第 3 章

语 句 进 阶

本章介绍 switch 语句、continue 语句、break 语句。switch 语句是多分支的条件语句，可根据整数相等关系匹配不同的分支。continue 语句只能用于循环语句中，作用是立即开始下一次循环，可跳过循环体中下面尚未执行的语句。break 语句可用于循环语句和 switch 语句中。如果循环体中的 break 被执行，则跳出本层循环；如果 switch 语句中的 break 被执行，则跳出 switch 而执行 switch 以后的语句。

3.1 switch 语句

实际问题中，常常需要用到多分支的选择语句。if 语句是双分支的条件语句，但可以在否定分支上进一步判断实现多分支。C 语言还提供了用于多分支选择的 switch 语句。switch 语句判断整数表达式和常量表达式的相等关系，并执行不同的语句序列。switch 语句的语法形式为

```
switch(整数表达式){
    case 常量表达式 1:  语句序列 1;  //注意 case 后有空格
    case 常量表达式 2:  语句序列 2;
    ⋮
    case 常量表达式 n:  语句序列 n;
    default:  语句序列 n+1;
}
```

switch 语句首先计算整数表达式(包括字符类型)的值，并逐个与常量表达式的值进行比较。当整数表达式的值与某个常量表达式的值相等时，即执行其后的语句序列，然后不再进行判断，继续执行所有 case 后的语句序列，直到遇到 break。如果整数表达式的值与所有 case 后面的常量表达式均不相等，则执行 default 后面的语句序列。default 是默认分支，通常放在最后。

例 3.1 从键盘输入一个整数，打印对应的星期名称。

```
//DayOfWeek.cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv){
    int day;
    cout << "Input an integer[1-7]: ";
    cin >> day;
    cout << day << " => ";
```

```
    switch(day){            //()里必须是整数表达式,只能判断相等关系
    case 1:                 //case后面一定要有空格
        cout << "Monday";
        break;
    case 2:
        cout << "Tuesday";
        break;
    case 3:
        cout << "Wednesday";
        break;
    default:
        cout << "Invalid number";
        break;              //default不在最后时需要加break
    case 4:
        cout << "Thursday";
        break;
    case 5:
        cout << "Friday";
        break;
    case 6:
        cout << "Saturday";
        break;
    case 7:
        cout << "Sunday";
        break;
    }
    cout<<endl;
    return 0;
}
```

【运行结果】

```
Input an integer[1-7]: 6↵
6 => Saturday
Input an integer[1-7]: 8↵
8 => Invalid number
```

【代码解读】

switch语句将变量day和case后面的各个数字依次比较是否相等,如果相等就执行数字后面的语句序列。比如输入“6”,就会执行“cout<<"Saturday";”和“break;”。如果输入的数字跟任何一个case后面的数字都不相等,就执行默认分支。比如输入8,就会执行“cout<<"Invalid number";”和“break;”。默认分支可以不放在最后,但此时默认分支的最后一条语句应为“break;”。

使用switch语句应注意下列问题。

(1) switch后面圆括号中的整数表达式的类型可以是整型、字符型或枚举表达式,case后面的常量表达式的类型必须与其匹配。

(2) 每个case常量表达式的值必须互不相同,否则就会出现编译错误。

(3) switch语句中case分支的语句序列可以是一条语句,也可以是多条语句,还可以没

有语句。

(4) 每个 case 后面的多条语句的最后通常是 break 语句,以保证多路分支的正确实现。在遇到 break 之前,switch 会继续执行下面不同的 case 分支中的语句。每个 case 只是一个入口标记,并不能确定执行的终止位置。

(5) 如果 case 后没有语句,则一旦执行到这个 case 分支,什么也不做,继续往下执行。当若干分支需要执行相同操作时,可以多个 case 共用一组语句。

(6) switch 语法中各个 case 分支和 default 分支的出现的次序在语法中没有规定,但通常 default 分支位于最后。

(7) switch 语句中 default 分支是可选的,若没有 default 分支且不和任何 case 后的值相等,switch 语句将什么也不做,直接执行后续语句。

例 3.2 输入月份判断是哪个季节。

```
//month.cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv){
    int month;
    cout << "Input an integer[1-12]: ";
    cin >> month;
    cout << month << " => ";
    switch(month){
    case 3:  case 4:  case 5:
        cout <<"春季(Spring)\n";
        break;
    case 6:  case 7:  case 8:
        cout<<"夏季(Summer)";
        break;
    case 9:  case 10:  case 11:
        cout<<"秋季(Autumn)";
        break;
    case 12:  case 1:  case 2:
        cout<<"冬季(Winter)";
        break;
    default:
        cout<<"无效数字(Invalid number)";
    }
    cout<<endl;
    return 0;
}
```

【运行结果】

```
Input an integer[1-12]: 9↵
9 => 秋季(Autumn)
```

【代码解读】

程序中多个 case 共用了同一组语句。输入 9 时,“case 9:”后面没有语句,程序继续往下执行,“case 10:”后面也没有语句,程序继续执行“case 11:”后面的输出语句和 break

语句。

下面给出月份对应季节问题的另外一种实现。

```
//month2.cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv){
    int month;
    cout << "Input an integer[1-12]: ";
    cin >> month;
    cout << month << " => ";
    if(month>=1 && month<=12){
        switch(month %  12 / 3){
        case 0: //12 1 2
             cout << "冬季(Winter)";
             break;
        case 1: //3 4 5
            cout << "春季(Spring)\n";
            break;
        case 2: //6 7 8
            cout << "夏季(Summer)";
            break;
        case 3: //9 10 11
            cout << "秋季(Autumn)";
            break;
        }
    }else{
        cout << "无效数字(Invalid number)";
    }
    cout << endl;
    return 0;
}
```

【代码解读】

对于1～12的数字，“对12求余再整除3”恰好可以把每个季节对应的三个月份变换成了相同的数字。“对12求余”将12变换成了0，其余数字没变。再“整除3”的结果：“0、1、2”得到0，“3、4、5”得到1，“6、7、8”得到2，“9、10、11”得到3。

3.2　continue语句

continue语句只能用在循环语句(while、do-while、for)中，不能单独使用。continue语句的作用是结束本次循环，即跳过循环体中下面尚未执行的语句，立即开始下一轮循环。

对于while循环和do-while循环，转去求解循环条件。对于for循环，转去执行for语句头中的第三个表达式(末尾循环体)。

例3.3　while循环中使用continue。

```
//ContinueWhile.cpp
#include <iostream>
```

```
using namespace std;
int main(int argc, char **argv){
    int i=1;
    while(i<=5){
        cout << i;
        if(3==i){          //== 判断相等
            i++;           //continue之前需要修改循环变量
            cout << endl;
            continue;      //转去判断循环条件
        }
        cout << ' ' << i << endl;
        i++;
    }
    return 0;
}
```

【运行结果】

```
1 1
2 2
3
4 4
5 5
```

【代码解读】

当循环变量i等于3时,执行continue转去计算循环条件"i<=5"。如果不在跳转前添加修改i的代码,将会导致死循环。

例 3.4 for循环中使用continue。

```
//ContinueFor.cpp
#include <iostream>
using namespace std;
int main(int argc, char **argv){
    int i;
    for(i=1; i<=5; i++){
        cout << i;
        if(3==i){
            cout << endl;
            continue;  //转去执行 for 头部第三个表达式 i++
        }
        cout << ' ' << i << endl;
    }
    return 0;
}
```

【代码解读】

当i等于3时,执行continue转去执行for头部第三个表达式"i++",跳转前不需要添加修改循环变量i的代码。运行结果同上。

例 3.5 do-while循环中使用continue。

```
//ContinueDoWhile.cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv){
    int i=1;
    do{
        cout << i;
        if(3==i){
            i=20;
            cout << endl;
            continue;     //转去判断循环条件
        }
        cout << ' ' << i << endl;
        i++;
    }while(i<=5);
    return 0;
}
```

【运行结果】

```
1 1
2 2
3
```

【代码解读】

当循环变量i等于3时，执行continue。为验证do-while循环中执行continue是转去计算循环条件，程序将i赋值为20。

如果转去循环开始位置，则会输出两个20；如果转去循环末尾判断循环条件，将导致循环结束。运行结果无后续输出，可见是转去循环末尾判断循环条件了。

例3.6　跳过奇数，输出偶数。

```
//ContinueEven.cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv){
    for(int i=1; i<10; i++){
        if(i%2==1){
            continue;       //跳过奇数
        }
        cout << i << " ";
    }
    cout<<endl;
    return 0;
}
```

【运行结果】

```
2 4 6 8
```

【代码解读】

当循环变量是奇数时，执行continue转去执行for头部的“i++”，这样就跳过了下面

的输出语句。当 i 等于“1、3、5、7、9”时 continue 语句都执行了，共执行 5 次。程序跳过了奇数，输出的就仅剩偶数了。

例 3.7 输入有效的表示月份的数字，给用户 5 次输入机会。

```
//ContinueMonth.cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv){
    int month;
    bool success=false;
    for(int i=0; i<5 && !success; i++){
        cout << "Input month: ";
        cin >> month;
        if(month<1 || month>12){
            continue;       //开始下一轮循环,转去执行 i++
        }
        success=true;
    }
    if(success){
        cout << "month=" << month << endl;
    }
    return 0;
}
```

【运行结果】

```
Input month: 15↵
Input month: 0↵
Input month: 9↵
month=9
```

【代码解读】

for 循环使用整数变量 i 和布尔变量 success 两个变量控制循环条件。如果输入的数字不在[1,12]内，则执行 continue 转去执行 for 语句头部的“i＋＋”。只有输入的数字在[1,12]内，才能执行后面的语句“success＝true;”，接着执行“i＋＋”，再判断循环条件中的“!success”为 false，导致循环条件不成立而结束循环。循环变量 i 从 0 到 4 时都可以进入循环，当 i 等于 5 时会导致循环条件不成立而结束循环。

3.3 break 语句

break 语句的作用：①终止本层循环；②在 switch 语句中，当执行完一个 case 后的语句序列时跳出 switch 语句。

continue 语句和 break 语句的区别：continue 语句只结束本次循环，而不终止整个循环的执行；break 语句则是结束整个循环过程，不再判断执行循环的条件是否成立。

for 循环内部的 break 语句执行时，是直接结束循环，不会执行 for 语句头部的第三个表达式，也不会判断循环条件。while 循环和 do-while 循环内部的 break 语句执行时，也不判断循环条件，直接结束循环。

循环嵌套时，位于内层循环中的 break 语句仅能终止内层循环，程序继续执行位于外层循环内部的跟在内层循环后面的语句。

例 3.8　在三种循环语句中使用 break。

```
//break.cpp
#include <iostream>
using namespace std;
int main(int argc, char** argv){
    int i=1;
    while(i<=5){
        cout << i;
        if(3==i){
            cout << endl;
            break;                    //结束循环
        }
        cout << " " << i << endl;
        i++;
    }
    cout << "i=" << i << "\n\n";      //break 跳到这条语句，输出 3
    for(i=1; i<=5; i++){
        cout << i;
        if(3==i){
            cout << endl;
            break;                    //结束循环
        }
        cout << " " << i << endl;
    }
    cout << "i=" << i << "\n\n";      //break 跳到这条语句，输出 3
    i=1;
    do{
        cout << i;
        if(3==i){
            cout << endl;
            break;                    //结束循环
        }
        cout << " " << i << endl;
        i++;
    }while(i<=5);
    cout << "i=" << i << endl;        //break 跳到这条语句，输出 3
    return 0;
}
```

【运行结果】

```
1 1
2 2
3
i=3
```

【代码解读】

三种循环执行的情况是一样的：当循环变量 i 等于 3 时，执行 break 语句结束整个

循环。

例 3.9 判断一个整数是否是素数。

素数即质数,是在大于 1 的自然数中,除了 1 和本身不再有其他因数的自然数。

```
//prime.cpp
#include <iostream>
#include <cmath>                          //数学函数库
using namespace std;
int main(int argc, char** argv){
    int n;
    cout << "Input n: ";
    cin >> n;
    if(n<2){
        cout << n << "不是素数\n";
    }
    bool is_prime = true;                 //假设 n 是素数
    for(int i=2; i<=sqrt(n); i++){        //sqrt: Square Root
        if(n %  i == 0){                  //发现 n 可以被 i 整除
            is_prime = false;             //可做出不是素数的判断
            cout << "i==" << i << " break\n";
            break;                        //立即结束循环
        }
    }
    if(is_prime)
        cout << n << "是素数\n";
    else
        cout << n << "不是素数\n";
    return 0;
}
```

【运行结果】

```
Input n: 35
i==5 break
35 不是素数
```

【代码解读】

对于正整数 n,循环变量 i 从 2 增长到 n－1,优化后到 sqrt(n)即可,判断 n 是否可以被 i 整除。一旦发现 n 可以被整除,立即使用 break 结束循环,如果不结束,后续的循环判断是没有意义的,因为只要被整除了,就不是素数。

假设存在一个比 sqrt(n)大的自然数 b 是 n 的因子,则存在自然数 a,满足“a * b＝＝n”。由于 b 大于 n 的平方根,则 a 必然小于 n 的平方根。而从 2 到 sqrt(n)的遍历过程已经检测过自然数 a,确定可以被 a 整除,不是素数,这样就不需要检测被 b 整除了。

例 3.10 跳出内层循环。

```
//NineBreak.cpp
#include <iostream>
#include <iomanip>          //控制输入输出格式的头文件,包含 setw()
using namespace std;
```

```
int main(int argc, char** argv){
    for(int i=1; i<=9; i++){
        for(int j=1; j<=9; j++){
            if(j>i){
               cout << " bk";
               break;       //i=1,j=2; i=2,j=3; i=3,j=4; ... i=8,j=9;时执行 break
            }
            cout << setw(3) << i * j;     //输出内容占 3 个字符宽,不足填充空格
        }
        cout << endl;                     //break 跳到这条语句
    }
    return 0;
}
```

【运行结果】

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

【代码解读】

在循环嵌套时,位于内层循环的 break 语句只能跳出内层循环。打印九九乘法表时,外层循环变量 i 从 1 循环到 9,控制输出 1～9 行;内层循环变量 j 从 1 循环到 9,在每一行输出 9 个乘积。在内层循环中,“j>i”时执行 break 语句,这仅仅会结束那一次的内层循环,跳到跟着内层循环后面的语句“cout<<endl;”继续执行。break 语句共执行了 8 次,结果是输出运行结果中的乘法表。

例 3.11　利用 goto 语句跳出多层循环。

```
//NineGoto.cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main(int argc, char** argv){
    for(int i=1; i<=9; i++) {
        for(int j=1; j<=9; j++) {
            if(j>i)
                goto MyTag;
            cout << setw(3) << i * j;
        }
        cout << endl;
    }
MyTag: cout << "\n 跳到这里";        //MyTag 是自己定义的标号
    return 0;
}
```

【运行结果】

```
  1
跳到这里
```

【代码解读】

虽然 goto 语句处于内层循环内部，但是可以使用 goto 语句跳到双层循环外面的标号。当“i＝1，j＝2”时，程序执行 goto 语句跳出了双层循环。

goto 语句也称为无条件转移语句，其一般格式为“goto 语句标号；”，其中语句标号是按标识符规定书写的符号，放在某一行语句的前面。语句标号后需要加“：”，即半角冒号，表示这是一个标号。语句标号起标识语句的作用，与 goto 语句配合使用。

习　　题

1. 第 1～4 名分别称为冠军、亚军、季军、殿军，第 5 名及 5 名以上，称为其他名次。输入一个名次，输出名次对应的荣誉称号。

2. 求 1～200 的整数中除了 7 的倍数的其他整数之和。

3. 输入一个日期（年 月 日），判断是这一年中的第几天。需考虑闰年。比如：输入“2008 8 8”，运行结果：8 月 8 日是 2008 年中的第 221 天。

闰年（Leap Year）是为了弥补因人为历法规定造成的年度天数与地球实际公转周期的时间差而设立的。补上时间差的年份为闰年。地球绕太阳运行周期为 365 天 5 小时 48 分 46 秒（合 365.242 19 天）即一回归年（Tropical Year）。公历的平年只有 365 日，比回归年短约 0.242 19 日，所余下的时间约为四年累计一天，故每四年于 2 月加 1 天，使当年的历年长度为 366 日，这一年就为闰年。判定公历闰年应遵循的一般规律：四年一闰，百年不闰，四百年再闰。

闰年的判断方法：①普通闰年能被 4 整除，不能被 100 整除。比如 2004 年是闰年。②世纪闰年能被 400 整除。如 2000 年是闰年，1900 年不是闰年。

4. 用二分法求方程 $2x^3-4x^2+3x-6=0$ 在[－10，10]的根。

二分法又称分半法，是一种求方程式根的近似值的方法。对于区间[a，b]上连续不断且 f(a)×f(b)＜0 的函数 y＝f(x)，通过不断地把函数 f(x)的零点所在的区间一分为二，使区间的两个端点逐步逼近零点，进而得到零点近似值的方法叫作二分法。

如果要求已知函数 f(x)＝0 的根，二分法求根的步骤如下：

(1) 先要找出一个区间[a，b]，使得 f(a)与 f(b)异号。根据介值定理，这个区间内一定包含着方程式的根。

(2) 求该区间的中点 m＝(a＋b)/2，并计算 f(m)的值。

(3) 若 f(m)与 f(a)正负号相同，则取[m，b]为新的区间，否则取[a，m]。

(4) 重复步骤(2)和(3)，使 f(m)趋近零，直到得到理想的精确度为止。