《单片机 C 语言入门》（修订版）

作者：华清远见

第 3 章 C51 语句

本章目标

第 2 章中介绍了 C51 的数据类型、运算符与表达式等内容，这些是构成 C51 程序的基本的元素。但是，要写出可以实际运行的 C51 程序，还需要把这些元素组合起来以使之有实际的意义。本章将介绍 C51 语句，C51 语句将这些孤立的元素组成了有机的整体。本章的主要内容如下：

- C51 的控制结构
- C51 的语句
- 如何使用 C51 语句实现控制结构

3.1 C51 控制结构概述

在介绍 C51 语句之前有必要首先介绍程序的 3 种基本的控制结构：顺序结构、选择结构和循环结构。所有实用的单片机程序，不管简单还是复杂，都要用到这 3 种基本结构的一种或多种。

按照顺序从头到尾完成依次一系列指令时就在使用顺序结构。比如，工厂车间的流水线就是一个很好的顺序结构的例子，需要严格按照从第一道工序到最后一道工序的顺序来生产产品。同样，顺序结构指示单片机按照程序中的顺序逐条处理程序指令。一个典型的顺序结构如图 3.1 所示。

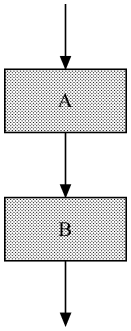


图 3.1　顺序结构

选择结构也称判断结构，选择结构首先做出判断，然后在此基础上采取适当的行动。驾车通过十字路口就是一个选择结构的例子，司机必须首先判断红绿灯的状态，然后按照判断的结果决定目前是否能够通过十字路口。同样，当单片机程序中使用选择结构时，提醒单片机需要做出选择，并按照判断结果决定程序的走向。选择结构还要提供基于选择结果所要采取的适当动作。一个典型的选择结构如图 3.2 所示。

当条件 c 为真时执行 A，当条件 c 为假时，执行 B，A 和 B 可以是简单语句或复合语句。

3 种控制结构中最后一种是循环结构。比如，洗衣服的过程就包含循环结构。洗衣服的过程通常包括“放入洗衣粉”、“搓洗”、“冲洗”这些步骤，直到把衣服洗干净。程序中使用的循环结构也就是重复，指示单片机重复一条或多条指令，直到满足某种条件，这时单片机才停止重复这些指令。循环结构分为事先测试循环结构和事后测试循环结构，事先测试循环结构和事后测试循环结构的流程图分别如图 3.3（a）和 3.3（b）所示：

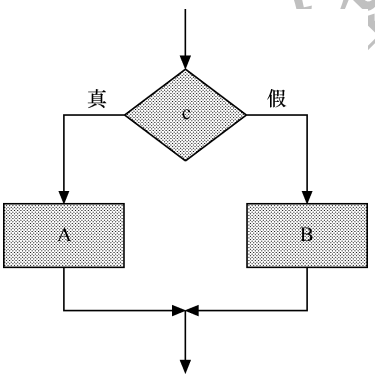
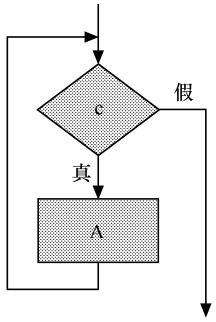
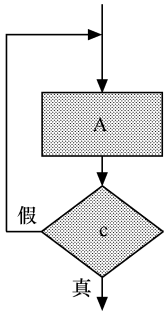


图 3.2　选择结构



（a）事先测试循环结构



（b）事后测试循环结构

图 3.3

3.2 C51 语句概述

C51 的语句用来向单片机发出操作指令。一个完整的 C51 程序包括数据描述和数据操作。数据描述定义数据结构和数据初值，由数据定义部分来实现；数据操作是对已提供的数据进行加工，这部分的功能就是由语句来实现的。这里的“数据”既包括与底层硬件无关的运算数据，也包括如特殊功能寄存器（SFR）等与底层硬件状态直接相关的数据。

在前面两章中已经接触到一些语句。例如，一个语句可以由一个表达式和一个分号构成。例如：

```
a = 1;
```

就是一个赋值语句，而

```
a = 1
```

是一个赋值表达式而非赋值语句。

可以看到，一个表达式的最后加上一个分号（;）就成了一个语句。分号是语句的终结符，一个语句必须在最后出现分号，分号是语句中不可缺少的一部分，在后面介绍空语句时会看到，甚至一个单独的分号也可以构成一个语句。

C51 的语句按其复杂度可以分为简单语句和复杂语句，上面的例子就是一个简单语句。可以用花括号“{”和“}”把一些语句组合在一起，使其在语法上等价于一个简单语句，这样的语句就称之为复合语句。下面的代码演示了复合语句的用法。

```
    if (a >= 0)
    {
        printf ( "|a| = %d", a);
        return a;
    }
 else
    {
        printf ( "|a| = %d", -a);
        return -a;
    }
```

当 a >= 0 时，执行 if 后的复合语句，否则执行 else 后的复合语句。

> **注意** 复合语句中最后一个语句中最后的分号不能忽略不写，否则会导致编译错误；结束一个复合语句的右花括号之后不能带分号，否则有时会导致编译错误或逻辑错误，在介绍条件语句时将要介绍复合语句的右花括号之后带分号可能导致的问题及其原因。

C51 的语句按其功能又可以分为以下 4 类：

- 说明语句；
- 表达式语句；
- 空语句；
- 控制语句。

其中控制语句又可以分为条件分支语句、循环语句、转移语句，共 3 类，各类控制语句的详细介绍参见 3.6 节。

3.3 说明语句

C51 的说明语句用来说明数据量的类型，同时也可以在必要的时候给数据量赋初值。

C51 语言中，所有用到的变量都要先定义，然后才可以使用。每一个变量都被指定为一个确定的类型，这样在编译的时候就能为其分配相应的存储单元，并且可以据此检查该变量所进行的运算是否合法，若不合法则报错，提示程序员修改。下面的代码展示了说明语句的作用。

```
void main(void)                 //注意，这是一个编译无法通过的程序
{
    unsigned int a=8;         //  说明语句 1
    unsigned int b=0;         //  说明语句 2
    float c=3.2;                    //  说明语句 3
    b=a%c;                          //  语句 4，编译时将出错
}
```

说明语句 1 将变量 a 定义为无符号整型变量，并赋初值为 8。这样，编译时会给 a 留出两个字节的存储单元，并将这个存储单元初始化为 8。同理，说明语句 2 和说明语句 3 使得编译时分别给 b、c 留出两个字节，4 个字节的存储单元并分别初始化为 0 和 3.2。在编译语句 4 时会根据变量的类型检查所进行的运算是否合法。在本例中，由于变量 c 被定义为单浮点类型，而单浮点类型是不能参与求余（%）运算的，因此编译器会判定此运算不合法。

常见的 C51 说明语句还有：

```
unsigned char chr= 'a';
unsigned char str[4]="abc";
sfr P1=0x90;
sbit CLK=P1^1;
bit flag_1;
```

3.4 表达式语句

任何表达式加上一个分号(;)后都可以直接形成表达式语句，例如：

```
3+2;
```

就可以构成一个语句。单片机可以执行该语句，但并不把“3+2”的结果赋给某个变量，也不改变程序的运行逻辑，从而也就没有实际意义。

当一些表达式组合起来，完成某一相对完整的功能后，再加一个分号表示结束，这就组成一条语句，最典型的是由赋值表达式和分号构成一个赋值语句。例如：

```
a=3+2;
```

这就是一行赋值语句，该语句改变了 a 的值。

3.5 空语句

顾名思义，空语句就是什么都不做的语句。常用的空语句有两种形式，可以是只有一个分号的语句，也可以是只有一个花括号的语句：

```
;
{}
```

C51 还专门提供了一个空函数语句:

```
void _nop_(void);
```

在使用这个语句时要把头文件 intrins.h 包括进来。

在单片机程序中，空语句一般用作延时或查询等待，例如：

```
for (i=1;i<=100;i++)
{}
```

完成了一段延时，这段延时的时间是把从 1 开始递增到 100 所用的时间，随着单片机的工作频率不同、编译器的不同而不同。对 for 语句的具体解释如 3.6.2 节所示。

而下面的代码段完成了查询等待的功能：

```
while (P1^1==1)
{}
```

这段代码循环检测管脚 P1^1 的状态，如果是高电平（1）就循环执行空语句，直到变为低电平后才去执行后面的语句。

3.6 控制语句

在 3.1 节已经介绍了 C51 的顺序、选择和循环这 3 种控制结构，下面分别介绍可以实现这 3 种控制结构的 C51 语句。

3.6.1 条件分支语句

条件分支语句主要包括 if…else…语句、if…语句、多级 if…else…语句和 switch 语句，共 4 种形式。前两种语句用于两分支选择分支，后两种语句用于多分支选择分支。

1．if…else…语句

if…else…语句的一般形式为：

```
if (表达式)
{
   分支一
}
else
{
   分支二
}
```

其中，“表达式”一般为逻辑表达式或关系表达式，单片机对表达式的值进行判断，若为非 0（即条件成立或称条件为真），则按“分支一”处理，若为 0（即条件不成立或称条件为假），则按“分支二”处理。可见，图 3.2 即是这种条件语句的流程图。与图 3.2 相对应，“表达式”即图中的 c，“分支一”即图中的 A，“分支二”即图中的 B。

例 3-1 是一个使用 if…else…语句的例子。该程序判断输入的无符号整数 a、b 的大小，如果 a>b,则输出字符串“a is larger than b”，否则输出“a is not larger than b”。

【例 3-1】使用 if…else…语句判断两个正整数的大小。

```
#include <reg51.h>
#include <stdio.h>
void main()
{
   unsigned int a,b;
   printf ("Please input a:");
   scanf ("%d",&a);
   printf ("\nPlease input b:");
   scanf ("%d",&b);

   if (a>b)
   {
      printf("\n");
      printf("a is larger than b");
   }
   else
   {
      printf("\n");
```

```
        printf("a is not larger than b");
    }

    while(1)
    { }
}
```

2．if…语句

if…语句的一般形式为：

```
if (表达式)
{语句体}
```

其中的“语句体”既可以是简单语句，也可以是复合语句，其流程如图 3.4 所示：

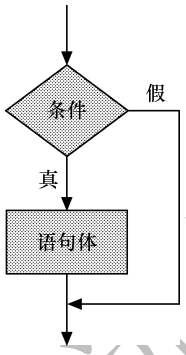


图 3.4　if…语句的流程图

对比 if…else…的流程图（图 3.2），在 if…语句中，当条件不成立时将“绕过”分支二的语句，直接执行后面的代码。

这里需要特别注意，if 和 else 一定要成对出现才能构成一个完整的语句。if…语句表面看上去没有 else，但其实是 if…else…语句的一种特殊形式，缺省了 if…else…语句中的 else…部分。也就是说：

```
if (表达式)
    {语句体}
```

等价于如下代码：

```
if （表达式）
    {语句体}
else
    { }     //这是一个空语句
```

可以看出，if…语句并不是没有执行分支二，而是因为分支二是空语句，在执行时没有任何动作。

至此，读者可以理解在 3.1 节中为什么说“结束一个复合语句的右花括号之后不能带分号，否则有时会导致编译错误或逻辑错误”。例如下面的例 3-2 就会在编译的时候出现错误：

【例 3-2】使用 if…else…语句判断两个正整数的大小但代码出现了语法错误。

```
#include <reg52.h>
#include <stdio.h>
void main()
{
        unsigned int a,b;
        printf ("Please input a:");
        scanf ("%d",&a);
        printf ("\nPlease input b:");
        scanf ("%d",&b);
```

```
        if (a>b)
        {
           printf("\n");
           printf("a is larger than b");
        };                          //注意，这一行的花括号右边加了一个分号
        else
        {
           printf("\n");
           printf("a is not larger than b");
        }

        while(1)
        { }
}
```

例 3.2 和例 3.1 基本相同，只是在添加注释的那一行多了一个分号，却会导致编译出错，这是因为编译器编译时，由于分号的存在，会认为以下代码段：

```
if (a>b)
{
   printf("\n");
   printf("a is larger than b");
};                      //注意，这一行的花括号右边加了一个分号
```

等价于如下代码段：

```
if (a>b)
{
   printf("\n");
   printf("a is larger than b");
}
else
{ }
```

从而导致例 3.2 程序中显式的 else 没有 if 与之对应，发生语法错误。

3．多级 if…else…语句

有时，需要创建从几个选项中进行选择的选择结构，例如可能需要编写一个程序，根据商品的不同数量来确定折扣水平。这时，就需要用到多级 if…else…语句或 switch 语句。

多级 if…else…语句的一般形式为：

```
if (表达式 1)
{
   分支一
}
else if (表达式 2)
{
   分支二
}
else if (表达式 3)
{
   分支三
}
```

```
…
else
{
    分支 n
}
```

多级 if…else…语句的流程图如图 3.5 所示。

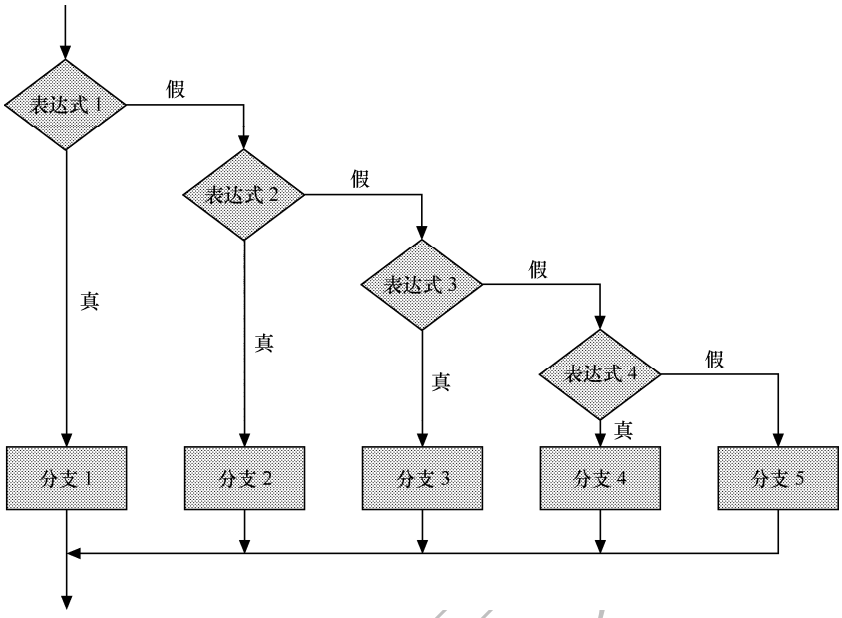


图 3.5　多级 if…else…语句的流程

这种结构从上到下逐个判断表达式的结果是否为 1，一旦发现表达式的结果为 1 就执行与之相关的语句，并跳过其他语句。例 3-3 就是一个用多级 if…else…语句根据商品数量来决定价格折扣的程序。

【例 3-3】使用多级 if…else…语句以一个根据商品数量来决定价格折扣。

```
#include <reg52.h>
#include <stdio.h>

void main()
{
    unsigned int num;
    float discount;
    num=30;

    if (num>=50)
    {
        discount=0.7;
    }
    else if (num>=40)
    {
        discount=0.8;
    }
    else if (num>=30)
    {
        discount=0.9;
    }
    else
    {
```

```
        discount=1;
    }

    printf("\n");
    printf("The discount is %f.\n",discount);
    while(1)
    { }
}
```

本例中，通过一个多级 if…else…语句对商品数量（num）作判断，若商品数量大于等于 50 个，折扣为 0.7；小于 50 个而大于等于 40 个，折扣为 0.8；小于 40 个而大于等于 30 个，折扣为 0.9；小于 30 个即折扣为 1（即不打折扣）。由于程序中已经给 num 赋值为 30，因此程序执行以下语句。

```
discount=0.9;
```

除了使用多级 if…else…语句来创建多分支选择结构外，还可以使用 switch 语句创建多分支选择结构。

4．switch 语句

当指令中的选择结构要从多个分支中进行选择时，使用 switch 语句往往要比使用多级 if…else…语句简洁明了。

switch 语句的一般形式为：

```
switch ( 整型或字符型变量 )
{
    case 变量可能值 1 :
        分支一;
        break;
    case 变量可能值 2 :
        分支二;
        break;
    …
    case 变量可能值 n:
        分支 n;
        break;
    default :
        分支 n+1 或空语句;
}
```

switch 语句的流程图如图 3.6 所示：

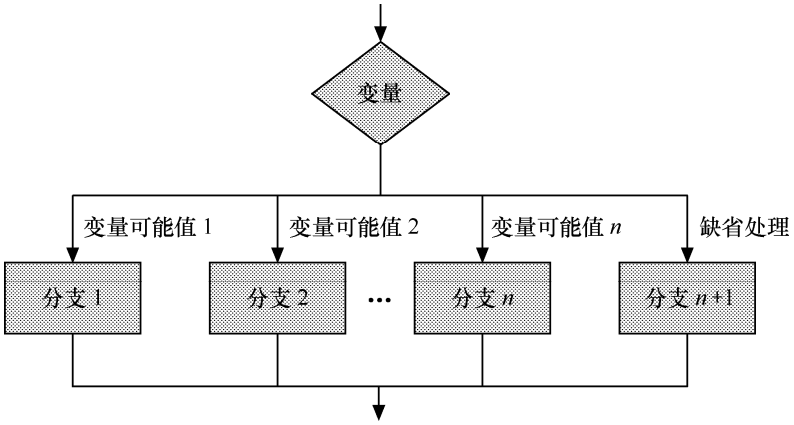


图 3.6　switch 语句流程图

switch 语句中，每一条单独的 case 子句都包含一个值，同时后面跟着一个冒号。该值的数据类型应该和选择表达式的数据类型一致，也就是说，若 switch 语句中的变量是一个整型变量，则在 case 子句中的值也应该是整型数字；同样，如果 switch 语句中的变量是一个字符型变量，则在 case 子句中的值也应该是字符。每个 case 语句的冒号后面是一条或多条语句。当 switch 语句中变量的值与 case 的值相匹配时，就执行这些语句。需要注意 case 子句中的语句不作为复合语句输入，即这些语句不括在花括号中。

在单片机执行了相应 case 子句中的指令以后，通常想让计算机退出 switch 语句，不再处理该语句中剩下的指令。可以将 break 语句作为最后一条语句包含在 case 子句中，就可以达到上述目的。break 语句告诉单片机在这一时刻跳出 switch 语句。如果没有 break 语句，单片机将继续处理该语句之后剩下的指令，这可能并不是程序的原意。

例 3-4 是一个 switch 语句中的变量为整型变量的例子。

【例 3-4】使用 switch 语句根据输入的数字来输出对应的一周中每天的名称。

```
#include <reg52.h>
#include <stdio.h>
void main()
{
    unsigned int day_num;
    printf("Please input a day number(1~7):\n");
    scanf("%d",&day_num);

    switch(day_num)        //switch 语句中的变量为整型变量
    {
        case 1:
            printf("\nMonday.\n");
            break;
        case 2:
            printf("\nTuesday.\n");
            break;
        case 3:
            printf("\nWednsday.\n");
            break;
        case 4:
            printf("\nThursday.\n");
            break;
        case 5:
            printf("\nFriday.\n");
            break;
        case 6:
            printf("\nSaturday.\n");
            break;
        case 7:
            printf("\nSunday.\n");
            break;
        default:
            printf("\nInput Error!\n");
    }
    while(1)
    { }
}
```

例 3-5 是一个 switch 语句中的变量是字符型变量的例子。

【例 3-5】使用 switch 语句根据输入的代表成绩等级的字符来输出对应的评价。

```
#include <reg52.h>
#include <stdio.h>
void main()
{
    unsigned char grade;
    printf("Please input the grade(A,B,C,D):\n");
    scanf("%c",&grade);

    switch(grade)          //switch 语句中的变量是字符型变量
    {
        case 'A':
            printf("\n Very good!\n");
            break;
        case 'B':
            printf("\n Good!\n");
            break;
        case 'C':
            printf("\n Pass!\n");
            break;
        case 'D':
            printf("\n Fail!\n");
            break;
        default:
            printf("\n Input Error!\n");
    }
    while(1)
    { }
}
```

3.6.2 循环语句

C51 中可以用 while 语句、for 语句和 do…while 语句来实现循环结构。其中，while 语句和 for 语句可以构成事先测试循环结构，即在循环体内部的指令执行之前，预先对条件求值；do…while 语句可以构成事后测试循环结构，即在循环体内部的指令执行之后再对条件求值。这两种循环中，使用较多的是事先测试循环。

1．while 语句

while 语句的一般形式为：

```
while(循环条件)
    {循环体}
```

使用 while 语句时必须提供圆括号中的循环条件。循环条件是其值要么为真要么为假的布尔表达式，包括变量、常量、函数、算术运算符、比较运算符和逻辑运算符。

除了提供循环条件，还必须提供当循环条件为真时需要处理的语句，即循环体。当循环条件的值为非 0 时就循环执行 while 语句中的循环语句。这是一种事先测试循环，在执行语句之前判断表达式，因此，循环可能没有执行循环体就退出。循环体既可以是简单语句，也可以是复合语句，还可以是一个空语句，例如：

```
while(1)
```

```
        { }
```

这个 while 语句仅仅提供了一个无限循环，以保证不会退出 main 函数。这是由于单片机程序和一般的运行在 PC 上的程序不同，单片机上一般没有操作系统，因此不能退出 main 函数。while 语句的流程如图 3.7 所示。

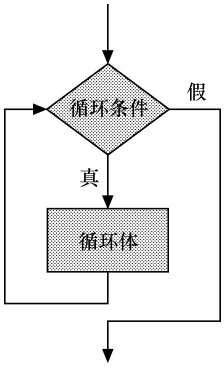


图 3.7　while 语句的流程

例 3-6 演示了 while 语句的使用方法。

【例 3-6】使用 while 语句实现从 1 到 100 的累加。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    unsigned int i=1;
    unsigned int sum=0;

    while (i<=100)
    {
        sum+=i;
        i++;
    }

    printf("Sum=%d",sum);
    while(1)
         { }
}
```

从这个例子中可以看到，在循环体中应该有使循环趋向于结束的语句。例如，在本例中循环结束的条件是“i>100”，因此在循环体中应该有使 i 增值以最终导致 i>100 的语句。本例中是通过“i++”来实现的。若无此语句，则 i 的值总是初始值 1，循环条件“i<=100”总是为真，循环也就永不结束。

2．for 语句

for 语句是在 C51 中用得最多，也最灵活的循环语句，可以在一条语句中包括循环控制变量初始化、循环条件、循环控制变量的增值等内容。for 语句的一般形式为：

```
for(表达式 1;表达式 2;表达式 3)
      {循环体}
```

其中，表达式 1 为循环控制变量初始化表达式，表达式 2 为循环条件表达式，表达式 3 为循环控制变量增值表达式。

需要说明的是，这里所谓的“增值”，既可能是循环控制变量加 1，也可能是循环控制变量加 2，还可能是循环控制变量减 1 或减 2。也就是说，“增值”仅仅是指循环控制变量发生了变化，不要按其字面意思去理解。

与 while 语句相同，for 语句也是事先测试循环结构，因此，如果循环条件在第一次测试的时候就不为真，那么循环体就一次都不会被执行。

for 语句的执行流程如图 3.8 所示：

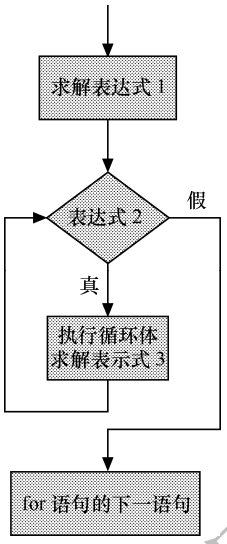


图 3.8　for 语句的执行流程

用语言描述 for 语句的执行过程如下。

（1）首先求解表达式 1，也就是给循环控制变量赋一个初值。

（2）测试循环条件是否为真，若为真，则执行循环体并求解表达式 3，即使循环控制变量增值；若为假，转到第（4）步。

（3）转回第（2）步继续执行。

（4）执行 for 语句的下一语句。

例 3-7 用 for 语句实现了例 3.6 所完成的功能。

【例 3-7】使用 for 语句实现 1～100 的累加。

```
#include <reg52.h>
#include <stdio.h>
void main (void)
{
    unsigned int i;
    unsigned int sum=0;

    for(i=1;i<=100;i++)
    {
        sum+=i;
    }

    printf("Sum=%d",sum);
    while(1)
    { }
}
```

对于 for 语句有如下几点需要说明。

- for 语句中的控制变量初始化表达式可以省略，但分号（;）不可省略。此时一般会在 for 语句之前对其初始化。例如上面的程序可以写成如下等价形式：

```
...
i=1;
for(;i<=100;i++)
{
    sum+=i;
}
...
```

- for 语句中的循环条件表达式可以省略。此时一般要在循环体中对循环条件进行判断并提供退出循环的措施，否则会导致“死循环”。例如上面的程序可以写成如下等价形式：

```
...
for(i=1;;i++)
{
    if(i>100)
        break;
    else
        sum+=i;
}
...
```

其中，用 if…else…语句测试循环控制变量，若循环条件不再满足，则用 break 语句退出循环。

- for 语句中的循环控制变量增值表达式也可以省略。除非有意为之，一般要在循环体中对循环条件进行判断并提供退出循环的措施，否则会导致“死循环”。例如上面的程序可以写成如下等价形式：

```
...
for(i=1;i<=100;)
{
    sum+=i;
    i++;
}
...
```

在循环体的最后提供了循环控制变量的增值方法。要说明的是，如果循环体中使用了循环控制变量，则要注意循环控制变量增值语句的位置。例如，上面的循环体如果写成如下形式就会得出错误的结果：

```
...
{
    i++;
    sum+=i;
}
...
```

- 可以同时省略 for 语句中的循环控制变量初始化表达式和增量表达式，这样的 for 语句完全等价于 while 语句。同样地，此时需要在 for 语句之前提供初始化语句，在循环体中提供增量语句。例如上面的程序可以写成如下等价形式：

```
...
i=1;
for(;i<=100;)
{
    sum+=i;
    i++;
}
```

```
...
```

3．do…while 语句

前面的 while 语句和 for 语句都是事先测试循环结构，其特点是先判断循环条件表达式，后执行循环体。而 do…while 语句是事后测试循环结构，其特点是先执行语句，后判断表达式。do…while 语句的一般形式为：

```
do {循环体}
while （循环条件）;
```

其中循环条件是一个结果为布尔型的表达式。需要注意，在 while（循环条件）后面要加分号。do…while 语句的流程图如图 3.9 所示。

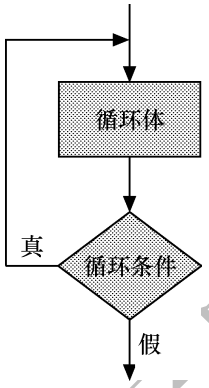


图 3.9　do…while 语句的流程图

可见，do…while 语句的特殊之处在于，第一次执行循环体之前并不测试循环条件，所以，即使循环条件根本就不成立，循环也将执行一次。

例 3-8 用 do…while 语句实现了例 3.6 所完成的功能。

【例 3-8】使用 do…while 语句实现 1～100 的累加。

```
#include <reg52.h>
#include <stdio.h>
void main (void)
{
   unsigned int i=1;
   unsigned int sum=0;

   do {
      sum+=i;
      i++;
      }
   while (i<=100);
   printf("Sum=%d",sum);
   while(1)
   { }
}
```

4．嵌套的循环结构

实际应用中，经常要用到嵌套的循环结构。在嵌套循环结构中，内层循环需要置于称之为外层循环的另一个循环中。

例如，日常生活中所用的时钟就使用了嵌套循环。最外层的循环是时针的循环，每 12 小时循环一次；次外层的循环是分针的循环，每 1 小时循环一次；最内层的循环是秒针的循环，每 1 分钟循环一次。

例 3-9 是一个用 for 语句构成的双层循环。

【例 3-9】使用嵌套循环结构打印一个由“*”组成的 4*4 的方阵（for 语句中嵌入 for 语句）。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
   unsigned int row,column;
   printf("\n");

   for (row=1;row<=4;row++)
   {
      for (column=1;column<=4;column++)
      {
         printf("*");
      }
      printf("\n");
   }

   while(1)
   { }
}
```

本例中，以下代码是外层循环，用来控制打印的行。

```
for (row=1;row<=4;row++)
{
…
}
```

以下代码是内层循环，用来控制打印的列。

```
for (column=1;column<=4;column++)
{
…
}
```

3 种循环语句还可以互相嵌套。上面的双层循环结构可以改写为以 for 语句作为外层循环而以 while 语句作为内层循环，如例 3-10 所示。

【例 3-10】使用嵌套循环结构打印一个由“*”组成的 4×4 的方阵（for 语句中嵌入 while 语句）。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
   unsigned int row,column;
   printf("\n");

   for (row=1;row<=4;row++)
   {
      column=1;
      while(column<=4)
```

```
        {
            printf("*");
            column++;
        }
        printf("\n");
    }

    while(1)
        { }
}
```

可以看到，在 for 语句的循环体中有对 column 的初始化语句，如下所示：

```
    column=1;
```

在 while 语句的循环体中有对 column 的增值语句，如下所示：

```
    column++;
```

在例 3-9 中，初始化语句和增值语句都是包含在 for 语句的表达式中的。

3.6.3 转移语句

前面所讨论的分支选择语句和循环语句都有着自己完整的流程结构，另外还有一些流程控制语句可以改变流程，但自身并不构成完整的流程结构，这样的语句包括 break 语句和 continue 语句。

1．break 语句

break 语句的一般形式为：

```
    break;
```

在介绍 switch 语句和 for 语句的时候已经接触到了 break 语句。在 switch 语句中，break 语句用来使流程跳出 switch 结构，继续执行 switch 之后的语句；在 for 语句中，break 语句用来使流程跳出循环体，接着执行循环后面的语句。

break 语句不仅可以用于 switch 语句和 for 语句，而且还可以用于其他的循环语句。例 3-11 是一个 break 语句用于 while 循环的例子。

【例 3-11】使用 break 语句退出 while 语句循环。该程序搜索 1～80 之间能被 4 整除的数并输出，但找到 10 个满足条件的数字之后就执行 break 语句停止搜索。

```
    #include <reg52.h>
    #include <stdio.h>

    void main (void)
    {
        unsigned int number,counter;
        number=0;
        counter=0;
        printf("\n");

        while(number<=80)
        {
            number++;
```

```
        if (number%4==0)
        {
            printf("%d ",number);
            counter++;
        }
        if (counter >= 10)
            break;
    }

    while(1)
        { }
}
```

2．continue 语句

continue 语句的一般形式为：

```
continue;
```

continue 语句的作用是跳过本次循环中剩余的循环体语句，立即进行下一次循环。continue 语句只能用在循环语句中，与 break 语句的区别在于执行 break 语句时，不仅跳过了本次循环中剩下的语句，而且还跳过了剩下的所有循环；而执行 continue 语句时，只是跳过了本次循环中剩下的语句，剩下的循环还要执行。例 3-12 是一个综合使用了 continue 语句和 break 语句的例子。

【例 3-12】使用 continue 语句强制进行下一次 while 语句循环，使用 break 语句退出 while 循环。该程序搜索 1～80 之间能被 4 整除但不能被 3 整除的数并输出。如果当前检测的数既能被 4 整除又能被 3 整除则执行 continue 语句，强制检测下一个数；如果已经找到 10 个满足条件的数字，则执行 break 语句就停止搜索。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    unsigned int number,counter;
    number=0;
    counter=0;
    printf("\n");

    while(number<=80)
    {
        number++;
        if (number%4==0)
        {
            if (number%3==0)
                continue;
            printf("%d ",number);
            counter++;
        }
        if (counter >= 10)
            break;
    }
```

```
    while(1)
        { }
}
```

在介绍完 C51 中主要的语句后，就可以用这些语句编写复杂的 C51 程序，例 3-13 就是一个综合运用说明语句、空语句、while 语句、do…while 语句、if 语句、switch 语句、break 语句、continue 语句等多种语句编写的一个 C51 程序。

【例 3-13】综合运用说明语句、空语句、while 语句、do…while 语句、if 语句、switch 语句、break 语句、continue 语句等多种语句。在购买商品时，经常会看到在收银台使用机器读取商品的条形码并自动计算总价格。下面的例子就是模拟计算各种商品的总价格的过程。为集中精力于 C51 语言的介绍对实际情况进行了简化，假设商品的种类只有 3 种（种类编号分别为 1、2、3），商品的种类和数量都通过手工输入，当输入商品的种类编号为 0 时表示已经输入完毕，可以输出总的价格。

```
#include <reg52.h>
#include <stdio.h>

#define PRICE_1 10;              //商品 1 的单价
#define PRICE_2 20;              //商品 2 的单价
#define PRICE_3 30;              //商品 3 的单价

void main(void)
{
    unsigned int  kind;       //商品的种类
    unsigned int  quanlity;   //商品的数量
    unsigned int sum;         //总的价格

    while(1)
    {
        sum=0;
        do
        {
            printf("Please input the food kind number(1,2 or 3), input 0 to end:\n");
            scanf("%d",&kind);

            if (kind!=1 && kind!=2 && kind !=3)
            {
                printf("Input Error!Input the kind again. \n");
                continue;           //若输入的商品分类不是 1、2 或 3，则重新输入
            }

            switch(kind)
            {
            case 1:
                printf("Please input the quanlity of food 1:\n");
                scanf("%d",&quanlity);
                sum+=quanlity*PRICE_1;
                break;
            case 2:
                printf("Please input the quanlity of food 2:\n");
                scanf("%d",&quanlity);
                sum+=quanlity*PRICE_2;
                break;
```

```
            case 3:
                printf("Please input the quanlity of food 3:\n");
                scanf("%d",&quanlity);
                sum+=quanlity*PRICE_3;
                break;
            default:
                {}
            }
        }
        while (kind!=0);        //end do…while
        printf("The total cost is %d.\n",sum);
    }       //end while()
}
```

3.7 小结

本章主要讲述了 C51 语言中的顺序、选择和循环 3 种控制结构。顺序结构比较简单，不需要专门的控制语句；选择结构可以分为两分支选择结构和多分支选择结构，前者可以由 if…else 语句或者 if 语句实现，后者可以由多级 if…else 语句或者 switch 语句实现；循环结构可以分为事先测试循环结构和事后测试循环结构，前者可以由 while 语句或者 for 语句实现，后者可以由 do…while 语句实现。此外，还有 break 语句、continue 语句等控制程序流程语句。

联系方式