第3章

程序控制结构

经过第2章的学习，我们已经能够合理地表达程序中的数据，并能够对这些数据进行基本的运算。但实际问题的求解过程往往是十分复杂的，它不仅包括顺序的计算步骤，还包括许多可供选择的不同计算路径，或者一些需要重复执行若干次的计算块。而这些计算路径的选择和计算块的重复控制都依赖于对某些设定条件的判断。对一个实际问题求解过程中所包含的各个计算块间的逻辑关系的完整描述就体现了程序的控制结构。

程序的控制结构包括顺序、选择、循环，这三种基本结构经过反复嵌套可以组成各种复杂程序。C语言提供了多种语句来实现这三种基本控制结构。本章介绍这些基本语句及其在常用算法（枚举法、迭代法）中的应用，并以此为基础使读者熟练掌握结构化程序设计方法，为后面各章的学习打下基础。

3.1 结构化程序设计

20世纪60年代末，国际著名计算机专家E. W. Dijkstra首次提出了“结构化程序设计”的思想。它规定了一套方法，使程序具有合理的结构，以保证程序的正确性。这种方法要求程序设计人员按照一定的结构形式来设计和编写程序，使程序易读、易理解、易修改和易维护。C语言是一种适合结构化程序设计方法的语言。

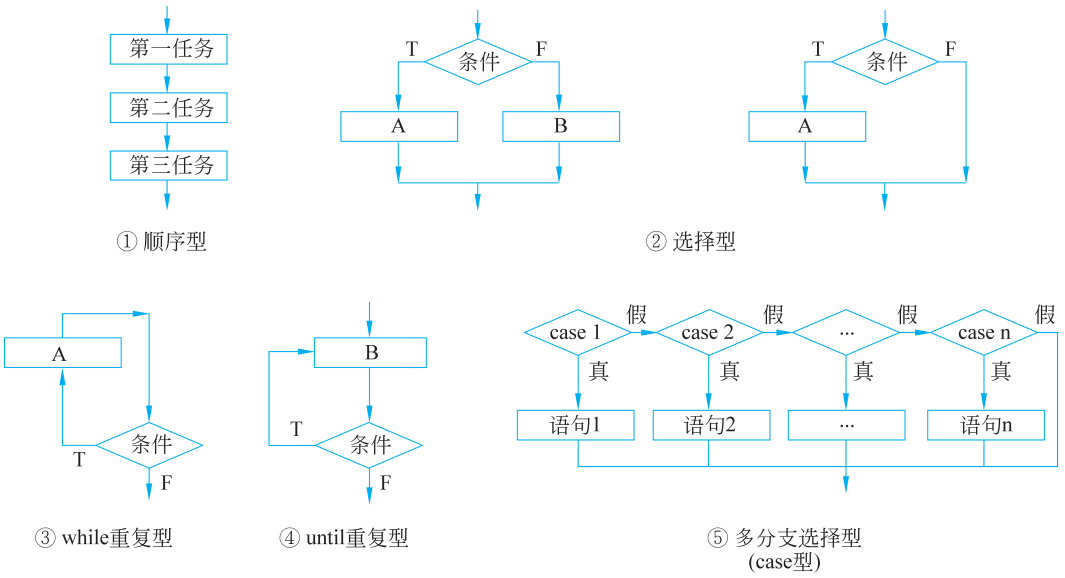
3.1.1 结构化程序

一般地，实际问题的求解过程可由顺序的计算步骤、可选择的不同计算路径、需要重复执行若干次的计算块三类子过程组成。在结构化程序中我们将之分别对应于顺序、选择、循环三种基本结构，这三种结构之间允许嵌套，由这三种基本结构经过反复嵌套即可构成任意复杂的结构化程序。

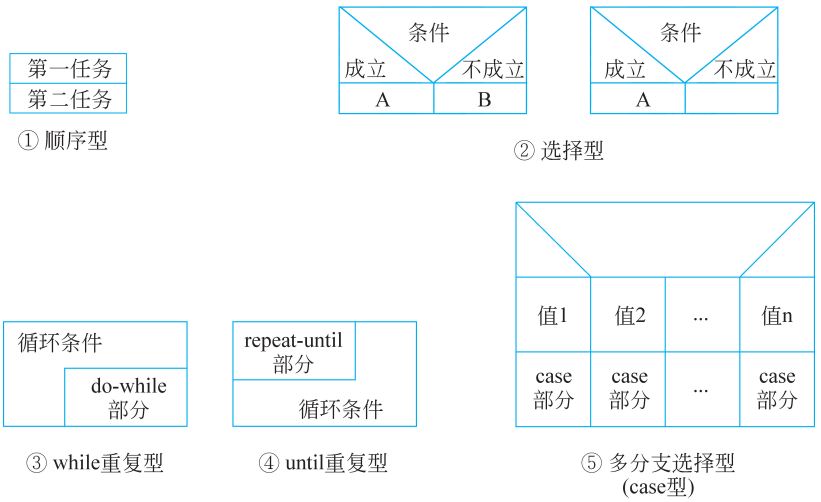
为使读者更好地理解结构化程序的概念，我们首先引入局部程序块的概念。局部程序块是指一对花括号之间的一段C语言程序，其在逻辑上是一个整体。因此，在逻辑上可以认为一个局部程序块是一条单一语句，C语言中称之为复合语句，或者语句块。在算法流程图中可以用矩形框来表示一个语句块，这样做能够隐藏语句块内部语句的逻辑关系，而使得整个程序中各个语句块间的

逻辑关系更为清晰。

顺序结构的功能是按照各个语句块排列的先后次序依次执行,其对应的流程图画法如图 3-1 中①所示。C 语言中的所有非转移语句(比如:赋值语句等)都支持顺序结构。



(a) 传统流程图



(b) N-S流程图

图 3-1 五种基本控制结构流程图的规范画法

选择结构的功能是根据给定的条件从两条或多条可能的路径中选择一条,其对应的流程图画法如图 3-1 中②、⑤所示。C 语言中的 if 语句和 switch 语句支持选择结构。

循环结构的功能是满足给定条件时反复执行内嵌语句块,直至条件不满足时离开。依据条件判定的先后可分为当型循环(先判定型)、直到型循环(后判定型)两种结构,其对

应的流程图画法如图 3-1 中③、④所示。C 语言中的 while 语句和 for 语句支持当型循环结构，do-while 语句支持直到型循环结构。

为使用流程图描述结构化程序，必须限制流程图只能使用图 3-1 所给出的 5 种基本控制结构。任何复杂的程序流程图都应由这五种基本控制结构组合或嵌套而成。这样规定的主要原因是，在传统流程图中表示程序控制流程的箭头可以不受任何约束的随意转移控制，这会破坏结构化程序设计的精神。

3.1.2　结构化程序设计方法

结构化程序设计方法是在语句级上的问题分解与抽象，即经过多层分解后的子问题是能够用 C 语句直接表达和计算的。因此，它是程序设计工作中最根本的方法，其基本思想可以概括为以下 3 点：

(1) 采用顺序、选择和循环三种基本结构作为程序设计的基本单元。这样设计的程序有以下 4 个特征：只有一个入口；只有一个出口；无死语句；无死循环。

(2) 尽量避免使用跳转语句。如果要使用，尽可能不使用多于一个跳转的语句标号；且只在一个单入口单出口的程序结构内使用跳转语句，并且只往前跳转，不允许往后跳转。

(3) 采用“自顶向下、逐步求精”的程序设计方法。

在运用计算机求解问题时，如果问题比较简单，则比较容易确定算法，甚至直接编出程序。因此，对于一个复杂的大问题，我们希望将其分解细化为若干个独立的小问题，然后逐一解决这些小问题，必要时再对每个小问题进一步分解，如此反复，直至分解后的小问题均能直接运用 C 语句运算。这种通过多层分解、逐步求精，最后完全确定算法的技术，通常称为“自顶向下、逐步求精”的程序设计方法，它也是结构化程序设计的最好方法。

自顶向下、逐步求精法程序设计的一般步骤如下：

(1) 对实际问题进行全局分析，确定数学模型。

(2) 确定程序的总体结构，将整个问题分解成若干相对独立的子问题。

(3) 确定子问题的具体功能及其相互关系。

(4) 将子问题进一步细化，直至能用高级语言表示为止。

为更好地说明结构化程序设计方法的应用过程，以例 3.1 的设计过程为例进行详细分析和讲解。

例 3.1　输入两个整数 start、end，编程求 start～end 内的所有素数。

问题分析：该问题本质是在一定范围内寻找满足特定条件的整数。这里的一定范围由输入量直接给出；特定条件即判断某个整数是否是素数。依据素数的定义，对于整数 i，需要在 2～i－1 范围内判断是否存在 i 的约数，若存在则不是素数；反之，即为素数，直接输出 i 即可。

“自顶向下、逐步求精”法程序设计过程如图 3-2 所示。根据问题的复杂程度，在其中的第(3)、(4)步之间还可以继续添加子问题的细化步骤，细化至每一个子问题都可以用 C 语句直接实现为止。

<table>
<tr><th>(1) 程序总体结构</th><th>(2) 子问题的细化</th><th>(3) 子问题的进一步细化</th><th>(4) 编码</th></tr>
<tr><td>子问题 1：数据表示
(用变量定义语句实现)</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="4">……</td></tr>
<tr><td>子问题 2：输入 start、end
(用输入语句实现)</td></tr>
<tr><td rowspan="3">子问题 3：输出 start～end 内的素数
解题思路：对范围内的每一个整数 i 进行判断和输出。(用循环结构实现)</td><td>设标志位 flag：0 表示非素数；1 表示素数
初始假设 i 是素数，flag 置 1
(用赋值语句实现)</td></tr>
<tr><td>子问题 3-1：i 是素数吗?
解题思路：对 2～i－1 范围内的每一个整数 j 进行判断。(用循环结构实现)</td><td>子问题 3－1－1：j 是 i 的约数吗?
解题思路：判断 i%j 是否为 0，为 0 则 flag 置 0。(用单分支 if 语句实现)</td></tr>
<tr><td>子问题 3-2：输出
判断 flag，是 1 则输出 i
(用单分支 if 语句实现)</td><td></td></tr>
</table>

图 3-2 “自顶向下、逐步求精”法程序设计过程示意图

温馨提示：设置标志位 flag 是程序设计中的一种常用技巧，它的引入可以使程序的逻辑更为清晰，使语句块之间的耦合关系更为松散，便于结构化程序设计方法的应用。

在本章后续内容的相应部分已给出本例题的关键代码段，请读者在学完本章后自行组合并调试解决该问题的完整程序。

3.2 顺序结构

3.2 顺序结构

顺序结构是最基本、最简单的程序控制结构，它由若干语句块组成，各块按照排列次序依次执行。这里所谓的块是指非转移语句(如表达式语句等)或者三种基本结构之一。顺序结构应用于比较简单的、不需要根据条件选择执行不同语句，也不需要反复(多次)完成特定操作的程序。一般地，所有的 C 程序都可以划分为：变量定义、数据输入、计算、结果输出 4 部分。即宏观来看，所有的 C 程序都是顺序结构程序。

本小节首先介绍输入、输出在 C 语言中的实现方法，然后再从结构化程序设计的角度分析顺序结构程序设计的要点及应注意的地方。

3.2.1 输入输出在 C 语言中的实现

所谓输入输出是以计算机为主体的输入和输出，即键盘输入和显示器输出。输入和输出是用户与程序实现交互的唯一途径。想使程序每次运行能对不同的数据进行计算就必须设置输入；一个有效的程序必须包含运算结果的输出。

在 C 语言中，数据的输入和输出主要是通过调用 scanf 函数和 printf 函数实现的

(scanf 和 printf 不是 C 语言标准中规定的语句,而是 C 编译系统提供的函数库中的标准函数)。因此,必须熟练掌握 scanf 函数和 printf 函数的应用。

在使用标准库函数的程序中应添加相应的预编译命令。例如:使用标准输入输出库函数时应在源程序开头包含以下预编译命令:

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

或

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

其中,stdio 是 standard input &outupt 的缩写。

常用的输入输出库函数包括:格式输入输出、单字符输入输出两类。

1. 格式输出函数

格式输出函数(printf 函数)关键字的尾字母 f 即为"格式"(format)之意。其功能是按用户指定的格式,把指定的数据输出到显示器屏幕上。在前面的例题中我们已多次使用过这个函数。printf 函数是一个标准库函数,它的函数原型在头文件"stdio.h"中。但作为一个特例,不要求在使用 printf 函数之前必须包含 stdio.h 文件。

printf 函数调用的一般形式为()

```
printf("格式控制串",输出表列)
```

其中:

格式控制串用于指定输出格式。格式控制串由格式说明和原样输出两部分组成。

(1) 格式说明是以%开头的字符串,在%后面跟有各种格式字符,以说明输出数据的类型、形式、长度、小数位数等。如:"%d"表示按十进制整型输出;"%ld"表示按十进制长整型输出;"%c"表示按字符型输出等。

(2) 原样输出部分可由任意字符组成,在输出中起提示作用。

输出表列中给出了各个输出项,要求格式说明和各输出项必须在数量和类型上一一对应,输出时将输出列表中各输出项的值按照格式说明的要求输出在对应格式说明的位置上。

比如:例 3.1 的子问题 3-2 中输出 i 的部分,采用 printf 函数可写为

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

例 3.2　printf 函数的应用。程序如下:

```
#include<"stdio.h">
int main(){
    int a=88,b=89;
```

```
    printf("%d %d\n",a,b);
    printf("%d,%d\n",a,b);
    printf("%c,%c\n",a,b);
    printf("a=%d,b=%d",a,b);
    return 0;
}
```

运行结果为

```
88 89↙
88,89↙
X,Y↙
a=88,b=89
```

温馨提示：本例中 4 次输出了 a、b 的值，但由于格式控制串不同，输出的结果也不相同。第 4 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 5 行的 printf 语句格式控制串中加入的是非格式字符逗号，因此输出的 a、b 值之间加了一个逗号。第 6 行的格式串要求按字符型输出 a，b 值。第 7 行中为了提示输出结果又增加了原样输出部分进行提示。

printf 输出格式说明的一般形式为

```
%[标志][输出最小宽度][.精度][长度]类型
```

其中，方括号[]中的项为可选项。各项的意义介绍如下：

(1) 类型：类型字符用以表示输出数据的类型，其格式符和意义如表 3-1 所示。

表 3-1 printf 格式字符

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 Ox)
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－、＋、＃、空格 4 种，其意义如表 3-2 所示。

表 3-2　printf 标志字符

标志	意　义
－	结果左对齐，右边填空格
＋	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：用十进制整数表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出，若实际位数少于定义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。本项的意义是：如果输出数字，则表示小数的位数；如果输出的是字符，则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

（5）长度：长度格式符为 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 3.3　printf 中格式说明的使用。程序如下：

```
#include<stdio.h>
int 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);
    return 0;
}
```

运行结果为

```
a=15,   15,17,f↙
b=123.123459, 123.123459,123.1235,1.231235e+002↙
c=12345678.123457, 12345678.123457, 12345678.1235↙
d=p,        p↙
```

温馨提示：本例第 7 行中以 4 种格式输出整型变量 a 的值，其中"％5d "要求输出宽度为 5，而 a 值为 15 只有两位故补三个空格。第 8 行中以 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"指定输出宽度为 5，精度为 4，由于实际长度超过 5 故应该按实际位数输出，小数位数超过 4 位部分被截

去。第 9 行输出双精度实数,"%8.4lf"由于指定精度为 4 位,故截去了超过 4 位的部分。第 10 行输出字符量 d,其中"%8c "指定输出宽度为 8 故在输出字符 p 之前补加 7 个空格。

2. 格式输入函数

格式输入函数(scanf 函数),即按用户指定的格式从键盘上把数据输入到指定的变量之中。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 函数相同。C 语言也允许在使用 scanf 函数之前不必包含 stdio.h 文件。

scanf 函数的一般形式为

```
scanf("格式控制串",地址表列);
```

其中:

格式控制串的作用与 printf 函数相同,但原样输入部分需要用户自己输入,且必须与源代码中的完全一致。因此,建议 scanf 中的格式控制串越简单越好,不要在数据间(格式说明间)加间隔符,以免用户输入时由于不知道源代码中写的间隔符是什么,而使得输入值不能被正确获取,最终导致程序出错。

地址表列中给出各变量的地址,地址是由地址运算符 & 后跟变量名组成的。例如:&a,&b,分别表示变量 a 和变量 b 的地址。这个地址就是编译系统在内存中给 a、b 变量分配的地址。在 C 语言中,使用地址这个概念,把变量的值和变量的地址这两个不同的概念区别开来。变量的地址是 C 编译系统分配的,用户不必关心具体的地址是多少。

例如,例 3.1 中子问题 2:输入 start、end。采用 scanf 函数可写为

```
scanf("%d%d",&start,&end);
```

scanf 格式说明的一般形式为

```
%[*][输入数据宽度][长度]类型
```

其中有方括号[]的项为任选项。各项的意义如下:

(1) 类型:表示输入数据的类型,其格式符和意义如表 3-3 所示。

表 3-3 scanf 格式字符

格式	字符意义	格式	字符意义
d,i	输入十进制整数	f 或 e	输入实型数(用小数形式或指数形式)
o	输入无符号八进制整数	c	输入单个字符
X,x	输入无符号十六进制整数	s	输入字符串
u	输入无符号十进制整数		

(2) "*"符:用以表示该输入项读入后不赋予相应的变量,即跳过该输入值。如:

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

当输入为：1　2　3 时，把 1 赋予 a，2 被跳过，3 赋予 b。

（3）输入数据宽度：用十进制整数指定输入的宽度（即字符数）。例如：

```
scanf("%5d",&a);
```

输入：12345678，只把 12345 赋予变量 a，其余部分被截去。又如：

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

输入：12345678，将把 1234 赋予 a，而把 5678 赋予 b。

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

使用 scanf 函数还必须注意以下几点：

（1）scanf 函数中没有精度控制，如“scanf("%5.2f",&a);”是非法的。不能企图用此语句输入小数为 2 位的实数。

（2）scanf 函数中要求给出变量地址，如给出变量名则会出错。如“scanf("%d",a);”是非法的，应改为“scnaf("%d",&a);”才是合法的。

（3）在输入多个数值数据时，若格式控制串中没有指定使用特定的原样输入部分作输入数据之间的间隔，实际输入时默认可用空格、Tab 或回车作间隔。C 编译在碰到空格、Tab、回车或非法数据（如对"%d"输入"12A"时，A 即为非法数据）时即认为该数据输入结束。

（4）在输入字符数据时，如果格式控制串中无原样输入部分，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);
```

输入为：d　e　f，则把'd'赋予 a，' '赋予 b，'e'赋予 c；只有当输入为：def 时，才能把'd'赋予 a，'e'赋予 b，'f'赋予 c。如果在格式控制中加入空格作为间隔，如：

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

则输入时各数据之间可加空格。

例 3.4　scanf 函数的应用。程序如下：

```
#include<stdio.h>
int main (){
    int a,b,c;
    printf("input a,b,c\n");
    scanf("%d%d%d",&a,&b,&c);
    printf("a=%d,b=%d,c=%d",a,b,c);
    return 0;
}
```

温馨提示：在本例中，由于scanf函数本身不能显示提示串，故先用printf语句在屏幕上输出提示，请用户输入a、b、c的值。执行scanf语句，则退出VC++屏幕进入用户屏幕等待用户输入。用户输入7　8　9后按下Enter键。此时，系统又将返回屏幕。在scanf语句的格式串中由于没有原样输入部分在“%d%d%d”之间作输入时的间隔，因此，在输入时要用一个以上的空格或回车键作为每两个输入数之间的间隔。如：

```
7 8 9
```

或

```
7
8
9
```

3. 单字符输出函数

字符输出函数(putchar函数)的功能是在显示器上输出单个字符。其一般形式为

```
putchar(字符变量/常量)
```

例如：

```
putchar('A');                /*输出大写字母A*/
putchar(x);                  /*输出字符变量x的值*/
putchar('\101');             /*输出字符A*/
putchar('\n');               /*换行*/
```

温馨提示：若输出为控制字符则执行相应的控制功能，不在屏幕上显示。

4. 单字符输入函数

单字符输入函数(getchar函数)的功能是从键盘上输入并返回一个字符。其一般形式为

```
getchar();
```

通常把输入的字符赋予一个字符变量，构成赋值语句，如：

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

例3.5　输入输出单个字符。程序如下：

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

```
    char c;
    printf("input a character:\n");          /*提示输入语句*/
    c=getchar();                             /*单字符输入*/
    putchar(c);                              /*单字符输出*/
    return 0;
}
```

运行结果为

```
input a character:↙
j↙
j
```

温馨提示：

（1）getchar 函数只能接受单个字符，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）使用本函数前必须包含文件" stdio.h "。

（3）在 VC++ 屏幕下运行含本函数程序时，将退出 VC++ 屏幕进入用户屏幕等待用户输入。输入完毕再返回 VC++ 屏幕。

（4）程序 5、6 行可用下面两行的任意一行代替：

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

知识小档案：行缓冲

行缓冲是指见到换行符的时候把缓冲区的内容送到指定位置。例如，在如下程序中，执行时输入了 a bc↙，此时缓冲区中有 a、空格、b、c、回车，共 5 个字符，程序执行 scanf 只会取走其中的前三个字符，紧接着再执行 getchar 就会使 end1 得到第四个字符 c，再执行 getchar 就会使 end2 得到第五个字符“↙”，因此程序执行最终的输出结果如图 3-3 所示。

```
#include<stdio.h>
#include<string.h>
int main(){
    char end1,end2, a,b,c;
    printf("input:");
    scanf("%c%c%c",&a,&b,&c);
    end1=getchar();
    end2=getchar();
    printf("output:a=%c b=%c c=%c  end1=%c end2=%c\n",a,b,c,end1,end2);
    return 0;
}
```

执行结果如图 3-3 所示。

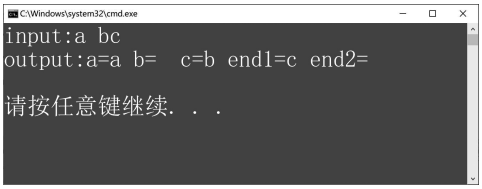


图 3-3 行缓冲示例的执行结果

3.2.2 顺序结构程序设计方法与示例

程序是由数据和运算组成的,而数据又是运算的基础,数据的表示形式也决定着可以对其进行哪些运算以及运算的先后次序。

比如:C 语言要求变量必须“先定义、再使用”,而且在利用变量的值参与运算之前必须保证该变量已经有了合理的初值,否则就需要先给它赋初值(赋值语句或者输入语句),然后才能用它参与各种运算。之所以这样规定,主要是因为当执行变量定义语句时,系统会在内存为该变量分配相应大小的空间(若干字节),并在变量名与内存空间之间建立关联关系(将该空间的首字节地址记为该变量的地址),但不会改变该空间内的数据值。因此,当直接使用该变量名读取数据时,只能得到一个随机数。

基于上述原因并结合结构化程序设计思想,我们认为 C 程序在宏观上都可以看作是顺序结构的,且由定义、输入、计算、输出 4 部分组成,其中前 3 部分可根据问题的实际需要选择省略或添加。

定义部分需要对问题进行分析,提取出解决问题过程中需要用到的各个量,包括变量和常量,确定每个量的名称、类型、初值、含义等。

输入部分主要实现变量赋初值,可以用赋值语句或者输入语句来完成。如果希望每次程序执行可以处理不同的数据,就必须使用输入语句。输入部分只需对完成运算必需的量赋初值,而对用于存放运算结果的量则不必赋值。

计算部分主要由各种表达式语句构成,确定好各个表达式语句的先后次序即可。

输出部分的功能就是用输出语句以要求的格式输出结果。有时,计算部分和输出部分是混杂在一起的,即边计算边输出,如例 3.1 就属于这种情况。

例 3.6 计算一元二次方程 $ax^2+bx+c=0$ 的实根,a、b、c 由键盘输入,设 $b^2-4ac>0$。

问题分析:由求根公式可知,若令:

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

则有:

$$x_1=p+q,\quad x_2=p-q$$

程序如下:

```
#include<stdio.h>
#include "math.h"     /*由于要调用数学函数库中的函数,所以必须用#include将数学函
                        数库的头文件 math.h 包含进程序*/
```

```
int main()
{
    float a,b,c disc,x1,x2,p,q;                    /*定义部分*/
    scanf("a=%f,b=%f,c=%f",&a,&b,&c);              /*输入部分*/
    disc=b*b-4*a*c;                                /*计算δ*/
    p=-b/(2*a);
    q=sqrt(disc/(2*a));                            /*sqrt()求平方根的函数*/
    x1=p+q;
    x2=p-q;                                        /*计算2个方程的根*/
    printf("\nx1=%5.2f\nx2=%5.2f\n",x1,x2);        /*输出部分*/
    return 0;
}
```

运行结果是：

```
输入：
   a=2.3,b=6.7,c=3.1↙
输出：
   x1=-0.58
   x2=-2.34
```

例 3.7　已知三条边 a、b、c，求三角形面积。

问题分析：由计算三角形面积的海伦公式可得：

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

其中：$s=(a+b+c)/2$。

设定义：

(1) 实型变量 a、b、c，表示三角形的三边长，需要输入。

(2) 实型变量 s、area，存放结果，包括中间结果和最终结果。

程序如下：

```
#include<stdio.h>
#include "math.h"
int main(){
    float a,b,c;                          /*定义部分*/
    float s,area;
    scanf("%f, %f, %f",&a,&b,&c);         /*输入三条边 a、b、c*/
    s=1.0/2*(a+b+c);                      /*计算部分*/
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("area=%8.3f \n",area);         /*输出部分*/
    return 0;
}
```

程序运行情况如下：

```
3,4,5↙
area=     6.000
```

请多试几组数据进行程序正确性的测试。想一想,这个程序还有什么地方需要改进?

温馨提示:注意此例中第 7 行 s =1.0/2 * (a+b+c)如写作 s =1/2 * (a+b+c),则无论输入的 a、b、c 为何值,其计算结果始终为 0。因为 1/2 在 C 表达式中的计算结果要保持为整型,即取为 0 而不是 0.5。这一点请读者在将数学表达式转换为 C 语言表达式时一定要特别注意。

3.3 选择结构

在程序设计中,经常要求计算机根据不同的条件或情况选择不同的处理过程。C 语言中的 if 语句和 switch 语句支持选择结构。

选择结构可分为单分支、双分支和多分支三种情况。一般的,采用 if 语句实现单选择结构程序;采用 if-else 语句实现双选择结构程序;采用 if-else-if 语句实现多选择结构。当 if 语句中的执行语句又是 if 语句时,则构成了 if 语句的嵌套。虽然 if-else-if 语句、嵌套 if 语句都能实现多选择结构程序,但用 switch 语句实现多选择结构的程序更加简洁明了。用 switch(k)语句实现的难点在于 k 表达式的设计和构造。


3.3.1 if 语句

3.3.1 if 语句

用 if 语句可以构成选择结构。它根据给定的条件进行判断,以决定是否执行某个分支程序段。C 语言的 if 语句经常使用 3 种基本形式:if 形式、if-else 形式和 if-else-if 形式。

1. 双分支 if-else 形式

if-else 是 if 语句的标准形式,它的语句形式为

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

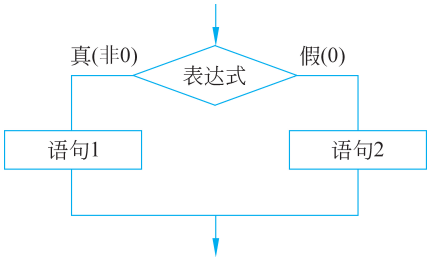


图 3-4 if-else 形式执行过程

其语义是:如果表达式的值为真(非 0),则执行语句 1,否则执行语句 2。为加以区别,称此处的表达式为条件判断表达式,称此处的语句 1 和语句 2 为内嵌语句。其执行过程可表示为图 3-4,即先计算条件判断表达式的值,根据条件判断表达式的值选择执行语句 1 或者语句 2。语句 1 和语句 2 只有一个将被执行。

注意：

(1) else 分句不允许单独使用。

(2) 关键字 if 应与关键字 else 对齐，内嵌语句缩进若干格。C 语言对缩进的要求没有 Python 那么严格，“分层缩进、对齐书写”是为了使程序具有更好的可读性。

使用 if 语句时还应注意以下问题(适用于 if 的 3 种基本形式)：

(1) if 语句中的条件判断表达式必须用括号括起来，且通常是逻辑表达式或关系表达式，但也可以是其他表达式(如赋值表达式等)，甚至也可以是一个变量。例如：

```
if(a=5) 语句;
if(b) 语句;
```

都是允许的。只要表达式的值为非 0，就表示逻辑“真”。又如：

```
if(a=5)…;
```

其中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，但在语法上是合法的。再又如，有程序段：

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

本语句的语义是，把 b 值赋予 a，如为非 0 则输出该值，否则输出“a＝0”字符串。这种用法在程序中是经常出现的。

(2) if 语句中的内嵌语句应为单个语句且语句之后必须加分号。如果要想在满足条件时执行一组(多个)语句，则必须把这一组语句用{}括起来组成一个复合语句。但要注意的是在“}”之后不能再加分号。例如：

```
if(a>b) {
    a++;
    b++;
}
else{
    a=0;
    b=10;
}
```

2. 单分支 if 形式

if 形式是 if-else 形式缺省 else 分句的特殊情况，它的语句形式为

```
if(表达式) 语句
```

其语义是：如果表达式的值为真(非 0)，则执行其后的语句，否则不执行该语句。其过程可表示为图 3-5。

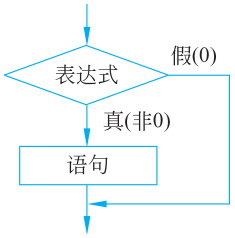


图 3-5　if 语句基本形式执行过程

比如：例 3.1 中子问题 3-1-1，判断 i%j 是否为 0，若为 0 则 flag 置 0。采用 if 形式可写为

```
f(i%j==0)   flag=0;
```

子问题 3-2，判断 flag，是 1 则输出 i。采用 if 形式可写为

```
f(flag)   printf("%d  ",i);
```

注意：该用单分支结构时千万不要画蛇添足而写成双分支结构，若将例 3.1 中子问题 3-1-1 写成如下双分支结构，则会造成灾难性的后果：输出 start～end 内的所有数！请大家自行分析原因。

```
if(i%j==0)   flag=0;   else   flag=1;
```

温馨提示：if-else 语句可以书写在一行上，只需将各语法单位用空格间隔开即可。虽然语法上允许，但考虑程序可读性，不提倡这样书写。

3. 条件运算符和条件表达式

如果在条件语句中只执行单个的赋值语句时，常可使用条件表达式来实现。不但使程序简洁，也提高了运行效率。

条件运算符由“?”和“:”组成，它是一个三目运算符，即有 3 个参与运算的量。

由条件运算符组成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：如果表达式 1 的值为真，则以表达式 2 的值作为条件表达式的值，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之中。

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;”执行该语句的语义是：如 a>b 为真，则把 a 赋予 max，否则把 b 赋予 max。

使用条件表达式时，还应注意以下几点：

(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“?”和“:”是一对运算符，不能分开单独使用。

(3) 条件运算符的结合方向是自右至左。例如：a>b?a:c>d?c:d 应理解为 a>b?a:(c>d?c:d)，这也就是条件表达式嵌套的情形，即其中的表达式 3 又是一个条件表达式。

例 3.8　输入两个整数，输出其中的大数。

方法一：单分支 if 语句

```
#include<stdio.h>
int main()
{
   int a,b,max;
   printf("input: ");
   scanf("%d%d",&a,&b);
   max=a;
   if(max<b)
     max=b;
   printf("max=%d",max);
   return 0;
}
```

方法二：双分支 if-else 语句

```
#include<stdio.h>
int main()
{
   int a, b;
   printf("input:");
   scanf("%d%d",&a,&b);
   if(a>b)
     printf("max=%d\n",a);
   else
     printf("max=%d\n",b);
   return 0;
}
```

方法三：条件表达式

```
#include<stdio.h>
int main()
{
   int a,b,max;
   printf("input:");
   scanf("%d%d",&a,&b);
   printf("max=%d",a>b?
a:b);
   return 0;
}
```

温馨提示：同一个问题可以运用不同的语法结构编写代码，但通常情况下，应选择其中逻辑表达最简单也最清晰的结构，这样的程序可读性较好。比如，该例中方法二就是首选结构。

4. 多分支 if-else-if 形式

if-else-if 是 if 语句用于多分支处理的形式，是程序编写中进行多路选择的常用方法，其语句形式为

```
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  if(表达式 m)
```

```
    语句 m;
else
    语句 n;
```

其语义是：依次判断表达式的值，当出现某个值为真时，则执行其对应的语句。然后跳转到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 n。然后继续执行后续程序。if-else-if 语句的执行过程如图 3-6 所示。

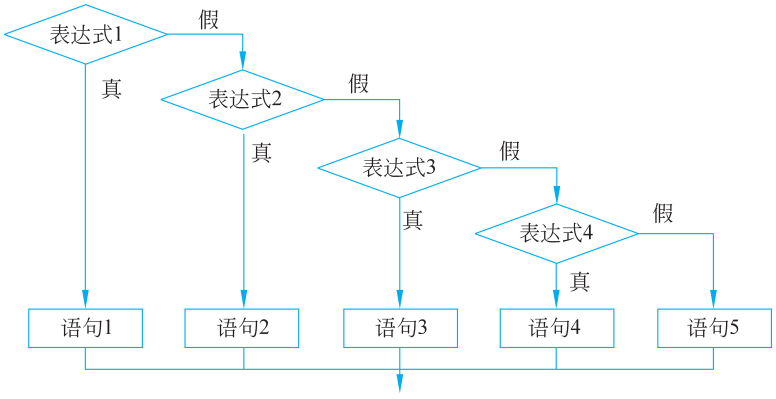


图 3-6　if-else-if 形式执行过程

5. if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句，则构成 if 语句的嵌套形式。其一般形式可表示如下：

```
if(表达式)
    if 语句;
```

或者为

```
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 if 和 else 的配对问题。

例如：

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

其中的 else 究竟是与哪一个 if 配对呢?

一种理解方式是:

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

另一种理解方式是:

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

为了避免这种二义性,C 语言规定,else 总是与它前面复合语句内最近的尚未配对的 if 配对,因此对上述例子应按前一种情况理解。

例 3.9　比较两个数的大小关系。

方法一:嵌套 if 形式

```
#include<stdio.h>
int main()
{
    int a,b;
    printf("please input a,b:    ");
    scanf("%d%d",&a,&b);
    if(a!=b)
        if(a>b)
          printf("a>b\n");    /*a>b*/
        else
          printf("a<b\n");    /*a<b*/
    else
      printf("a=b\n");        /*a==b*/
    return 0;
}
```

方法二:if-else-if 形式

```
#include<stdio.h>
int main()
{
    int a,b;
    printf("please input a,b:    ");
    scanf("%d%d",&a,&b);
    if(a==b)
      printf("a=b\n");      /*a==b*/
    else if(a>b)
      printf("a>b\n");      /*a>b*/
    else
      printf("a<b\n");      /*a<b*/
    return 0;
}
```

温馨提示:本例用 if-else-if 语句实现,程序逻辑更加清晰。因此,在一般情况下,应尽量避免使用 if 语句的嵌套结构,如果非用不可,建议用复合语句的形式表明其逻辑关系而不是仅仅通过缩进对齐的形式说明其配对关系。

学习完 if 语句后,请改进例 3.7,实现在计算三角形面积之前,先判断输入的三条边是否合法,而后再进行计算。并且当用户输入的三条边不合法时(比如:3、4、8),能给出“输入数据不合法”的提示。


3.3.2 switch 语句

3.3.2 switch 语句

在用 if-else-if 或者 if 嵌套实现的多分支结构中,随着分支数的增加,其结构的复杂性也会显著增加,不仅会减弱程序的可读性还会降低执行效率。C 语言还提供了另一种用于多分支选择的 switch 语句,其一般形式为:

```
switch(表达式){
    case 常量表达式 1:  语句序列 1;
    case 常量表达式 2:  语句序列 2;
    ⋮
    case 常量表达式 n:  语句序列 n;
    default:           语句序列 n+1;
}
```

其语义是:先计算 switch 后表达式的值,将该表达式的值依次与 case 后常量表达式的值比较,若相等,则从该常量表达式后的语句序列开始执行,直到遇到 break 语句或遇到 switch 语句的结尾花括号为止;若都不相等,则执行 default 后的语句序列。

使用 switch 语句还应注意:

(1) case 和 default 不能单独使用。

(2) 各 case 和 default 子句的先后顺序可以变动,而不会影响程序执行结果。

(3) default 子句可以省略,也可以出现在任意位置。

(4) case 和 default 后的语句序列可包含多条语句,且不必使用复合语句的形式。

(5) 在 case 后的各常量表达式的值不能相同,否则会出现错误。

(6) 多条 case 分句可以共用同一个语句序列。

(7) 一般地,case 后的语句序列应以 break 语句结束。如果没有 break 语句,则会顺序依次执行之后 case 的语句序列,直至遇到 break 或者 switch 语句的结尾花括号。

例 3.10 输入一个 1～7 的整数,转换成星期输出。

程序如下:

```
#include<stdio.h>
int main()
{
    int a;
    printf("input an integer number:    ");
    scanf("%d",&a);
    switch (a){
        case 1:  printf("Monday\n");
        case 2:  printf("Tuesday\n");
        case 3:  printf("Wednesday\n");
        case 4:  printf("Thursday\n");
```