

第3章

C程序的控制结构

任何一个结构化程序都可以由顺序结构、选择结构和循环结构这三种基本结构来表示，C 语言的程序也不例外。C 语言程序包含若干条执行语句（简称语句），不同的语句可以实现不同的功能，其中，控制语句可以控制程序的执行流程，用来实现选择结构和循环结构。本章将会介绍 C 程序语句，并通过 C 语言程序实例来介绍如何使用这三种基本结构解决实际问题。

3.1 C 程序语句及三种基本结构

3.1.1 C 程序语句

C 程序的结构如图 3-1 所示，一个 C 语言程序可以由一个或多个源程序文件组成，一个源程序文件则可以由预处理指令、相关全局变量以及一个或多个函数构成。一个函数又由内部变量定义和执行语句两部分组成。执行语句经过编译后产生若干条机器指令，作用是要求计算机系统执行相应的指令操作，内部变量定义不产生机器指令，只是有关数据的声明。

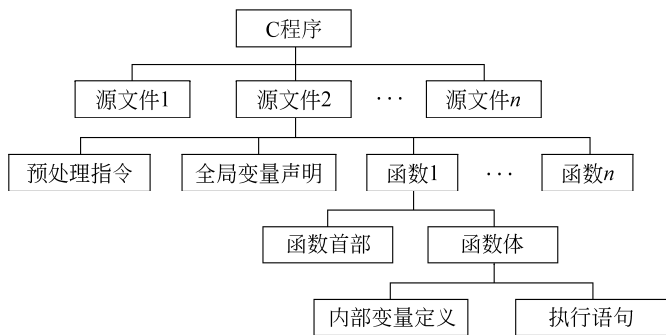


图 3-1 C 程序结构图

语句是程序最基本的执行单位，程序就是通过执行一系列语句来实现其功能的。C 语言中的语句有如下几种形式。

(1) 表达式语句：由表达式加分号组成。例如，在赋值表达式 `i=5` 后面加上分号即“`i=5;`”，就是赋值语句。

(2) 控制语句：用于控制程序的流程，以实现程序的各种结构。主要有以下几种。

- 选择语句：if()…else…
- 多分支语句：switch()…case…
- 循环语句：for()…、while()…、do…while()
- 辅助控制语句（与分支和循环语句搭配使用）：continue（结束本次循环语句）、break（中止执行 switch 或循环语句）、goto（转向语句）
- 返回语句（返回函数结果）：return

以上语句中，()内表示一个判断条件，“…”表示内嵌的语句，例如“if()…else…”语句可以具体写成：

```
if(i>5) i=i-1;
else i=i+1;
```

表示如果“i>5”成立，就执行“i=i-1;”语句，否则执行“i=i+1;”语句。

(3) 函数调用语句：由一个函数调用加一个分号构成一个语句，例如：

```
printf("Hello, World! \n");
```

就是调用函数 printf()的语句。

(4) 空语句：只有一个分号的语句，什么也不做。空语句有时用来作循环语句中的循环体。例如：

```
while (i<0);
```

当 i<0 成立时执行分号表示的空语句，即什么也不做。

(5) 复合语句：用{}把一些语句和声明括起来构成复合语句，又称为块或语句块。例如：

```
if(i>j) {i--; j++;}
```

用花括号括起来的的就是复合语句，表示如果 i>j 成立，则“i--;”和“j++;”两条语句都要执行。

3.1.2 C 程序的三种基本结构

结构化的程序不管是简单还是复杂，主要由顺序结构、选择结构和循环结构这三种基本结构组成，简单程序可以只有其中一种结构，复杂程序可以包含三种结构，这三种结构的运行特点如下。

(1) 顺序结构：顺序结构的程序设计是最简单的，只要按照解决问题的顺序写出相应的语句即可，它的执行顺序是自上而下，依次执行。

(2) 选择结构：选择结构用于判断给定的条件，根据判断的结果来控制程序的流程。条件成立或不成立分别执行不同分支的语句。

(3) 循环结构：循环结构是指在程序中需要反复执行某个功能而设置的一种程序结构。它根据循环条件来判断是继续执行某个功能还是退出循环。根据判断条件的时间顺序不同，循环结构又可细分为两种形式：先判断后执行的循环结构和先执行后判断的循环结

构。循环结构可以减少源程序重复书写的工作量，一般用来描述重复执行某段算法的问题。循环结构的三个要素：循环变量、循环体和循环终止条件。

3.2 顺序结构程序设计

顺序结构的程序中没有控制语句，也不必做任何判断，只需要从上而下一个语句接一个语句地执行，直到执行完所有语句。就跟日常生活中大家排队坐车一样，排在前面的先上，排在后面的后上。顺序结构执行时，一般是先输入需要处理的数据（包括初始化数据），然后运用各种运算对数据进行加工处理，最后将计算结果输出。其流程图如图 3-2 所示。

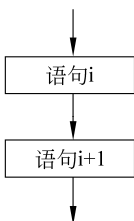


图 3-2 顺序结构流程图

假设顺序结构的代码如下：

```
语句 1;  
语句 2;  
⋮  
语句 n;
```

则执行顺序为：语句 1→语句 2→…→语句 n。

【例 3-1】 两个整数求和。

```
#include<stdio.h>  
int main()  
{  
    int a,b,c;  
    a=3;  
    b=5;  
    c=a+b;  
    printf(" c=%d\n ",c);  
    return 0;  
}
```

程序从上而下依次执行，运行后结果为：

c=8

程序直接给变量 a 和变量 b 赋值，运行后输出结果，如果要计算不同数值的结果，直接改 a 和 b 的值即可。

【例 3-2】 输入三角形的三边，计算三角形的面积，结果保留两位小数。

```
#include <stdio.h>
#include <math.h>           //用到数学函数 sqrt(), 因此需要包含该头文件
int main( )
{
    double a,b,c,d,s;
    printf("请输入三角形的三边: ");
    scanf("%lf%lf%lf",&a,&b,&c);    //double 类型输入格式为%lf
    d=(a+b+c)/2;              //计算周长的一半
    s=sqrt(d*(d-a)*(d-b)*(d-c));  //sqrt( )是求算术平方根的函数
    printf("s=%.2lf\n",s);
    return 0;
}
```

从键盘输入三边 3 4 5↵，则程序的运行结果为：

```
请输入三角形的三边: 3 4 5
s=6.00
```

【例 3-3】 从键盘输入一个小写字母，在屏幕上显示其对应的大写字母。

```
#include <stdio.h>
int main( )
{
    char ch1,ch2;
    int d='a'-'A';          //小写字母与大写字母间的 ASCII 码差值
    ch1=getchar( );         //用 getchar( )函数输入小写字母
    ch2=ch1-d;              //将小写字母转换为大写字母
    putchar(ch2);           //以字符格式输出 ch2 的值
    putchar('\n');
    return 0;
}
```

从键盘输入小写字母 a，程序的运行结果为：

```
a
A
```

3.3 选择结构程序设计

在日常生活中，很多情况都需要根据某个条件来做判断，然后做出选择。例如，出门时如果下雨了就带伞；如果是工作日就上班；遇到十字路口，需要选择一个方向前行等，诸如此类的情況非常多。计算机在处理问题时也可能需要通过判断做出选择，这便是选择结构需要解决的问题。

选择结构（又称分支结构）可以表示更加复杂的逻辑结构。C 语言常用关系运算或逻辑运算来判断条件是否得到满足，并根据计算的结果决定程序的不同流程。C 语言有两种选择语句：if 语句，实现单分支或者双分支的选择结构；switch 语句，实现多分支的选择结构。

3.3.1 if 语句

if 语句实现单分支结构时，语句的一般形式为：

```
if(表达式)
{
    语句组
}
```

单分支 if 语句的执行过程如图 3-3 所示。首先计算表达式的值，如前文所述，选择结构应该先判断表达式给定的条件是否成立，成立时和不成立时执行不同的分支。而计算机在执行程序时，根据表达式计算后的值为 0 还是非 0 来进行判断。值为 0 即“假”，表示条件不成立；值为非 0 即“真”，表示条件成立。因此，计算机在执行如图 3-3 所示的 if 结构时，如果表达式的值为“真”（非 0，即条件成立），则执行表达式后面的语句组；否则，跳过语句组执行 if 结构后面的语句。

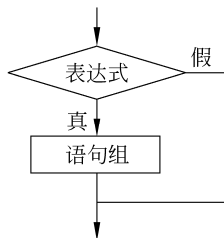


图 3-3 单分支 if 结构

说明：

(1) if (表达式) 中的“表达式”可以是常量、变量、关系表达式、逻辑表达式等。如果表达式的值为“真”，则 if 语句内的语句组将被执行。如果表达式的值为“假”，则 if 语句结束后的语句（闭括号后）将被执行。C 语言把任何非零和非空的值定为“真”，把零或 NULL 定为“假”。例如：

```
① if(2)
    printf("真的");
② if(a==b)
    printf("a=b");
③ if(a-b>0)
    printf("a>b");
```

(2) if (表达式) 后面的语句组可以是一条语句，也可以是多条语句组合而成的复合语句。如果是复合语句，则需要用花括号把复合语句括起来。例如：

```
if (i<j)
{
    i++;
    j--;
}
```

如果 $i < j$ 是“真”的，则 $i++$ 和 $j--$ 两条语句都要执行。

【例 3-4】 输入一个实数，要求输入这个数的绝对值（保留两位小数）。

```
#include <stdio.h>
int main( )
{
    float a,b;
    printf("请输入一个实数: ");
    scanf("%f",&a);
    b=a;
    if(a<0)
        b=-a;           //如果 a<0, 则执行此语句, b 的值为 -a
    printf("该数的绝对值是: %.2f\n",b);
    return 0;
}
```

① 输入 4.5↵，程序的运行结果为：

```
请输入一个实数: 4.5
该数的绝对值是: 4.50
```

② 输入 -3↵，程序的运行结果为：

```
请输入一个实数: -3
该数的绝对值是: 3.00
```

程序分析：一开始将变量 a 的值赋值给变量 b ，如果 $a < 0$ ，就把 $-a$ 的值重新赋值给 b ，如果 $a > 0$ ， b 的值则不变，这样就能保证 b 的值为 a 的绝对值。

【例 3-5】 输入两个整数，按数值从小到大的顺序输出。

```
#include<stdio.h>
int main()
{
    int a,b,t;
    printf("请输入两个整数: ");
    scanf("%d%d",&a,&b);
    if(a>b)           //条件成立则实现 a 和 b 值的交换
    {
        t=a;
        a=b;
        b=t;
    }
    printf("按由小到大顺序输出为: %d %d\n",a,b);
    return 0;
}
```

① 输入 3 19↵，程序的运行结果为：

```
请输入两个整数: 3 19
按由小到大顺序输出为: 3 19
```

② 输入 8 2↵，程序的运行结果为：

```
请输入两个整数：8 2
按由小到大顺序输出为：2 8
```

程序分析：如果输入的值满足条件 $a > b$ ，则借助中间变量 t ，交换 a 、 b 的内容，使 a 里面放的是较小值， b 里面是较大值。

3.3.2 if-else 语句

if 语句实现双分支结构时，语句的一般形式为：

```
if(表达式)
{
    语句组 1
}
else
{
    语句组 2
}
```

双分支 if 语句的执行过程如图 3-4 所示。首先计算表达式的值，如果其值为“真”时，执行语句组 1；否则，执行语句组 2。语句组 1 和语句组 2 只能选择其中一个执行。

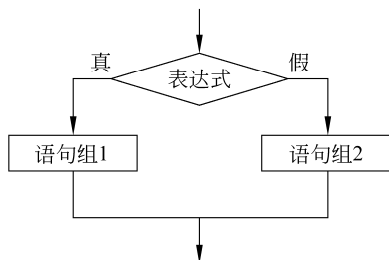


图 3-4 双分支 if 结构

说明：else 子句（可选）是 if 语句的一部分，必须与 if 配对使用，不能单独使用。

【例 3-6】 输入两个整数，求两个数中的较大值。

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

输入 4 8✓后，程序的运行结果为：

```
4 8
The max of 4,8 is 8
```

程序分析：输入 4 和 8 分别存入变量 a 和 b 后，a 等于 4，b 等于 8，由于 $a < b$ ，所以 $a > b$ 条件不成立，执行 else 后面的“c=b;”语句，因此最后的结果 c 为 8。

【例 3-7】 输入一个学生的成绩，判断是否合格。

```
#include<stdio.h>
int main()
{
    float a;
    printf("请输入1个0~100的分数: ");
    scanf("%f",&a);
    if(a<60)
        printf("成绩不合格\n");
    else
        printf("成绩合格\n");
    return 0;
}
```

① 输入 58.9✓，程序的运行结果为：

```
请输入1个0~100的分数: 58.9
成绩不合格
```

② 输入 67✓，程序的运行结果为：

```
请输入1个0~100的分数: 67
成绩合格
```

3.3.3 嵌套的 if 语句

当 if（表达式）或 else 后的语句组合中包含另一个 if 语句结构时，就形成了 if 语句的嵌套结构。选择结构的嵌套形式较多，可根据需要选择嵌套的形式。在 if（表达式）和 else 里面都嵌套 if-else 结构的嵌套形式如下。

```
if(表达式 1)
{
    if(表达式 2)
    { 语句组 1 }
    else
    { 语句组 2 }
}
else
{
    if(表达式 3)
    { 语句组 3 }
    else
    { 语句组 4 }
}
```

} 内嵌 if-else 结构

} 内嵌 if-else 结构

```
}
```

以上形式可根据需要再增加 if 嵌套，或者省略 else 语句（包括外层的和内嵌的），例如：

```
if(表达式 1)
{
    if(表达式 2)
    { 语句组 1 }
    else
    { 语句组 2 }
}
```

内嵌 if-else 结构

如果再省略内嵌的 else 语句，则嵌套形式又变为：

```
if(表达式 1)
{
    if(表达式 2)      //内嵌 if 结构
    { 语句组 1 }
}
```

由于嵌套形式繁多，就不一一举例。需要注意的是，当出现多个 if 和 else 时，配对原则为：在同一个复合语句括号“{ }”作用域内，else 总是与它上面最近的未配对的 if 配对。

if-else 语句嵌套后可实现多分支结构，例如，在 else 部分又嵌套多层 if 语句的一般形式为：

```
if(表达式 1)
{
    语句组 1
}
else if(表达式 2)
{
    语句组 2
}
:
else if(表达式 n)
{
    语句组 n
}
else
{
    语句组 n+1
}
```

此结构的执行过程如图 3-5 所示。首先计算表达式 1 的值，若其值为“真”，则执行语句组 1；否则，若表达式 2 的值为“真”，则执行语句组 2……以此类推，直到判断表达式 n，若表达式 n 的值为“真”，则执行语句组 n，否则执行语句组 n+1。最后的 else 也可以根

据需要省略。

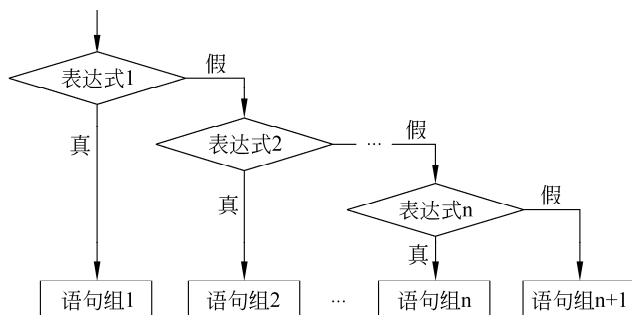


图 3-5 多分支 if 结构

【例 3-8】 输入一个学生的成绩，对此成绩进行分级。成绩小于 0 分或者大于 100 分提示“输入错误”，成绩在 90 分以上为“优”，80~89 分为“良”，70~79 分为“中”，60~69 分为“及格”，60 分以下为“不及格”。

```

#include <stdio.h>
int main( )
{
    int score;
    printf("请输入学生的成绩: ");
    scanf("%d",&score);
    if(score<0||score>100)
        printf("输入错误! \n");
    else if(score>=90)
        printf("优! \n");
    else if(score>=80)
        printf("良! \n");
    else if(score>=70)
        printf("中! \n");
    else if(score>=60)
        printf("及格! \n");
    else
        printf("不及格! \n");
    return 0;
}
  
```

① 输入 101↵，运行结果为：

```

请输入学生的成绩: 101
输入错误!
  
```

② 输入 56↵，运行结果为：

```

请输入学生的成绩: 56
不及格!
  
```

③ 输入 95↵，运行结果为：

```
请输入学生的成绩：95
优！
```

说明：如果输入的成绩不满足 `if(score<0||score>100)` 中的条件，则在 `else if(score>=90)` 中，`else` 就已经排除了 `score<0` 和 `score>100` 的情况，所以虽然条件只写了 `score>=90`，但实际条件相当于 `score>=90&&score<=100`。后面的 `else` 也都表示排除了前面的所有可能。

此程序除了用多分支 `if` 结构实现，还可以用单分支 `if` 结构实现，用单分支 `if` 结构实现代码如下。

```
#include <stdio.h>
int main( )
{
    int score;
    printf("请输入学生的成绩: ");
    scanf("%d",&score);
    if(score<0||score>100)
        printf("输入错误! \n");
    if(score>=90&&score<=100)
        printf("优! \n");
    if(score>=80&&score<=89)
        printf("良! \n");
    if(score>=70&&score<=79)
        printf("中! \n");
    if(score>=60&&score<=69)
        printf("及格! \n");
    if(score>=0&&score<=59)
        printf("不及格! \n");
    return 0;
}
```

输入 74✓ 后，程序的运行结果为：

```
请输入学生的成绩：74
中！
```

【例 3-9】 输入 x 的值，要求根据如下公式输出对应 y 的值。

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

用嵌套的 `if` 结构实现，方法如下。

方法一：在 `if` 里面嵌套。

```
#include <stdio.h>
int main( )
{
    int x,y;
    printf("请输入 x 的值: ");
```

```
scanf("%d",&x);
if(x<=0)
{
    if(x==0)
        y=0;
    else y=-1;
}
else y=1;
printf("y=%d\n",y);
return 0;
}
```

① 输入-3✓，运行结果为：

```
请输入x的值: -3
y=-1
```

② 输入0✓，运行结果为：

```
请输入x的值: 0
y=0
```

③ 输入7✓，运行结果为：

```
请输入x的值: 7
y=1
```

方法二：在 else 里面嵌套。

```
#include <stdio.h>
int main( )
{
    int x,y;
    printf("请输入 x 的值: ");
    scanf("%d",&x);
    if(x<0)
        y=-1;
    else
    {
        if(x==0)
            y=0;
        else
            y=1;
    }
    printf("y=%d\n",y);
    return 0;
}
```

① 输入-5✓，运行结果为：


```
请输入x的值: -5
y=-1
```

② 输入 0✓，运行结果为：

```
请输入x的值: 0
y=0
```

③ 输入 5，运行结果为：

```
请输入x的值: 5
y=1
```

除了以上两种嵌套的 if 结构，该例题还可以用单分支及多分支的 if 结构实现。读者在熟悉 if 语句后，可以根据需要选择合适的结构解题。

3.3.4 switch 语句

实际问题中，多分支选择情况还是很多，如成绩分类、学历划分、工资税率问题等，虽然 if 语句可以处理多分支选择，但是当分支较多时，用 if...else 来处理，嵌套层次就会增多，导致程序可读性较差。在 C 语言中，switch 语句可直接实现多分支选择。

switch 语句是多分支选择语句，也称为开关语句。switch 语句的一般形式：

```
switch (表达式)
{
    case 常量表达式 1: 语句组 1; break;
    case 常量表达式 2: 语句组 2; break;
    :
    case 常量表达式 n: 语句组 n; break;
    default: 语句组 n+1; break;
}
```

switch 语句的执行过程如图 3-6 所示。首先计算 switch 后面括号内“表达式”的值，然后将此值与各 case 后面的常量表达式的值比较。如果与某个 case 后的常量表达式的值相等，则执行该 case 后面的语句组，遇到 break 语句，switch 结构终止，控制流程跳转到 switch 语句后的下一行；如果与所有 case 后的常量表达式的值都不相等，则执行 default 后的语句组。

说明：

- (1) switch 后面括号内的“表达式”只能是整型、字符型或枚举型表达式。
- (2) 各个 case 后的常量表达式的值必须不相同。
- (3) case 和常量表达式之间一定要有空格，常量表达式后有冒号。
- (4) 各个 case 及 default 的前后顺序改变，不影响程序执行结果。
- (5) 多个 case 可以共用一个语句块。
- (6) default 可以省略，当 switch 表达式的值与各个 case 后面的常量表达式的值都不匹配时，不执行任何语句，直接跳出 switch 结构。
- (7) break 语句用于终止 switch 结构，使控制流程跳转到 switch 语句后的下一行。如

果不使用 `break`，则从与表达式匹配的那个 `case` 语句开始，后面所有的语句组都会被执行。

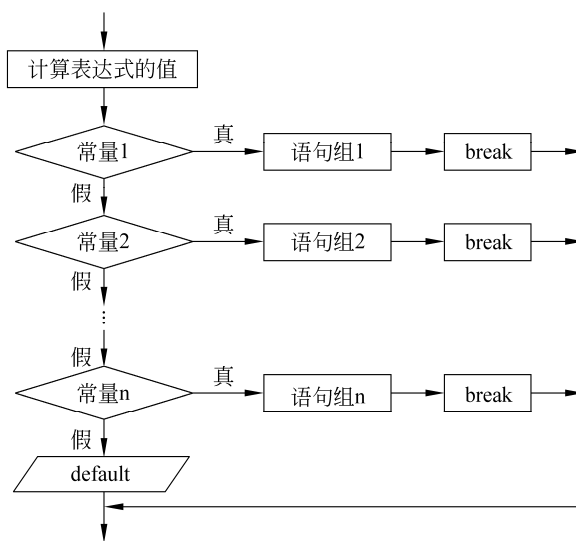


图 3-6 switch 多分支结构

【例 3-10】 根据输入的月份，输出相应的季节。春季 3、4、5 月，夏季 6、7、8 月，秋季 9、10、11 月，冬季 12、1、2 月。

```

#include <stdio.h>
int main( )
{
    int month;
    printf("请输入月份: ");
    scanf("%d",&month);
    switch(month)
    {
        case 3:
        case 4:
        case 5: printf("春季! \n"); break;
        case 6:
        case 7:
        case 8: printf("夏季! \n"); break;
        case 9:
        case 10:
        case 11: printf("秋季! \n"); break;
        case 12:
        case 1:
        case 2: printf("冬季! \n"); break;
        default: printf("输入错误! \n");
    }
    return 0;
}
  
```

输入 6↵，程序的运行结果为：

```
请输入月份：6
夏季！
```

说明：程序中多个 case 共用一个语句，如 case 3、case4、case5 共用“printf("春季! \n"); break;”语句。当输入值为 1~12，都可以找到相应 case 后的常量匹配，从而执行后面的语句，如果输入值超出 1~12，则执行 default 后的语句。

【例 3-11】 从键盘输入一个由运算符（+、-、*或/）连接的运算式，要求根据输入的运算符计算两个数的运算结果（保留两位小数）。

```
#include <stdio.h>
int main( )
{
    float a, b, c;
    char op;
    int flag=1;           //设 flag 为标志位，以判断运算符输入是否正确
    printf("请输入运算式: ");
    scanf("%f%c%f", &a, &op, &b);
    switch(op)
    {
        case '+': c = a + b; break;
        case '-': c = a - b; break;
        case '*': c = a * b; break;
        case '/': c = a / b; break;
        default: flag = 0;
    }
    if(flag==1)           //flag 值未变，说明输入了正确运算符
        printf("%.2f%c%.2f=%.2f\n", a, op, b, c);
    else printf ("运算符输入错误! \n");
    return 0;
}
```

① 输入 3.5+8↵，程序的运行结果为：

```
请输入运算式：3.5+8
3.50+8.00=11.50
```

② 输入 29/5↵，程序的运行结果为：

```
请输入运算式：29/5
29.00/5.00=5.80
```

说明：程序增加了一个 flag 标志，flag 初始值为 1，如果输入的是规定的运算符，则会匹配其中一个 case，执行其后面的语句，flag 的值不会改变，正常输出运算式的结果；如果输入的不是规定运算符，没有匹配的 case，则执行 default 语句，使 flag=0，结果输出“运算符输入错误!”。

3.4 循环结构程序设计

前面简单提到过循环结构，其特点在于“循环”，即重复地执行某部分功能代码。而日常生活中需要重复处理的问题非常多，例如，每天早上起来晨跑，每天按时上下班，统计全班每一个同学各科的总分及平均分等。在程序处理中也会遇到很多需要重复处理的问题，例如，计算 $1+2+3+\cdots+100$ 或者计算 $1\times 2\times 3\times\cdots\times 20$ 等，这类问题通常用循环结构来实现。循环结构有三个要素：循环变量、循环体和循环终止条件，当未达到循环终止条件（即满足循环条件）时，就重复执行循环体，改变循环变量，直到达到循环终止条件，就结束循环。循环变量主要用以使循环条件向终止方向靠近，否则容易造成死循环。C 语言中主要有 3 种循环语句：`while` 语句、`do-while` 语句和 `for` 语句。

3.4.1 while 语句

`while` 语句是“当型循环”控制语句，其特点是：先判断条件，后执行循环。它的一般形式为：

```
while(表达式)
{
    循环体
}
```

执行过程如图 3-7 所示，首先判断表达式的值，如果值为“假”，直接终止循环，转去执行循环体外的下一行；如果值为“真”，执行循环体，再重复判断表达式的值，执行循环体，直到表达式的值为“假”为止。

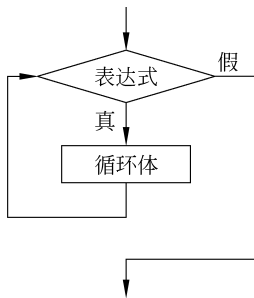


图 3-7 while 语句执行流程图

说明：

(1) 循环体如果只有一条语句，可以省略其外面的花括号“{ }”，如果有多条语句，则不能省略，必须是用花括号“{ }”括起来的复合语句。

(2) `while(表达式)`后如果添加分号“;”，则表示循环体为空语句。

(3) 如果“表达式”的值在第一次判断时为假，则循环体一次都不会被执行。

【例 3-12】 用 `while` 语句计算 $1+2+3+\cdots+100$ 的值。

```
#include<stdio.h>
int main( )
{
    int i=1, sum=0;           //定义变量 i 和 sum, 并初始化
    while(i<=100)             //判断 i 的值是否小于或等于 100, 是则执行循环体
    {
        sum=sum+i;           //将 i 的值累加在 sum 中
        i++;                 //循环变量 i 的值加 1
    }
    printf("1+2+3+...+100=%d\n", sum);
    return 0;
}
```

程序的运行结果为:

1+2+3+...+100=5050

说明:

(1) 程序中累加的数据是有规律的, 每次加的数都比前一个数多 1, 因此可以用 `i++` 或者 `++i` 来实现。

(2) 变量 `i` 和 `sum` 都要赋初值, 如果不赋值, 则为随机值, 会导致结果错误。

(3) 循环体包含两条语句, 因此必须用花括号 “{ }” 括起来, 否则 `while` 语句的范围只能管到 `while` 后面第一条语句。

(4) 变量 `i` 的初值为 1, 满足循环条件, 随着循环的执行, `i` 的值逐渐增加, 直至大于 100, 循环结束, 循环体执行了 100 次。

【例 3-13】 输入两个整数, 求两个数的最大公约数。

```
#include<stdio.h>
int main( )
{
    int a,b,c;
    printf("请输入两个整数:");
    scanf("%d%d",&a,&b);
    c=(a>b)?b:a;           //使 c 为 a, b 中的较小值
    while(a%c!=0||b%c!=0)  //如果 a 或者 b 不能被 c 整除, 则执行循环体
    {
        c--;               //c 的值最多减少至 1 时, 就会使循环条件不满足而跳出循环
    }
    printf("%d 和 %d 的最大公约数是:%d\n ",a,b,c);
    return 0;
}
```

① 输入 12 18↵, 程序输出结果为:

请输入两个整数:12 18
12和18的最大公约数是:6

② 输入 4 7↵, 程序输出结果为:

请输入两个整数:4 7
4和7的最大公约数是:1

说明：这里使用了穷举法，由于两个数的最大公约数有可能是其中的小数，所以用条件运算符“ $c=(a>b)?b:a$ ”使 c 的值为 a, b 中的较小值，然后 a, b 分别都与 c 相除，只要有一方不能被 c 整除，就继续循环，循环中 $c--$ ， c 逐次减少，直到 a, b 都能同时被 c 整除，循环结束。当然， c 最小减少到 1 时， a, b 都能被其整除。

3.4.2 do-while 语句

do-while 语句是“直到型循环”控制语句，其特点是：先执行循环，再判断条件。它的一般形式为：

```
do
{
    循环体
}while(表达式);
```

执行过程如图 3-8 所示，先执行 do 后面的循环体，然后判断表达式的值，如果值为“假”，结束循环，转向执行循环体外的下一行；如果值为“真”，继续执行循环体，再重复判断表达式的值，执行循环体，直到表达式的值为“假”为止。

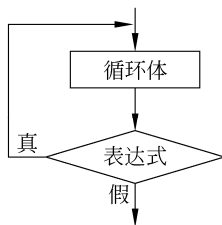


图 3-8 do-while 语句执行流程图

说明：

- (1) while(表达式)后面的分号“;”是结构的一部分，必须有。
- (2) 循环体至少执行一次，当第一次判断“表达式”为假时，循环体已经执行了。
- (3) 循环体如果只有一条语句，可以省略其外面的花括号“{ }”，如果有多条语句，则不能省略。例如：

循环体无花括号：

```
i=10;
do
    printf("%d",i--);
while(i>0);
```

循环体有花括号：

```
i=10;
do{
    printf("%d",i--);
}while(i>0);
```

由于 do-while 结构最后本来就有个分号“;”，为增加程序的可读性，即使循环体只有一条语句，也建议用花括号“{ }”括起来。

【例 3-14】 用 do-while 语句计算 $1+2+3+\cdots+100$ 的值。

```
#include<stdio.h>
int main( )
{
    int i=1, sum=0;
    do{
        sum=sum+i;
        i++;
    }while(i<=100);
    printf("1+2+3+...+100=%d\n", sum);
    return 0;
}
```

程序的运行结果为：

```
1+2+3+...+100=5050
```

说明：使用 do-while 实现与例 3-12 中使用 while 实现基本思路是一样的，只是 do-while 将循环体放到 do 后先执行，再判断条件，两种结构循环的次数一致，而最后执行结果也一致。

在大多数情况下，能用 while 处理的题目也能够用 do-while 处理，但是在第一次判断就为“假”的情况下，两者执行结果不一致，while 语句的循环体一次都不执行，而 do-while 语句的循环体会执行一次。

【例 3-15】 输入一个正整数 n，要求输出从 n 到 1 的整数，即输出 n, n-1, ..., 2, 1。

(1) 用 while 语句实现。

```
#include<stdio.h>
int main( )
{
    int n;
    printf("请输入一个正整数: ");
    scanf("%d",&n);
    while(n>0)
    {
        printf("%d ",n);
        n--;
    }
    putchar('\n');
    return 0;
}
```

① 当输入 6✓时，程序的运行结果为：

```
请输入一个正整数: 6
6 5 4 3 2 1
```

② 当输入 -3✓时，程序的运行结果为：

请输入一个正整数: -3

(2) 用 do-while 语句实现。

```
#include<stdio.h>
int main( )
{
    int n;
    printf("请输入一个正整数: ");
    scanf("%d",&n);
    do{
        printf("%d ",n);
        n--;
    }while(n>0);
    putchar('\n');
    return 0;
}
```

① 当输入 6✓时, 程序的运行结果为:

请输入一个正整数: 6
6 5 4 3 2 1

② 当输入-3✓时, 程序的运行结果为:

请输入一个正整数: -3
-3

说明: 当输入正整数时, 满足循环条件, while 结构和 do-while 结构运行结果一致; 而当输入负数时, while 结构结束循环, 没有输出结果, 而 do-while 结构执行了一次循环, 将输入的负数输出了一次, 然后结束循环。

3.4.3 for 语句

for 语句是三种循环结构中使用最广泛的, while 语句能处理的问题都可以用 for 语句来处理, 它的一般形式为:

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

其中三个表达式各自表示如下。

- (1) 表达式 1: 只执行一次, 主要是给循环变量设置初值, 一般为赋值表达式。
 - (2) 表达式 2: 为循环条件, 一般为关系表达式或逻辑表达式。
 - (3) 表达式 3: 主要用于调整循环变量的值, 使循环变量值改变后循环趋于结束。
- 因此, for 语句还可以写成下面这种更容易理解的形式。