5.3.3 程序实例

【例 5-2】输入一行字符，分别统计出字母、数字和其他字符的个数。

【解题思路】要统计字符出现的个数，可以设定一个计数器，通过累加的方式来完成计数工作。根据题目要求，应该设置 3 个计数器，通过扫描整个输入串来使计数器累加，其中的判断由条件语句来完成。

在判断字符 c 属于哪一类的问题上，需要一定的技巧。假设字符 c 是字母，那么它一定是'A'-'Z'或'a'-'z'其中之一。这可以通过一条 switch 语句来实现，但其中的 case 分支会非常得多。仔细分析可以发现，字符类型是一种有序类型，即字符是按各自的 ASCII 码进行升序排序的，这样就有'A'<'Z'和'a'<'z'成立。因此，可以用 c>='A'&&c<='Z'的逻辑运算结果来判断 c 是否是大写字母，用 c>='a'&&c<='z'的逻辑运算结果来判断 c 是否是小写字母，而这两个逻辑表达式只要满足一个即可认为 c 是字母，因此二者是"或者"的关系。如果用语句来实现，就是

```
if (c>='A' && c<='Z' || c>='a' && c<='z') …
```

对于数字字符和其他字符，判断过程相似，这里就不再赘述。

剩下的问题就是如何判断输入结束。一般地，可以使用 getchar()来完成字符输入。虽然 getchar()函数只能一次输入一个字符，但由于它是一种带缓冲的输入（意思是所有的输入都会暂存到内存中，即使只读入一个字符，其他的字符仍然会保存在缓冲区中），每一次的 getchar()调用都会使当前的读入位置后移，因此可以用一个循环来读入后续的字符。又因为 getchar()函数用换行符作为输入结束的标志，而这个标志也会被存放到输入缓冲中，所以当读到一个换行符时就可以认为输入结束了。

具体程序如下。

```
//5-2.c
#include <stdio.h>

int main()
{
    char c;                                    //输入的字符
    int alpha = 0, digit = 0, others = 0;  //这 3 个变量分别作为字母、数字和其他字符的计数器

    printf("请输入一行字符:");
    c = getchar();
    while (c != '\n')
    {
        if (c >= 'A' && c < 'Z' || c >= 'a' && c <= 'z')
            ++alpha;
        else if (c >= '0' && c <= '9')
            ++digit;
        else
            ++others;
        c = getchar();
    }
    printf("字母共%d 个\n 数字共%d 个\n 其他字符共%d 个", alpha, digit, others);

    return 0;
}
```

程序的运行结果如下。

```
请输入一行字符:a9b7$ndi#3ze*(wz↙
字母共 9 个
数字共 3 个
其他字符共 4 个
```

请读者注意循环前 getchar()和循环体内 getchar()的调用。如果在循环前没有调用，那么循环将没有初始条件而使结果未知；如果没有循环体内的调用，循环控制变量 c 将继续保持第一次调用的结果，因而导致最终的结果不正确。

5.4 do-while 语句

当遇到循环结束条件明确、循环次数要求至少 1 次的情况下可以使用 do-while 循环。即如果题目明确说明无论在什么条件下至少要执行一次循环体，那么选择 do-while 比 while 更适合。

5.4.1 do-while 语句的语法

do-while 语句的语法如下：

```
do
{
    循环体
} while(表达式);
```

其功能是：先执行循环体语句，再测试表达式的值，如果其值为真（非 0 值），则重复执行循环体，以此类推；如果表达式的值为假（0 值），则终止循环。其流程图如图 5-4 所示。可以看到，do-while 也是一种当型循环。

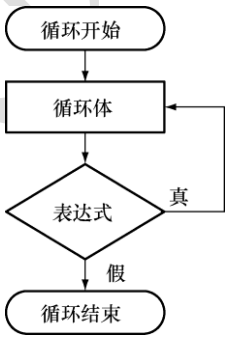

图 5-4 do-while 语句的流程图

do-while 的循环体至少要执行一次，这与 while 语句不同。不过，只要循环次数多于 1 次，二者完全可以互换。

5.4.2 迭代法

迭代法（*iterative method*）是一种循环算法。这里用几个实例来说明迭代的方法。

【例 5-3】已知 $e^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}$，请用级数前 21 项的和来求 e 的近似值。其中 e 是自然底数，0! =1，0^0=1。

【解题思路】取 x=1，然后将和式展开，即得到如下表达式。

$e=1^0/0!+1^1/1!+1^2/2!+\ldots+1^n/n!=1+1+1/2!+\ldots+1/n!$

这是一个典型的累加和形式，不过其中的每一项都不是一个简单的数，而是在按一定的规律变化。设 i 是从 0 到 n 的整数，a_i 是级数中的第 i 项（i=0,1,2…n），那么 a_i 是 i 的函数，其函数关系为：

$a_i = f(i) = 1/i!$

那么，根据前面的经验，求累加和的算法框架可以写成：

```
e = 1.0;
i = 1;
do
{
    e += f(i);
    ++i;
} while (i <= n);
```

算法中一个比较麻烦的事情是，必须要使用函数 *f(i)*来计算每一次的 a_i。目前，C 语言没有求阶乘的库函数，读者也还没有学到关于自定义函数的内容，因此须用其他的方法来完成这项计算工作。

实际上，上述级数中的每一项都不是杂乱无章排列的，而是相邻前后两项具有某种规律性变

化的关系，这种关系称为“**递推关系**（*recursive relation*）”。找出递推关系，就可以通过当前项 a_i 推导出后一项 a_{i+1}，而不必使用函数来计算后项的值。

现在就尝试找找 e 级数的递推关系。已知 a_i=1/i!，a_{i+1}=1/(i+1)!，则有

a_{i+1}=1/(i+1)!=1/(i!(i+1))=1/i!*1/(i+1)=a_i/(i+1)

至此，根据这个递推公式，就可以在累加循环中递推出后项并求出累加和。即

```
累加项=初始值;
循环语句（条件）
{
    sum+=累加项;
    改变循环控制变量;
    根据递推公式，以累加项和控制变量的当前值求出累加项的后续值并回存到累加项;
}
```

以上的这种反复使用同一个累加项，并通过递推关系来获得累加项的后续值的循环模式称为“**迭代**（*iteration*）”，是循环程序设计中常用的方法。

现在来设计本例的迭代程序。对于求解值 e，可以设其初始值为 1，可以避免求 0! 这样的特殊值，而迭代项 a 的初始值也就顺理成章地设为第二项的值 1。同理，循环控制变量 i 也应该从 1 开始计数，直到 20 结束。程序的流程图如图 5-5 所示。

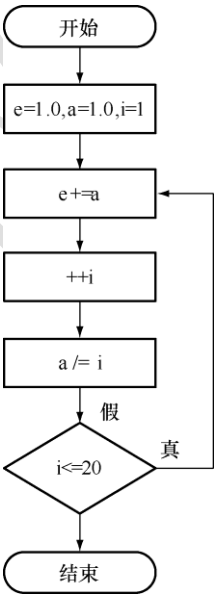

图 5-5 【例 5-3】的流程图

```
//5-3.c
#include <stdio.h>

int main()
{
    int i = 1;
    double e = 1.0; //初始值：e=1/0!
    double a = 1.0; //a = 1/i!=1/1!

    do
    {
        e += a;      //累加
        ++i;
        a /= i;      //求 ai+1
    } while (i <= 20);

    printf("e=%lf\n", e);

    return 0;
}
```

程序的运行结果如下：

```
e=2.718282
```

经过验算，这是一个正确的结果。

【例 5-4】对于【例 5-3】这样的计算，人为确定求和项数来求级数的近似值可能产生较大误差。因此要求改写【例 5-3】，将结束条件改为：前后两次计算结果之间误差的绝对值小于 1^{-8}。

【解题思路】程序的主体仍然可以沿用【例 5-3】的代码。这里需要处理的只是如何求两次相邻运算的结果的差值。显然，这需要用两个变量 laste 和 e 来分别保存上一次和本次的计算结果。当两者之差的绝对值不满足结束条件时，将 laste 赋值为 e，用于下一次计算。在第一次循环中，e 是计算出来的，而 laste 就必须在循环前人为指定。由于级数之和一定是正数，因此可以将 laste 设为任意负数，这里就简单设为-1.0。

求浮点数的绝对值要用到 fabs()函数，这个函数的声明包含在头文件 math.h 中。

```
//ex5-4.c
#include <stdio.h>
```

```
#include <math.h>

int main()
{
    int i = 1;
    double e = 1.0; //初始值：e=1/0!
    double a = 1.0; //a = 1/i!=1/1!
    double laste = -1.0;
    const double epsilon = 1e-8;
    int notEnd = 1;

    do
    {
        e += a; //累加
        ++i;
        a /= i;
        if (fabs(laste - e) < epsilon) notEnd = 0;
        laste = e;
    } while (notEnd);

    printf("i=%d, e=%lf\n", i, e);

    return 0;
}
```

程序的运行结果为：

```
i=13, e=2.718282
```

程序示例了一种结束循环的方法：用一个标志变量确定循环是否继续。例中，标志变量 notEnd 预设为真，表示假定要继续循环；当计算精度达到要求后，将其值置为假，从而结束循环。