3-3(a)所示，程序先执行判断，如果判断结果为“真”，就执行 A 程序块，否则结束循环；“直到型循环”是先执行，后判断，如图 3-3(b)所示，程序先执行 A 程序块，然后再进行判断，如果判断结果为“假”，则继续运行 A 程序块，否则结束循环。

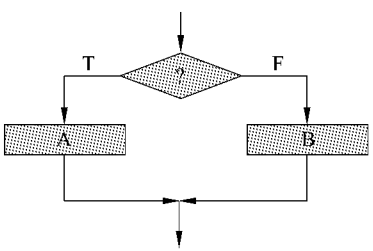
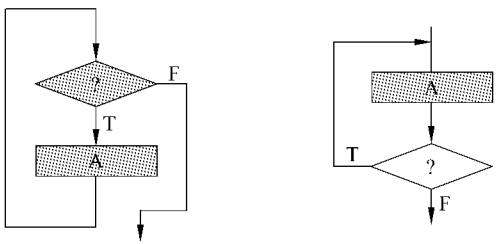



图 3-2　分支结构示意图




(a) 当循环　　(b) 直到型循环

图 3-3　循环结构示意图

## 3.2 顺序结构程序设计

顺序结构的程序设计是最简单的程序设计。顺序结构的程序由一组顺序执行的程序块所组成。最简单的程序块是由若干顺序执行的语句所构成的,这些语句可以是赋值语句、输入输出语句等。

计算机处理的问题难易程度差别很大。简单问题的程序,其结构可能完全是顺序的,即在执行时是顺序逐句执行。而复杂问题的程序则不仅包含顺序结构,还可能包含分支结构和循环结构。本节主要介绍顺序结构的程序设计方法。

下面通过一些简单的顺序结构程序设计的例子,使读者了解并总结顺序结构程序设计的方法与特点。

**【例 3-2】** 从键盘上输入一个小写字母,要求用其对应的大写字母输出。

对于这个问题,算法设计实际上比较简单,大家已经清楚,大小写相应字母的 ASCII 值相差 32,若已知小写字母,只要将其 ASCII 值减 32,就变成相应大写字母的 ASCII 值。

```
#include "stdio.h"                                  //嵌入头文件
void main()
{
    char c1,c2;                                     //定义字符型变量 c1 和 c2
    printf("please input a Lower-case:");           //输出提示语句
    c1=getchar();                                   //从键盘接收一个字符
    c2=c1-32;                          //将小写字母通过 ASCII 值的计算转换成大写字母
    printf("\n  Upper-case  is%c",c2);              //输出转换后的结果
}
```

运行时的输出情况如下:

```
please  input  a  Lower-case: a<回车>
Upper-case  is  A
```

这就是一个典型的顺序结构的程序,其执行步骤是一步一步顺序往下执行的,没有任何转向操作。

编程的关键是设计算法。一个简单的问题,其处理算法比较简单,就能很快把它的算法设计出来。但对处理较复杂的问题的算法,就很难一下子分析得十分透彻,而需要逐步分析清楚,即需要先从总体上分析其粗略的算法框架,再对粗略算法的每一步进一步细化,如此下去,直至细化到每一步都足够简单,能用一个或几个 C 的语句来实现。于是就出现算法的顶层设计、第二层设计……这是十分有效而且常用的算法设计方法,即"自顶向下,逐步求精"的设计方法。

## 3.3　分支结构程序设计

分支结构也是程序的基本结构之一。所谓分支结构，就是根据不同的条件，选择不同的处理块。

在 C 语言中，条件分支结构可通过 if 语句和 switch 语句实现。if 语句有 if、if-else 和 if-else if 三种形式，而且条件判断语句还可以嵌套。

### 3.3.1　if 分支

if 分支是条件判断语句中最简单的形式，具体形式如下：

```
if (表达式) 语句
```

例如，下面的语句：

```
if (a>b) printf("a");
```

就是判断如果 a 的值大于 b 的值，打印字符 a。

### 3.3.2　if-else 分支

if-else 分支是标准形式的条件分支，其语句形式如下：

```
if(条件表达式)
{
    程序块 1;
}
else
{
    程序块 2;
}
```

其中，“程序块 1”和“程序块 2”可以是语句或者复合语句。

if-else 语句的处理过程是先计算 if 后面的“条件表达式”，若结果为非 0 值（即逻辑上为真），则执行“程序块 1”，否则执行“程序块 2”。可见，if-else 是二选一的分支结构。下面举例说明 if-else 结构的应用方法。

**【例 3-3】** 输入一个字符，判断它是否是 0～9 的阿拉伯数字。

对于这个问题，首先要弄清楚判断输入的内容是否是阿拉伯数字。从 ASCII 码表可知，0～9 阿拉伯数字的 ASCII 值范围为 48～57，因此就可以判断条件，如果 ASCII 值在 48～57 范围内，就是阿拉伯数字，否则不是。参考程序如下：

```
#include<stdio.h>
void main()
{
    char c;
```

```
    printf("\n  please  input  a  character:");
    c=getchar();                              //从键盘输入一个字符
    if(c>=48&&c<=57)                          //根据 ASCII 码判断其是否是阿拉伯数字
        printf("It is a number.\n");          //若是数字,输出相应的提示
    else
        printf("It is not a number.\n");      //若不是数字,也输出相应的提示
}
```

根据题义,这时把 0～9 看成是一字节的字符,而不是二字节的整型数。该程序执行时,当输入数不在 48～57 范围之间时,则显示“It is not a number.”,否则显示“It is a number.”。if 后面用圆括号“(”和“)”括住的表达式经常是条件表达式或逻辑表达式(但可以是任何表达式),表达式必须用圆括号括住。

### 3.3.3 多分支 if-else if-else 形式

上面谈到的分支实际上都比较简单,但更多的可能是利用多分支形式,多分支形式如下:

```
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 m) 语句 m
else 语句 n
```

下面通过实例介绍多分支结构的应用。

【例 3-4】 有一个函数:

$$y = f(x) = \begin{cases} 0 & (x < 0) \\ x & (0 \leqslant x \leqslant 50) \\ x^2 & (x > 50) \end{cases}$$

用多分支结构编写一程序,根据用户输入的自变量 x 的值,计算函数值。

```
#include<stdio.h>
void main()
{   float x;
    printf("input  x: \n");
    scanf("%f",&x);
    if(x<0.0)
        printf("y=0\n");
    else if(x>=0.0&&x<=50.0)
        printf("y=%f\n",x);
    else
        printf("y=%f\n",x*x);
}
```

从上面例子可以看出，当条件 x<0.0 满足时就输出“y=0”；当条件 x<0.0 不满足时，就进行另一个分支的判断“x>=0.0&&x<=50.0”，如果这个条件满足，就输出 y=x 的值（参见题目中的分段函数）；如果这个条件还不满足，就输出 $y=x^2$ 的结果。这就是多分支判断。

值得注意的是，通常情况下，if-else-语句和 if-else if-else 语句可以互相嵌套使用，而且和下面将要讲到的 switch 语句之间是可以相互转换的。

### 3.3.4　条件分支的嵌套

在实际编程中，还经常会遇到条件分支的嵌套。所谓条件分支嵌套就是在一个分支中可以嵌套另一个分支。比如，有如下的结构就是条件分支嵌套的情形之一。

```
if (条件表达式 a)
    程序段 a1;
    if (条件表达式 b)
        程序段 b1;
    else
        程序段 b2;
    程序段 a2;
else
    程序段 a3;
```

从上面的形式可以看出，在程序段 a1 和程序段 a2 中又包含了条件分支，利用条件分支嵌套可以实现多分支控制。下面是一个条件分支嵌套的例子。

【例 3-5】　求解 $ax^2+bx+c=0$ 的完全解。

```
1:   #include<stdio.h>
2:   #include<math.h>                          //由于用到平方根函数,要用此头文件
3:   void main()
4:   {
5:   float a,b,c,x1,x2,p,q,m;                  //定义实型变量
6:   scanf("%f%f%f",&a,&b,&c);                  //从键盘输入方程的三个系数
7:   if ((a==0.0)&&(b==0)&&(c==0))              //条件判断
8:      printf("any value");
9:   else if ((a==0)&&(b!=0))
10:     printf("x1=x2=%f",-c/b);
11:   else
12:     {
13:      m=b*b-4.0*a*c;
14:      if(m>=0)                               //嵌套判断,不存在虚根
15:      {  x1=(-b+sqrt(m))/(2.0*a);
16:         x2=(-b-sqrt(m))/(2.0*a);
17:         printf("x1=%f\n",x1);
18:         printf("x2=%f\n",x2);
```

```
19:       }
20:    else                                    //存在虚根
21:       {  p=-b/(2.0*a);
22:          q=sqrt(-m)/(2.0*a);
23:          printf("x1=%f+%fi\n",p,q);
24:          printf("x2=%f-%fi\n",p,q);
25:       }
26:      }
27:    }
```

对于求解一元二次方程,实际上大家很熟悉,在这里就是介绍如何通过计算机求解。大家知道,对于一元二次方程,如果输入的三个系数都是 0,这是极端的情况,那么,此时的方程根为任意值,这在程序的第 7 和第 8 行表现出了此情形;如果系数 a 为 0,那实际上一元二次方程就转化为一元一次方程,这时由于方程的本身是一元二次方程,那么就要体现两个根,这才符合数学上的概念,此时的根是两个相等的根,第 9 行和第 10 行体现了这种情形;如果三个系数都不是 0,那么这就要考虑虚根的问题,这时候,在第 11 行的 else 后面就要出现是否出现虚根的条件判断,从程序中的第 7、第 9 和第 11 行构成了一个 if-else if-else 条件判断结构,从第 12 行到第 26 行实际上是上述 if-else if-else 条件判断结构的 else 后面的语句,它是一个程序块,在这个程序块中嵌套了条件判断语句,这个嵌套判断从第 14 行开始,第 25 行结束,在这个嵌套判断中解决了虚根的问题。

值得注意的是,在这个程序中有几对括号,这几对括号对于清晰程序结构起到了重要的作用。如第 15 和第 19 行、第 21 和第 25 行,分别是一对括号。如果没有第 21 和第 25 行的括号,那么在逻辑上,只有第 21 行属于嵌套条件判断的 else 后面的语句,而第 22 行～24 行语句则属于第 7、第 9 和第 11 行构成的 if-else if-else 条件判断的 else 语句后面的内容。这样整个逻辑关系破坏了,程序的运行结果就会出错。如果没有第 21 和第 26 行的一对括号,也会引起类似的问题。因此,内嵌的括号如果仅括住了一个语句,则可以省略,如果被括住的是多个语句,则括号不能省略。其实,为了使程序清晰易读并减少出错几率,建议对所有的嵌套 if-else 语句组,都使用括号来明确其配对关系。请读者运行一下这个程序,并消去上述的几对括号,看看结果是否仍然与保留括号时的结果一致。这样就可以加深体会了。

此外,读者可能已经注意到,这个例题中的代码并不是向左对齐的,这样是为了清晰地体现嵌套结构,这对于提高程序的可读性很重要,建议大家在编写程序时一定要注意程序的层次结构,尤其带有嵌套的情况。更应当注意的是,即将在后面介绍的循环也存在嵌套的情况,也请大家注意层次结构。

### 3.3.5 开关分支

开关(switch)分支是分支结构的另一种形式,执行时它根据条件的取值来选择程序中的一个分支。

其程序形式如下:

```
switch (表达式 e)
{
    case  常量表达式 c1:
          程序段 1;
          break;
    case  常量表达式 c2:
          程序段 2 ;
          break;
    …
    case  常量表达式 cn:
          程序段 n;
          break;
    default:
          程序段 n+1;
          break;
}
```

其执行流程是首先计算表达式的值，然后判断表达式的值与 c1、c2、…、cn 中的哪个值相等，若与某个 ci 值相等，则执行其下的相应的语句组。若不与任何一个 ci 值相等，则执行 default 下面的语句组。在执行某一分支中的语句组时，遇 break 语句则退出 switch-case 结构，即程序控制转移至该结构中花括号之后的语句。

**【例 3-6】** 根据学生成绩的等级打印出分数段。

```
#include<stdio.h>
void main()
{  char grade;                                          //定义字符型数据
   printf("input the grade(A,B,C,D,E):");               //提示语句
   scanf("%c",&grade);                                  //输入字符
   switch (grade)                                       //switch 分支
   {    case 'A' : printf("90- 100\n"); break;
        case 'B': printf("80-89\n");   break;
        case 'C': printf("70-79\n");   break;
        case 'D': printf("60-69\n");   break;
        case 'E': printf("0-59\n");    break;
        default: printf("error\n");
   }
}
```

在使用 switch-case 分支时，应注意以下几点。

(1) switch 后面的表达式可以是整型、字符型或枚举类型表达式。

(2) case 后面的判断值要求是一个整常量表达式，它可以是一个整数、字符常量、枚举常量或其他整常量表达式。

(3) 各分支语句组中的 break 语句使控制退出 switch 结构。若没有 break 语句，则程序将继续执行下面一个 case 中的语句组。

如果把上述代码变成下面的形式:

```
#include<stdio.h>
void main()
{    char grade;                                          //定义字符型数据
     printf("input the grade(A,B,C,D,E):");               //提示语句
     scanf("%c",&grade);                                  //输入字符
     switch (grade)                                       //switch 分支
     {    case 'A': printf("90-100\n");
          case 'B': printf("80-89\n");
          case 'C': printf("70-79\n");
          case 'D': printf("60-69\n");
          case 'E': printf("0-59\n");
          default: printf("error\n");
     }
}
```

在上面的开关分支语句中,所有的 break 语句都没有了。若此时输入的字符是 A,则 A～E 的分支和 default 分支全部执行;若输入的字符是 B,则 B～E 的分支和 default 分支全部执行。也就是说,分支语句的执行,是从相匹配的分支开始,遇到第一个 break 语句结束。

(4) 在开关分支结构中,各个 case 及 default(default 之后有 break 语句时)的次序是任意的,但各个 case 后的判断值必须不同。

(5) 在开关分支中,default 部分不是必需的,如果没有 default 部分,则当表达式的值与各 case 的判断值都不一致时,则程序不执行该结构中的任何部分。例如:

```
switch (表达式 C)
{
    case  0:
        x+=5; break;
    case  1:
        y+=5; break;
    case  2:
        z+=5; break;
}
```

当 C 的值既不是 0 也不是 1 或 2 时,则程序不执行该开关分支中的任何语句。

增加 default 分支会给逻辑检查带来很多方便。例如,如果用 switch 语句来处理数目固定的条件,而且认为这些条件之外的值都属于逻辑错误,则可以增加一个 default 分支来识别逻辑错误。

(6) 尽管最后一个分支之后的 break 语句可以省略,但推荐保留它。在最后一个分支之后有 break 语句是程序设计的好习惯。因为用户写的程序要被自己或他人维护,例如要在最后一个分支之后增加几个 case 分支,如果没有注意到最后一个分支之后没有