CHAPTER 3

第3章 程序控制结构

本章要点

- 选择控制及其嵌套
- 多路选择控制 switch 语句
- 循环控制及多重循环
- break 和 continue 语句

在学习了C++的数据类型、表达式和输入输出控制之后，就能够编写一些可以完成简单功能的程序了。

之前编写的一些程序都仅包含一些顺序执行的语句。实际上，客观世界远不是这么简单，通常解决问题的方法也不是仅用顺序步骤就可以描述清楚的。例如，有一分段函数如下，要求输入一个 x 值，求出相应的 y 值。

$$y=\begin{cases}1 & (x\geqslant 0)\\ -1 & (x<0)\end{cases}$$

这是一个很简单的选择判断算法，人工解决这个问题并不复杂，但问题是如何将这个选择判断让计算机来完成。显然，用顺序执行的语句序列是无法描述这个算法的，这里需要根据 x 值进行选择判断，以决定 y 的取值。

再举一个例子，统计某一个年级某门课程的平均成绩。这个统计算法中的一个主要部分就是进行累加，这种大量重复的相同操作显然也不适宜用顺序执行的语句来罗列，这就需要一种能控制进行重复操作，并能决定什么时候结束重复操作的程序结构。

算法是计算机解决问题的方法和步骤，它由一系列控制结构组成。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。顺序控制结构是系统预置的，即除非特别指定，计算机总是按指令编写的顺序一条一条地执行，顺序控制结构比较简单，之前的程序中已使用过，本章着重介绍选择和循环这两种控制结构。

3.1　选择控制结构

设计程序的关键是使程序具有决策能力，C++提供了多种选择型控制语句来支持程序选择决策，这些选择型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，执行另一个操作。又如，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在可选择的操作中做出决定。

3.1.1　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
```

其中，语句 1 和语句 2 不仅可以是一条语句，而且可以是大括号括起来的多条语句(称为复合语句或语句块)。

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 3-1(a)所示。

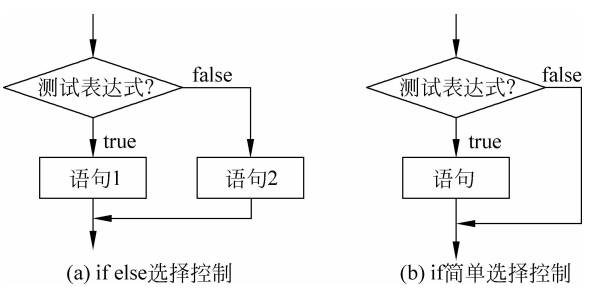

图 3-1　if else 语句控制流程图

例如，如下的程序代码：

```
if(x>=0)
    cout<<"y="<<1<<endl;
else
    cout<<"y="<<-1<<endl;
```

实现了根据 x 的取值，输出相应 y 值的功能。

另外，if 语句中的语句 2 可以为空，当语句 2 为空时，else 可以省略，其执行流程如图 3-1(b)所示，这时的语句形式如下：

```
if(测试表达式)  语句
```

例如，若 ch 字符等于字符'b'，则响铃：

```
if(ch == 'b')     cout <<'\a';
```

【例 3-1】 在两个数中找最大值。

问题分析：使用 if else 语句可以很容易地实现在两数中找最大值。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;
}
```

程序运行情况：

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

3.1.2 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。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 是在两个数中找最大值，如果是在三个数中找最大值，程序又如何实现呢？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，但对初学者不那么好理解。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。条件表达式“?:”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 max 变量。

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;
```

初学者最好用规范的格式书写程序，因为程序的可读写性是非常重要的，真正的商业程序是绝对规范的，张三写的程序和李四的程序格式大致相同，各种标识符的命名规则一样，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。

3.1.3　if else 语句的嵌套

有很多问题是一次简单的判断所解决不了的，需要进行多次判断选择。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 3-2 所示，读者可以依此试着画出形式 1 的 if…else 的执行流程图。

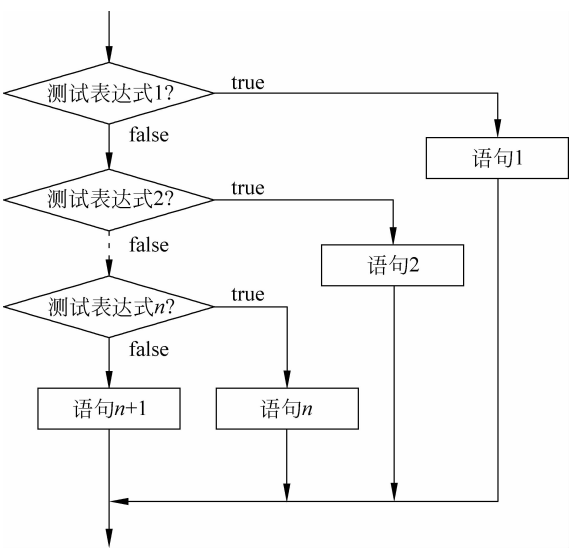

图 3-2　if…else if 语句流程图

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，如果省略了某一个 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
```

这时，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;
}
```

程序运行情况：

```
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。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';
…
```

由此可见，在测试表达式中，灵活地运用关系运算符和逻辑运算符组成高效的表达式是非常重要的，恰当的表达式不仅可以使程序逻辑关系清晰，还可以简化程序，提高程序运行

效率。

3.1.4 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一个嵌套的 if 语句中都计算一次表达式的值。为此，C++中有 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";
csae 65:      cout <<"this is 65\n";          //错误,'A'等值于 65
```

(3) 每个 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 考试成绩分级。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;
}
```

程序运行情况：

```
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
```

3.2　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。

3.2.1　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

while 语句的执行顺序是：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。如果测试表达式的值为 true，则执行循环体，循环体由一条语句或花括号“{}”括起来的语句块组成。执行完循环体后，程序返回测试表达式，对它进行重新计算。如果该测试表达式的值为 true，则再次执行循环体。测试和执行将一直进行下去，直到测试表达式的值为 false 时结束。如图 3-3 所示是 while 语句的执行流程图。

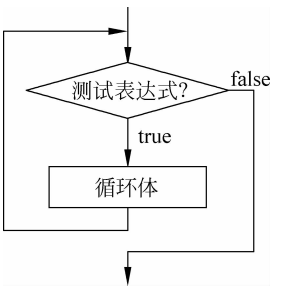

图 3-3　while 语句的执行流程图

应用 while 语句时应该注意：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：本题需要用累加算法，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;
}
```

程序运行情况：

```
sum=5050
```

上述程序中，累加操作被控制执行了 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，这里为循环控制变量 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++;
}
```

还可以用更简洁的代码替代：

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

控制循环次数，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？

3.2.2　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 语句的执行顺序是：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。do-while 语句的执行流程图如图 3-4 所示。

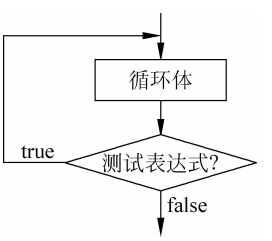

图 3-4　do while 语句执行流程图

现在用 do while 重做例 3-3 求自然数 1～100 之和，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;
}
```

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 循环和 do while 循环的结果是相同的。但是如果一开始循环测试表达式的值就为 false，这两种循环的执行结果就不同了，do while 循环至少执行一次循环体，while 循环却一次都不执行。

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。

3.2.3 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，使得程序又精炼又可读。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 3-5 所示，具体步骤如下。

(1) 计算初始化表达式的值。

(2) 计算测试表达式的值，如果测试表达式的值为 false，则退出循环。

(3) 如果测试表达式的值为 true，则执行一次循环体，然后计算更新表达式的值。

(4) 转回(2)，求测试表达式的值，以决定是否继续执行循环体。

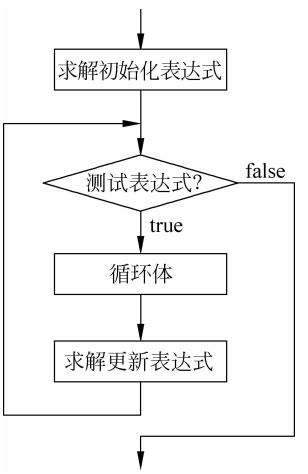

图 3-5 for 语句的执行流程图

关于 for 语句的几点说明如下。

(1) 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。

(2) 初始化表达式一般用于给循环控制条件赋初值，也可以是与循环控制变量无关的其他表达式。例如：

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

(3) 初始化表达式和更新表达式可以是任何表达式。例如：

```
for(sum = 0,i = 1;i <= 100;i++) sum = sum + i;    //初始化表达式为逗号表达式
```

(4) 更新表达式一般用于改变循环控制条件。例如，上述代码中的更新表达式 i++，每循环一次使 i 递增，用于改变循环的条件，当 $i=100$ 时，结束循环。

(5) 如果省略初始化表达式和更新表达式，只有测试表达式，则完全等同于 while 语句。例如，下列两个程序片断完全等同：

```
① for (;i <= 60;)                    ②while(i <= 60)
      sum += i++;                          sum += i++;
```

for 循环使得所有的循环细节都可在语句中描述，简单方便，用 for 语句几乎可以解决编程中的所有循环问题。

现在用 for 重做例 3-3 求自然数 1～100 之和，程序代码如下：

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

```
{
    int i = 1,sum = 0;
    for(i = 1;i <= 100;i++)
    {
        sum += i;
    }
    cout <<"sum = "<< sum << endl;
    return 0;
}
```

上述程序中的程序段：

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

可以缩写成：

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

没有最好，只有更好，不断地优化、精炼程序是提高编程水平的很好方法。相信有一天，你会突然发现自己编写的程序越来越专业了。

3.2.4　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，无限循环就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而且容易理解。

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 3-4】 统计输入的字符个数。

```
#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 <<"个字母。"<< endl << endl;
    return 0;
}
```

程序运行情况：

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

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个数的功能，注意二者的区别。

```
#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<<endl;
    return 0;
}
```

上述程序中，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。

3.2.5 循环嵌套

如果要解决更大的问题，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、do while 和 for 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 3-5】 输出乘法九九表。

*	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：乘法九九表是一个二维表，在屏幕上输出乘法九九表相当于从上到下、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。

现在，如果想输出如下所示的乘法九九表，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";
```

3.3 其他控制语句

3.3.1 break 语句

break 语句只用于 switch 语句或循环体中时，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。

3.3.2 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。

现在对例 3-2 考试成绩分级程序进行修改，注意程序中通过使用 while、break 和 continue 语句，实现了键盘输入信息控制循环的目的。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<<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 << endl;
    }
    return 0;
}
```

运行这段程序可实现多次输入学生百分制成绩,并分别对其进行成绩分级,直到输入－1结束程序。

3.3.3　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 的语法格式为:

```
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,看下面的代码。

```
...
    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 << 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 3-6】　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;
}
```

程序运行情况 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
```

程序运行情况 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，应该少用或不用。

3.4 程序控制编程案例

【例 3-7】 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 的范围内寻找水仙花数，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，这是一个已知循环次数的循环，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;
}
```

程序运行情况：

```
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 3-8】 输入一个年份，判断是否闰年。

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;
}
```

程序运行情况：

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

【例 3-9】 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),然后通过输入 1～3 选择不同的图形,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;
}
```

程序运行情况：

```
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 3-10】 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，都像“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{
        cout<<"请输入菱形的行数(奇数): ";
        cin>>k;
        if(k%2==0)
        {
            cout<<"输入数据有错,请重新输入!";
            continue;
        }
        else
            break;
    }while(true);

    //菱形的上三角
    n=(k+1)/2;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)            //输出每行前面的空格
            cout<<"";
```

```
        cout << setw(N)<<'A';                    //输出左边的 A

        for(j = 1;j < i;j++)                     //输出中间的 B
            cout << setw(N)<<'B';

        if(i!= 0)cout << setw(N)<<'A';           //输出右边的 A
        cout << endl;
    }

    //菱形的下三角
    n = n - 1;
    for(i = n;i > 0;i -- )
    {
        for(j = 0;j < n - i + 1;j++)
            cout <<"";

        cout << setw(N)<<'A';

        for(j = 2;j < i;j++)
            cout << setw(N)<<'B';

        if(i!= 1)cout << setw(N)<<'A';
        cout << endl;
    }
    return 0;
}
```

【例 3-11】 求 $a\sim b$ 的数段内所有的素数。

问题分析：本例题分两步考虑：首先判断给定的一个正整数 m 是否为素数，然后求数段 $a\sim b$ 内所有的素数。

(1) 判断任意给定的一个正整数 m 是否为素数。

素数是指只能被 1 和它自己整除的数。如果正整数 m 依次除以 2,3,…,$m-1$,结果都除不尽(有余数),则 m 肯定是一个素数；反之,如果正整数 m 能被 2,3,…,$m-1$ 中的任何一个数除尽(只要除法中有一次余数为零),则 m 肯定不是一个素数。其实数学已经证明：只要正整数 m 不能被 2～$\sqrt{m}$ 内的数整除,则 m 就是素数。程序代码如下：

```
#include < iostream >
#include < math.h >
using namespace std;
int main()
{
    long m;
    cout <<"Input a number:";
    cin >> m;
    double sqrtm = sqrt(m);                  //该函数包含在 math.h
    for(int i = 2;i < = sqrtm;i++)
        if(m % i == 0)                       //只要正整数 m 能被一个数整除,就退出此循环
            break;                           //此时,i < = sqrtm 或 i > sqrtm 的值肯定为 false
    if(i > sqrtm)                          //当 i > sqrtm 时,表示上述循环正常结束,没有被中断过
        cout << m <<" is prime.\n";
    else
```

```
        cout << m << " is not prime.\n";
    return 0;
}
```

程序测试 1：

```
Input a number:53
53 is prime.
```

这时，sqrtm＝sqrt(53)＝7.28，i＝int(sqrt(53))＋1＝7＋1＝8，所以 i＞sqrtm。因此，输出 53 is prime。

程序测试 2：

```
Input a number:9
9 is not prime.
```

这时，sqrtm＝sqrt(9)＝3，当 i＝3 时，9％3＝＝0，退出循环，所以 i＝sqrtm，即 i＞sqrtm 的值为 false。因此，输出 9 is not prime.

程序测试 3：

```
Input a number:81
81 is not prime.
```

这时，sqrtm＝sqrt(81)＝9，当 i＝3 时，81％3＝＝0，退出循环，所以 i＜sqrtm，即 i＞sqrtm 的值为 false。因此，输出 81 is not prime.

(2) 求数段 $a \sim b$ 内所有的素数。

因为要求找出数段 $a \sim b$ 内的所有素数，所以需要对 $a \sim b$ 之间的所有奇数(偶数能被 2 整除，所以肯定不是素数)都进行上述是否为素数的判断，这时只需要在上述判断素数的程序代码外加一个循环，即循环嵌套，就可以实现查找数段 $a \sim b$ 内所有素数的目的。程序代码如下：

```
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
int main()
{
    long a,b,m;
    int i,l = 0;

    cout <<"Input two number:";
    cin >> a >> b;
    cout <<"Primes from "<< a <<" to "<< b <<" is:";
    if(a % 2 == 0)
        a++;
    for(m = a;m <= b;m += 2)
    {
        double sqrtm = sqrt(m);
        for(i = 2;i <= sqrtm;i++)
            if(m % i == 0)
                break;
```

```
        if(i > sqrtm)
        {
            if(l++ % 10 == 0)                    //10 个数换行
                cout << endl;
            cout << setw(5)<< m;
        }
    }
    cout << endl;
    return 0;
}
```

程序运行情况：

```
Input two number:2  10
Primes from 2to 10 is:
    3    5    7
```

【例 3-12】 输出三角数列。

(1) 输出以下简单的三角数列。

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

问题分析：输出这个三角数列需要双层循环，外循环控制行，内循环控制列。每一行又分为两部分，首先输出每行前面的空格，其次输出该行的数字。这个三角数列之所以比较简单是因为三角形的左右两边都是 1，中间的数字正好是其所在的行号。程序代码如下：

```
#include < iostream >
#include < iomanip >
using namespace std;
const int N = 4;
int main()
{
    int n,i,j;
    cout <<"请输入三角数列的行数: ";
    cin >> n;
    for(i = 0;i < n;i++)
    {
    for(j = 0;j < n - i - 1;j++)                 //输出每行前面的空格
        cout <<"";

    cout << setw(N)<< 1;                         //输出左边的 1,此宽度是上面的 2 倍

    for(j = 1;j < i;j++)                         //输出中间的数
        cout << setw(N)<< i;

    if(i!= 0)cout << setw(N)<< 1;                //输出右边的 1
```

```
        cout << endl;
    }
    return 0;
}
```

（2）输出以下杨辉三角数列。

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
      ...
```

问题分析：杨辉三角数列与上面介绍的三角数列差不多，只是杨辉三角数列中间的数字的规律比较复杂。程序代码如下：

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int n,i,j,c;
    cout <<"请输入扬辉三角的行数: ";
    cin >> n;

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n - i - 1;j++)            //输出每行前面的空格
            cout <<" ";

        cout << setw(6)<< 1;                    //输出左边的 1

        for(j = 1,c = 1;j <= i;j++)             //输出右边的各数
        {
            c = c * (i - j + 1)/j;
            cout << setw(6)<< c;
        }

        cout << endl;
    }
    return 0;
}
```

3.5　小结与知识扩展

3.5.1　小结

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。

(1) 顺序控制结构比较简单,它是系统预置的,是指按程序编写的顺序一条一条地执行语句。

(2) 选择控制结构根据测试表达式的值决定程序的执行方向,常用的语句有以下两种。

① 基本的 if else 选择语句,其形式如下:

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

② switch 语句用于多路选择,其形式如下:

```
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
}
```

(3) 循环控制结构支持程序中需要重复执行的操作,循环分有限次循环和无限循环。

有限次循环就是循环次数确定的循环,构造有限次循环的三要素为循环控制变量初始化、循环结束条件和循环控制变量更新。for 语句的形式如下:

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

它将有限次循环的三要素集中描述,使用更灵活,程序更精练。

无限循环就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。构建无限循环通常采用 while 语句,其形式如下:

```
while(测试表达式)
    循环体
```

或 do while 语句,其形式如下:

```
do
    循环体
while(测试表达式);
```

一般采用在循环体内输入信息,再通过选择控制语句的测试表达式确定是否跳出循环。

3.5.2 字符函数库

C++从 C 语言继承了一个与字符相关的、非常方便的函数软件包,它可以简化诸如确定字符是否为大写字母、数字、标点符号等操作。这些函数的原型都是在 cctype 头文件中定义的。常用的字符函数如表 3-1 所示。

表 3-1　常用的字符函数

函 数 名 称	返　回　值
isalnum()	如果参数是字母或数字，函数返回 true
isalpha()	如果参数是字母，函数返回 true
iscntrl()	如果参数是控制字符，函数返回 true
isdigit()	如果参数是数字，函数返回 true
isgraph()	如果参数是除空格之外的打印字符，函数返回 true
islower()	如果参数是小写字母，函数返回 true
isprint()	如果参数是打印字符，函数返回 true
ispunct()	如果参数是标点符号，函数返回 true
isspace()	如果参数是标准空白字符，如空格、回车、制表符等，函数返回 true
isupper()	如果参数是大写字母，函数返回 true
isxdigit()	如果参数是十六进制数字，函数返回 true
tolower()	如果参数是大写字母，则函数返回其小写字母，否则返回该参数
toupper()	如果参数是小写字母，则函数返回其大写字母，否则返回该参数

使用这些函数比使用逻辑运算符更方便。例如，测试字符变量 ch 是不是字母字符的代码：

```
if((ch>='a'&&ch<'z')||(ch>='A'&&ch<'Z')        //使用运算符"&&"
if(isalpha(ch))                                //使用 isalpha()
```

【例 3-13】 字符函数的使用。

```
#include<iostream>
#include<cctype>
using namespace std;
int main()
{
    int space=0,digits=0,chare=0,punct=0,others=0;
    char ch;
    cout<<"请输入一个字符串,以@结束:";
    cin.get(ch);
    while(ch!='@')
    {
        if(isalpha(ch))                        //如果是字母
            chare++;
        else if(isspace(ch))                   //如果是空格
            space++;
        else if(isdigit(ch))                   //如果是数字
            digits++;
        else if(ispunct(ch))                   //如果是标点符号
            punct++;
        else                                   //如果是其他字符
            others++;
        cin.get(ch);
    }
    cout<<chare<<" letters,"
```

```
        << space << " space,"
        << digits << " digits,"
        << punct << " punctuations,"
        << others << " others.\n";
    return 0;
}
```

习题

3-1　选择题

(1) 设 i、j、k 均为 int 型变量，则执行完以下 for 循环语句后，k 的值是[　　]。

```
for(i = 0,j = 10;i <= j;i++,j--)
    k = i + j;
```

A. 100　　B. 10　　C. 12　　D. 7

(2) while(!x)中的(!x)与下列条件[　　]等价。

A. x==0　　B. x==1　　C. x!=1　　D. x!=0

(3) 若 x、y 是 int 型变量，则执行以下语句后，x 的值为[　　]、y 的值为[　　]。

```
for(y = 1,x = 1;y <= 50;y++)
{   if(x >= 10)break;
        if(x % 2 == 1)
            {x += 5;continue;}
        x -= 3;
}
```

[1] A. 1　　B. 6　　C. 7　　D. 10

[2] A. 6　　B. 2　　C. 4　　D. 8

3-2　简答题

(1) if(x=3)和 if(x==3)两条语句的差别是什么？

(2) 简述 break 语句在循环语句和 switch 语句中的作用。

(3) 用传统流程图表示求解以下问题的算法。

① 找出两数之间不能被 2 和 3 整除的数。

② 找出三个数 a、b、c 中最大的数。

③ 求 1+2+3+…+100。

④ 求 $n!$。

⑤ 求两个数 m 和 n 的最大公约数和最小公倍数。

3-3　阅读下列程序，写出程序运行结果

(1) 程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int m,sum = 0;
```

```
    for(m = 10;m <= 100;m++)
    {
        if(m % 3 == 0||m % 2 == 0) continue;
            cout << m <<'\t';
    }
    cout << endl;
    return 0;
}
```

程序的运行情况：

(2) 程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int m = 8,n = 12,r,p,temp;
    if(m > n)
    {
        temp = m;
        m = n;
        n = temp;
    }
    p = n * m;
    while(m!= 0)
    {
        r = n % m;
        n = m;
        m = r;
    }
    cout << n << endl;
    cout << p/n << endl;
    return 0;
}
```

程序的运行情况：

3-4　阅读下列程序说明和程序代码，填空完成程序

(1) 下列程序的作用是求以下算式中 X、Y、Z 的值，请填入正确的内容。

$$XYZ+YZZ=532$$

程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int x,y,z,i,result = 532;
    for(x = 1; ①  ;x++)
        for(1 = 1; ②  ;y++)
            for(z = ③ ; ④ ;z++)
            {
```

```
                i = (  ⑤  ) + (100 * y + 10 * z + z);
                if(i == result)
                    cout <<"x = "<< x <<"   y = "<< y <<"   z = "<< z << endl;
            }
    return 0;
}
```

(2) 有 1020 个西瓜,第一天卖一半多两个,以后每天卖剩下的一半多两个,问几天以后能卖完? 完成下列程序。

```
#include < iostream >
using namespace std;
int main()
{
    int day,x1,x2;
    day = 0;x1 = 1020;
    while(  ①  )
    {     x2 =  ②  ;x1 = x2;day++;     }
    cout <<"day = "<< day << endl;
    return 0;
}
```

(3) 根据以下函数关系,对输入的每个 x 值,计算出相应的 y 值。请在程序中填空。

$$y=\begin{cases}0 & x<0\\ x & 0\leqslant x<10\\ 10 & 10\leqslant x<20\\ -0.5x+20 & 20\leqslant x<40\\ -2 & 40\leqslant x\end{cases}$$

```
#include < iostream >
using namespace std;
int main()
{
    int x,c;
    float y;
    cin >> x;
    if(  ①  )c = -1;
    else c =  ②  ;
    switch (c)
    {
        case -1:y = 0;break;
        case 0:y = x;break;
        case 1:y = 10;break;
        case 2:
        case 3:y = -0.5 * x + 20;break;
        default:y = -2;
    }
    if(  ③  ) cout <<"y = "<< y << endl;
    else cout <<"error\n";
```

```
    return 0;
}
```

程序运行情况：

```
请输入一个字符串,以@结束:I am a student. My age is 18.@
18 letters,7 space,2 digits,2 punctuations,0 others.
```

3-5　编程题

(1) 有一函数：

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

编程实现输入一个 x 的值，输出 y 的值。

(2) 编程求 1000 之内的所有“完数”。所谓“完数”是指一个数恰好等于它的因子之和。例如 6 是完数，因为 6＝1＋2＋3。

(3) 打印三个相邻的字母。

(4) 猴子吃桃问题：有一天，猴子摘了一些桃子，吃了一半，觉得不过瘾，就多吃了一个。以后每天如此，到第十天想吃时，发现就只剩下一个桃子。请计算第一天猴子摘的桃子的个数。

(5) 小学生算术加法测试：随机自动生成 10 道小学生算术加法试题(100 以内)，学生每做对一道加 10 分，并给出“正确”信息；做错时，提示“错误，再做一遍”，最后给出学生的测试成绩(使用 cstdlib.h 中的函数 rand())。

(6) 换硬币：将一个 1 元人民币换成 5 分、2 分、1 分的硬币，有多少种换法？

(7) 谁打烂了玻璃：有四个孩子踢球，不小心打烂了玻璃，老师问是谁干的？

A 说：不是我

B 说：是 C

C 说：是 D

D 说：他胡说

现已知三个孩子说的是真话，一个孩子说谎。根据这些信息，编程找出打烂玻璃的孩子。