第3章 程序控制结构

本章要点

- 顺序控制。
- 选择控制及其嵌套。
- 多路选择控制。
- 循环控制及循环嵌套。
- 输入信息控制循环。

在学习了C++的数据类型、表达式和简单的输入/输出控制之后，就能够编写一些可以完成一定功能的程序了。在程序设计中，算法的实现都是由一系列控制结构完成的。在经典的结构化程序设计中，最基本的控制结构有顺序、选择和循环，它们是构成复杂算法的基础。

3.1 顺序控制结构

C++程序由若干语句构成，程序中的大部分语句都是按顺序执行的。顺序控制结构是系统预置的，除非特别指定，计算机总是按指令编写的顺序一条一条地执行，顺序控制的流程如图3-1所示。

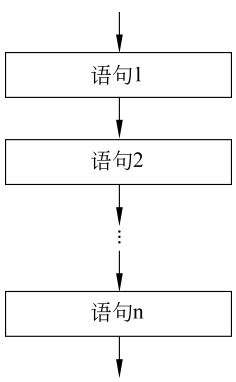

图3-1 顺序控制的流程

从图3-1可以看出，在顺序控制结构中，语句按先后顺序依次执行语句1，语句2，…，语句n。

例如，下面的程序代码将一个三位整数n的百位数、十位数和个位数分别取出，并分别赋值给变量i、j和k。

```
int n,i,j,k;                    //声明四个整型变量n、i、j和k
cout <<"输入一个三位整数:";     //屏幕显示输入提示信息
cin >> n;                       //键盘输入数据并存入n中
i = n/100;                      //获取n的百位数,并赋值给i
j = (n/10) % 10;                //获取n的十位数,并赋值给j
k = n % 10;                     //获取n的个位数,并赋值给k
```

这段程序代码是典型的顺序控制。假设输入n的值为567，则n/100的结果为整数5，所以i=5；这时n只是被使用，并没有被重新赋值，因此n仍为567，则n/10为56，56%10的结果为6，所以j=6；因为n仍为567，n%10的结果为7，所以k=7。

3.2 选择控制结构

设计程序的关键是使程序具有决策能力,C++提供了多种选择控制语句来支持程序选择决策,这些选择控制语句构成了适合不同情况下使用的选择控制结构。例如,当某一条件满足时,程序执行某一操作;条件不满足时,则执行另一个操作。又例如,如何根据用户选择的菜单项执行特定的程序代码等。

下面就从最简单的 if…else 语句开始,看看 C++如何使用各种不同的选择控制语句进行选择操作。

3.2.1 选择控制语句 if…else

使用 if…else 语句可以让程序根据测试表达式的值决定执行两条语句中的哪一条。这种语句对于二者选择其中之一很好用,其语法形式为:

```
if(测试表达式)    语句 1
else              语句 2
```

if…else 语句的执行顺序是:首先计算测试表达式的值,若测试表达式的值为 true,则执行语句 1;否则执行语句 2,执行流程如图 3-2(a)所示。

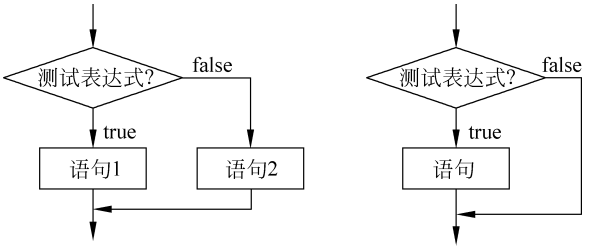

图 3-2 if…else 语句控制流程

例如,如下的 if…else 语句可以根据 x 的值是否大于或等于 0,实现分别给 y 赋值为 1 或－1 的功能。

```
if(x>=0)
    y=1;          //语句 1
else
    y=-1;         //语句 2
```

在使用 if…else 选择控制语句时,应注意以下两点。

(1) 其中的语句 1 和语句 2 可以是一条语句,也可以是由花括号({})括起来的语句序列,称为复合语句或语句块。

(2) 当语句 2 为空时,else 可以省略,构成 if 简单选择控制语句,其语法形式如下:

```
if(测试表达式)    语句
```

if 简单选择控制语句的执行顺序与 if…else 语句基本相同,只是当测试达式的值为

false 时，不执行任何语句，直接结束选择控制，如图 3-2(b)所示。

例如，以下的 if 简单选择控制语句可以判断变量 x 的值是否为偶数：

```
if(x % 2 == 0)
    cout <<"x 是偶数";
```

【例 3-1】 在两个数中找较大值。

问题分析：使用 if…else 语句可以很容易地实现在两个数中找较大值。假设有两个数 a 和 b，则测试表达式为 a＞b，程序代码如下：

```
#include < iostream >
using namespace std;
int main()
{
    int a,b,max;
    cout <<"Input a,b:";
    cin >> a >> b;
    if(a > b)
        max = a;
    else
        max = b;
    cout <<"The max is :"<< max << endl;
    return 0;
}
```

程序运行结果：

```
Input a,b:4    5
The max is :5
```

3.2.2 条件运算符(?:)代替 if…else 语句

C++中常用条件运算符(?:)代替 if…else 语句来实现选择控制。例如，用语句

```
max = a > b?a:b;
```

替换例 3-1 中的 if else 语句，即

```
if(a > b)
    max = a;
else
    max = b;
```

程序运行后，结果是相同的。

例 3-1 是在两个数中找较大值，如果是在三个数中找最大值，程序又应如何实现呢？程序代码如下：

```
#include < iostream >
using namespace std;
int main()
{
```

```
    int a,b,c,max;
    cout <<"Input a,b,c:";
    cin >> a >> b >> c;
/*  if(a > b)          max = a;
    else           max = b;
    if(c > max)     max = c;
 */ max = (a > b?a:b)> c?(a > b?a:b):c;                    //用?:运算符实现
    cout <<"The max is :"<< max << endl;
    return 0;
}
```

请读者注意程序中用条件运算符(?:)实现的语句:

```
max = (a > b?a:b)> c?(a > b?a:b):c;
```

与if…else语句相比,条件运算符"?:"更简洁,但对初学者来说并不容易理解。虽然,一般情况下条件运算符"?:"都可以代替if…else语句,但这两种方法之间也有区别。条件运算符"?:"生成一个表达式,因此是一个值,可以将其放入另一个更大的表达式中或将其赋给变量,在三个数中找最大值的程序中就将条件表达式a>b?a:b放入另一个更大的表达式中,并将更大的条件表达式(a>b?a:b)>c?(a>b?a:b):c的值赋给了变量max。

注意:上述程序的if…else语句采用了紧凑格式。使用C++语言的紧凑格式虽然可以简洁、灵活、方便地编写程序,但是使用紧凑格式要有度。例如:

```
if(a > b)max = a;
else     max = b;
if(c > max)   max = c;
```

或

```
cout <<"The max is :"
```

都是允许的。但以下代码是不允许的:

```
cout <<"The max
    is :"<< max << endl;
cout <<"The max is :"<   < max << endl;
```

初学者最好使用规范的格式书写程序,因为程序的可读性是非常重要的,真正的商业程序是绝对规范的。张三写的程序和李四写的程序格式应大致相同,各种标识符的命名规则一致,否则大家都看不懂其他人编写的程序。

规范的格式应包括长标识符命名、代码缩进和一对大括号范围不超过一屏幕等。

3.2.3 if…else语句的嵌套

在编程中,有许多问题是一次简单的判断所解决不了的,需要进行多次判断。这时需要**嵌套的if语句**,其形式有以下两种。

形式1:

```
if(测试表达式1)
    if(测试表达式2)          语句1
```

```
    else                    语句 2
else
    if(测试表达式 3)        语句 3
    else                    语句 4
```

形式 2：

```
if(测试表达式 1)            语句 1
else if(测试表达式 2)       语句 2
    …
else if(测试表达式 n)       语句 n
else                        语句 n + 1
```

可以看出，形式 1 的嵌套分别发生在 if 和 else 之后的再选择，形式 2 的嵌套则仅发生在 else 后的再选择。形式 2 的 if…else if 语句的执行流程如图 3-3 所示，读者可以依此试着画出形式 1 的 if…else 的执行流程图。

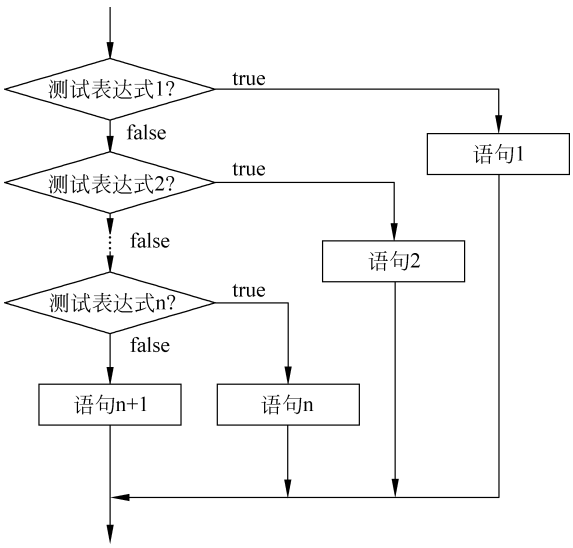

图 3-3　if…else if 语句的执行流程

无论是何种形式，应当注意的是 if 与 else 的配对关系，else 总是与它上面最近的 if 配对，如果省略了某个 else，if 与 else 的数目就不一样了，这时为实现程序设计者的意图，有必要用花括号“{}”括起该层的 if 语句，以确定层次关系。例如：

```
if( )
{    if()  语句 1    }
else      语句 2
```

其中，花括号({})限定了内嵌 if 语句的范围，因此，这里的 else 与第一个 if 配对。

另外，在多层 if 嵌套中简化逻辑关系是非常重要的。

【例 3-2】 考试成绩分级。

问题分析：首先输入一个考试成绩(整数)，通过“处理”输出相应的五分制成绩。设 90 分以上为 A，80～89 分为 B，70～79 分为 C，60～69 分为 D，60 分以下为 E。程序代码如下：

```
#include<iostream>
using namespace std;
int main()
{
    int score;
    char result;

    cout<<"请输入学生百分制成绩(0～100):";
    cin>>score;

    if(score>=90)
        result='A';
    else
        if(score>=80)
            result='B';
        else
            if(score>=70)
                result='C';
            else
                if(score>=60)
                    result='D';
                else
                    result='E';

    cout<<"百分制成绩"<<score<<"对应的成绩等级为:"<<result<<endl;
    return 0;
}
```

程序运行结果:

```
请输入学生百分制成绩(0～100):87↵
百分制成绩87对应的成绩等级为:B
```

上述程序是从90分以上为A开始向下筛选,直到60分以下为E为止。也可以从60分以下为E开始向上筛选,直到90分以上为A为止。相应的程序代码如下:

```
…
if(score<60)
    result='E';
else
    if(score<70)
        result='D';
    else
        if(score<80)
            result='C';
        else
            if(score<90)
                result='B';
            else
                result='A';
…
```

3.2.4 多路选择控制语句 switch

如果在算法中需要进行多次判断选择，但都是判断同一个表达式的值，这时就没有必要在每个嵌套的 if 语句中都计算一次表达式的值。为此，C++中的 switch 语句专门来解决这类问题。switch 语句的语法形式如下：

```
switch(测试表达式)
{
    case    常量表达式 1:              语句 1
    case    常量表达式 2:              语句 2
    …
    case    常量表达式 n:              语句 n
    default:                           语句 n + 1
}
```

switch 语句的执行顺序是：首先计算 switch 语句中的测试表达式的值，然后在 case 语句中寻找值相等的常量表达式，并以此为入口标号，开始顺序执行。如果没有找到相等的常量表达式，则从“default:”开始执行。

使用 switch 语句时应注意下列问题。

(1) switch 后面括号内的“测试表达式”的值只能是整型、字符型、枚举型。例如：

```
float f = 4.0;
switch(f)                                  //错误,测试表达式 f 的值应该为整型
{
    //…
}
```

代码中错误地用浮点型作为 switch 的表达式，将会引起编译错误。

(2) 各常量表达式的值不能相同，且次序不影响执行结果。例如，下面的代码中出现相同的常量值：

```
case 'A':    cout <<"this is A\n";
case 65:     cout <<"this is 65\n";          //错误,'A'等值于 65
```

(3) 每条 case 语句只是一个入口标号，通常只需执行一个 case 后的语句，因此，每个 case 选择的最后应该加 break 语句，用来结束整个 switch 结构。否则，将会从入口点开始一直执行到 switch 结构的结束点。

(4) 当若干选择需要执行相同操作时，可以使多个 case 选择共用一组语句。

现在用 switch 语句重做例 3-2。程序代码如下：

```
#include < iostream >
using namespace std;
int main()
{
    int score;
    char result;
    cout <<"请输入学生百分制成绩(0～100):";
    cin >> score;
```

```
    switch(score/10)
    {
        case 10:
//            result = 'A';
        case 9:
            result = 'A';
            break;
        case 8:
            result = 'B';
            break;
        case 7:
            result = 'C';
            break;
        case 6:
            result = 'D';
            break;
        default:
            result = 'E';
    }
    cout <<"百分制成绩"<< score <<"对应的成绩等级为: "<< result << endl;
    return 0;
}
```

程序运行结果:

```
请输入学生百分制成绩(0～100): 95 ↵
百分制成绩 95 对应的成绩等级为: A
```

上述程序中,用 score/10 作为 switch 语句的测试表达式,score 是整型,所以 score/10 的结果默认取整。例如,当成绩 score 为 95 时,score/10=9,与 case 9 相匹配,因此,执行其后的顺序语句系列"result='A';"和"break;",之后跳出 switch 语句,输出结果。从这段程序可以看出,使用多路选择控制语句 switch 进行编程,逻辑关系更加清晰,程序可读性也更强。

3.3 循环控制结构

循环控制结构是 C++提供的另一种决策方式,它支持程序中需要重复执行的操作。

循环一般分有限次循环和无限循环。有限次循环是指循环次数确定的循环;无限循环则是指循环次数未知的循环,无限循环的极致就是死循环。死循环是指程序的控制流程一直在重复运行某一段代码,并且无法退出的情形,在程序设计中要尽量避免死循环的发生。

C++中有三种循环控制语句:while,do…while 和 for 语句。下面将一一讨论这些语句及其使用方法。

3.3.1 while 语句

while 语句中没有初始化和循环控制变量的更新,只有测试条件和循环体。while 语句

的语法形式如下：

```
while(测试表达式) 循环体
```

其中，循环体是一条语句或由花括号({})括起来的复合语句。

while 语句的执行流程如图 3-4 所示，具体执行顺序如下。

(1) 首先判断测试表达式的值是否为 true，以此决定是否应当进入和执行循环体。

(2) 如果测试表达式的值为 false，结束循环；如果测试表达式的值为 true，则执行循环体。

(3) 返回(1)，对测试表达式进行重新计算和评估，以决定是否继续循环。

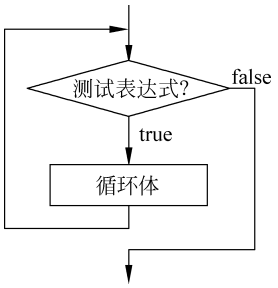

图 3-4　while 语句的执行流程

应用 while 语句构成循环时应特别注意：如果希望循环最终可以结束，循环体中的代码必须完成某种可以影响测试表达式取值为 false 的操作，否则便会造成死循环。

【例 3-3】 求自然数 1～100 之和。

问题分析：本例题需要用累加操作，累加操作是一个典型的循环过程，可以用 while 语句实现。程序代码如下：

```
#include<iostream>
using namespace std;
int main()
{
    int i=1,sum=0;
    while(i<=100)
    {
        sum+=i;
        i++;
    }
    cout<<"sum="<<sum<<endl;
    return 0;
}
```

程序运行结果：

```
sum=5050
```

上述程序中，累加操作被控制执行 100 次。这里变量 i 是关键，它控制了循环的次数，因此 i 被称为循环控制变量。循环控制变量一般在 while 语句之前定义且赋初值，本例题为循环控制变量 i 赋初值 1，循环的测试表达式为 i<=100，循环体中包含了 i++，这样每执行一次循环体，循环控制变量 i 的值都加 1，以此影响循环测试表达式的值。在循环 100 次之前(包括第 100 次)，i 虽然每次加 1，但测试表达式 i<=100 的值一直为 true，循环继续；当循环 100 次之后，i 值加 1 后变为 101，这时测试表达式 i<=100 的值变为 false，循环结束。

上述程序中的代码段：

```
while(i<=100)
{
```

```
        sum += i;
        i++;
    }
```


还可以用以下更简洁的代码替代：

```
while(i <= 100)
    sum += i++;
```

控制循环次数是实现有限次循环，是初学者必须掌握的编程技巧。读者可以想想，例3-3中，同样是实现循环100次，程序代码还可以怎样编写？

3.3.2 do…while 语句

do…while语句结构使循环至少执行一次。do…while语句的语法形式如下：

```
do    循环体
while{测试表达式};
```

do…while语句的执行流程如图3-5所示，其具体的执行顺序为：当程序流程执行到do时，立即执行循环体，与while语句一样，循环体由一条语句或花括号({})定义的语句块组成，然后判断测试表达式的值。当测试表达式的值为true时，继续执行循环体；当测试表达式的值为false时，结束循环。

现在用do…while重做例3-3，程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int i = 1,sum = 0;
    do{
        sum += i;
        i++;
    }while(i <= 100);
    cout <<"sum = "<< sum << endl;
    return 0;
}
```

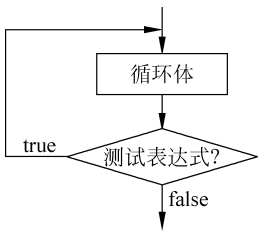

图3-5 do…while语句执行流程

同while语句一样，一般在do…while语句之前也需要定义一个循环控制变量且赋初值(如上述程序中的变量i)，且在循环的测试表达式中包含该循环控制变量，并配合在循环体中包含类似i++这样的操作更新循环控制变量的值，以此影响循环测试表达式的值，用于决定循环是否结束。

do…while与while语句都可以实现循环控制结构，两者的区别是：while语句先判断测试表达式的值，值为true时，再执行循环体；而do…while语句是先执行循环体，再判断测试表达式的值。在大多数情况下，如果循环控制条件和循环体中的语句都相同，while语句和do…while语句的结果是相同的。但是如果开始时循环测试表达式的值就为false，那么这两种循环的执行结果就不同了，do…while语句至少执行一次循环体，while语句却一次都不执行。

在例 3-3 中，i = 1; i < = 100; i++三者联合，实现了循环 100 次。其实，i = 0; i<100; i++也可以实现循环 100 次。同理，i = 100;i > 0;i--和 i = 99;i > = 0;i--都可以实现循环 100 次。

由此可见，循环控制的实现与程序中循环控制变量初始化、循环结束条件和循环控制变量更新三者紧密相关，它们是控制有限次循环的关键。因此，循环控制变量初始化、循环结束条件和循环控制变量更新被称为有限次循环的三要素。

3.3.3 for 语句

for 语句的使用最为灵活，它将有限次循环的三要素，即循环控制变量初始化、循环结束条件和循环控制变量更新集中描述，使得程序既精炼又可读。for 语句的语法形式如下：

```
for(初始化表达式;测试表达式;更新表达式)
    循环体
```

for 语句的执行流程如图 3-6 所示，其具体的执行顺序如下。

(1) 求解初始化表达式的值。

(2) 计算测试表达式的值，并判断是否为 true，以此决定是否进入和执行循环体。

(3) 如果测试表达式的值为 false，则退出循环；如果测试表达式的值为 true，则执行一次循环体。

(4) 求解更新表达式的值。

(5) 转回(2)，判断测试表达式，以决定是否继续执行循环。

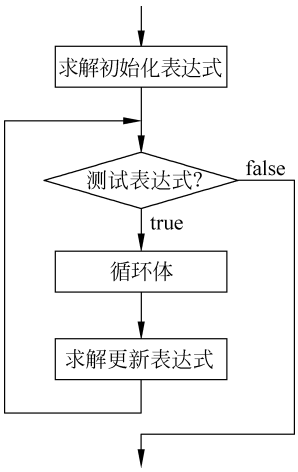

图 3-6　for 语句的执行流程

现在用 for 语句重做例 3-3，程序代码如下：

```
#include<iostream>
using namespace std;
int main()
{
    int i,sum=0;
    for(i=1;i<=100;i++)
    {
        sum+=i;
    }
    cout<<"sum="<<sum<<endl;
    return 0;
}
```

关于 for 语句有以下几点需要特别说明。

(1) 更新表达式用于改变循环控制条件，为了遍历数据，一般设置本次循环与下次循环间的步长为 1，如 i++。步长也可以根据需要设定为大于 1 的整数，如 i=i+2 和 i=i+10 分别表示步长为 2 和 10。

(2) for 语句中，测试表达式是循环控制条件，所以一般不能省略。如果省略，将会出现

死循环。

(3) 初始化表达式和更新表达式在一定条件下可以省略,并可以是任何表达式。例如,可以将上述程序中有关循环的代码缩写成如下形式:

```
for(sum = 0,i = 1;i <= 100;) sum += i++;
```

这里的 for 语句中省略了更新表达式,但前面的分号(;)不能随之省略,另外初始化表达式为逗号表达式,除了给循环变量初始化为 1 之外,还为存取求和结果的变量 sum 进行清零。

没有最好,只有更好,不断地优化、精简程序可以提高编程水平,相信有一天,你会突然发现自己编写的程序越来越专业了。

3.3.4 循环嵌套

如果程序要解决更大的问题,实现更复杂的算法,单循环是远远不够的。在很多情况下需要循环嵌套,即一个循环体内包含另一个完整的循环结构,构成多重循环。while、for 和 do…while 三种循环语句可以自己嵌套或互相嵌套,但要求内循环必须被完全包含在外循环的循环体中。

【例 3-4】 输出九九乘法表。

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

问题分析:九九乘法表是一个二维表,在屏幕上输出九九乘法表相当于从上到下、从左到右,如扫描一般输出内容。其中,从上到下是输出每一行,从左到右是输出一行中的每一列。因此,需要构成双重循环,外循环控制行,内循环控制列。程序代码如下:

```
#include <iostream>
using namespace std;
int main()
{
    int i,j;

    cout <<" * \t";
    for(i = 1;i <= 9;i++)
        cout << i <<"\t";

    cout <<"\n----------------------------------"
        <<"----------------------------------------\n";
```

```
    for(i = 1;i < = 9;i++)
    {
        cout << i <<"\t";
        for(j = 1;j < = 9;j++)
            cout << i * j <<"\t";
        cout << endl;
    }

    return 0;
}
```

上述程序中，for(i = 1;i < = 9;i++){…}是外循环，从第 1 行开始“遍历”每一行；for(j = 1;j < = 9;j++){…}是内循环，被完全包含在外循环体中，它是外循环体的一部分，对第 i 行，从第 1 列开始“遍历”每一列。

如果想输出如下所示的九九乘法表，上述程序需要怎样修改呢？

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

这时，上述程序基本保持不变，只需将程序中内循环用如下代码替换即可：

```
for(j = 1;j < = i;j++)
    cout << i * j <<"\t";
```

3.4 程序控制进阶

在比较复杂的程序设计中，顺序、选择和循环通常是混合应用的，三种控制结构中，比较难掌握的是循环控制。

下面首先介绍在选择和循环中常用的三个控制语句，之后再讲解如何通过输入信息控制循环。

3.4.1 其他控制语句

1. break 语句

break 语句只用于 switch 语句或循环体中，其作用是使程序从 switch 语句内跳出或结束循环，继续执行逻辑上的下一条语句。

2. continue 语句

continue 语句仅用于循环体中，其作用是结束本次循环，接着开始判断循环条件，决定

是否继续执行下一次循环。

在实际的程序设计中,break 和 continue 语句应用得比较多,为了更好地理解 break 和 continue 语句在控制循环中的作用和区别,我们将循环比作在操场上跑圈,循环 10 次就是跑 10 圈,每跑完一圈计数器加 1。假如你现在跑到第 5 圈的一半,此时因为某种原因,触发了执行 break 语句的条件,这时意味着剩下的 4 圈半你不需要再跑了;如果触发的是执行 continue 的条件,则意味着第 5 圈剩下的半圈不用跑,但你还需要从第 6 圈开始继续跑。

灵活运用 break 和 continue 语句可以实现一些特殊功能,使程序运行时更"智能"。下面对考试成绩分级程序进行修改,使之可以多次输入成绩,并进行成绩分级处理,直到输入-1 时,结束程序。注意程序中 while、break 和 continue 语句的使用。程序代码如下:

```
#include<iostream>
using namespace std;
int main()
{
    int score;
    char result;
    while(true)
    {
        cout<<"请输入学生百分制成绩(0～100,输入-1结束):";
        cin>>score;
        if(score==-1)
            break;
        if(score<0||score>100)
        {
            cout<<"输入学生百分制成绩有错,请重新输入!"<<endl;
            continue;
        }
        switch(score/10)
        {
        case 10:
        case 9:
            result='A';
            break;
        case 8:
            result='B';
            break;
        case 7:
            result='C';
            break;
        case 6:
            result='D';
            break;
        default:
            result='E';
        }
        cout<<"百分制成绩"<<score<<"对应的成绩等级为:"
```

```
            << result << endl;
    }
    return 0;
}
```

上述程序用 while(true){}构建一个无限循环，在此循环内，首先用 if 简单语句判断输入的成绩 score 是否等于－1，如果等于－1，则执行 break 语句结束循环。之所以用－1，是因为－1 不可能是一个成绩的值。之后，通过另一个 if 简单语句判断输入的成绩是否为 0～100，如果不是，则显示“输入学生百分制成绩有错，请重新输入!”，并执行 continue 语句退出本次循环，并开始下一次循环。重新输入成绩，如果输入的成绩为 0～100，则继续执行下面的 switch 语句，对成绩进行多分支处理。运行这段程序可实现多次输入成绩，并完成成绩分级，直到输入－1，结束程序。

3. goto 语句

goto 语句的作用是使程序的执行流程跳转到语句标号所指定的语句。goto 的语法形式如下：

```
goto <语句标号>
```

其中，“语句标号”是用来表示语句的标识符，放在语句的最前面，并用冒号“:”与语句分开。例如，上述程序中的 continue 语句就可以用 goto 语句替代，程序代码如下：

```
…
    while(true)
    {
a1:
    cout <<"请输入学生百分制成绩(0～100,输入－1 结束):";
    cin >> score;
    if(score == －1)
        break;
    if(score < 0||score > 100)
    {
        cout <<"输入学生百分制成绩有错,请重新输入!"<< endl;
        goto  a1;
    }
    …
}
…
```

程序段中的 a1 为标识符，goto a1 表示转至 a1: cout <<"请输入学生百分制成绩(0～100,输入－1 结束):"处。

【例 3-5】 一元二次方程求解。

问题分析：对于一元二次方程 $ax^2+bx+c=0$，系数 a 不能为 0，否则需要重新输入系数。程序代码如下：

```
#include < iostream >
#include < math.h >
using namespace std;
int main()
```

```
{
    double a,b,c,d,x1,x2;
    cout <<"一元二次方程: ax * x + bx + c = 0\n 请输入系数 a,b,c:";
n1:cin >> a >> b >> c;
    if(a == 0)
    {
        cout <<"请重新输入系数 a,b,c:";
        goto n1;
    }
    else
    {
        d = b * b - 4 * a * c;
        if(d > = 0)
        {
            x1 = ( - b + sqrt(d))/(2 * a);
            x2 = ( - b - sqrt(d))/(2 * a);
            cout <<"x1 = "<< x1 << endl;
            cout <<"x2 = "<< x2 << endl;
        }
        else
            cout <<"此方程无解!\n";
    }
    return 0;
}
```

程序运行结果1:

```
一元二次方程:ax * x + bx + c = 0
请输入系数 a,b,c:1  5  6
X1 = - 2
X2 = - 3
```

程序运行结果2:

```
一元二次方程:ax * x + bx + c = 0
请输入系数 a,b,c:1  2  3
此方程无解!
```

注意:goto 语句的使用会破坏程序的结构,应该少使用或不用。

3.4.2 输入信息控制循环

输入信息控制循环通常控制的是无限循环,无限循环是循环次数不确定的循环。当遇到利用条件反复判断确定何时结束任务的时候,借助构建无限循环和跳出循环的方法,可以有效地避免死循环,这样做的好处是比传统的解决方法编写的代码精简且容易理解。

通常采用 while 或 do…while 语句构建无限循环,在循环体内输入信息,并通过循环控制语句的测试表达式确定是否跳出循环,以实现输入信息控制循环的目的。

【例 3-6】 统计输入的字符数。

```
#include < iostream >
using namespace std;
```

```
int main()
{
    char ch;
    int count = 0;
    cout <<"输入字符串,以＃结束: ";
    while(true)
    {
        cin >> ch;
        if(ch == '＃')
            break;
        count++;
    }
    cout <<"共输入"<< count <<"个字符。"<< endl;
    return 0;
}
```

程序运行结果:

```
输入字符串,以＃结束:abcdefg＃
共输入 7 个字符。
```

上述程序中,while 语句的测试表达式为 true,也就是循环条件始终满足,如果没有其他干预,循环将不停地继续,不会自动结束。所以在循环体中,用 cin 输入的单字符 ch 控制循环是否结束。当输入非'＃'字符时,选择语句 if 的测试表达式 ch == '＃'的值为 false,计数变量 count 加 1,并继续循环。再次输入单字符,直到输入'＃'字符,选择语句 if 的测试表达式 ch == '＃'的值变为 true,这时执行 break 语句,跳出循环。

以下的程序代码同样可以实现统计输入字符数的功能,注意观察与前者的区别。

```
＃include < iostream >
using namespace std;
int main()
{
    char ch;
    int count = 0;
    cout <<"输入字符串,以＃结束: ";
    cin >> ch;                                  //循环外输入
    while(ch!= '＃')                            //以 ch!= '＃'作为循环结束条件
    {
        count++;
        cin >> ch;
    }
    cout <<"共输入"<< count <<"个字符."<< endl;
    return 0;
}
```

这段程序中,巧妙地用 ch! = '＃'作为 while 语句的测试表达式。在 while 之前首先用 cin 输入单字符,当输入非'＃'字符时,测试表达式 ch! = '＃'的值为 true,进入循环体,计数变量 count 加 1,再次输入单字符,并继续循环,直到输入'＃'字符,测试表达式 ch! = '＃'的值变为 false,结束循环。

3.5 程序控制综合编程案例

【例 3-7】 输入一个年份,判断是否为闰年。

问题分析:闰年的年份可以被4整除而不能被100整除,或者能被400整除。因此,首先输入年份存放到变量year中,如果表达式((year%4==0&&year%100!=0)‖(year%400==0))的值为true,则为闰年,否则就不是闰年。程序代码如下:

```
#include<iostream>
using namespace std;
int main()
{
    int year;
    bool IsLeapYear;
    cout<<"Input a year:";
    cin>>year;
    IsLeapYear=((year%4==0&&year%100!=0)||(year%400==0));
    if(IsLeapYear)
        cout<<year<<" is leap year"<<endl;
    else
        cout<<year<<" is not leap year"<<endl;
    return 0;
}
```

程序运行结果:

```
Input a year:2000
2000 is a leap year
```

【例 3-8】 计算相应图形的面积。

问题分析:首先需要构造如下的菜单。

```
图形类型:
1-圆形
2-长方形
3-正方形
请输入你的选择(1~3):
```

然后,通过cin键盘输入1~3中的任意整数作为选项,之后再通过多路选择控制语句switch根据选项选择不同的图形,并计算相应图形的面积。程序代码如下:

```
#include<iostream>
using namespace std;
const double PI=3.1416;
int main()
{
    int iType;
    double radius, a, b, area;
    cout<<"图形类型:\n1-圆形\n2-长方形\n3-正方形\n请输入你的选择(1~3):";
```

```
    cin >> iType;
    switch(iType)
    {
    case 1:
        cout <<"\n 圆的半径为: ";
        cin >> radius;
        area = PI * radius * radius;
        cout <<"面积为: "<< area << endl;
        break;
    case 2:
        cout <<"\n 矩形的长为: ";
        cin >> a;
        cout <<"矩形的宽为: ";
        cin >> b;
        area = a * b;
        cout <<"面积为: "<< area << endl;
        break;
    case 3:
        cout <<"\n 正方形的边长为: ";
        cin >> a;
        area = a * a;
        cout <<"面积为: "<< area << endl;
        break;
    default:
        cout <<"\n 不是合法的输入值!"<< endl;
    }
    return 0;
}
```

程序运行结果：

```
图形类型:
1 - 圆形
2 - 长方形
3 - 正方形
请输入你的选择(1～3): 1

圆的半径为: 2
面积为: 12.5664
```

【例 3-9】 求水仙花数。如果一个三位整数的个位数、十位数和百位数的立方和等于该数自身，则称该数为水仙花数。

问题分析：要求在 100～999 内寻找水仙花数，所以必须对这个范围内所有的数据进行一一检验，看是否符合水仙花数的条件。而判断一个三位整数是否为水仙花数的关键是得到这个整数 n 的百位数、十位数和个位数，假设 i 为其百位数、j 为其十位数、k 为其个位数，则：

```
i = n/100;
j = (n/10) % 10;
k = n % 10;
```

显然,这是一个确定循环次数的循环,故可以使用 for 语句。程序代码如下:

```
#include<iostream>
using namespace std;
int main()
{
    int n,i,j,k;
    for(n=100;n<=999;n=n+1)
    {
        i=n/100;                          //取出 n 的百位数
        j=(n/10)%10;                      //取出 n 的十位数
        k=n%10;                           //取出 n 的个位数
        if(n==i*i*i+j*j*j+k*k*k)
            cout<<n<<"="
                <<i<<"^3+"
                <<j<<"^3+"
                <<k<<"^3\n";
    }
    return 0;
}
```

程序运行结果:

```
153=1^3+5^3+3^3
370=3^3+7^3+0^3
371=3^3+7^3+1^3
407=4^3+0^3+7^3
```

【例 3-10】 输出图形:

```
        A
      A   A
    A   B   A
  A   B   B   A
A   B   B   B   A
  A   B   B   A
    A   B   A
      A   A
        A
```

问题分析:在打印机和屏幕上输出二维图形,就像"扫描"一样,从左到右、从上到下输出图形,所以一般都需要用到循环嵌套,外循环控制行,内循环控制列。程序代码如下:

```
#include<iostream>
#include<iomanip>
using namespace std;
const int N=4;
int main()
{
    int k,n,i,j;

    do{
```