第5章 循环结构

本章学习目标

- 熟练掌握循环结构的设计
- 熟练掌握三种循环结构的使用场景及相互等价转换
- 掌握循环结构中 continue 和 break 的区别
- 掌握循环的嵌套设计

C 语言中，把重复执行一组“相同”或“相似”操作的流程结构称为循环，该组“相同”或“相似”的语句集合被称为循环体。C 语言中支持 while、do-while、for 等三种形式的循环语句。

本章首先简要介绍三种基本的循环语句，接着介绍循环流程跳转的两种语句 break 和 continue 及其区别，最后重点介绍循环的嵌套结构。

5.1 while 循环

当循环体中的语句多于一条时，要用{}把这些语句括起来形成一条复合语句，如下所示。

```
while(Exp_cntrl)
{
    Statement_1;
    Statement_2;
    …
}
```

当循环体为一条简单语句时，可以省略{}，即：

```
while(Exp_cntrl)
    Simple_Statement;              //循环体
```

while 循环的执行流程：

首先判断循环控制表达式 Exp_cntrl 的值，当该表达式的值为逻辑真(非 0)时，会一直执行循环体，直到表达式的值为逻辑假(0)时，结束循环。while 循环流程图如图 5-1 所示。

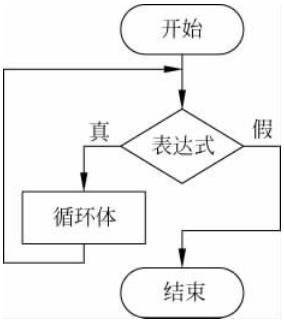

图 5-1 while 循环流程图

通常把循环控制表达式 Exp_cntrl 中含有的变量，称为循环控制变量。为了避免程序陷入死循环，必须要有能改变循环

控制变量的语句，使循环控制表达式 Exp_cntrl 的值趋于逻辑假，以便使循环趋于终止。

例 1 统计输出 100 以内的所有奇数之和。

【分析】

通过分析，本题是重复执行“把 100 以内的当前奇数 1、3、5、7，…累加求和”的相似操作，故采用循环结构。循环算法的关键是要确定循环条件表达式和循环体。

循环控制变量及初始条件确定：由题意可知，奇数 i 作为循环控制变量，初值为第一个奇数，即 i=1。其他变量初始值：求和变量 sum=0。

循环条件表达式的确定：循环控制变量 i 为[1,100]间的奇数。故循环条件表达式为 i<=100。

循环体确定：该题循环体中包含以下两部分操作。

（1）把当前奇数变量 i 累加到求和变量 sum 中，即 sum+=i;

（2）为计算当前奇数的下一个奇数做准备，也就是控制变量的增量部分，即 i+=2。

流程图如图 5-2 所示。

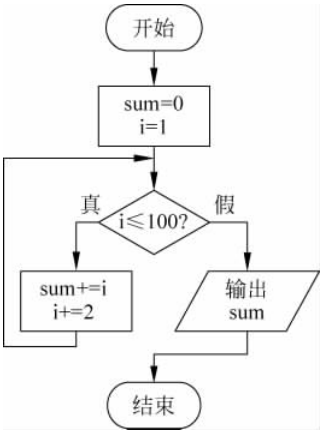

图 5-2 例 1 流程图

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int sum=0,i=1;                  //i 初始为第一个素数
    while(i<=100)                   //循环执行的判断条件
    {
        sum+=i;
        i+=2;                       //控制变量的增量
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

【运行结果】

```
sum=2500
```

【说明】

必须在零的基础上进行累加，故 sum 需要初始化为 0，否则将是无意义的随机值。循环控制条件不必刻意去思考最后一个奇数是否包含 100，让程序根据奇数的定义及相邻奇数的差值自行计算确定 100 以内的最后一个奇数。

例 2 计算输出 1-3+5-7+…-99 的值。

【分析】

通过分析，本题是重复执行“把当前数据项 item，如 1、-3、5、-7，…累加到求和变量 s 上”的相似操作，故采用循环结构。循环算法的关键是要确定循环条件表达式和循环体

语句。

每个数据项 item 由符号位 sign 和数值位 n 两部分组成。

循环控制变量及初始条件确定：由题意可知，数据项的数值位 n 可以作为该题的循环控制变量，初值为 n=1。其他变量初始值：求和变量 s=0，符号位 sign。由于第一个数据项为 1 即+1，故符号位初始为 sign=1。

循环条件表达式的确定：循环控制变量 n 的起止值，起始值为 1，终止值为 99。故循环条件表达式为 n<=99。

循环体确定：该题的循环体语句包括以下三部分操作。

(1) 组建当前数据项 item(由符号位 sign 和数值位 n 组成)，即 item=sign* n。

(2) 并把各个当前数据项累加到求和变量 s 上，即 s+=item。

(3) 然后改变下一个数据项的符号位 sign 及数值位 n，符号位与前一项相反，即 sign*=-1，数值位 n 的改变也就是循环控制变量的增量部分，比前一项大 2，即 n+=2。

该算法的流程图表示如图 5-3 所示。

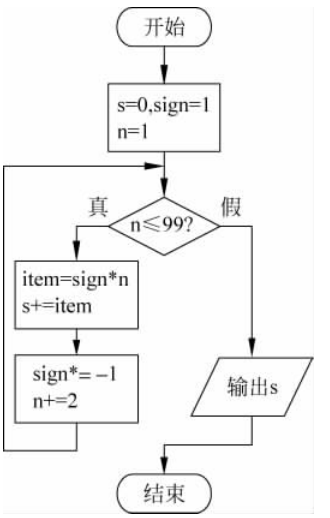

图 5-3　例 2 流程图

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int s=0,sign=1,n=1,item;
    while(n<=99)
    {
        item=sign*n;                    //组建当前项
        s+=item;                        //累加当前项
        sign*=-1;                       //改变下一项的符号位
        n+=2;                           //改变下一项的数值位
    }
    printf("sum=%d\n",s);
    return 0;
}
```

【运行结果】

```
sum=-50
```

例 3　任意输入一个十进制正整数，把其“反序”后输出(若输入：1234，则输出：4321)。

【分析】

经分析本题涉及两个主要操作：①把原数据从最低位到最高位逐位分离，如 4、3、2、1。②按照分离的顺序，用分离出的数字组成新的十进制整数 4321。

(1) 把数据 n=1234 从低位到高位逐位分离的方法和步骤如下。

① 分离个位数字：4。

用该数值 n(1234)除以 10 取余，即 n%10，可得个位数 4，然后用该数值 n(1234)除以 10 取整，即 n=1234/10=123。

② 分离十位数字：3。

用该数值 n(123)除以 10 取余，即 n%10，可得个位数 3，然后用该数值 n(123)除以 10 取整，即 n=123/10=12。

③ 分离百位数字：2。

用该数值 n(12)除以 10 取余，即 n%10，可得个位数 2，然后用该数值 n(12)除以 10 取整，即 n=12/10=1。

④ 分离千位数字：1。

用该数值 n(1)除以 10 取余，即 n%10，可得个位数 1，然后用该数值 n(1)除以 10 取整，即 n=1/10=0；即当 n=0 时，原数据的各位数字均已分离出来。

分析可知，上述各位数字的分离过程是重复执行 t=n%10;和 n=n/10;两条语句，直到 n=0 为止。故可采用 while 循环结构，代码框架如下。

```
while(n)
{
    t=n%10;                    //t:当前 n 分离出的低位数字
    //…
    n/=10;                     //去除已分离出的当前低位后的数值，为下一次分离次低位做准备
}
```

(2) 把各位数字按分离出的顺序组成一个新的十进制整数：先分离出的位(原数据的低位)作为新十进制数的高位，后分离出的位(原数据的高位)作为新十进制数的低位。

若依次用逐位分离出来的 4、3、2、1 组成一个新的十进制整数 m，则组建 m 的步骤如下。

① 设新组成的十进制数 m 初始为 0。

② m(0)扩大 10 倍，即原 m 的各位均向左(高位)移动一位，再加上刚分离出的数位 4，即 m=m×10+4=0×10+4=0+4=4。

③ m(4)扩大 10 倍，即原 m 的各位均向左(高位)移动一位，再加上刚分离出的数位 3，即 m=m×10+3=4×10+3=40+3=43。

④ m(43)扩大 10 倍，即原 m 的各位均向左(高位)移动一位，再加上刚分离出的数位 2，即 m=m×10+2=43×10+2=430+2=432。

⑤ m(432)扩大 10 倍，即原 m 的各位均向左(高位)移动一位，再加上刚分离出的数位 1，即 m=m×10+1=432×10+1=4320+1=4321。

完善上述代码框架如下。

```
int t,m=0;
while(n)
{
    t=n%10;                    //分离 n 的当前低位数字
    m=m*10+t;                  //用原 m 的各位作为高位，刚分离出的 t 作为低位，组成新的 m
    n/=10;                     //去除已分离出的当前低位后的数值，为下一次分离次低位做准备
}
```

【参考代码】

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

```
{
    int n,t,m=0;
    printf("输入十进制整数:");
    scanf("%d",&n);
    while(n)
    {
        t=n%10;
        m=m*10+t;
        n/=10;
    }
    printf("对应反序后数字:%d\n",m);
    return 0;
}
```

【运行结果】

```
输入十进制整数:1234
对应反序后数字:4321
```

【复习思考题】

1. 输入一个 n 值,求算术表达式 1+1/2+1/3+…+1/n 的值。

2. 输入一个十进制正整数,计算并输出该整数是几位数。如 123 是 3 位数,1000 是 4 位数。

5.2 do-while 循环

do-while 循环的格式如下。

```
do
{
    Statement _1;
    Statement _2;
    …
}while(Exp_cntrl);                    //分号不可丢
```

当循环体为一条简单语句时,可以省略{},即:

```
do
    Simple_Statement;                 //循环体
while(Exp_cntrl);
```

注意:在 do-while 结构中,while 括号后的分号不能丢。

do-while 循环的执行流程:首先无条件地执行一次循环体,然后再根据循环控制表达式的值来判断是否继续执行循环体。若为真,则继续执行;若为假,则停止执行,退出 do-while 循环。即 do-while 循环至少执行一次循环体。

do-while 循环和 while 循环的主要差别是:前者至少执行一次循环体,后者有可能一次也不执行循环体。

do-while 循环的执行流程图，如图 5-4 所示。

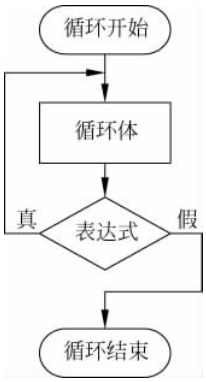

图 5-4　do-while 循环流程图

do-while 循环主要用在一直进行尝试性的操作，直到满足条件为止的情景。

例 4　编程实现猜数字游戏，假设谜底为 0~10 的整数，猜谜者每次输入一个整数，直到猜对为止。

【分析】

本题属于先输入所猜数字才能判断是否猜中，如果猜中，游戏结束，如果没猜中，继续猜，直到猜中为止。故该题符合 do-while 循环的使用场景。

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int pwd=7,gs;                    //pwd:谜底
    printf("\tGames Begin\n");
    do
    {
        printf("Please guess(0~10):");
        scanf("%d",&gs);
    }while(gs!=pwd);
    printf("\tSucceed!\n");
    printf("\tGames Over\n");
    return 0;
}
```

【运行结果】

```
      Games Begin
Please guess(0~10):3
Please guess(0~10):5
Please guess(0~10):8
Please guess(0~10):7
      Succeed!
      Games Over
```

【复习思考题】

1. 阐述 while 循环和 do-while 循环的差异。
2. 理解 while 循环及 do-while 循环结构的应用场景。
3. 分析以下程序输出结果。

```
int x=4;
do
{
    printf("%d\n",x-=3);
}while(!(--x));
```

5.3 while 和 do-while 的关系

5.3.1 while 和 do-while 的等价关系

在多数情况下，while 循环和 do-while 循环是等价的，如下例所示。

例 5 计算表达式 1-1/2+1/3-1/4+1/5-…-1/100 的值。

【分析 1】

通过观察可以发现，该表达式是把分母从 1 开始到 100 为止的所有数据项：1、-1/2、1/3、…、-1/100 累加求和。也就是说先判断分母是否小于等于 100，如果是，再组建该项，并把该项累加到求和变量上。符合循环条件前置的特点，故可选择 while 循环实现。

【参考代码 1】

```
#include<stdio.h>
int main(void)
{
    double s=0.0,item;
    int sign=1,n=1;                 //n：分母
    while(n<=100)
    {
        item=sign*(1.0/n);          //组建当前项，注意 1.0 的作用
        s+=item;                    //累加当前项
        sign*=-1;                   //改变下一项的符号位
        n++;                        //改变下一项的分母
    }
    printf("sum=%lf\n",s);          //double 的格式控制符为%lf
    return 0;
}
```

【运行结果 1】

```
sum=0.688 172
```

【分析 2】

由于事先知道求和变量 s 中至少包含一项 1，故第一次累加的分母判断条件可以去掉，从第二项开始，首先判断分母是否小于等于 100，再组项，然后累加。故该例子也符合循环条件后置的情况，所以本例也可以使用 do-while 循环，实现求该表达式的值。

【参考代码 2】

```
#include<stdio.h>
int main(void)
{
    double s=0.0,item;
    int sign=1,n=1;                 //n:分母
    do
    {
        item=sign*(1.0/n);          //组建当前项，注意 1.0 的作用
        s+=item;
```

```
        sign*=-1;
        n++;
    }while(n<=100);                //勿漏分号
    printf("sum=%lf\n",s);
    return 0;
}
```

【运行结果 2】

```
sum=0.688 172
```

【说明】

(1) 每一项的组建均是：分数值(分子与分母相除的结果)与符号位相乘的结果，尽管在本例中写成 term=sign*1.0/n;同样能得到正确结果，但不提倡这种写法。建议分数值表示部分显式加上括号，即：item=sign*(1.0/n);这是一种规范的写法，这样可避免因编译器的差异而造成结果不确定的情况，即**增强了代码的可移植性**。

(2) 参考代码 item=sign*(1.0/n);中的 1.0 如果误写成 1，即：item=sign*(1/n);则输出错误结果：sum=1.000 000。原因是除了第一项 n 为 1 时，1/n=1 外，其余当 n≥2 时，1/n 分子、分母同为整数值，结果为取整，故从第二项开始每一项的结果均为 0。

5.3.2 while 和 do-while 的不等关系

并不是所有的 while 循环都可等价替换为 do-while 循环结构。当 while 循环第一次循环条件就不满足时，此时不能把该 while 循环转换为 do-while 循环。如例 6 所示。

例 6 分析如下两段代码的输出结果，总结 while 循环和 do-while 循环的差异。

【参考代码 1】

```
#include<stdio.h>
int main(void)
{
    int s=0,i=15;
    while(i<=10)
    {
        s+=i;
        i++;
    }
    printf("s=%d\n",s);
    return 0;
}
```

【参考代码 2】

```
#include<stdio.h>
int main(void)
{
    int s=0,i=15;
    do
    {
        s+=i;
```

```
        i++;
    }while(i<=10);
    printf("s=%d\n",s);
    return 0;
}
```

【分析】

【参考代码 1】 使用 while 循环结构，循环判断条件前置，先判断 i 是否满足小于等于 10 时，如果满足，则把 i 累加到 s 上；否则，循环结束。本例中 i 初始为 15，不满足 i 小于等于 10，故循环体一次也不执行。s 为 0。

【参考代码 2】 使用 do-while 循环结构，循环判断条件后置，先无条件执行一次循环体，即先把 i 的初始值 15 累加到 s 上，i 自增 1 变为 16，然后判断 i 的值 16 是否小于等于 10，结果为假，故循环终止。s 的值为 15。

【运行结果 1】

```
s=0
```

【运行结果 2】

```
s=15
```

5.4 for 循环

for 循环的格式如下。

```
for(Exp_init;Exp_cntrl;Exp_incr)
{
    Statement _1;
    Statement _2;
    …
}
```

当循环体为一条简单语句时，{}可以省略，即：

```
for(Exp_init;Exp_cntrl;Exp_incr)
    Simple_Statement;            //循环体
```

关键字 for 后的()内含有三个表达式，用两个分号隔开。Exp_init 为初始化表达式，主要用于对循环控制变量做初始化操作，仅执行一次；Exp_cntrl 为循环控制表达式，如果该表达式的值为真，则执行循环体，如果为假，则不再执行循环体，循环结束。Exp_incr 为增量表达式，一般为修改循环控制变量的表达式。

for 循环的执行流程如下。

第 1 步：先执行初始化表达式 Exp_init。

第 2 步：判断循环控制表达式 Exp_cntrl 的值，若为真(非 0)，则执行循环体。然后接着执行第 3 步；若为假(0)，则结束整个循环。即跳转到第 4 步，继续执行循环结构后面的语句。

第 3 步：执行增量表达式 Exp_incr，然后跳转到第 2 步继续往下执行。

第 4 步：循环结束，继续执行循环结构后面的语句。

for 循环流程图如图 5-5 所示。

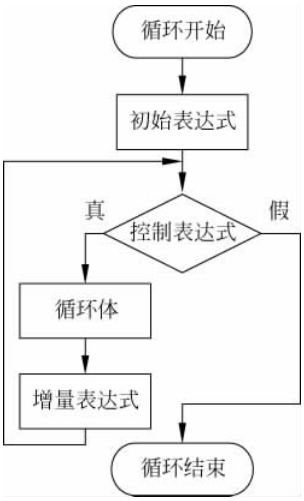

图 5-5　for 循环流程图

for 循环结构中的三个表达式中可以省略一个或者两个，甚至三个均省略，但两个分号均不能省略，当三个表达式全部省略时表示无限循环，或称为“死循环”。例如：

```
for(;;)                                    //正确，表示无限循环
    Simple_Statement;                      //循环体
```

例 7　计算输出 1~100 以内的所有偶数之和，使用 for 循环结构实现。

【分析】

定义求和变量 s，初始为 0，本题是重复执行把 100 以内的每个偶数都累加到 s 上，故使用循环结构。循环体为把当前偶数 i 累加到 s 上这一条简单语句，即 s+=i，故可省略循环体起止标志的一对大括号。

若采用 for 循环结构，则表达式 1 为循环控制变量 i 的初值表达式，即第一个偶数 2，i=2;。表达式 2 为循环控制表达式，只要该偶数小于等于 100 均可以累加到 s 上，即循环控制表达式为 i<=100;。表达式 3 为循环控制变量的增量表达式，即在当前偶数的基础上加 2 为下一个偶数，故循环控制变量 i 的增量表达式为 i+=2;。

【参考代码】

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

【运行结果】

```
sum=2550
```

【说明】

(1) 在 for 循环结构的表达式 1 中，可以同时对多个变量做初始化操作，中间用逗号把各个赋值操作隔开，如下所示。

```
int s,i;
for(s=0,i=2;i<=100;i+=2)          //正确，但不提倡，见说明(2)
    s+=i;
```

(2) 为了使 for 循环结构清晰明了，建议在 for 循环结构的各个表达式中，都是仅对循环控制变量的判断或其他操作。本例中的循环控制变量即表达式 2 中涉及的变量只有 i，故在表达式 1 中，最好也仅为 i 赋初值，故不提倡说明(1)中的写法。

(3) 把表达式1中对循环控制变量赋初值的操作提前,即放到for循环前面的写法也是不提倡的,因为这样失去了for循环设计为三个表达式的意义。例如:

```
int s=0,i=2;
for(;i<=100;i+=2)                    //正确,但不提倡
    s+=i;
```

(4) 把表达式3的操作,移到循环体里的下部,如下形式的写法,虽然正确,但也是不提倡的。

```
int s=0,i;
for(i=2;i<=100;)                     //正确,但不提倡
{
    s+=i;
    i+=2;
}
```

例8 打印输出所有的"水仙花数"。所谓"水仙花数"是指一个三位数,其各位数字的立方和等于该数本身。例如,153是一个"水仙花数",因为:$1^3+5^3+3^3=153$。

【分析】

(1) 由于"水仙花数"为三位数,从100~999每个三位数都进行验证,如果是则输出,所以循环控制变量n初值为100,循环控制表达式为n<=999,循环控制变量的增量表达式为n++。

如何判断每个n是不是水仙花数?先分离n的个位、十位、百位等数字。然后判断各位数字的立方和是否为n本身,若是则说明n是"水仙花数",输出n。故程序框架如下。

```
for(n=100;n<=999;n++)
{
    //分离n的个位、十位、百位数字
    //判断各位数字立方和是否为n,若是,输出n
}
```

(2) 在循环体中,分离出每个数的个位、十位、百位,分离的方法不唯一,例如,分离153各位数的方法如下。

把153除以10取余即可得个位数字3,即n0=n%10。

把153除以10取整的结果15,再除以10取余即可分离出十位数5,即n1=n/10%10。

把153除以100取整即可得百位数字1,即n2=n/100。

(3) 判断各位数字立方和是否等于该数本身n。

求幂次方使用math.h头文件中的pow函数,如n^2=pow(n,2)、n^3=pow(n,3)等。故该判断条件为:if(pow(n0,3)+pow(n1,3)+pow(n2,3) == n)。

(4) 由于每个"水仙花数"均为三位数,故输出格式可选择占5位输出,可保证每位数之间有两位间隔。"%5d":右对齐,占5位输出。"%-5d":左对齐,占5位输出。

填充程序框架如下。

```
for(n=100;n<=999;n++)
{
    n0=n%10;                         //个位
```

```
    n1=n/10%10;                    //十位
    n2=n/100;                      //百位
    if(pow(n0,3)+pow(n1,3)+pow(n2,3) == n)
        printf("%-5d",n);          //左对齐输出,每个数据占 5 位宽
}
```

【参考代码】

```
#include<stdio.h>
#include<math.h>                   //pow()函数头文件
int main(void)
{
    int n,n0,n1,n2;
    printf("水仙花数:");
    for(n=100;n<=999;n++)
    {
        n0=n%10;                   //个位
        n1=n/10%10;                //十位
        n2=n/100;                  //百位
        if(pow(n0,3)+pow(n1,3)+pow(n2,3) == n)
            printf("%-5d",n);      //左对齐输出,每个数据占 5 位宽
    }
    return 0;
}
```

【运行结果】

```
水仙花数:153  370  371  407
```

5.5 循环的嵌套结构

在一个循环结构内,又嵌入另一个循环结构,称为循环结构的嵌套或多重循环,常用的为二重循环。通常把外层的循环称为外循环,内层的循环称为内循环。do-while 循环、while 循环、for 循环可以相互嵌套。

例 9 从键盘输入一个正整数 n,求 s=1+(1+2)+(1+2+3)+…+(1+2+3+…+n)的值。

【分析】

(1) 求和变量 s 可以看成 n 项之和,设当前项用 term 表示。第 1 项 term=1、第 2 项 term=(1+2)、第 3 项 term=(1+2+3),…、第 n 项 term=(1+2+3+…+n),把每一项 term 累加到求和变量 s 上,使用外层循环控制该操作,循环累加 n 次,设 i 为循环控制变量,则循环结构如下。

```
for(i=1;i<=n;i++)
{
    //计算第 i 项的 term 值
    s+=term;
}
```

以上循环控制变量 i 控制的循环称为外层循环,负责把 n 项的值累加到 s 上。

(2) 计算第 i 项的 term 值，term=(1+2+3+…+i)，即把 1、2、3、…、i 累加到 term 上(term 初始为 0)，设 j 为循环控制变量，则第(1)步中，“计算第 i 项的 term 值”的代码如下。

```
term=0;                              //每一项的初始值均为 0
for(j=1;j<=i;j++)
    term+=j;
```

以上循环控制变量 j 控制的循环称为内层循环，负责计算第 i 项的值。

(3) 把第(2)步代码嵌入到第(1)步的 for 循环中，即可得如下代码。

```
for(i=1;i<=n;i++)
{
    term=0;
    for(j=1;j<=i;j++)
        term+=j;
    s+=term;
}
```

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int s=0,n,i,j,term;
    printf("Input a Integer:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        term=0;
        for(j=1;j<=i;j++)
            term+=j;
        s+=term;
    }
    printf("sum=%d\n",s);
    return 0;
}
```

【运行结果】 假设某次运行输入 10，然后回车(↙)，输出结果如下。

```
Input a Integer:10↙
sum=220
```

例 10 编程输出如下矩阵：

```
1  2  3  4
2  4  6  8
3  6  9  12
```

【分析】

(1) 以上输出矩阵为三行，每一行都是执行相同的操作(输出该行的 4 列)，所以该程序要重复执行三次输出行的操作，设行的循环控制变量为 i。使用外层 for 循环控制行的输出。循环体内为输出该行的 4 列，然后换行。故程序框架如下。

```
for(i=1;i<=3;i++)
{
    //输出第 i 行的 4 列
    printf("\n");                    //输出完第 i 行的 4 列后换行
}
```

(2) 输出第 i 行的 4 列,也就是要重复执行 4 次输出列的操作,故也符合循环结构。设控制列的循环控制变量为 j,则输出 4 列的循环代码如下。

```
for(j=1;j<=4;j++)                    //循环输出 4 列
{
    //打印输出 i 行 j 列的值
}
```

由于该循环嵌入在控制行的外层循环中,故称该控制列的循环为内层循环。

(3) 把第(2)步中控制列的 for 循环结构填入第(1)步中控制行的外层循环中,即可得如下代码。

```
for(i=1;i<=3;i++)                    //控制行操作的外层循环
{
    for(j=1;j<=4;j++)                //控制列操作的内层循环
    {
        //输出第 i 行、第 j 列的值
    }
    printf("\n");                    //换行
}
```

(4) 分析第 i 行、第 j 列对应数值的数学规律,如表 5-1 所示。

表 5-1　第 i 行 j 列值的规律

i	j			
	1	2	3	4
1	1×1	1×2	1×3	1×4
2	2×1	2×2	2×3	2×4
3	3×1	3×2	3×3	3×4

由表 5-1 可得：i 行 j 列对应的值为 i×j。

可编写如下代码实现输出第 i 行、第 j 列的值。

```
printf("%-3d",i*j);                  //每个数值占三列,左对齐输出
```

(5) 把第(4)步中输出第 i 行、第 j 列值的代码代入第(3)步代码框中。可得如下代码。

```
for(i=1;i<=3;i++)
{
    for(j=1;j<=4;j++)
    {
        printf("%-3d",i*j);
    }
    printf("\n");
}
```

由于控制列的内层循环体中仅含一条简单语句，故可省略内层循环体起止标志的一对大括号。

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int i,j;
    for(i=1;i<=3;i++)
    {
        for(j=1;j<=4;j++)
            printf("%-3d",i*j);
        printf("\n");
    }
    return 0;
}
```

【说明】

(1) 循环嵌套程序设计的一般思路是先把外层循环的框架搭起来，确定循环体操作，如果不能写出，先在该位置加功能注释，由外而内逐步实现。

(2) 本题最容易出错的地方是输出换行的位置，如下所示将无法打印输出题目要求的矩阵形式。

```
for(i=1;i<=3;i++)
{
    for(j=1;j<=4;j++)
    {
        printf("%-3d",i*j);
        printf("\n");

    }
}
```

【复习思考题】

1. 从键盘输入大于等于 1 的正整数 n 值，求表达式 s=1!+2!+…+n!的值。

2. 打印九九乘法表，输出格式如下所示。

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

提示：

(1) 第 1 行有 1 列，第 2 行有 2 列，…，第 i 行有 i 列。

(2) 列对齐可采用输出完每列后输出一个制表符\t 位宽的空格。

5.6 执行流程跳转语句

控制程序流程跳转的通常有 goto、break、continue、return 等语句。本节主要讲解前三种流程跳转语句，return 语句通常用在被调函数执行结束后，返回调用者处的流程跳转，将在函数一章中讲解 return 语句。

5.6.1 goto 语句

goto 语句是一种无条件流程跳转语句，通常 goto 语句与 if 语句结合使用，当满足一定条件时，程序流程跳转到指定标号处，接着往下执行。

定义语句标识的格式如下。

```
语句标识：语句;
```

其中，“语句标识”可以是任一个合法的标识符，如 pos_1、pos_2、label_1、label_2 等都是合法的语句标识。

注意：语句标识后的冒号不能省略。

goto 语句的调用语法格式为：

```
goto 语句标号;
```

程序将从对应“语句标号”的代码处开始往下执行。

例 11 分析以下程序，了解 goto 语句的使用。

【程序代码】

```
#include<stdio.h>
int main(void)
{
    int n;
pos_1:
    printf("请输入一个正整数:");
    scanf("%d",&n);
    if(n<0)
    {
        printf("输入错误!\n");
        goto pos_1;
    }
    printf("成功输入正整数:%d\n",n);
    return 0;
}
```

【分析】

在上述程序代码中，有一个语句标号 pos_1。该程序的执行流程如下。

(1) pos_1 标号处。先提示用户“请输入一个正整数:”。

(2) 如果用户输入的是正整数，则提示“成功输入正整数：***”。执行 (4)。

(3) 如果用户输入的是负数,则进入循环体,提示“输入错误!”。程序执行流程跳转到 pos_1 处,即跳转到第(1)步,继续往下执行。

(4) 程序结束。

【运行结果】 假设某次运行,依次输入-2、-6、3 等数字,其运行结果如下。

```
请输入一个正整数:-2
输入错误!
请输入一个正整数:-6
输入错误!
请输入一个正整数:3
成功输入正整数:3
```

【说明】

(1) 在一般 C 程序开发环境如 VC++ 6.0 中,语句标号处的位置较正常代码突出,就是为了让其更明显,不要刻意和其他代码对齐。

(2) 通过上述执行流程及运行结果的分析,可以发现该例中使用 goto 跳转语句实现了循环的功能。故可以使用循环结构的代码替换该 goto 结构的代码。

【等价代码】

```
#include<stdio.h>
int main(void)
{
    int n;
    printf("请输入一个正整数:");
    scanf("%d",&n);
    while(n<0)
    {
        printf("输入错误!\n");
        printf("请输入一个正整数:");
        scanf("%d",&n);
    }
    printf("成功输入正整数:%d\n",n);
    return 0;
}
```

使用 goto 语句可能会造成程序层次不清晰,可读性差,故在实际编程中,应尽量少使用或避免使用 goto 语句。

5.6.2 break 语句

当执行到循环体中的 break 语句时,将终止 break 所在该层的循环,从该层循环体之后的语句开始继续执行。

break 的执行流程如下所示。

单重循环情况:选用 while 循环结构示意,do-while 循环、for 循环同样适用。

```
while(循环判断表达式)
{
    …
```

```
    if(条件表达式)
          break;
    循环体中 break 后的语句;
}
循环结构后的第 1 条语句;
循环结构后的第 2 条语句;
          …
```

【说明】

在循环体中，当满足一定条件执行到 break 语句时，终止 break 所在的该层循环，即“循环体中 break 后的语句”部分将不再被执行，程序执行流程转向从“循环结构后的第 1 条语句”处，开始继续往后执行。

双重循环情况：使用双重 for 循环嵌套结构示意，其他类型的循环嵌套组合同样适用。

```
for(;循环判断表达式 1;)
{
    …
    for(;循环判断表达式 2;)
    {
        …
        if(条件表达式)
            break;
        内层循环体中 break 后的语句;
    }
    内层循环结构后的第 1 条语句;
    内层循环结构后的第 2 条语句;
    …
}
```

【说明】

在内层循环中，当满足一定条件执行到 break 语句时，仅结束 break 所在内层循环的执行 (在本轮外层循环中，“内层循环体中 break 后的语句”部分不再被执行)，程序执行流程跳转到“内层循环结构后的第 1 条语句”所在的外层循环体处，开始继续执行。

例 12 分析以下程序输出结果，掌握 break 语句的使用方法。

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int s=0,i;
    for(i=1;i<=10;i++)
    {
        if(6==i)
            break;
        s+=i;
    }
    printf("sum=%d\n",s);
    return 0;
}
```

【分析】

当 i<6 时，均不执行 break 语句，可以将其忽略。即 i<6 时，等价于如下代码。

```
for(i=1;i<=10;i++)
{
    s+=i;
}
```

相当于执行了加法运算 s=1+2+3+4+5=15。

当 i=6 时，执行 break 语句，立即终止 break 语句所在的该层 for 循环，故 i=6 并没有累加到 s 上，接着从 for 循环结构后的第一条语句 printf("sum=%d\n",s);处开始继续执行。

【运行结果】

```
sum=15
```

例 13 打印输出如下矩阵。

```
1  2   3   4   5
2  4
3  6   9  12  15
4  8  12  16  20
```

【分析】

(1) 通过分析可知，如果第 2 行正常输出为 2 4 6 8 10，输出完整 4 行 5 列上述规律矩阵时的代码段如下。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++)
    {
        printf("%-3d",i*j);
    }
    printf("\n");
}
```

(2) 当执行到第 2 行第 3 列时，立即终止控制列输出的内层 for 循环，并换行输出。故在内层循环体中，输出第 i 行、第 j 列数值的语句之前，设置 break 条件语句。程序代码如下。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++)
    {
        if(i==2 && j==3)          //当执行到第 2 行第 3 列时，执行 break 语句
            break;
        printf("%-3d",i*j);
    }
    printf("\n");
}
```

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int i,j;
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=5;j++)
        {
            if(2==i && 3==j)
                break;
            printf("%-3d",i*j);
        }
        printf("\n");
    }
    return 0;
}
```

【说明】

(1) 表示“第 2 行第 3 列”时应该是第 2 行和第 3 列的逻辑与。

(2) 正确掌握设置 break 语句的位置,思考如果改变 break 语句的位置如以下代码段所示,分析输出结果。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++)
    {
        printf("%-3d",i*j);
        if(2==i && 3==j)
            break;
    }
    printf("\n");
}
```

5.6.3 continue 语句

在循环体中设置 continue 语句,同样可以改变循环的执行流程,只不过它不像 break 那样结束整个循环体,而是仅结束本次循环体的执行,提前进入下一次循环。

continue 语句的执行流程如下所示。

选用 for 循环结构示意。

```
for(初始表达式;循环判断表达式;增量表达式)
{
    …
    if(条件表达式)
        continue;
    循环体中 continue 后的所有语句;
}
```

【说明】

当在上述 for 循环中执行到 continue 语句时，本次循环体的执行流程将跳过“循环体中 continue 后的所有语句”，接着执行“增量表达式”部分，然后执行“循环判断表达式”，即提前进入下一次循环的准备工作。

例 14 分析以下程序输出结果，掌握 continue 语句的使用方法。

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int s=0,i;
    for(i=1;i<=10;i++)
    {
        if(6==i)
            continue;
        s+=i;
    }
    printf("sum=%d\n",s);
    return 0;
}
```

【分析】

(1) 当 i≠6 时，也就是 i≤5 时，忽略 continue 语句，相当于：

```
for(i=1;i<=10;i++)
{
    s+=i;
}
s=1+2+3+4+5
```

(2) 当 i=6 时，执行 continue 语句，本次循环(i=6)体中 continue 后的语句 s+=i; 将被忽略，接着执行增量表达式 i++，相当于 i=6 没有累加到 s 上。

(3) 后续的循环中 i 不再可能等于 6，故 continue 语句也将被忽略。故程序功能相当于把 1~10 中除 6 之外的 9 个数累加到 s 上。即 s=1+2+3+4+5+7+8+9+10=49。

【运行结果】

```
sum=49
```

例 15 编程输出如下形式的矩阵。

```
1  2   3   4   5
2  4       8   10
3  6   9   12  15
4  8   12  16  20
```

【分析】

通过分析可知，如果第 2 行正常输出为 2 4 6 8 10，输出完整 4 行 5 列上述规律矩阵时的代码段如下。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++)
    {
        printf("%d\t",i*j);        //每列数值后空制表符位宽,便于对齐
    }
    printf("\n");
}
```

执行到第 2 行第 3 列时,输出空格,而忽略输出该列对应数值 6,接着正常输出第 4 列、第 5 列,此处恰好符合 continue 语句的执行规律。故在内层循环体中,输出第 i 行、第 j 列数值的语句之前,设置 continue 条件语句。代码如下所示。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=5;j++)
    {
        if(2==i && 3==j)
        {
            printf("\t");          //此处输出制表符位宽的空格
            continue;
        }
        printf("%d\t",i*j);
    }
    printf("\n");
}
```

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int i,j;
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=5;j++)
        {
            if(2==i && 3==j)
            {
                printf("\t");
                continue;
            }
            printf("%d\t",i*j);
        }
        printf("\n");
    }
    return 0;
}
```

【说明】

复习运算符优先级,关系运算符 (==、!=、>、>=、<、<=) 的优先级高于逻辑运算符 (&&、||)

的优先级，故 if((2==i) && (3==j))等价于 if(2==i && 3==j)。

【复习思考题】

1. 以下编程试图实现计算输出 1+2+3+4+5+7+8+9+10 值的功能。试分析如下 while 循环结构的程序能否正常输出，指出存在的问题，并说明在使用 while 循环结构的情况下，如何修改使其输出正确结果。

```
#include<stdio.h>
int main(void)
{
    int s=0,i=1;
    while(i<=10)
    {
        if(6==i)
            continue;
        s+=i;
        i++;
    }
    printf("sum=%d\n",s);
    return 0;
}
```

2. 编程输出如下形式的矩阵。提示：综合使用 continue 和 break 语句。

```
1  2   3   4   5
2  4          10


4  8  12  16  20
```

5.7 综 合 举 例

例 16 某人摘了一堆桃子。第一天卖掉一半，吃了一个，第二天卖掉剩下的一半，又吃了一个，第三天、第四天，…，以此类推。第 6 天发现只剩下一个桃子。问此人共摘了多少个桃子，并打印输出每天初始的桃子个数。

【方法 1】

由当前天初始桃子个数推算出前一天初始桃子个数，当前天从第 6 天开始到第 2 天结束。

【分析 1】

由“第 6 天发现只剩下一个桃子”可知，第 6 天初始桃子的个数 n=1，根据每天即当前天初始桃子的个数推算出前一天初始桃子的个数，推算关系为：当前天初始个数=前一天初始个数/2-1，即前一天初始桃子的个数=(当前天初始桃子的个数+1)×2。

从当前天第 6 天即 d=6 开始，往前推算第 5 天、由第 5 天推算第 4 天、…、由第 2 天推算第 1 天初始桃子的个数。为重复做 5 次类似的推算操作，故采用循环结构。注意，由于第 1 天并没有前一天，也就没有做“卖掉一半，吃掉 1 个”的操作，故第 1 天不在循环操作中。

循环控制变量及初始条件确定：当前天 d 为循环控制变量，确定 d 的起止值，初始为第 6 天即 d=6，因为第一天不再有前一天，故终止为第 2 天即 d=2。当前天初始桃子的个数 n，

初值为第 6 天初始桃子的个数 1，即 n=1。

循环条件表达式的确定：d 的范围为[6,2]递减整数，故循环条件表达式为 d>=2。

循环体的确定：该题的循环体语句包括以下三部分操作。

（1）输出当前天初始桃子的个数，即：printf("第%d天，初始桃子的个数为：%d\n",d,n);。

（2）为计算前一个当前天也就是前一天初始桃子个数做准备。由当前天初始桃子的个数 n，计算前一个当前天也就是前一天初始桃子的个数为 (n+1)*2，即：n=(n+1)*2。

（3）当前天 d--;。

流程图如图 5-6 所示。

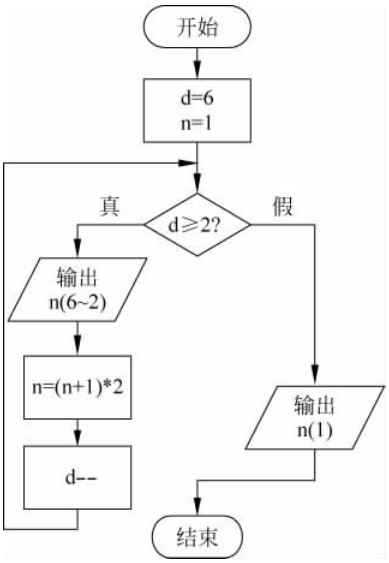

图 5-6　例 16 方法 1 流程图

【参考代码 1】

```
#include<stdio.h>
int main(void)
{
    int d=6,n=1;                    //当前天 d 从第 6 天开始，初始桃子个数 n 为 1
    while(d>=2)
    {
        printf("第%d天，初始桃子的个数为：%d\n",d,n);          //输出当前天初始桃子个数
        n=(n+1)*2;                  //计算前一天初始桃子个数
        d--;
    }
    printf("第%d天，初始桃子的个数为：%d\n",d,n);
    return 0;
}
```

【运行结果 1】

```
第 6 天，初始桃子的个数为：1
第 5 天，初始桃子的个数为：4
第 4 天，初始桃子的个数为：10
第 3 天，初始桃子的个数为：22
第 2 天，初始桃子的个数为：46
第 1 天，初始桃子的个数为：94
```

【方法 2】

由当前天剩余桃子个数推算出当前天初始桃子个数，当前天从第 5 天开始到第 1 天结束。

【分析 2】

由“第 6 天发现只剩下一个桃子”可知，第 5 天剩余桃子的个数 r=1，根据当前天剩余桃子的个数推算出当前天初始桃子的个数 n，推算关系为：当前天剩余个数=当前天初始个数/2-1，即当前天初始桃子的个数=(当前天剩余桃子的个数+1)×2。

从当前天第 5 天即 d=5 开始，由第 5 天剩余个数推算第 5 天初始个数、由第 4 天剩余个

数推算第 4 天初始个数、…、由第 1 天剩余个数推算第 1 天初始个数。为重复做 5 次类似的推算操作，故采用循环结构。注意，由于第 6 天并没有做“卖掉一半，吃掉 1 个”的操作，故第 6 天不在循环操作中。

循环控制变量及初始条件确定：当前天 d 为循环控制变量，确定 d 的起止值，d 初始为第 5 天即 d=5，终止为第 1 天即 d=1。当前天剩余桃子的个数 r，初值为第 5 天剩余桃子的个数 1，即 r=1。当前天初始桃子个数用 n 表示。

循环条件表达式的确定：d 的范围为[5,1]递减整数，故循环条件表达式为 d>=1。

循环体的确定：该题的循环体语句包括以下 4 部分操作。

（1）为计算当前天初始桃子个数 n，由当前天剩余桃子的个数 r，计算当前天初始桃子的个数为 (r+1)*2，即：n=(r+1)*2;。

（2）输出当前天初始桃子的个数，即：printf("第%d 天，初始桃子的个数为：%d\n",d,n);。注意：此处的 d 不包含第 6 天，需单独考虑。

（3）循环控制变量当前天 d 的增量，d--;。

（4）把当前天的初始个数 n 作为前一天的剩余个数 r，即 r=n;。

特殊情况：第 6 天的初始个数为第 5 天的剩余个数 r，故输出语句为：

```
printf("第%d天,初始桃子的个数为：%d\n",d+1,r);
```

【参考代码 2】

```
#include<stdio.h>
int main(void)
{
    int d=6-1,r=1,n;                    //r、n:均初始化为第 5 天的剩余个数、初始个数
    printf("第%d天,初始桃子的个数为：%d\n",d+1,r);    //第 6 天初始个数=第 5 天剩余个数
    while(d>=1)
    {
        n=(r+1)*2;                      //由剩余的 r 求当天初始为(r+1)*2
        printf("第%d天,初始桃子的个数为：%d\n",d,n);
        d--;
        r=n;                            //当前天初始个数 n,作为前一天剩余个数 r
    }
    return 0;
}
```

【运行结果 2】

```
第 6 天,初始桃子的个数为：1
第 5 天,初始桃子的个数为：4
第 4 天,初始桃子的个数为：10
第 3 天,初始桃子的个数为：22
第 2 天,初始桃子的个数为：46
第 1 天,初始桃子的个数为：94
```

【说明】

注意两种方法的区别，尤其是循环控制变量当前天 d 的起止范围。特殊天的处理：方法 1 中第 1 天，方法 2 中第 6 天的特殊处理。

例 17 只能被 1 和它本身整除的数，称为素数。从键盘输入一个正整数 n，判断并输出其是否为素数。使用 for 循环结构实现。

【分析】

(1) 根据素数定义的描述，如果用一个正整数 n 分别去除以 i：i 属于集合[2,n-1]，故为循环结构。若每一个 i 均不能整除，则说明该数为素数；只要有一个 i 能整除，则停止判断，即退出循环，故可以使用 break 语句。循环结构如下。

```
for(i=2;i<n;i++)
{
    if(n%i==0)
        break;
}
```

该循环体中只有一条 if 语句，故循环体起止标志的一对大括号可以省略。

(2) 上述循环若正常终止(即一直没有执行 break 语句)，则循环结束后，i 不再小于 n，如果循环结束后 i 依然小于 n，说明是执行了 break 语句使得循环提前终止的，肯定不是素数。否则是素数。

【参考代码】

```
#include<stdio.h>
int main(void)
{
    int n,i;
    printf("输入大于等于 2 的整数:");
    scanf("%d",&n);
    for(i=2;i<n;i++)
        if(n%i==0)
            break;
    if(i<n)
        printf("%d 不是素数.\n",n);
    else
        printf("%d 是素数.\n",n);
    return 0;
}
```

【运行结果】

```
输入大于等于 2 的整数:17
17 是素数.
```

例 18 打印输出如下图形。

```
   *
  ***
 *****
*******
```

【分析】

(1) 该图依次输出 4 行，故外层循环控制行输出，在每行内，先输出若干(可为 0)个空

格,然后输出若干个*,最后换行。故程序框架如下。

```
for(i=1;i<=4;i++)
{
    //输出 i 行空格的代码
    //输出 i 行*的代码
    printf("\n");
}
```

(2) i 行与该行空格数 j 的关系如表 5-2 所示。

表 5-2 行数和空格数的关系

第 i 行	1	2	3	4
空格个数 j	3	2	1	0

根据表 5-2 可得,行数 i 和空格个数 j 的关系:i+j=4,故 j=4-i,输出 i 行空格的代码如下。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=4-i;j++)
        printf("");
    //输出 i 行*的代码
    printf("\n");
}
```

(3) i 行与该行*符号个数 k 的关系如表 5-3 所示。

表 5-3 行数和*符号数的关系

第 i 行	1	2	3	4
*符号个数 k	1	3	5	7

根据表 5-3 可得,行数 i 和*号个数 k 的关系:k=2*i-1,输出 i 行*符号的代码如下。

```
for(i=1;i<=4;i++)
{
    for(j=1;j<=4-i;j++)
        printf("");
    for(k=1;k<=2*i-1;k++)
        printf("*");
    printf("\n");
}
```

【参考代码】

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

```
{
    int i,j,k;
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=4-i;j++)
            printf("");
        for(k=1;k<=2*i-1;k++)
            printf("*");
        printf("\n");
    }
    return 0;
}
```

小　　结

1. 本章主要知识点梳理

本章主要介绍了常见的三种循环结构：while 循环、do-while 循环和 for 循环。重点介绍了循环结构中的 break 和 continue 语句及循环的嵌套。本章知识点小结如表 5-4 所示。

表 5-4　本章主要知识点梳理

知　识　点	示　　例	说　　明

do-while 循环结构	do
	{
	　　//循环体
	}while(Exp_cntrl);
		首先无条件地执行一次循环体，然后再根据条件表达式的值来判断是否继续执行循环体，若为真，则继续执行；若为假，则停止执行，退出循环。即 do-while 循环至少执行一次循环体。一般循环控制变量的增量放在循环体中

while 循环结构	while(Exp_cntrl)
	{
	　　//循环体
	}
		当条件 Exp_cntrl 为逻辑真时，一直执行循环体，当条件为假时，循环结束。即 while 循环结构中，循环体有可能一次也没有执行。一般循环控制变量的增量放在循环体中

for 循环	for(Exp_init;Exp_cntrl;Exp_incr)
	{
	　　//循环体
	}
		与 while 循环功能相似，仅是三个表达式的位置不同而已

循环流程控制语句	break;退出该层循环。 continue;本次循环中，continue 后的语句不再执行，提前进入下一次循环条件的判断	goto 语句容易造成流程的混乱，不建议使用

2. 本章易错知识点

本章易错知识点如表 5-5 所示。

表 5-5　本章易错知识点

易错知识点	错误示例	说　明

while、do-while 缺少循环控制变量的增量	计算 1+3+5+…+99
	int i=1,s=0;
	while(i<=99)
	{
	　　s+=i;
	}
	或者
	do
	{
	　　s+=i;
	}while(i<=99);
	以上两种循环结构中均缺少循环控制变量的增量，为死循环
		循环结构必须具备三个表达式：循环控制变量的初值表达式、循环控制表达式、循环控制变量的增量表达式。缺一不可
		int i=1,s=0;
		while(i<=99)
		{
		　　s+=i;
		　　i+=2;
		}
		或者
		do
		{
		　　s+=i;
		　　i+=2;
		}

缺少分号	do {…}while(…) 编译时报语法错误	正确格式如下 do {…}while(…);

while 多余分号	例如：计算 2+4+6+…+100 的值
	int i=2,s=0;
	while(i<=100);
	{s+=i;i+=2;}
	该程序为死循环
		该 while 循环既无编译错误，也无运行时错误。只是把循环体当成空语句，且没有执行循环控制变量的增量，故为死循环

for 多余分号	例如：计算 2+4+6+…+100 的值
	int i,s=0;
	for(i=2;i<=100;i+=2);
	{s+=i;}
	运行输出结果为：s=102
	属于逻辑错误
		(1) 该 for 循环既无编译错误，也无运行时错误。仅是循环体为空，该循环含有 i 初值 2，含循环控制条件 i<=100，也含有循环控制变量 i 的增量，i+=2。故是完整意义上的循环。
		(2) 该 for 循环结束时，i 不再满足<=100，而等于 102，接着执行循环后的语句块中语句 s+=i;，故 s=102。属于逻辑错误

习　题

1. while 循环

(1) 语句 while (E);中的条件 E 等价于(　　)。

A. E==0　　B. E!=1　　C. E!=0　　D. ~E

(2) 分析以下程序的输出结果，并掌握循环控制变量的起止值。

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

```
{
    int n=0;
    while (n++<=2)
    {
        printf("%d\n",n);
    }
    printf("%d\n",n);
    return 0;
}
```

(3) 使用 while 循环编写求 n!的程序。

(4) 判断一个整数是否为对称数,如 123321 是对称数,而 12325 不是对称数。

提示:使用 while 循环依次把原整数从低位到高位分离出来,组成一个新的整数(原整数的低位作为新整数的高位),如果新整数与原整数相等,则该数为对称数。

(5) 输入一个长整型数 n,把 n 从低位到高位依次分离出来,用其中的奇数位依次组成一个新的十进制数 m,并输出 m 的值。如 n=123456789,则 m=97531。

2. do-while 循环

分析以下程序的输出结果。

```
#include<stdio.h>
int main(void)
{
    int x=3;
    do
    {
        printf("%d\t",x-=3);
    }while(!x);
    return 0;
}
```

3. while 和 do-while 的关系

以下关于 while 语句和 do-while 语句的描述中,错误的是(　　)。

A. while 语句和 do-while 语句都可以使一段程序重复执行多遍

B. while 语句和 do-while 语句都包含控制循环的表达式

C. while 语句和 do-while 语句都包含循环体

D. while 语句和 do-while 语句的循环体至少都会执行一次

4. for 循环

(1) 使用 for 循环打印输出 26 个小写字母'a'~'z',每行输出 13 个字母。

(2) 分别使用 while、do-while、for 循环实现计算输出 1+2+3+…+100。

(3) 分别使用 while、do-while、for 循环实现计算输出 1×2×3×…×10 即 10!。

(4) 分析以下程序段执行的次数,并输出结果。

```
int i;
for(i=3;i;)
    printf("%d",i--);
```

(5) 分析以下程序段是否正确,能否得到运行结果,如果有结果,是多少。

```
int i,s=0;
for(i=1;i<=100;i++);
{
    s+=i;
}
printf("s=%d\n",s);
```

5. 循环的嵌套结构

(1) 打印输出如下图形。

```
#########
 #######
  #####
   ###
    #
```

(2) 输出 100~200 以内的所有素数,每行输出 10 个。

6. 执行流程跳转语句

(1) 分析以下程序的输出结果。

```
#include <stdio.h>
int main(void)
{
    int i;
    for (i=4;i<=10;i++)
    {
        if (i%3==0)
            continue;
        printf("%d",i);
    }
    return 0;
}
```

(2) 打印输出如下矩阵(提示:综合使用 break 和 continue 语句)。

```
1  2   3   4   5
2           8  10

4  8  12  16  20
```

7. 综合举例

(1) 分析以下程序。

```
#include <stdio.h>
int main(void)
{
    int n=0;
    char c;
    while((c=getchar()) != '#')
    {
        if (c>='0' && c<='9')
```

```
            n=n*10+c-'0';
        }
        printf("value=%d\n", n);
        return 0;
    }
```

① 当运行以上程序时，输入 A2b0c1Y7#↙(回车符)，分析程序输出结果。

② 如果上述 while 循环条件误写成 while(c=getchar() != '#')，分析输出是否与原题相同，说明原因。(提示：从运算符优先级角度思考。)

(2) 分析以下程序的输出结果，并说明该程序代码中的 break 与改变循环执行流程的 break 的区别。

```
#include<stdio.h>
int main(void)
{
    int k=4,n=0;
    while(k>0)
    {
        switch(k)
        {
            case 1:
            case 4:n+=1;k--;break;
            default:n=1;k--;
            case 2:
            case 3: n+=2;k--;break;
        }
    }
    printf("%d\n%d\n",n,k);
    return 0;
}
```

(3) 分析以下程序的输出结果。代码中的 continue 可以去掉吗？

```
#include<stdio.h>
int main(void)
{
    int a,b=1;
    for(a=1;a<=15;++a)
    {
        if(b%3==1)
        {
            b+=3;
            continue;
        }
        if(b>=10)
            break;
    }
    printf("%d\n",a);
    printf("%d\n",b);
    return 0;
}
```