6 选择结构与循环结构

从上到下依次执行的程序被称为顺序结构，但有时很多情况并不使用于顺序结构来处理，所以C语言中提供了另两种的结构，选择结构（分支结构）与循环结构来处理相适用的情况。

6.0 选择结构

6.0.0 if , if-else 语句

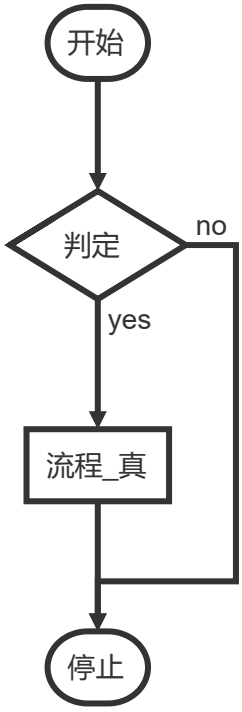

图 6-1-1 选择结构流程

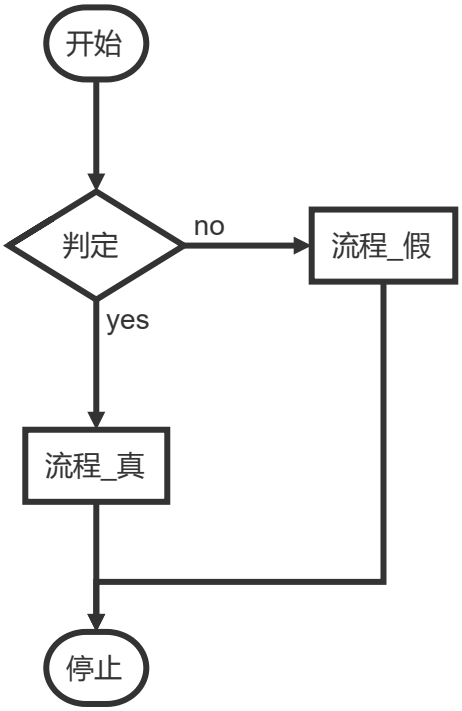

图 6-1-2 选择结构流程

if 语句分为两种：

- if(表达式) {语句_真;}
- if(表达式) {语句_真;} else {语句_假;}

其中，表达式的类型为标量（字符，整型，浮点数或指针）类型，后面可以是任意类型的语句，包括复合语句、表达式语句、选择语句、循环语句和跳转语句。第一种的含义为，当表达式的值为真（非零）时，执行语句_真，否则什么都不做。第二种的含义为，当表达式的值为真时，执行语句_真，否则执行语句_假。

选择结构的语法很容易理解，重点在于正确地利用表达式来表达实际问题中的逻辑关系。

```c
#include <stdio.h>
// 判断用户输入的整数是奇数还是偶数
int main()
{
    int n;
    scanf("%d, &n");
    if (n&1) {puts("odd");}
    else {puts("even");}
    // 当 n 为奇数时，二进制最低位一定为1，表达式 n&1 的值为1，条件为真，所以输出奇数；否则输出偶数
    return 0;
}
```

```c
#include <stdio.h>
#include <math.h>
// 判断用户输入的整数是否为完全平方数
const double eps = 1e-6;
int main()
{
    int n, r;
    scanf("%d", &n);
```

```
    r = sqrt(n) + eps;  // 使用一个极小量防止 double 截断误差
    if (r*r == n) {puts(Yes);}
    else {puts("No");}
    // 利用平方根的平方来判断是否为完全平方数
    return 0;
}
```

```
#include <stdio.h>
// 用户输入一个英文字母，对其进行大小写转换
int main()
{
    char c;
    c = getchar();
    if (c>='a' && c<='z') {c += 'A'-'a';}
    else {c += 'a'-'A';}
    // 利用 ASCII 编码中字母编码来进行大小写的判断与转换，要注意判断变量范围的表达式要写准确
    putchar(c);
    return 0;
}
```

```
#include <stdio.h>
// 闰年判断
int main()
{
    int year;
    scanf("%d", &year);
    if (y%400==0 || y%100!=0 && y%4==0) {
        puts("Leap year");
    }
    else {puts("Common year");}
    return 0;
}
```

6.0.1 条件表达式

条件表达式由条件运算组成，这是C语言中唯一一个三元操作符，具体格式为：

```
条件 ? 表达式_真 : 表达式_假
```

它将先计算条件表达式的值，如果为非零，那么执行表达式_真，否则执行表达式_假。条件表达式的值为被执行的表达式的值，其类型与表达式_真与表达式_假的类型相关。这与选择结构相仿，但条件表达式还是存在一些限制，主要表现在表达式_真与表达式_假的类型规定上：

- 两个表达式都为算术类型
- 两个表达式都为相同的结构体或联合体类型
- 两个表达式都为void类型
- 两个表达式都为指针类型，且指向的元素为兼容类型，忽略cvr（constant、volatile、restrict）类型限定符
- 一个表达式是指针而另一个是空指针常量（例如 NULL）
- 一个表达式是指向对象指针而另一个是指向void（可以限定）指针

在实际使用时尽量使用相同类型的表达式_真与表达式_假，也尽量不要使用复杂的表达式作为条件表达式中的条件部分或执行部分，以提高代码的可读性，在前面判断整数奇偶的程序中可以使用

```
puts(n&1?“odd”:“even”);
```

来替代以简化代码。

6.0.2 if - else 嵌套

if 语句中语句真与语句假本身也可以是 if 语句，所以可以实现 if - else 的嵌套结构，来应对更加复杂的情况。

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

    scanf("%d", &year);
    if (y%400 == 0) {puts("Leap year");}
    else if (y%100 == 0) {puts("Commom year");}
    else if (y%4 == 0) {puts("Leap year");}
    elae {puts("Common year");}
    /*
    char *p = "Common year";
    if (y%100 == 0) {
        if (y%400 == 0) {p = "Leap year";}
    } else {
        if (y%4 == 0) {p = "Leap year";}
    }
    puts(p);
    */
    return 0;
}
```

但这也会是 if-else 的对应关系易产生歧义，所以C语言中约定 else 总是和它最近的前接 if 匹配。注释中的另一种写法就是另一种 if-else 的匹配结构。

实际使用中，会经常使用 if-else 的嵌套结构，完成类似于图 6 - 2 的所谓“多路开关”的程序控制流程。

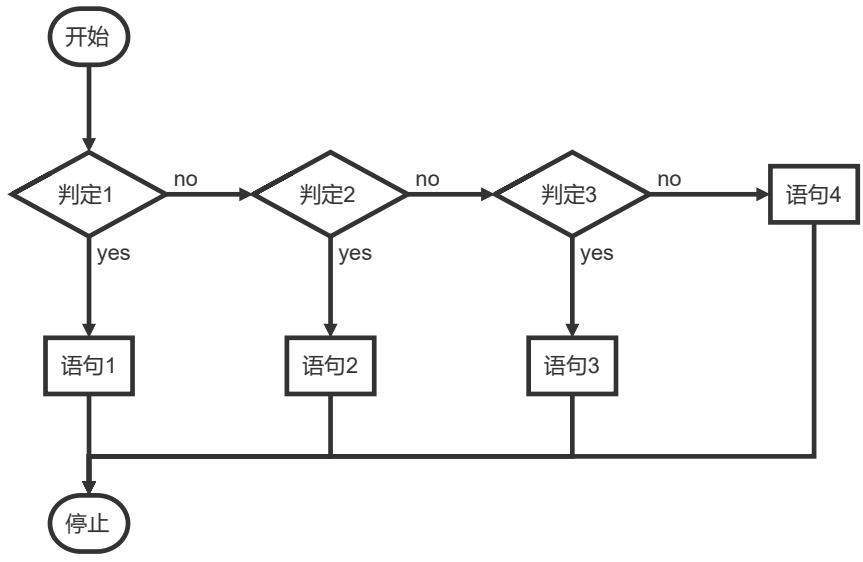

图 6-2 “多路开关”程序控制流程

```c
#include <stdio.h>
// 用户输入一个加减乘除的运算，格式为“数字操作符数字”，计算结果并输出
int main()
{
    double x, y;
    char op;
    scanf("%lf%c%lf", &x, &op, &y);
    if (op == '+') {printf("%f + %f = %f\n", x, y, x+y);}
    else if (op == '-') {printf("%f - %f = %f\n", x, y, x-y);}
    else if (op == '*') {printf("%f * %f = %f\n", x, y, x*y);}
    else if (op == '/') {printf("%f / %f = %f\n", x, y, x/y);}
    else {printf("operator illegal");}
    return 0;
}
```

在逻辑情况比较复杂时，很容易导致逻辑上的错误，这是最好将所有逻辑分支情况理清，之后再编写程序。

6.0.3 switch语句

在处理诸如图 6 - 3 的多路开关控制时，除了使用 if-else 嵌套外，C语言还提供了switch 语句来实现更便捷的操作，比如前面加减乘除表达式的程序也可以这样实现：

```c
#include <stdio.h>
int main()
{
    double x, y;
    char op;
    scanf("%lf%c%lf", &x, &op, &y);
    switch (op) {
        case '+':
            printf("%f + %f = %f\n", x, y, x+y);
            break;
        case '-':
            printf("%f - %f = %f\n", x, y, x-y);
            break;
        case '*':
            printf("%f * %f = %f\n", x, y, x*y);
            break;
        case '/':
            printf("%f / %f = %f\n", x, y, x/y);
            break;
        default:
            puts("operator illegal");
            break;
    }
    return 0;
}
```

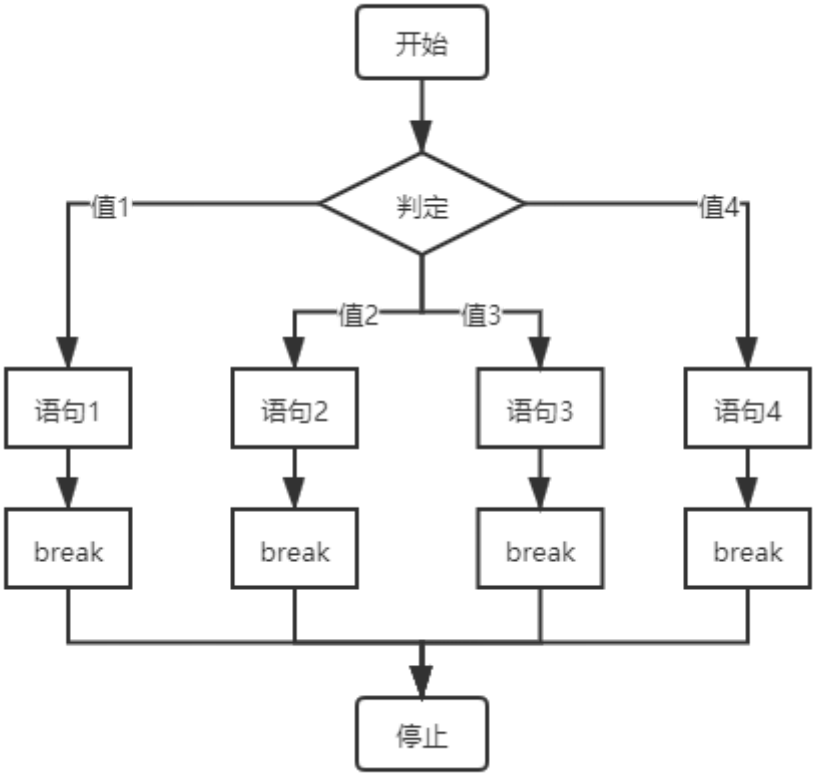

图 6-3 根据判断表达式值的多路开关

switch 语法格式为：

```
switch（表达式）语句
```

其中，表达式必须为整数类型，包括 char ，有符号或无符号整数，或枚举。语句一般都为复合语句，允许在语句中有“case:”和“default:”标号，而“break;”语句有特殊含义。“case:”标号的格式为

```
case 整数常量表达式: 语句
```

，“default:”标号的格式为

```
default: 语句
```

。在switch的使用中要注意以下几点：

- 表达式必须是整数类型不能是浮点数或数组等其他类型。
- 可以有多个“case:”标号，但其后的表达式必须为整数常量表达式，不能是浮点数或字符串常量，更不能是关系或逻辑表达式。所以“case:”标号后的常量表达式的值要唯一。注意，“case 1,2,3:”这种标号值为逗号表达式，其值为3，并不是1、2或3中的任意一个。
- 最多有1个“default:”标号。
- 表达式求值后，如果匹配某个“case:”标号的值，那么程序跳转到这个标号处执行。否则，跳转到“default:”标号执行；如果不存在“default:”标号，那么跳转出switch语句体。
- 在语句中任何位置遇到“break;”，跳出语句体。如果没有遇到“break;”，那么仍按照语句顺序执行，这时就可能执行多个“case:”标号语句。所以，除非多个标号的处理程序相同，否则每个标号的语句后都应该加入“break;”语句。

综述，多路开关控制可以使用 if-else 嵌套或者 switch 语句来实现，两者各有特点：

- if-else 嵌套更加灵活，能适用于各种场景，而 switch 语句只适用于根据整数表达式的值来进行多路选择的场景。
- 在 switch 的适用场景中，它比 if-else 嵌套的结构层次更清晰，可读性更强。

```c
#include <stdio.h>
#include <math.h>
/* 计算一种特殊的分段函数
f(x) =  x,          0<=x<2
        (2x)^(1/2),  2<=x<4
        (x+4)^(1/2), 4<=x<10
        0,          others
*/
int main()
{
    double x, y;
    scanf("%lf", &x);

    // if-else 嵌套结构
    if (x<0.0) {y = 0.0;}
    else if (x<2.0) {y = x;}
    else if (x<4.0) {y = sqrt(2.0*x);}
    else if (x<10.0) {y = sqrt(4.0+x);}
    else y = 0.0;
    /* switch 语句结构
    int k;
    k = floor(x/2.0);
    switch (k) {
        case 0: y = x; break;
        case 1: y = sqrt(2.0*x); break;
        case 2:
        case 3:
        case 4: y = sqrt(4.0+x); break;
        default: y = 0.0; break;
    }
    */ // 但这种写法有很多细节，如果不注意很容易出错，所以要使用这种方式时必须仔细检查细节

    printf("%f\n", y);
    return 0;
}
```

6.1 循环结构

C语言中提供了 while 、do-while 语句以及 goto 语句来实现需要根据某些条件重复执行的情况，这也被称为循环结构（重复结构）。

循环结构有两种：

- 当型循环结构　当条件成立时，反复执行某操作，直到条件为假时停止循环。
- 直到型循环结构

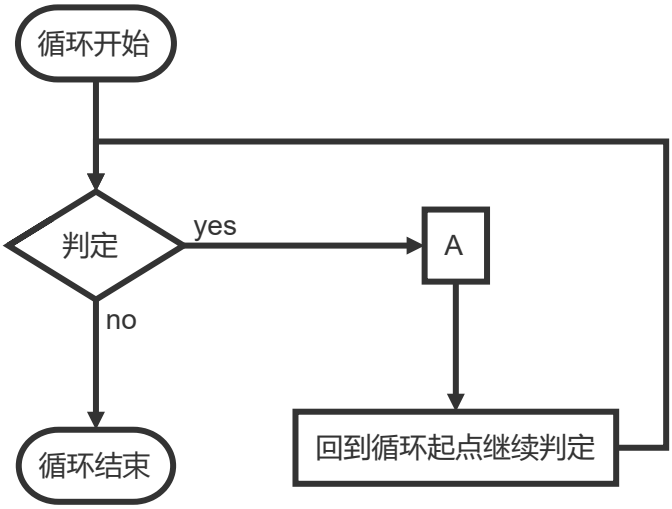

图 6-4 循环图（当型）

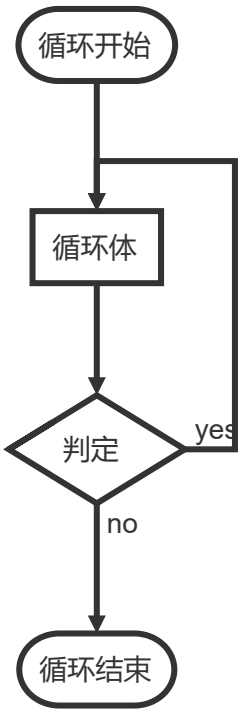

图 6-4 循环图（直到型）

6.1.0 while语句

在 C 语言中一般使用 while 语句来实现当型循环，它的一般形式为：

```
while (exp)
    stat;
```

其中，exp 是任意合法的关系表达式或逻辑表达式，也可以是基本类型变量或常量，被称为循环条件测试表达式。stat 是一条语句，被称为循环体。语法上规定，循环体只能是一条语句，因此，当需要重复执行多条语句时，应使用复合语句（使用大括号括起）。

while 语句的执行流程是：首先对表达式 exp 求值，若其值为真（非零），则执行循环体；循环体执行完毕后再次去判断表达式 exp，若其值为真，则又去执行循环体……如此反复，直到表达式 exp 的值为假，就退出循环。这就意味着，循环体中要存在一个影响 exp 值的语句，否则 exp 的值永远不会改变，也就不会存在 exp 为假的情况，循环永远不会结束，这就造成了“死循环”。另外，不正确的 exp 表达式也可能造成永真，构成“死循环”。

while 语句的特点是先判断，后执行，因此循环体除死循环外也有可能一次也不执行。

```c
#include <stdio.h>
// 依次输出1、2、3……100
int main()
{
    int cnt = 1;         // 赋初值
    while (cnt<=100) {  // 循环到100为止
        printf("%d\n", cnt);
        cnt++;          // 变量自增
    }
    return 0;
}
```

对于一个循环而言，要注意以下几点：

- 循环变量赋初始值
- 循环的终止条件，防止死循环
- 循环体有多条语句时要用大括号括起

```c
#include <stdio.h>
// 求1+2+3+……+100之和
int main()
{
    int sum = 0;     // 累加器清零，作为每次加法运算的结构存储变量
    int cnt = 1;     // 加数赋初值
    while (cnt<=100) {
        sum = sum + cnt;     // 累加
        cnt++;
    }
    printf("sum = %d", sum);
    return 0;
}
```

这两个小程序在结构上很相似，只是前者的循环体中输出数据，后者将数据累加。这两个程序也是大多数循环问题的基础。

```c
#include <stdio.h>
// 数字逆序输出
/* 比如输入正整数32496，逆向输出为69423。
    如果要逆序输出，可以将原数的各个数位的数据分离，然后再逆向分别输出。这是一个十进制的整数，分离数位可以考虑利用对10取余，即可得到最低位数据，然后再对原数对10取商，再取余得到倒数第二位的数据，依次循环直到所有的数位都被提取出。
*/
int main()
{
    int number, remainder;
    printf("请输入一个十进制整型数：\n");
    scanf("%d", &number);
    printf("逆向输出的数字是：\n");
    while (number>0) {               // 循环直到数字变为 0 为止
        remainder = numder % 10;      // 分离末位
        printf("%d", remainder);
        number = number / 10;
    }
    return 0;
```

```
}
```

6.1.1 break语句

简单的循环结构只能等到循环条件为假时才能结束循环，也就是 while 本身只提供了一个出口。但有时我们需要程序状态满足一定条件后就即刻跳出循环，这就需要使用 break 语句。之前讨论 switch 语句时就已经接触到了 break 语句，它的作用正是跳出当前层次结构。在循环体中使用后，即可跳出当前一层循环，之所以说一层只因为循环也可以进行嵌套，循环嵌套之后再进行讨论。

要注意 break 语句只能用于循环体和 switch 语句之中。

break 语句的形式很简单，就只有：

```
break;
```

```c
#include <stdio.h>
// 使用 break 提前退出循环
int main()
{
    int i = 1;
    while (i<=100) {
        printf("%d\n", i);
        i++;
        if (i>=10) {
            break;
        }
    }
    return 0;
}
```

在循环体中，break 语句通常写在 if 语句中，否则第一次循环就跳出循环，循环就失去了意义。break 语句为循环提供了两个或者更多的出口，程序的可读性会降低，所以不能滥用 break ，每一次使用都值得注意。

```c
#include <stdio.h>
// 判断输入数的位数
int main()
{
    int number;
    int cnt = 1;        // 计数器置 1 ，最少也是一个一位数
    printf("请输入一个正整数");
    scanf("%d", &number);
    while (1) {
        number = number / 10;
        if (number>0) {
            cnt++;
        } else {
            break;
        }
    }
    printf("这是一个%d位数", cnt);
    return 0;
}
```

6.1.2 continue语句

除了 break 语句外，C语言中还有另一种用于辅助控制循环流程的语句：continue。该语句的功能是，使当前执行的循环体中止，即跳过 continue 语句后面尚未执行的该循环体中的所有语句，但不结束整个循环，而是继续进行下一轮循环。

continue 语句只能在循环体中使用。如果它出现在 while 循环中，只要执行该语句，会立即跳转到 while 的循环条件表达式处，进行条件测试，以确定是否进行下一次循环。

```c
#include <stdio.h>
// 输出[1, 100]之间不能被 3 整除的数
int main()
{
    int i = 0;
    while (i<100) {
        i++;
        if (i%3 == 0) {
            continue;
        }
        printf("%4d\n", i);
    }
    // 不使用 continue 语句也可以实现这样的功能，这里只是示例 continue 的用法
    return 0;
}
```

6.1.3 do-while语句

另一种直到型循环结构使用 do-while 语句来实现，其语法形式为：

```
do {
    stat;
} while (exp);
```

其中，stat 即为循环体；exp 是循环结束的条件，它与while中的exp的限定类型是一样的。同样，这个大括号也是为了多条语句的复合而写，但多数情况下，即使循环体中只有一条语句，也会打上大括号，以作区分，不易和 while 语句混淆。

do-while 语句的功能是，重复执行由 star 构成的循环体，直到紧跟在 while 后的表达式 exp 的值为假时循环结束。

与 while 语句相比：

- do-while 语句总是先执行循环体，然后再判断循环结束条件。因此，循环体至少被执行一次。
- do-while 循环本身被看成是一条语句，所以 while(exp) 后需要一个分号终止符。

```c
#include <stdio.h>
// 使用 do-while 语句对 1 到 100 累加求和
int main()
{
    int sum = 0;            // 累加器清零
    int cnt = 1;            // 加数赋初值
    do {
        sum = sum + cnt;     // 累加
        cnt++;
    } while (cnt<=100);
    printf("sum = %d", sum);
    return 0;
}
```

一般能用 while 语句解决的问题经过转化后也可以使用 do-while 语句来解决，反之亦然。通常情况下都会使用 while 语句来编写程序，不过在某些特定的情况下（比如至少要执行一次循环体），使用 do-while 语句会使程序更简洁。

```c
#include <stdio.h>
// 从键盘输入学生成绩，求平均值
/* 要求平均值，要求出学生人数，与总成绩。总成绩就是一个累加的过程，但人数如果需要手动输入，难免因为人工统计产生错误，所以一般由计算机自己来统计人数。分数为非负数，所以可以通过一个特殊值来表示输入完毕，这种方法在一些问题中很常用。
*/
int main()
{
    double score, sum=0;            // 累加器清零
    int cnt = 0;                    // 计数器赋初值
    printf("请依次输入学生成绩，负数表示输入结束：\n");

    // 使用 do-while 语句
    do {
        scanf("%lf", &score);
        if (score>0) {
            sum = sum + score;
            cnt++;
        }
    } while (score>=0);
    /* 使用 while 语句
    scanf("%lf", &score);           // 先读入一个供判断的值
    whiel (score>=0) {
        sum = sum + score;          // 以后类似的语句可以简写为：sum += score
        cnt++;
        scanf("%lf", &score);
    }
    */

    if (cnt>0) {
        printf("sum = %f\ncnt = %d\naverage = %f\n", sum, cnt, sum/cnt);
    }
    return 0;
}
```

```c
#include <stdio.h>
// 求两个正整数的最大公约数和最小公倍数
// 输入两个整数m n，利用辗转相除法求得最大公约数q，最小公倍数即为m/q*n
int main()
{
    int m, n, sm, sn;
    printf("请输入两个正整数：\n");
    scanf("%d %d", &m, &n);
    sm = m; sn = n;             // 保存两个整数

    // 辗转相除求最大公约数
    while (n) {
        r = m%n;
        m = n;
        n = r;
    } // 循环结束，最大公约数存储在 m 中
    printf("%d和%d的最大公约数是：%d\n", sm, sn, m);
```

```
    printf("%d和%d的最小公倍数是：%d\n", sm, sn, sm/m*sn);
    return 0;
}
```

前面的 break 语句与 continue 语句在 do-while 语句中同样适用。

6.1.4 for语句

C 语言中还有另外一种当型循环结构：for 循环。for 语句使用最为灵活，不仅适用于循环次数已经确定的情况，也适用于循环次数不确定而只给出循环条件结束的情况，熟练使用后几乎很少使用前两种，for 循环完全可以独挑大梁，但也要根据具体情况。

for 语句的一般形式为：

```
for (exp1, exp2, exp3) {
    stat;
}
/*
for (循环变量赋值;循环条件;循环变量改变值) {
    语句;
}*/
```

其中，

- exp1 是循环初值表达式。它在第一次循环开始前计算，且仅计算一次，其作用是给循环控制变量赋初值。
- exp2 是循环条件测试表达式。它在每次循环即将开始前计算，以决定是否进入循环，因此，正常情况下，改表达式决定了循环的次数。
- exp3 是循环控制变量调整表达式。它在每次循环结束时计算，以更新循环控制变量的值。
- stat 是循环体。可以是简单语句或复合语句。当有多条语句时，必须使用大括号括起，最简单的循环体只有一个空语句。

for 循环的执行过程：首先计算 exp1 ，然后再计算 exp2 ，若 exp2 的值为真，则执行循环体；否则，退出 for 循环，执行 for 循环后面的语句。如果执行了循环体，则循环体每执行完一次，都要重新计算 exp3 ，然后计算 exp2 ，依次循环，直至 exp2 的值为假。

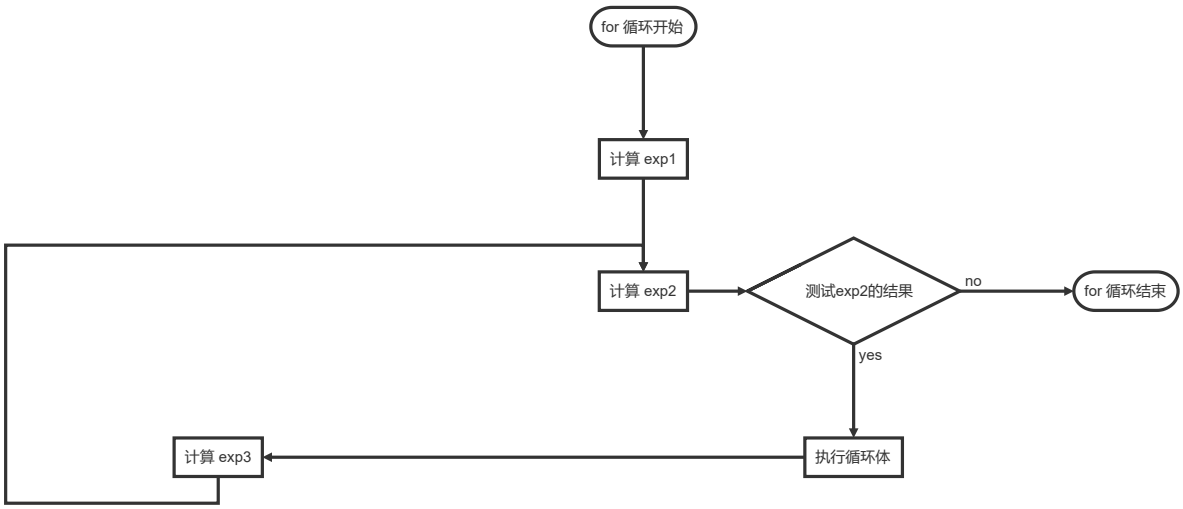

图 6 - 5 for 语句执行流程

```c
#include <stdio.h>
// 利用 for 语句对 1 到 100 累加求和
int main()
{
    int sum = 0;
    int cnt;
    for (cnt=1;cnt<=100;cnt++) {
        sum += cnt;
    }
    printf("sum = %d", sum);
    return 0;
}
```

显然，使用 for 语句比前两种方法更为简单方便，而且在 for 语句中不容易忽略对循环变量值的修改，犯错几率会小一点。但在使用 for 语句时，也要注意：

- for 语句中的 exp1 可以省略，此时应在循环前完成赋初值，但 exp1 可省，分号不能省。如 for(;cnt<=100;cnt++)
- 如果 exp2 省略，即不判断循环条件，这将无限循环下去，形成永真式，相当于 while (1) {;}
- exp3 也可以省略，但此时应另设方法保证循环能正常结束，如 for(cnt=1;cnt<=100;) {cnt++;}
- 如果同时省略 exp1 与 exp3 ，只给出 exp2 ，那么 for 循环结构将与 while 循环的结构极其相似。
- 3个表达式都省略，如 for (;;) {语句;} 这就相当于 while (1) {语句;}，除非有跳出循环的方式，否则构成死循环。

- exp1 与 exp3 可以是一个简单的表达式，也可以是逗号表达式，即包含多个赋值或改变值；同时 exp2 也可以是一个复杂的逻辑表达式，与前两种循环的 exp 限定一致。
- 同时可以把循环体和一些与循环控制无关的操作作为 exp1 或 exp3 出现，这样可以使程序短小简洁。但也不要过分的利用这一点，否则会使 for 语句显得杂乱，降低可读性。

```c
#include <stdio.h>
// 求 n! = 1*2*3*……*n，设 n=10
int main()
{
    int product = 1;            // 累乘器赋初值
    int cnt;
    for (cnt=1;cnt<=10;cnt++) {
        product *= cnt;
    }
    printf("10! = %d", product);
    // 要注意 int 型变量的范围，13以上的阶乘结果就会产生溢出，如果要求一个较大整数的阶乘结果就要使用数组来存储数据
    return 0;
}
```

```c
#include <stdio.h>
// 求斐波那契（Fibonacci）数列的前 30 项
int main()
{
    int F1=1, F2=1, F3, i;
    printf("%8d\n%8d\n", F1, F2);       // 输出前两位
    for (i=3;i<=30;i++) {
        F3 = F1 + F2;
        printf("%8d\n", F3);
        // 开始迭代，保存最近的两项
        F1 = F2;
        F2 = F3;
    }
    // 斐波那契数列的增长速度很快，到47项就会超出 int 型变量的存储范围
    return 0;
}
```

在 for 语句中也可以使用 break 语句或 continue 语句来控制程序流程。但使用这两种方式会降低程序可读性，如果没有注释说明，很容易错误理解程序，所以很少在 for 循环中使用到，但也不是绝对的视情况而定。

6.1.5 循环嵌套

与选择结构类似，在循环体再中添加循环语句即形成循环嵌套。3 种循环结构都可以相互嵌套，但要注意的是各个循环必须完整，相互之间绝不允许交叉。

C 语言没有规定最多嵌套层数，但如果嵌套太多，会影响程序的可读性，所以嵌套层次最好不要超过 3 层，如果有必要嵌套多层，可将内层循环另外写成函数供外层循环调用。还要注意内层循环的次数，外层循环每执行一次，内层循环都要循环完毕，也就是说，如果外层循环要执行n次，内层循环要执行 m 次，则内层循环中的语句实际上要执行 n*m 次。

```c
#include <stdio.h>
// 依次输出1、2、3、……、100，每行10个数字，共10行。
// 其实可以使用一层循环来实现，但用双层循环嵌套更为直观
int main()
```

```c
{
    int i, j, cnt;

    for (cnt=i=1;i<=10;i++) {        // 外层循环，控制行数
        for (j=1;j<=10;j++) {        // 内层循环，控制每行输出的数字数目
            printf("%4d", cnt);
            cnt++;
        }
        printf("\n");               // 每 10 个数字输出后需要换行
    }

    /* 但其实可以直接使用i和j来输出数字，而不使用另外的变量cnt
    for (i=0;i<10;i++) {
        for (j=10*i+1;j<=10*(i+1);j++) {
            printf("%4d", j);
        }
        printf("\n");
    }
    */ // 这种方式比前面的会简短一些，但相比之下也会更难想到，但这种技巧十分好用，希望牢记
    return 0;
}
```

```c
#include <stdio.h>
/* 输出一个如下所示的直角三角形（行数可以增多）
*
**
***
****
*****                              */
int main()
{
    const int N = 5;             // 输出 5 行
    int i, j;
    for (i=1;1<=N;i++) {         // 外层循环，控制输出5行
        for (j=1;j<=N;j++) {     // 内存循环，控制每行输出 * 的数目
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

```c
#include <stdio.h>
/* 输出一个如下所示的等腰三角形（行数可以增多）
    *
   ***
  *****
 *******
*********                            */
int main()
{
    const int N = 5;             // 输出 5 行
    int i, j;
    for (i=1;1<=N;i++) {         // 外层循环，控制输出5行
        for (j=1;j<=N-1;j++) {       // 内存循环，控制 * 前的空格
```

```
            printf(" ");
        }
        for (j=1;j<=2*i-1;j++) {    // 内存循环，控制每行输出 * 的数目
            printf("*");
        }
        printf("\n");
    }
    // 重点是理清行数与每行打印内容的关系
    return 0;
}
```

```
#include <stdio.h>
// 打印九九乘法表
int main()
{
    int i, j;
    for (i=1;i<=9;i++) {
        for (j=1;j<=i;j++) {
            printf("%d * %d = %-3d\t", j, i, i*j);
        }
        printf("\n");
    }
    return 0;
}
```

下图为程序运行结果：

```
1 * 1 = 1
1 * 2 = 2       2 * 2 = 4
1 * 3 = 3       2 * 3 = 6       3 * 3 = 9
1 * 4 = 4       2 * 4 = 8       3 * 4 = 12      4 * 4 = 16
1 * 5 = 5       2 * 5 = 10      3 * 5 = 15      4 * 5 = 20      5 * 5 = 25
1 * 6 = 6       2 * 6 = 12      3 * 6 = 18      4 * 6 = 24      5 * 6 = 30      6 * 6 = 36
1 * 7 = 7       2 * 7 = 14      3 * 7 = 21      4 * 7 = 28      5 * 7 = 35      6 * 7 = 42      7 * 7 = 49
1 * 8 = 8       2 * 8 = 16      3 * 8 = 24      4 * 8 = 32      5 * 8 = 40      6 * 8 = 48      7 * 8 = 56      8 * 8 = 64
1 * 9 = 9       2 * 9 = 18      3 * 9 = 27      4 * 9 = 36      5 * 9 = 45      6 * 9 = 54      7 * 9 = 63      8 * 9 = 72      9 * 9 = 81

Process returned 0 (0x0)   execution time : 0.056 s
Press any key to continue.
```