第3章

C/C++流程控制

结构化程序的基本结构包括顺序结构、选择结构和循环结构。结构化程序解决任何一个实际问题，无论简单或复杂，都可以由这三种基本结构来完成。本章主要介绍上述三种基本结构的语句及一些常用的算法。

通过本章学习，应该重点掌握以下内容：

- 结构化程序设计的程序结构。
- 选择结构和实现选择结构的语句。
- 循环结构和实现循环结构的语句。
- 结构化程序设计的典型算法和应用。

3.1 算法与流程图

人们在处理日常生活中的一些事情时，都有一定的方法和步骤，先做哪一步，后做哪一步。例如，邮寄一封信，大致可以将寄信的过程分为以下 4 个步骤：写信、写信封、贴邮票、投入信箱。将信投入信箱后，寄信过程结束。同样，在面向过程的程序设计中，程序设计者必须指定计算机执行的具体步骤，如何设计这些步骤，如何保证它的正确性和高效率，就是算法需要解决的问题。

3.1.1 算法的概念

所谓算法，是指为了解决一个问题而采取的方法和步骤。当利用计算机来解决一个具体问题时，也要首先确定算法。

对于同一个问题，往往会有不同的解题方法。

例如，要计算 $S = 1 + 2 + 3 + \cdots + 100$，可以先进行 1 加 2，再加 3，再加 4，一直加到 100，得到结果 5050，也可以采用另外的方法，$S = (100 + 1) + (99 + 2) + (98 + 3) + \cdots + (51 + 50) = 101 \times 50 = 5050$。当然，还可以有其他方法。比较两种方法，显然第二种方法比第一种方法简单。所以，为了有效地解决问题，不仅要保证算法正确，还要考虑算法质量，要求算法简单、运算步骤少、效率高，能够迅速得出正确结果。

利用计算机解决问题，实际上也包括了设计算法和实现算法两部分工作。首先设计出解决问题的算法，然后根据算法的步骤，利用程序设计语言编写出程序，在计算机上调试运行，得出结果，最终实现算法。可以这样说，算法是程序设计的灵魂，而程序设计语言是表达算法的形式。

3.1.2 算法的描述

表示算法的形式有很多种，其中有流程图和 N-S 图。

1. 流程图

流程图由一些特定意义的图形、流程线及简要的文字说明构成，它能清晰、明确地表示程序的运行过程，传统流程图由如图 3-1 所示的图形组成。

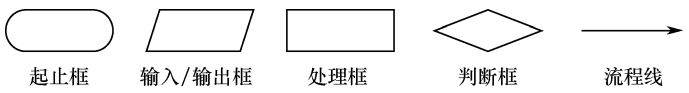


图 3-1　传统流程图的常用图形

（1）起止框：说明程序的起点和结束点。

（2）输入/输出框：输入、输出操作步骤写在该框中。

（3）处理框：算法大部分操作写在此框中，如下面的处理框用于实现加 1 操作。

i←i+1

（4）判断框：代表条件判断，以决定如何执行后面的操作。

例如，网上购物的流程图如图 3-2 所示。

2．N-S 图

在使用流程图的过程中，人们发现流程线不一定是必需的，为此设计了一种新的流程图——N-S 图，它是一种较理想的方式。它是 1973 年由美国学者 I.Nassi 和 B.Shneiderman 提出的。在这种流程图中，全部算法写在一个大矩形框内，该框中还可以包含一些从属于它的小矩形框。例如，网上购物的 N-S 图如图 3-3 所示，N-S 图可实现传统流程图功能。

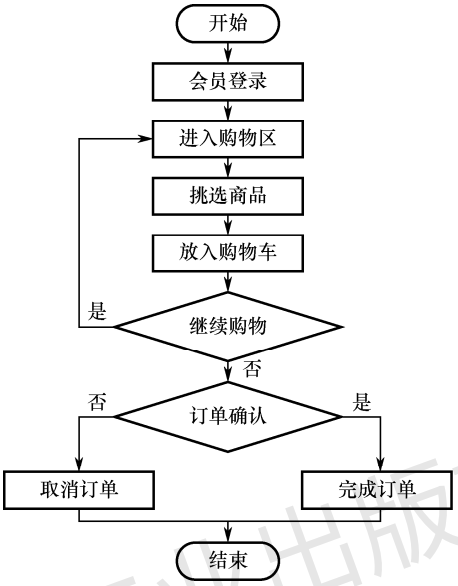


图 3-2　网上购物的流程

图 3-3　网上购物的 N-S 图

注意，在 N-S 图中，基本元素框在流程图中的上下顺序就是执行时的顺序，程序在执行时，也按照从上到下的顺序进行。

对初学者来说，先画出流程图很有必要。根据流程图编写程序，会避免不必要的逻辑错误。本书采用 N-S 图表示算法。

3.2　语句和程序的三种基本结构

3.2.1　语句

语句（statement）是程序中最小的可执行单位。一条语句可以完成一种基本操作，若干条语句组合在一起就能实现某种特定的功能。C/C++中的语句可以分为以下 4 种形式。

1．表达式语句

在任何一个表达式后面加上分号“;”就构成一条简单的 C/C++表达式语句。例如：

```
c=a+b;
b++;
```

2．空语句

仅由单个分号构成的语句，称为空语句。

空语句不进行任何操作。该语句被用在从语法上需要一条语句，但实际却又不进行任何操作的地方。

3．复合语句

复合语句是用一对花括号{ }括起来的语句块。复合语句在语法上等效于一个单一语句。

使用复合语句应注意：

（1）花括号必须配对使用。

（2）花括号外不要加分号。

4．控制语句

控制语句改变程序执行的方向，如 if 语句、for 语句等。

3.2.2　程序的三种基本结构

在过程化程序设计中，按照结构化设计的思想，程序由三种基本结构构成，它们分别是顺序结构、选择结构和循环结构。

1. 顺序结构

程序按照语句的书写顺序依次执行，语句在前的先执行，语句在后的后执行，顺序结构只能满足设计简单程序的要求。

2. 选择结构（也称分支结构）

在选择结构中，程序根据判断条件是否成立来选择执行不同的程序段。也就是说，这种程序结构，能有选择地执行程序中的不同程序段。

3. 循环结构

在循环结构中，程序根据判断条件是否成立来决定是否重复执行某个程序段。

程序的执行流程和顺序是由程序中的控制语句来完成的，而控制流程的主要方式是分支和循环。需要说明的是，基本结构之间是可以互相嵌套的，在一个基本结构中可以包含一个或多个基本结构。

3.2.3　结构化算法

解决任何一个复杂的问题，都可以由三种基本结构来完成。由这三种基本结构构成的算法称为结构化算法，它不存在无规律的转移，只有在本结构内才允许存在分支或向前、向后的跳转。由结构化算法编写的程序称为结构化程序。结构化程序便于阅读和修改，提高了程序的可读性和可维护性。

3.3 顺序结构程序

顺序结构是程序设计中最简单、最常用的基本结构。C/C++程序是由一条条语句组成的，在顺序结构中，各语句按照出现的先后顺序依次执行。顺序结构是任何程序主体的基本结构，即使在选择结构或循环结构中，也常以顺序结构作为其子结构。

顺序结构流程图如图 3-4（a）所示，N-S 图如图 3-4（b）所示。

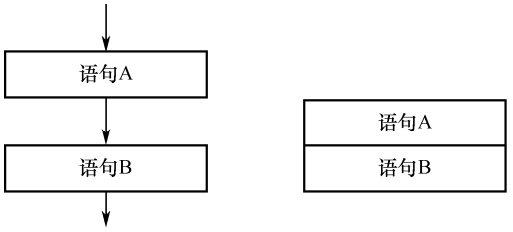


图 3-4　顺序结构流程图

【例 3.1】 “鸡兔同笼”问题。鸡有 2 只脚，兔有 4 只脚，如果已知鸡和兔的总头数为 h，总脚数为 f。问笼中鸡和兔各有多少只？

分析：设笼中有鸡 x 只，兔 y 只，由条件可得方程组：

$$\begin{cases} x+y=h \\ 2x+4y=f \end{cases}$$

解方程组得：

$$\begin{cases} x=\dfrac{4h-f}{2} \\ y=\dfrac{f-2h}{2} \end{cases}$$

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main( )
{
    int h,f,x,y;
    cout<< " 请输入鸡和兔的总只数: " ;
    cin>>h;
    cout<< " 鸡和兔的总脚数（偶数）: " ;
    cin>>f;
    x = (4 * h-f) / 2;
    y = (f-2 * h) / 2;
    cout<< " 则笼中鸡有 "  << x << " 只，兔有 "  <<y<<  " 只。 " <<endl;
    return 0;
}
```

运行程序，输入数据后，程序将计算出结果。

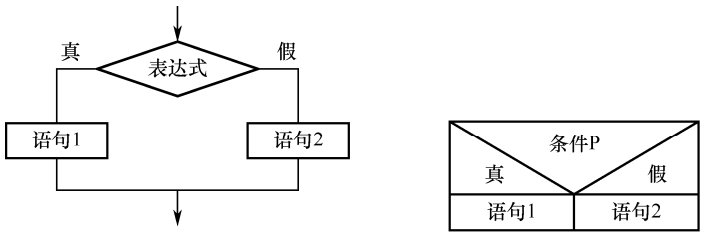
```
请输入鸡和兔的总只数:30↙          (输入 h 的值，↙表示回车)
鸡和兔的总脚数（偶数）:100↙        (输入 f 的值)
则笼中鸡有 10 只，兔有 20 只。
```

注意，本书中运行时输入部分加下画线，以区分 cout 输出的信息。

3.4 选择结构程序

在信息处理、数值计算及日常生活中，经常会碰到需要根据特定情况选择某种解决方案的情况。选择结构是在计算机语言中用来实现上述分支现象的方法，它能根据给定条件，从事先编写好的各个不同分支中执行并且仅执行某一分支的相应操作。

选择结构又称为分支结构，其流程图如图 3-5（a）所示，N-S 图如图 3-5（b）所示。该结构能根据表达式（条件 P）成立与否（真或假），选择执行语句 1 或语句 2。



（a）选择结构流程图　　（b）选择结构N-S图

图 3-5　选择结构

为实现选择结构，C/C++提供以下三种分支语句：if 语句、嵌套 if 语句、switch 语句。

3.4.1 if 语句

if 语句的语法格式为：

```
if(表达式)
    语句 1;
else
    语句 2;
```

如果表达式为真（或非 0 值），则执行语句 1；若表达式为假（或 0 值），则执行语句 2。执行的流程图如图 3-5 所示。

例如，从 *x* 和 *y* 中选择较大的一个值并输出。

```
if (x>y)
    cout<<x;
else
    cout<<y;
```

if 语句的语句 2 可以为空。当语句 2 为空时，else 可以省略，写为如下形式：

```
if(表达式)    语句 1;
```

【例 3.2】 输入一个年份，判断是否为闰年。闰年的年份必须满足以下两个条件之一：

（1）能被 4 整除，但不能被 100 整除；

（2）能被 400 整除。

分析：设变量 year 表示年份，判断 year 是否满足以上两个条件。

条件（1）的逻辑表达式是：year%4==0&&year%100!=0

条件（2）的逻辑表达式是：year%400==0

两者取“或”，即得到判断闰年的逻辑表达式为：

```
(year%4= =0&&year%100!=0)||year%400= =0
```

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main( )
{   int year;
    cout<< " 输入年份: " <<endl;
    cin>>year;
    if(year%4==0 && year%100!=0 || year%400==0)     //注意运算符的优先级
        cout<<year<< " 是闰年 " <<endl;
    else
        cout<< year<< " 不是闰年 " <<endl;
    return 0;
}
```

【例 3.3】 输入 3 个不同的数，将它们从大到小排序输出。

分析：

（1）先将 *a* 与 *b* 比较，把较大者放入 *a* 中，较小者放入 *b* 中。

（2）再将 *a* 与 *c* 比较，把较大者放入 *a* 中，较小者放入 *c* 中，此时 *a* 为三者中的最大者。

（3）最后将 *b* 与 *c* 比较，把较大者放入 *b* 中，较小者放入 *c* 中，此时 *a*、*b*、*c* 已按由大到小的顺序排列。其 N-S 图如图 3-6 所示。

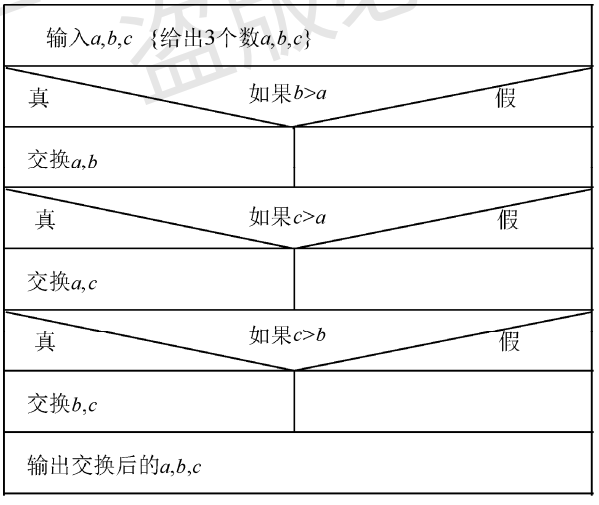


图 3-6　对 3 个数从大到小排序的 N-S 图

根据 N-S 图编写代码：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main( )
```

```
{
   int a,b,c,t;
   cin>>a>>b>>c;
    if (b > a){ t = a; a = b; b = t;}
    if (c > a){ t = a; a = c; c = t;}
    if (c > b){t = b; b = c; c = t;}
   cout<< "   从大到小排序输出：  " <<a<<'\t'<<b<<'\t'<<c<<endl;
   return 0;
}
```

3.4.2 嵌套 if 语句

if 语句中，如果内嵌语句又是 if 语句，就构成了嵌套 if 语句。一个 if 语句可实现二选一分支，而嵌套 if 语句则可以实现多选一的多路分支情况。

嵌套有两种形式，第一种是嵌套在 else 分支中，形成 if…else…if 语句：

```
if(表达式 1)      语句 1;
      else if (表达式 2)      语句 2;
      else if (表达式 3)      语句 3;
      …
      else if (表达式 n)      语句 n;
      else   语句 n+1；
```

执行的流程图如图 3-7 所示。

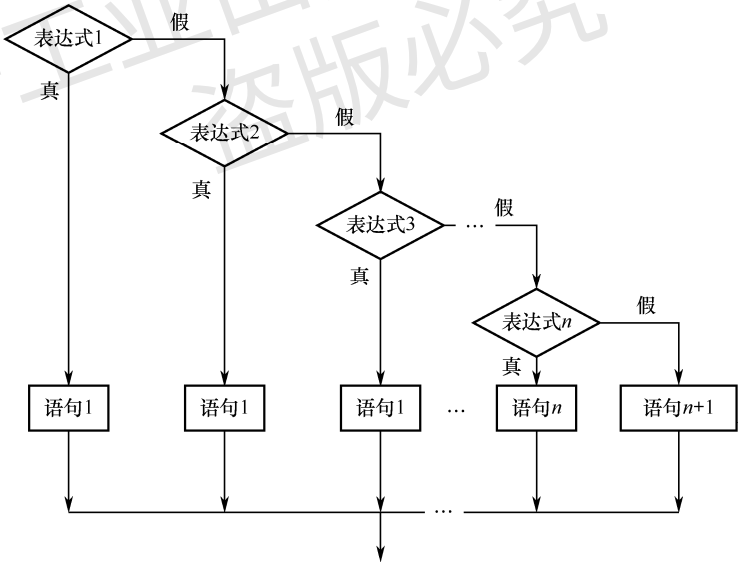


图 3-7　嵌套在 else 分支中的流程图

【例 3.4】 输入学生的成绩 score，按分数输出其等级：score≥90 为优，90>score≥80 为良，80>score≥70 为中等，70>score≥60 为及格，score<60 为不及格。

程序如下：

```
#include <iostream>
```

```
using namespace std;                                    //标准命名空间 std
int main()
{
   int score;
   cout <<  " 请输入成绩 " ;
   cin >> score;
   if (score >= 90)
        cout <<  " 优 "  << endl;
   else if (score >= 80)
        cout <<  " 良 "  << endl;
   else if (score >= 70)
        cout <<  " 中 "  << endl;
   else if (score >= 60)
        cout <<  " 及格 "  << endl;
   else
        cout <<  " 不及格 "  << endl;
   return 0;
}
```

第二种是嵌套在 if 分支中，形成 if…if…else 语句：

```
if (表达式 1)
     if (表达式 2)  语句 1；
     else    语句 2;
```

要特别注意 else 和 if 的配对关系。C/C++规定了 if 和 else 的“就近配对”原则，即相距最近且还没有配对的一对 if 和 else 首先配对。按上述规定，第二种嵌套形式中的 else 应与第二个 if 配对。如果根据程序的逻辑需要改变配对关系，则要将属于同一层的语句放在一对“{}”中。如第二种嵌套形式中，要让 else 和第一个 if 配对，则语句必须写成：

```
if (表达式 1)
{
     if (表达式 2)    语句 1 ;
}
else  语句 2 ;
```

第二种嵌套形式较容易产生逻辑错误，而第一种嵌套形式的配对关系则非常明确，因此从程序可读性角度出发，建议尽量使用第一种嵌套形式。

请看以下两个语句。

语句 1：

```
if(n%3==0)
   if(n%5==0)
        cout<<n<<  " 是 15 的倍数 " <<endl;
   else
        cout<<n<< " 是 3 的倍数但不是 5 的倍数 " <<endl;
```

语句 2：

```
if(n%3==0){
   if(n%5==0)
```

```
        cout<<n<< " 是 15 的倍数 " <<endl;
}
else
    cout<<n<< " 不是 3 的倍数 " ;
```

两个语句的差别只在于一个“{}”，但表达的逻辑关系却完全不同。

if 语句在某些情况下可以用条件运算符“?:”来简化表达。

例如，求两个数 *a*、*b* 中较大的数，采用 if 语句如下：

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

也可用条件运算符实现：

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

【例 3.5】 输入一个字符，判别它是否为大写字母，如果是，则它转换成小写字母；如果不是，则不转换。然后输出最后得到的字符。

分析：将大写字母转换成小写字母，由于小写字母的 ASCII 值比大写字母的 ASCII 值大 32，所以加 32 可以实现大写字母转换成小写字母。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main( )
{
   char ch;
   cin>>ch;
   ch=(ch>='A' && ch<='Z')?(ch+32):ch;      //判别 ch 是否为大写字母，是则转换
   cout<<ch<<endl;
   return 0;
}
```

3.4.3 switch 语句

用嵌套 if 语句可以实现多分支的情况。另外，C/C++中还提供了一个 switch 语句，称为多分支语句（开关语句），可以用来实现多选一程序结构。

switch 语句语法格式为：

```
switch (表达式)
{
   case   常量表达式 1: 语句序列 1;     break;
   case   常量表达式 2: 语句序列 2;     break;
                …
   case   常量表达式 n: 语句序列 n;     break;
   default:语句序列 n+1;
}
```

说明：

（1）当 switch 后面括号中表达式的值与某一个 case 分支中的常量表达式匹配时，就执行该分支。如果所有的 case 分支中的常量表达式都不能与 switch 后面括号中表达式的值匹配，则执行 default 分支。

（2）break 语句可选，如果没有 break 语句，则每一个 case 分支都只作为开关语句的执行入口，执行完该分支后，还将接着执行其后的所有分支。因此，为保证逻辑的正确实现，通常每个 case 分支都与 break 语句联用。

（3）每个常量表达式的取值必须各不相同，否则会引起歧义。各 case 后面必须是常量，而不能是变量或表达式。各个 case（包括 default）分支出现的次序可以任意，通常将 default 分支放在最后，并且 default 分支是可选的。

（4）允许多个常量表达式共用同一个语句序列。

例如：

```
char score;
cin>>score;
switch (score) {
    case   'A':
    case   'a': cout<< " excellent " ;
                break;
    case   'B':
    case   'b': cout<< " good " ;
                break;
    default: cout<< " fair " ;
}
```

（5）从形式上看，switch 语句的可读性比嵌套 if 语句好，但不是所有多选一的问题都可由开关语句完成，这是因为开关语句中限定了 switch 后面括号中表达式的类型，它的类型必须是整型、字符型、枚举类型，而不能是浮点型。

【例 3.6】 用 switch 语句实现例 3.4 的功能。

分析： 由题意可知，10 分为一个等级，采用 score/10 整除可以得到等级。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
    int score;
    int a;
    cout<< " Input score(0~100): " ;
    cin>>score;
    a=score/10;
    switch(a)
    {   case   10:
        case   9: cout <<  " 优  "  << endl; break;
        case   8: cout <<  " 良  "  << endl; break;
        case   7: cout <<  " 中  "  << endl; break;
        case   6: cout <<  " 及格  "  << endl; break;
        default: cout<< " 不及格 "  << endl;
    }
```

```
    return 0;
}
```

【例 3.7】 switch 语句应用。设计一个计算器程序，实现加、减、乘、除运算。

分析：由于加、减、乘、除是 4 种运算，所以需要使用多分支语句。

程序如下：

```
#include <iostream.h>
int main( )
{   float num1,num2,result;
    char op;
    cout<< " 输入操作数 1，运算符，操作数 2： " <<endl;
    cin>>num1>>op>>num2;
    switch(op){
        case   '+' : result = num1+num2; break;
        case   '-' : result = num1-num2; break;
        case   '*' : result = num1*num2; break;
        case   '/' : result = num1/num2; break;
        default :   cout<<op<< " 是无效运算符! " ;
    }
    if(op=='+'||op=='-'||op=='*'||op=='/')
        cout<<num1<<op<<num2<< " = " <<result<<endl;
    return 0;
}
```

程序运行结果：

```
输入操作数 1，运算符，操作数 2：
5+8↙                    (输入 num1，op，num2 的值)
5+8=13
```

可以使用这个程序输入算术题目，获得正确答案。当我们运行这个程序时，每次仅能得到一道题目的答案，能否获取多道题目的正确答案呢，那就需要使用循环结构来设计程序。

3.5 循环结构程序设计

当需要在指定条件下反复执行某一操作时，可以用循环结构来实现。使用循环可以简化程序，提高效率。C/C++提供了以下三种循环语句：while 语句、do-while 语句和 for 语句。

3.5.1 while 语句

while 语句的语法格式为：

```
while (表达式)
{
    循环体语句
}
```

其作用是：当指定的条件表达式为真（非 0）时，执行 while 语句中的循环体语句。其流程图和 N-S 图如图 3-8 所示。其特点是先判断表达式，后执行语句。while 循环又称为当型循环。

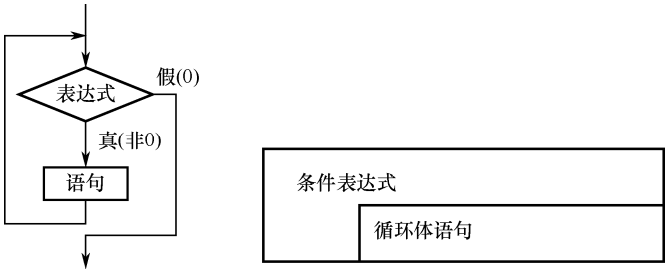


图 3-8　while 循环结构流程图和 N-S 图

【例 3.8】 求 1+2+3+…+100。

分析：计算累加和需要两个变量，变量 *sum* 存放累加和，变量 *i* 存放加数。重复将加数 *i* 加到 *sum* 中，根据分析可画出 N-S 图，如图 3-9 所示。

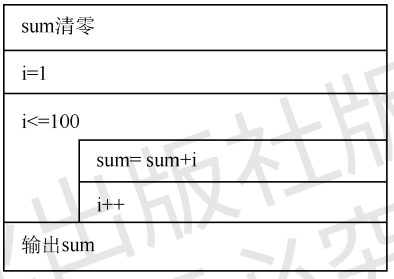


图 3-9　累加中的 N-S 图

根据流程图写出程序：

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

程序运行结果：

```
    sum=5050
```

说明：

（1）循环体如果包含一条以上的语句，则应该用花括号括起来，以复合语句的形式出现。如果不加花括号，则 while 语句的循环体只到 while 后面第一个分号处。

（2）在循环体中应有使循环趋向结束的语句。

【例 3.9】 求 *sum* = 1! + 2! + 3! + … + *n*!，当 *sum*≥1000 时 *n* 的值。

分析：计算阶乘累加和需要用变量 *sum* 存放累加和，变量 *t* 存放阶乘。重复将 *t* 加到 *sum* 中且变量 *t* 变成下一个数的阶乘。根据分析可画出 N-S 图，如图 3-10 所示。

程序如下：

```
#include <iostream>
using namespace std;                          //标准命名空间 std
int main( )
{
    int i=0,t=1,sum=0;
    while (sum<1000)
     {
         i++;
         t=t*i;
         sum=sum+t;
     }
     cout<< " sum= " <<sum<<'\t'<< " i= " <<i<<endl;
     return 0;
}
```

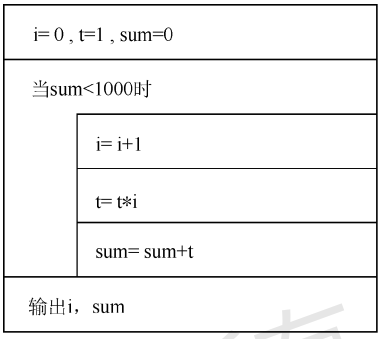


图 3-10　阶乘累加 N-S 图

程序运行结果：

```
sum=5913     i=7
```

【例 3.10】 输入一个非负的整数，将其反向后输出。例如，输入 24789，变成 98742 输出。

分析：将整数的各位数字逐个分开，一个一个分别输出。将整数各位数字分开的方法是，通过对 10 进行求余得到个位数输出，然后将整数缩小 1/10，再求余，并重复上述过程，分别得到十位、百位……，直到整数的值变成 0 为止。

程序如下：

```
#include <iostream>
using namespace std;                                  //标准命名空间 std
int main( )
{
    int n;
    cin>>n;
    while(n>0)
    {
        cout << n%10;
        n = n/10;
    }
    cout<<endl;
    return 0;
}
```

3.5.2 do-while 语句

do-while 语句的语法格式如下：

```
do
{
    循环体语句
} while (表达式);
```

do-while 语句的执行过程为：先执行一次循环体语句，然后判别表达式，当表达式的值为真（或非 0）时，继续执行循环体语句，如此反复，直到表达式的值为假（或等于 0）为止，此时循环结束。可以用图 3-11 表示其流程。

图 3-11　do-while 循环结构流程图和 N-S 图

【例 3. 11】 用 do-while 语句求 1+2+3+…+100。N-S 图如图 3-12 所示。

程序如下：

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

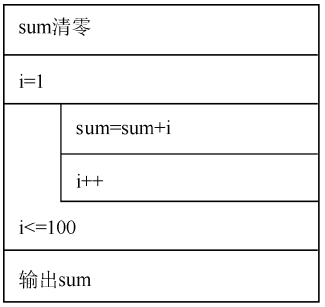


图 3-12　do-while 循环求和的 N-S 图

说明：在循环体相同的情况下，while 语句和 do-while 语句的功能基本相同。二者的区别在于，当循环条件一开始就为假时，do-while 语句中的循环体至少会被执行一次，而 while 语句则一次都不执行。

【例 3.12】 输入两个正整数，求它们的最大公约数。

分析：求最大公约数可以用“辗转相除法”，方法如下。

（1）比较两数，并使 m 大于 n。

（2）将 m 作为被除数，n 作为除数，相除后余数为 r。

（3）将 $m \leftarrow n$，$n \leftarrow r$；

（4）若 $r=0$，则 m 为最大公约数，结束循环。若 $r \neq 0$，则执行步骤（2）和步骤（3），根据此分析画出 N-S 图，如图 3-13 所示。

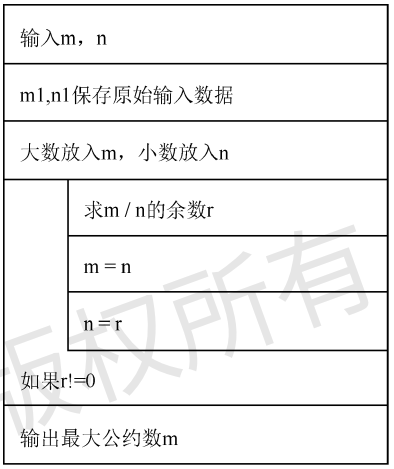


图 3-13　求最大公约数的 N-S 图

程序如下：

```
#include <iostream>
using namespace std;          //标准命名空间 std
int main( )
{
   int m,n,r,t;
   int m1,n1;
   cout<< " 请输入第 1 个数: " ;cin>>m;
   cout<< " 请输入第 2 个数: " ;cin>>n;
   m1=m;   n1 = n;
   //保存原始数据供输出使用
   if(m < n)
         {t = m; m = n; n = t; } //m,n 交换值
   do
   {
      r = m % n;
      m = n;
      n = r;
   }while(r!=0);
   cout<<m1<< " 和 " <<n1<< " 的最大公约数是 " <<m;
   return 0;
}
```

说明：

（1）由于在求解过程中，m 和 n 已经发生了变化，故可以将其保存在另外两个变量 m_1 和 n_1 中，以便输出时可以显示这两个原始数据。

（2）求两个数的最小公倍数，只需将两数相乘除以最大公约数即可，即 $m_1 \times n_1 / m$。

【例 3.13】 计算并输出下列级数和：

$$\text{sum} = \frac{1}{1\times 2} - \frac{1}{2\times 3} + \frac{1}{3\times 4} + \cdots + \frac{(-1)^{k+1}}{k(k+1)} + \cdots$$

直到某项的绝对值小于 10^{-4} 为止。

分析：从通项可以看出，相邻两项的符号相反，为此设

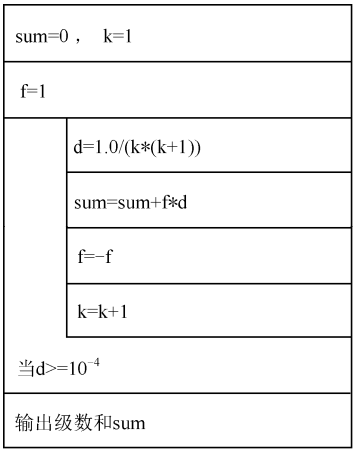


图 3-14　求级数和的 N-S 图

立 f 变量并赋初值 1，表示对应项的符号，重复进行 $f=-f$，从而得到对应项的符号。重复累加操作直到该项的绝对值小于 10^{-4} 为止。根据分析画出 N-S 图，如图 3-14 所示。

程序如下：

```
#include<stdio.h>
int main()
{
    int   k;
    double   sum,d,f;
    sum= 0;k= 1; f= 1;
    do{
      d= 1.0/(k*(k+1));
      sum= sum+f*d;
      k= k+1;
      f= -f;
    } while(d>=1.0e-4);
    printf( " sum= %lf\n " ,sum);
    return 0;
}
```

这里使用 C 语言语法编写程序，C 语言输入/输出使用的是<stdio.h>的 scanf()和 printf()函数。

3.5.3 for 语句

1．for 语句的语法

for 语句的语法格式如下：

```
for(表达式 1;表达式 2;表达式 3)
{
        循环体语句
}
```

该语句的执行过程：

（1）执行 for 后面的表达式 1。

（2）判断表达式 2，若表达式 2 的值为真（或非 0），则执行 for 语句的内嵌语句（即循环体语句），然后执行第（3）步；若表达式 2 的值为假（或等于 0），则循环结束，执行第（5）步。

（3）执行表达式 3。

（4）返回继续执行第（2）步。

（5）循环结束，执行 for 语句循环体下面的语句。

可以用图 3-15 表示其流程。

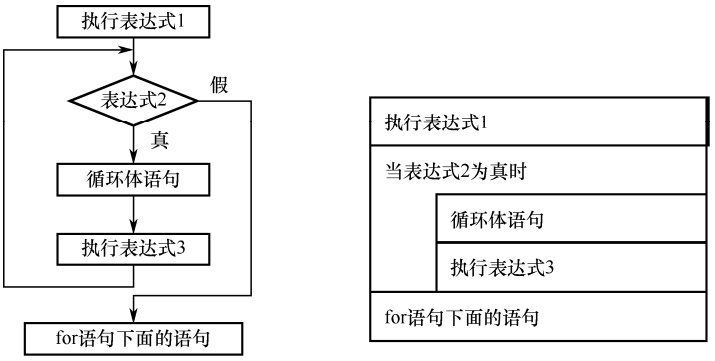


图 3-15　for 循环的流程图和 N-S 图

2．省略 for 循环语句中的表达式

for 循环语句中的 3 个表达式都可省略。

（1）for 语句中的“表达式 1”一般用于给循环变量赋初值，如果省略表达式 1，此时应在 for 语句之前给循环变量赋初值。

（2）for 语句中的“表达式 2”一般用于判断循环结束条件，如果省略表达式 2，则认为表达式 2 始终为真循环将无终止地进行下去。。

（3）for 语句中的“表达式 3”一般用于循环变量的增量，如果省略表达式 3，则循环变量的增量应在循环体内实现。

（4）如果省略 for 语句中的 3 个表达式，则变为以下形式：

```
for( ; ; )
{
    循环体语句
}
```

这就相当于下面的 while 语句：

```
while(1)
{
    循环体语句
}
```

此时，循环条件永远为真，从而无休止地执行循环体。为了终止循环，就要在循环体中加入 break 语句或 goto 语句等。

【例 3.14】 求 1+2+3+…+100。用 for 语句实现循环。

程序如下：

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

【例 3.15】 打印出所有的“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 $153=1^3+5^3+3^3$。

分析：利用 for 循环控制 100～999 之间的数，每个数分解出个位、十位和百位，然后判断它们的立方和是否等于该数本身。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
   int a,b,c;
   for(int i=100;i<1000;i++)
   {
      a=i%10;          //分解出个位
      b=(i/10)%10;     //分解出十位
      c=i/100;         //分解出百位
      if(a*a*a+b*b*b+c*c*c==i)
          cout<<i<<'\t';
   }
   cout<<endl;
   return 0;
}
```

程序运行结果：

```
153    370    371    407
```


3.5.4 循环的嵌套

前面曾经介绍过 if 语句的嵌套，而在一个循环的循环体中又包含另一个循环语句，称为循环嵌套。嵌套层次一般不超过 3 层，以保证可读性。

C/C++的三种循环语句可以相互嵌套，构成循环嵌套。

例如：

① for(;;)

```
{
        for(;;)
        {
            …
        }
}
```

② while()

```
{
        for(;;)
        {
            …
        }
}
```

说明：

（1）循环嵌套时，外层循环和内层循环间是包含关系，即内层循环必须被完全包含在外层循环中，不得交叉。

（2）当程序中出现循环嵌套时，程序每执行一次外层循环，则其内层循环必须循环所有的次数（即内层循环结束）后，才能进入到外层循环的下一次循环。

【例 3.16】 输出一个金字塔图形，如图 3-16 所示。

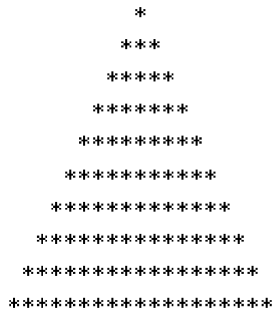


图 3-16　输出金字塔图形

分析：利用双重 for 循环，外循环控制层，内循环控制星号个数。该图形有 10 层，可用外循环控制。第 1 层有 1 个星号，第 2 层有 3 个星号，第 3 层有 5 个星号……，可用公式 $j=2*i-1$ 表示。其中，i 表示层数，j 表示该层星号的数量。还需要用 setw()控制每层输出星号的起始位置。

程序如下：

```
#include<iomanip>
#include <iostream>
using namespace std;                          //标准命名空间 std
int main()
{
    int i,j;
    for(i=1;i<=10;i++)
    {
        cout<<setw(20-i);
        for(j=1;j<2*i;j++) cout<< " * " ;
        cout<<endl;
    }
    return 0;
}
```

说明：可以改变输出图形的形状，如改为矩形、直角三角形、菱形等，请读者自行设计。

【例 3.17】 输出九九乘法表，如图 3-17 所示。

分析：输出 9 行 9 列乘法表是一个典型的双循环问题。计算机的输出是按行进行的，九九乘法表每行乘积数据是一组有规律的数，每个乘积数据的值是其所在行与列的乘积。

程序如下：

```
#include <iostream>
#include <iomanip>
using namespace std;                                    //标准命名空间 std
int main()
{
    cout<<'*';
```

```
    for(int i=1;i<=9;i++)
        cout<<setw(8)<<i;                                   //先用一个循环语句输出第一行表头
    cout<<endl;
    for(i=1;i<=9;i++){
        cout<<i;                                            //输出行号(被乘数)
        for(int j=1;j<=i;j++)
        cout<<setw(3)<<i<<'*'<<j<<'='<<setw(2)<<i*j;  //输出表中数据
        cout<<endl;                                         //准备输出下一行
    }
    return 0;
}
```

```
*        1        2        3        4        5        6        7        8        9
1   1*1= 1
2   2*1= 2   2*2= 4
3   3*1= 3   3*2= 6   3*3= 9
4   4*1= 4   4*2= 8   4*3=12   4*4=16
5   5*1= 5   5*2=10   5*3=15   5*4=20   5*5=25
6   6*1= 6   6*2=12   6*3=18   6*4=24   6*5=30   6*6=36
7   7*1= 7   7*2=14   7*3=21   7*4=28   7*5=35   7*6=42   7*7=49
8   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   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
```

图 3-17　九九乘法表

【例 3. 18】 计算并输出 10 以内（包括 10）所有自然数的阶乘值，即计算 1!，2!，3!，4!，5!，6!，7!，8!，9!，10!。

分析：采用的方法是对于 10 以内的每一个自然数分别求它们的阶乘值。其 N-S 图如图 3-18 所示。显然，这是一个二重循环结构。

程序如下：

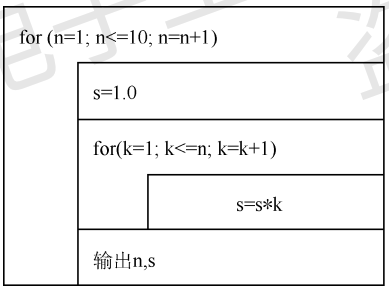


图 3-18　计算 10 以内自然数阶乘值的 N-S 图

```
#include <stdio.h>
int main()
{   int   n,k;
    double   s;
    for (n=1;n<=10;n=n+1)
    {   s=1.0;
        for (k=1; k<=n; k=k+1)
                    s=s*k;
        printf( " %2d!=%lf\n " ,n,s);
    }
    return 0;
}
```

注意，如果 s＝1.0;放到外循环 for (n=1;n<=10;n=n+1)的前面可以吗？请读者思考。

【例 3.19】 对一个正整数分解质因数。例如，输入 100，打印出 100=2*2*5*5。

分析：对 *n* 进行分解质因数，应先从最小的质数 i=2 开始，然后按下述步骤完成。

（1）如果 *n* 能被 *i* 整除，则应打印出 *i* 的值，并用 *n* 除以 *i* 所得的商作为新的正整数 *n*，重复执行此步。

（2）如果 *n* 不能被 *i* 整除，则用 *i*+1 作为 *i* 的值，重复执行第一步。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
    int n,i;
    cout<< " please input a number: " ;
    cin>>n;
    cout<<n<<'=';
    for(i=2;i<=n;i++)
    {
        while(n%i==0)
        {
            if(n!=i)
                cout<<i<<'*';
            else
                cout<<i;
            n=n/i;
        }
    }
    return 0;
}
```

程序运行结果：

```
please input a number:100↙
100=2*2*5*5
```

3.5.5 跳转语句

这一类语句的功能是改变程序的流程，使程序从其所在的位置转向另一处执行。这类语句是非结构化语句。

C/C++提供的跳转语句包括 break 语句、continue 语句、goto 语句和 return 语句。

1. break 语句

break 语句的一般格式为：

```
break;
```

该语句只能用于两种情况：

（1）用在 switch 结构中，当某个 case 分支执行完后，使用 break 语句跳出 switch 结构。

（2）用在循环结构中，用 break 语句来结束循环。如果放在嵌套循环中，则 break 语句只能结束其所在的那层循环。

【例 3.20】 任意输入若干个正整数（不多于 50 个），计算已输入的正整数之和，直到输入了负数为止。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
```

```
{   int i,n,sum;
    sum=0;
    for(i=0;i<50;i++)
    {   cout<< " \n Input number: " ;
        cin>>n;
        if(n<0) break;
        sum+=n;
    }
    cout<< " sum= " <<sum<<endl;
    return 0;
}
```

2. continue 语句

continue 语句的一般格式为：

```
continue;
```

该语句只能用在循环结构中。当在循环结构中遇到 continue 语句时，则跳过 continue 语句后的其他语句，结束本次循环，并转去判断循环控制条件，以决定是否进行下一次循环。

【例 3.21】 输出 0～100 之间所有不能被 3 整除的数。

程序如下：

```
#include <iostream>
using namespace std;                          //标准命名空间 std
int main()
{
     int i;
     for(i=0;i<=100;i++)
     {     if(i%3==0)
                    continue;
           cout<<i<<endl;
     }
     return 0;
}
```

3．goto 语句

goto 语句为无条件转向语句，它既可以向下跳转，也可以往回跳转。goto 语句与标号语句一起使用，所谓标号语句是用标识符标志的语句。goto 语句控制程序从 goto 语句所在的地方转移到标号语句处。goto 语句会导致程序结构混乱，可读性降低，而且它所完成的功能完全可以用算法的三种基本结构实现，因此一般不提倡使用 goto 语句。

goto 语句最大的好处就是可以一次性跳出多重循环，而 break 语句却不能做到这一点。

```
loop：average += score;
          n++;
          cin>>score;
          if (score>=0)       //表达式为真，转移到 loop 标号处
          goto loop;
```

4．return 语句

return 语句用于结束函数的执行，返回到主调函数，如果是主函数 main()，则返回至操作系统。

利用一个 return 语句可以将一个数据返回给调用者（主调函数）。通常，当函数的返回类型为 void 时， return 语句可以省略，如果使用，也仅作为函数或程序结束的标志。

3.5.6 三种循环的比较

（1）三种循环可以相互代替，并且都可以使用 break 语句和 continue 语句控制循环转向。

（2）while 语句和 for 语句是先判断条件，后执行循环体，而 do-while 语句是先执行循环体，后判断条件。

（3）for 语句功能最强，可完全取代 while 和 do-while 语句。

（4）while 语句和 do-while 语句中的循环变量初始化应该在循环前完成，并在 while 后指定循环条件，循环体中要包含使循环趋于结束的语句，而 for 循环可把这些操作放在 for 语句中。

3.6 常用算法及应用实例

3.6.1 累加与累乘

累加与累乘是最常见的一类算法，这类算法就是在原有基础上不断加上或乘以一个新的数。例如，求 1+2+3+…+*n*、求 *n* 的阶乘、计算某个数列前 *n* 项的和，以及计算一个级数的近似值等。

【例 3. 22】 求数列 2/3, 4/5, 6/7,… 前 30 项的和。

分析：该数列的通式为 2*n*/(2*n*+1)，*n*=1, 2, 3,…, 30。

程序如下：

```
#include <iostream>
using namespace std;                          //标准命名空间 std
int main()
{
    double sum=0;
    for(int i=1;i<=30;i++)
      sum+=2.0*i/(2.0*i+1);
    cout<<sum<<endl;
    return 0;
}
```

说明：也可将 for 循环改为下面的情形，请比较两者的可读性。

```
for(int i=2;i<=60;i+=2)
  sum+=i/(i+1.0);
```

【例 3.23】 求自然对数 e 的近似值，近似公式为：

$$e=1+ 1/1!+ 1/2!+ 1/3!+ \cdots + 1/n!$$

分析：这是一个收敛级数，可以通过求其前 n 项和来实现近似计算。通常，该类问题会给出一个计算误差，如可设定当某项的值小于 10^{-5} 时停止计算。

此题既涉及累加，又包含累乘，程序如下：

```
#include <iostream>
using namespace std;                          //标准命名空间 std
int main()
{
    double p,t,sum_e;
    int i = 1;
    p = 1; sum_e = 1;
    do
    {
        p=p*i;      //计算 i 的阶乘
        t=1/ p;
        sum_e=sum_e+t;
         i = i + 1;         //为计算下一项做准备
    }while(t>0.00001);
    cout<<sum_e<<endl;
    return 0;
}
```

程序运行结果：

```
2.71828
```

在开始循环前，必须为相关变量赋初值，这是初学者容易忽略和出错的地方。本例从级数的第 2 项开始进入循环，因此 sum_e（存放累加和）的初值为 1（级数第 1 项的值）；而 i = 1，p=1，则为进入循环后计算第 2 项的分母（1 的阶乘）做好了准备。

3.6.2 求最大数、最小数

求数据中最大数和最小数的算法是类似的，可采用“打擂”算法。以求最大数为例，可先用其中第一个数作为最大数，再用其与其他数逐个比较，并将找到的较大数替换为最大数。

【例 3.24】 求区间[100, 200]内 10 个随机整数中的最大数、最小数。

分析：随机函数 rand()返回一个 0～32767 之间的随机整数，为了生成区间[m, n]之间的随机整数，可使用公式 rand()%(n－m + 1) + m，故产生区间[100, 200]内随机整数的计算公式为 rand()%101+100。

程序如下：

```
#include <iostream>
#include<iomanip>
#include<cstdlib>                    //或者#include <stdlib.h>
#include <ctime>                     //或者#include <time.h>
using namespace std;                 //标准命名空间 std
int main()
```

```
{
    int max, min,x;
    x=rand()%101+100;          //产生一个在区间[100, 200]内的随机数 x
    cout<<setw(4)<<x;
    max = x; min =x;           //设定最大数和最小数
    for(int i=1;i<10;i++)
    {
        x=rand()%101+100;      //再产生一个在区间[100, 200]内的随机数 x
        cout<<setw(4)<<x;
        if(x > max)max = x;    //若新产生的随机数大于最大数，则进行替换
        if(x < min)min = x;    //若新产生的随机数小于最小数，则进行替换
    }
    cout<<endl<< " 最大数： " <<max<< " ,最小数： " <<min<<endl; //输出最大数和最小数
    system( " pause " );
    return 0;
}
```

程序运行结果类似如下：

```
141 167 134 100 169 124 178 158 162 164
最大数：178，最小数：100
```

说明：以上程序每次运行结果都一样，原因是 rand()函数（在调用它时头文件要包含 stdlib.h 或 cstdlib）可以用来产生随机数，但是这不是真正意义上的随机数，是一个伪随机数，它是以一个数（可以称它为种子(seed)）为基准，依据某个递推公式推算出来的一系列数（随机序列）。但这不是真正的随机数，当计算机正常开机后，这个种子的值是确定的。为了改变这个种子值，C/C++提供了 srand()函数，它的原型是 void srand(int a)，功能是初始化随机产生器，即把 rand()函数的种子的值改成 a。当需要不同的随机序列时可以用 srand(time(0))，其中 time 函数（在调用它时头文件要包含 time.h 或 ctime）的功能是返回从 1970/01/01 00:00 到现在的秒数，因为每一次运行程序的时间是不同的，所以用它来作为 rand() 的种子值，可以产生不同的随机序列。

对程序进行如下修改：

```
srand(time(0));            //srand()函数以当前时间产生随机种子
x=rand()%101+100;          //产生一个在区间[100, 200]内的随机数 x
```

由于每次运行的时间不同（只要两次运行的时间间隔超过 1s），所以 time(0)返回的当前时间（以秒为单位）也会不同，这样就可以保证每次运行时可以得到不同的随机数序列。

若要 0～1 之间的随机小数，则可以先取 0～10 之间的整数，然后均除以 10 即可得到随机到十分位的 1 个随机小数；若要得到随机到百分位的随机小数，则需要先得到 0～100 的整数，然后均除以 100，其他情况以此类推。

3.6.3 求素数

素数是除 1 和本身外，不能被其他任何整数整除的整数。判断一个数 m 是否为素数，只要依次用 2, 3, 4, …, $m-1$ 做除数去除 m 即可。只要有一个数能整除，则 m 就不是素数。

【例 3.25】 从键盘上输入一个大于 2 的自然数，判断其是否为素数。

分析：可使用一个逻辑变量 flag 来表示自然数 m 是否为素数。首先标志 flag 默认为 true，然后循环判断 m 能否被 2, 3, 4, …, m−1 整除，只要有一个数能整除，就可以停止循环并修改标志 flag，根据标志 flag 是否被修改即可知道 m 是否是素数。

程序如下：

```
#include <iostream>
using namespace std;                                  //标准命名空间 std
int main()
{
     int m,i;
     bool flag;
     cout<< " 输入整数 m：  " <<endl;
     cin>>m;
     flag = true;                                     //设标志为 true
     for(i=2;i<m;i++)
     if (m%i==0)                                      //能被 i 整除
     {
        flag=false;                        //设标志为 false
        break;                             //只要有一个数能整除，即知 m 不是一个素数，就可停止
     }
     if(flag==true)                        //根据标记 flag 输出判断结果
        cout<< m<< " 是素数 " <<endl;
     else
        cout<< m<< " 不是素数 " <<endl;
     return 0;
}
```

运行上述代码，对于一个非素数而言，判断过程往往即可结束。例如，判断 30009 时，因为该数能被 3 整除，所以只需判断 i = 2, 3 两种情况。而判断一个素数，尤其当该数较大时，如判断 30011，则要从 i = 2, 3, 4, …，一直判断到 30010 都不能被整除，才能得出其为素数的结论。实际上，只要从 2 判断到 $\sqrt{n}$，若 n 不能被其中任何一个数整除时，则 n 即为素数，故语句 for(i=2;i<m;i++)可改为 for(i=2;i<=sqrtf(m);i++)。

【例 3.26】 输出在 100 与 200 之间的所有素数。

分析：上例已经可以判断一个数 m 是否为素数，只需加一层外循环，控制 m 的变化范围为 100 到 200 之间即可。

程序如下：

```
#include <iostream>
#include <iomanip>
#include <cmath>                          //或者#include <math.h>
using namespace std;                      //标准命名空间 std
int main(    )
{   int i, m,count=0;
    bool flag;                            /*  用 flag 作标志  */
    for(m=100; m<=200; m++)               //控制 m 变化范围为 100 到 200 之间
    {
        flag = true;
        for(i=2;i<=sqrtf(m);i++)
```

```
            if (m%i==0)
            {
                flag=false;          //m 不是一个素数
                break;               //只要有一个数能被整除，就可停止
            }
          if(flag)                   /* m 是素数 */
          {
                cout<<setw(5)<<m;
                count++;
                if (count % 8 == 0) cout<<endl;   //每输出 8 个素数就换行
          }
    }                                              /* 测试下一个 m */
    system("pause");
    return 0;
}
```

3.6.4 穷举法

穷举法又称枚举法，穷举法将所有可能出现的情况一一进行测试，从中找出符合条件的所有结果。如计算“百钱买百鸡”问题，又如列出满足 xy=100 的所有组合等。

【例 3.27】 公鸡每只 5 元，母鸡每只 3 元，小鸡 3 只 1 元，现要求用 100 元钱买 100 只鸡，问公鸡、母鸡和小鸡各买几只？

分析：设公鸡 x 只，母鸡 y 只，小鸡 z 只。根据题意可列出以下方程组：

$$\begin{cases} x+y+z=100 \\ 5x+3y+z/3=100 \end{cases}$$

由于两个方程式中有 3 个未知数，属于无法直接求解的不定方程，故可采用枚举法进行试根，即逐一测试各种可能的 x、y、z 组合，并输出符合条件的数。

程序如下：

```
#include <iostream>
#include <iomanip>
using namespace std;                                  //标准命名空间 std
int main()
{
    int x,y,z;
    for(x=0;x<=100;x++)                               //可优化为 x<=19
        for(y=0 ;y<=100;y++)                          //可优化为 y<=33
        {
            z=100-x-y;
            if(5*x+3*y+z/3.0==100)
              cout<<setw(5)<<x<<setw(5)<<y<<setw(5)<<z<<endl;
        }
    return 0;
}
```

程序运行结果：

```
    0    25    75
```

```
4    18   78
8    11   81
12   4    84
```

说明：上述程序循环体内的语句块将被执行 101×101 次。考虑到最多只能买 19 只公鸡，33 只母鸡，故可将两个 for 语句优化为 x<=19 和 y<=33。优化后的循环次数为 20×34 次，从而大大提高了运行效率。

如果将 if(5*x+3*y+z/3.0==100)中的条件改为 if(5*x+3*y+z/3==100)，会出现什么情况？

3.6.5 递推与迭代

1．递推

利用递推算法或迭代算法，可以将一个复杂的问题转换为一个简单过程的重复执行。这两种算法的共同特点是，通过前一项的计算结果推出后一项。不同的是，递推算法不存在变量的自我更迭，而迭代算法则在每次循环中用变量的新值取代其原值。

【例 3.28】 输出斐波那契（Fibonacci）数列的前 20 项。该数列的第 1 项和第 2 项为 1，从第 3 项开始，每一项均为其前面两项之和，即 1，1，2，3，5，8，…

分析：设数列中相邻的 3 项分别为变量 f_1、f_2 和 f_3，则有如下递推算法。

（1）f_1 和 f_2 的初值为 1。

（2）每次执行循环，用 f_1 和 f_2 产生后项，即 $f_3=f_1+f_2$。

（3）通过递推产生新的 f_1 和 f_2，即 $f_1=f_2$，$f_2=f_3$。

（4）如果未达到规定的循环次数，则返回步骤（2），否则停止计算。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
    long f1, f2, f3;
    f1 = 1; f2 = 1;                                     //初始条件
    cout<<f1<<endl<<f2<<endl;
    for(int i=3;i<=20;i++)
    {
        f3=f1+f2;                                       //递推公式
        cout<<f3<<endl;
        f1 = f2; f2 = f3;                               //递推公式
    }
    return 0;
}
```

说明：解决递推问题必须具备两个条件，即初始条件和递推公式。本题的初始条件为 f_1=1 和 f_2=1，递推公式为 $f_3=f_1+f_2$，$f_1=f_2$，$f_2=f_3$。

【例 3.29】 有一分数序列：2/1, 3/2, 5/3, 8/5, 13/8, 21/13, …求出这个数列的前 20 项之和。

分析：注意分子与分母的变化规律，可知后项分母为前项分子，后项分子为前项分子与分母之和。

程序如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
   int n,number=20;
   float a=2,b=1,t,s=0;
   for(n=1;n<=number;n++)
   {
         s=s+a/b;
         t=a;a=a+b;b=t;                                 /*这部分是程序的关键*/
   }
   cout<<s;
   return 0;
}
```

2．迭代

迭代算法也称为辗转法，是一种不断用变量的旧值递推新值的过程。迭代算法是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组指令（或一定步骤）重复执行，在每次执行这组指令（或这些步骤）时，都从变量的原值推出它的一个新值。

【例 3.30】 用迭代算法求 a 的平方根。求平方根的公式为 $x_{n+1}=(x_n+a/x_n)/2$，求出的平方根的精度是前、后项差的绝对值小于 10^{-5}。

分析：迭代算法求 a 的平方根的步骤如下。

（1）设定一个 x 的初值为 x_0（在如下程序中取 $x_0=a/2$）。

（2）用求平方根的公式 $x_1=(x_0+a/x_0)/2$ 求出 x 的下一个值 x_1。可以肯定，求出的 x_1 与真正的平方根相比，误差很大。

（3）判断 x_1-x_0 的绝对值是否满足大于 10^{-5}，如果满足，则将 x_1 作为 x_0，重新求出新 x_1，如此继续下去，直到前后两次求出的 x_1 和 x_0 之差的绝对值满足小于 10^{-5} 为止。

程序如下：

```
#include <iostream>
#include <cmath>                                //或者#include <math.h>
using namespace std;                            //标准命名空间 std
int main()
{
    float a;                                    /*  被开方数  */
    float x0, x1;                               /*  分别代表前一项和后一项  */
    cout<<   " Input a positive number: " <<endl;
    cin>>a;
    x0 = a / 2;                                 /*  任取初值  */
    x1 = (x0 + a / x0) / 2;
    while (fabs(x1 - x0) >= 1e-5)               /*fabs(x)函数用来求参数 x 的绝对值*/
```

```
    {
        x0 = x1;
        x1 = (x0 + a / x0) / 2;
    }
    cout<< " The square root is   " <<x1;
    return 0;
}
```

程序运行结果:

```
Input a positive number:
2↙
The square root is 1.41421
```

在工程技术中，经常使用数值算法来求解超越方程和代数方程的根。解决这类问题一般采用迭代算法，如下面介绍的牛顿迭代法。

设有方程 $f(x)=0$，使用牛顿迭代法求根 α 的近似值，算法如下。

（1） 在根 α 附近取一点 x_0，作为 α 的第 0 次近似值。

（2） 以曲线 $y=f(x)$ 在点$(x_0,f(x_0))$ 的切线与 x 轴交点的横坐标 x_1 作为 α 的第 1 次近似值。切线方程为 $y-f(x_0)=f'(x_0)(x_1-x_0)$，令 $y=0$，得

$$x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

（3）以曲线 $y=f(x)$ 在点 $(x_1,f(x_1))$ 的切线与 x 轴交点的横坐标 x_2 作为 α 的第 2 次近似值，则

$$x_2=x_1-\frac{f(x_1)}{f'(x_1)}$$

（4）一般有

$$x_n=x_{n-1}-\frac{f(x_{n-1})}{f'(x_{n-1})}$$ （牛顿迭代公式）

x_1，x_2，…，x_n，逐次逼近 α，如图 3-19 所示。

（5）依次求出 x_1，x_2，…，x_n，直到前、后两项之差的绝对值$|x_n-x_{n-1}|\leqslant\varepsilon$ 为止（ε 是指定的允许误差），此时就认为 x_n 是足够接近真实根的近似值。

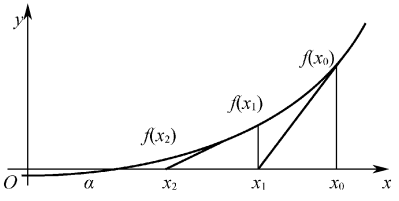


图 3-19　牛顿迭代法求方程根

【例 3.31】 编写程序，用牛顿迭代法求方程 $x+\ln x-2.13=0$ 的近似实根。

分析：通过对根的大致估算，可设迭代初值为 2，误差为 0.0001。其中 ln$x$ 对应 C/C++函数 log(x)；fabs(x)函数用来求参数 x 的绝对值。

程序如下：

```
#include <iostream>
```

```
#include <cmath>                          //或者#include <math.h>
using namespace std;                      //标准命名空间 std
int main()
{
    double x0, x1, e,f,f1;
    cout<< " 请输入迭代初值: " ;
    cin>>x1;                              //运行时输入初值 2
    cout<< " 输入迭代误差: " ;
    cin>>e;                               //运行时输入误差值 0.00001
     do
     {
          x0 = x1;
          f = x0+log(x0)−2.13;            //方程 f(x)
          f1 = 1 + 1 / x0;                //方程 f(x)的一阶导数
          x1 = x0 − f / f1;               //牛顿迭代公式
    }while(fabs(x0 − x1)>e);
    cout<<x1<<endl;
    return 0;
}
```

程序运行结果：

```
请输入迭代初值:2↙
输入迭代误差:0.0001
1.63708
```

说明：在计算过程中，要用 x 的新值 x_1 替换其原值 x_0，再使用牛顿迭代公式由新的 x_0 产生下一轮的 x_1。由于在进入循环后的第 1 条语句为 x0= x1，故应将迭代初值赋给 x_1 而非 x_0。

3.7 应用实例

完成小学数学四则运算测试程序。要求如下：

（1）用户可以指定测试题目数量。

（2）要求系统随机生成 0～100 之内的整数。

（3）对回答错误的题目应该可重新回答一次。

（4）能够统计回答正确题目的个数及正确率（百分比形式）。

（5）显示效果，如 12+56=?

注意：减法题目结果不允许出现负数，除法题目必须能够整除且被除数大于除数，且除数是 10 以内的数字。

分析：由于是多道题测试，明显可采用循环语句来完成。但是，如何让计算机随机产生加、减、乘、除各种题型题目呢？这里采用一个技巧，随机函数 rand()产生 1～4 之间的整数 op，如果 op 是 1，则认为是加法，如果 op 是 2，则认为是减法，如果 op 是 3，则认为是乘法，如果 op 是 4，则认为是除法，而一个题目需要的两个运算数 num1、num2 也是由随机函数 rand()产生的，这样解决不同题型的问题。

```
#include<iostream>
#include<cstdlib>                 //C 语言是#include<stdlib.h>
```

```
#include<ctime>                    //C 语言是#include<time.h>
using namespace std;
int main()
{
    int num1, num2 , result, op, right=0,m;
    char c;
    cout <<  " 请输入要做的题数: " ;
    cin >> m;
    srand(time(0));
    for (int j = 1; j <= m; j++)
    {
        num1 = rand() % 101;
        num2 = rand() % 101;
        op = rand() % 4 + 1;
        switch (op){
            case 1:
                c = '*';
                result = num1*num2; break;                    // result 存储正确答案
             case 2:
                c = '+';
                result = num1+num2; break;
             case 3:
                c = '-';
                if (num1<num2)
                {
                    t = num1; num1 = num2; num2 = t;
                }
                result = num1-num2; break;
             case 4:
                c = '/';
                if (num1<num2)
                {
                    t = num1; num1 = num2; num2 = t;
                }
                num1 = num1 - num1%num2;                      //减去余数自然就能整除
                result = num1/num2; break;
                default:cout <<  " 错 "  << endl;
        }
        cout <<  " 请输入 "  << num1 << c << num2 <<  " =?\n " ;
        cin >> n;
        if (n == result)
            right++;                                          //正确数加 1
        else {
            cout <<  " 请再次输入 "  << num1 << c << num2 <<  " =?\n " ;
            cin >> n;
            if (n == result)
                right++;
        }
    }
    cout <<  " 共做对 "  << right <<  " 题 "  << endl;
```

```
    cout <<  " 正确率是：  "  << right * 100 / m << '%' << endl;
    cin >> c;
    return 0;
}
```

程序运行结果如图 3-20 所示。

图 3-20　小学数学四则运算测试程序运行结果

3.8　程序的调试

程序的错误可以分为编译错误、连接错误及运行错误。

Visual Studio 2013 集成环境中，编译器能够发现编译错误（即语法错误）和连接错误。编译错误通常是编程者违反了 C/C++语言的语法规则，如保留字输入错误、大括号不匹配、语句少分号等。连接错误通常由未定义或未指明要连接的函数，或函数调用不匹配等引起，对系统函数的调用必须通过“include”来说明。

在程序运行时，也会产生运行错误。对于运行错误，在 Visual Studio 2013 集成环境中提供了调试器，能跟踪程序的运行过程。它可以一行一行（单步）地执行程序源代码，以观察程序运行过程中，哪些语句执行了，哪些语句没有执行、执行的顺序如何，以及内存中各变量当前的值，从而确定应用程序在运行的各个阶段发生了什么，且是如何发生的。Visual Studio 2013 提供的调试技术包括单步运行和断点运行、设置中断表达式和监视表达式、显示变量的动态值等。

要显示“调试”（Debug）工具栏，可在工具栏上单击鼠标右键，在弹出的快捷菜单中选择“调试”选项。

在如图 3-21 所示的“调试”工具栏上提供了几个很有用的按钮。表 3-1 列出的是 Visual Studio 2013 （Visual C++ 2013 ）“调试”工具栏中按钮的用途。

图 3-21　“调试”工具栏

表 3-1　“调试”工具栏主要按钮的用途

工 具 名 称	按　钮	用　途
逐语句(F11)		单步执行应用程序的下一个可执行语句行，并跟踪到函数中
逐过程(F10)		单步执行应用程序的下一个可执行语句行，但不跟踪到函数中

（续表）

工具名称	按　钮	用　途
跳出		跳出当前函数
代码图	代码图	通过代码图集成可视化调试
重新启动		重新开始运行程序
停止调试		终止调试
继续		继续运行程序

3.8.1　进入调试

首先完成编译连接，只有编译连接正确，才能开始调试。不能通过调试来检查编译或连接的错误，调试属于程序运行阶段。

使用“调试”菜单，选择“逐语句”，程序将开始单步调试，也可以直接按“Fl1”键。如果没有图 3-21 所示的“调试”工具栏，可在工具栏上单击鼠标右键，在弹出的快捷菜单中选择“调试”选项。

3.8.2　单步调试

单步执行时单击“调试”工具栏中的“逐过程”按钮 或按“F10”键。如果遇到自定义函数调用，想进入函数进行单步执行，可单击“逐语句”按钮 或按“F11”键。对不是函数调用的普通语句来说，“F11”键与“F10”键的作用相同。但一般对系统函数（如 scanf、cin 语句）不要使用“F11”键。建议使用“F10”键进行单步调试。

通过使用单步执行，可以比较直观地看到程序执行流程，尤其对于分支语句 if-else, 可以看出程序到底执行了条件为真的语句 1，还是执行了条件为假的语句 2。图 3-22 左侧的箭头指示当前即将要执行的语句，每单步执行一次，箭头位置就会发生变化。

```
#include <iostream>
using namespace std;                                    //标准命名空间std
int main()
{
    int n, number = 20;
    float a = 2, b = 1, t, s = 0;
    for (n = 1; n <= number; n++)
    {
        s = s + a / b;
        t = a; a = a + b; b = t;          /*这部分是程序的关键*/
    }
    cout << s;
    return 0;
}
```

图 3-22　程序单步执行

在单步执行时，还要关注运行窗口，尤其在执行 scanf、cin 语句时，一定要在程序运行

窗口中输入数据后再回到调试窗口，否则单步无法继续。

3.8.3 查看变量、表达式的值

调试器支持查看变量、表达式的值。所有这些观察都必须在调试情况下进行。查看变量的值最简单，当箭头指示将运行的语句是所需查看变量所在的语句时，把光标移动到这个变量上，停留一会儿就可以看到变量的值。

自动窗口（单击“调试”→“窗口”→ “自动窗口”，就可以打开自动窗口）会自动显示当前程序执行中所有可见的变量值。特别是当前语句涉及的变量，以红色显示。在自动窗口的下部有 3 个选项卡：自动窗口、局部变量和监视 1。选中不同的选项卡，将会在该窗口中显示不同类型的变量。

查看变量、表达式的值也可通过监视窗口［单击“调试”→“窗口”→ “监视”→ “监视 1（有 1～4 共 4 个监视）”，就可以打开监视窗口，可以同时打开 4 个监视窗口］实现。在监视窗口中输入变量或表达式，就可以观察变量或表达式的值。

快速监视：先选定希望监视的表达式，然后在其上面单击鼠标右键，在出现的菜单中选择“快速监视”，这时会直接打开快速监视窗口并添加已选定的表达式为监视项。

3.8.4 停止调试

如果用户想停止调试，可以使用“调试”菜单下的“停止调试”选项，或“调试”工具栏，终止调试 ■ 按钮，从而回到正常的运行状态。

下面通过实例说明单步调试的过程。

有一分数序列：2/1, 3/2, 5/3, 8/5, 13/8, 21/13,…求出这个序列的前 20 项之和。参考代码如下：

```
#include <iostream>
using namespace std;                                    //标准命名空间 std
int main()
{
   int n,number=20;
   float a=2,b=1,s=0;
   for(n=1;n<=number;n++)
   {
      s=s+a/b;                                          //a 是数据项的分子，b 是分母
      a=a+b;b=a;
   }
   cout<<s;
   return 0;
}
```

调试时可以观察每次循环求出的数据项 a/b 的值是否正确。具体步骤如下：

（1）首先按“F11”键，系统进入调试状态，程序开始运行并会暂停在程序入口 main 函数的开始“{”处，箭头指示将运行的语句。

（2）每按“F11”键一次，执行一条语句。当执行 for 语句体 s=s+a/b 时，在“自动窗口”中

看到 *s* 及其他 *a*、*b*、*n* 等变量的值。此时是第 1 次循环时 *s*、*a*、*b* 等变量的值，如图 3-23 所示。

（3）如果想观察第 2 次循环时 *s*、*a*、*b* 等变量的值，可以在“调试”工具栏中单击按钮或按“F11”键单步执行，再次运行到 s=s+a/b;这一程序代码。在图 3-24 的“自动窗口”中观察第 2 次循环时 *s*、*a*、*b* 等变量的值是否与预期的变量值一致。从“自动窗口”中得知 a=3，b=2，实际上第 2 个数据项为 3/2，相符则说明程序代码在此处没有问题，否则需要修改。

图 3-23　第 1 次循环时 s、a、b 等变量的值

图 3-24　第 2 次循环时 s、a、b 等变量的值

调试最重要的是思考，要猜测程序可能出错的地方，然后运用调试器来验证猜测。熟练使用上面的工具无疑会加快这个过程。

实验三　程序控制结构

一、实验目的

通过本次实验，初步建立算法的概念，掌握解决一个实际问题的方法和步骤，并能用流程图表示出来。理解一个完整的结构化程序由顺序、选择和循环三种基本结构组成。熟练掌握并能于实际问题中应用选择结构和循环结构的相关语句，如 if 语句、for 语句和 while 语句。

二、实验要求

1．掌握顺序结构程序的设计。
2．掌握选择结构及选择嵌套结构的程序设计。
3．掌握循环结构及循环嵌套结构的程序设计。
4．掌握常用算法。

三、实验内容与步骤

1．编写程序。输入一个三位正整数，找出其中最大的一位数字并输出。

分析：先将三位正整数的各个数位上的数字分离出来，再进行比较。

程序 1 如下：

```
#include <iostream>
using namespace std;
int main()
{
    int a1, max_a, a2, a3, a;
```

```
    cin >> a;
    a1 =a % 10;
    a2 =a/10 % 10;
    a3 =a/100;
    if (a1>a2)
        max_a =a1;
    else
        max_a =a2;
    if (max_a>a3)
        max_a = max_a;
    else
        max_a =a3;
    cout << max_a << endl;
    system("pause");
    return 0;
}
```

思考问题：如何实现在 5 位正整数中找出其中最大的一位数字并输出？

2．编写程序，求级数 1*2+2*3+3*4+4*5+……+n*(n+1)+…前 n 项的和。

分析：这是一个计数循环，每一个加数可以由数列的通项公式求得。

程序 2 如下：

```
#include <iostream>
using namespace std;
int main()
{
   int n,s=0,i,a;
   cin >> n;
   for(i=1;i<=n;i++)
   {
       a=i*(1+i);
       s+=a;
   }
   cout << s << endl;
   system("pause");
   return 0;
}
```

思考问题：求和问题的算法设计实际应用了迭代算法的思想。请自行理解。

3．编写程序。一个小球从 *n* 米高度落下，每次落地后反弹，上升为原来高度的一半，再落下，球在 10 次落地后，未再弹起时，共经过了多少米（要求 *n* 为偶数）？

分析：这是一个 9 次的计数循环求和，其中的加数（小球弹起高度）又应用了迭代算法。（提示：需要将和初始化为 *n*。）

程序 3 如下：

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

```
int main()
{
    double n,s,i,a;
    cin >> n;
    s=n;
    for(i=1;i<=9;i++)
    {
        n=n/2;
        s+=2*n;
    }
    cout << s << endl;
    system("pause");
    return 0;
}
```

4．编写程序。一个整数，它加上 100 后是一个完全平方数（如果一个正整数 *a* 是某一个整数 *b* 的平方，那么这个正整数 *a* 叫做完全平方数），再加上 168，又是一个完全平方数。请输出 *n* 以内符合这一特征的整数个数。

分析：这是一个双重循环嵌套应用穷举法的题目。外层循环遍历 n 以内的所有整数，两个内层循环又分别应用穷举算法判断此数加上 100 及此数加上 268 后得到的两个新数是否是完全平方数。

程序 4 如下：

```
#include <iostream>
using namespace std;
int main()
{
    int n,k=0,i,j,a,flag1,flag2;
    cin >> n;
    for(i=1;i<=n;i++)
    {
        a=i+100;flag1=0;
        for(j=1;j<=a;j++)
            if(a==j*j){flag1=1;break;}
        a=i+268;flag2=0;
          for(j=1;j<=a;j++)
              if(a==j*j){flag2=1;break;}
          if(flag1 && flag2)
              k++;
    }
    cout << k << endl;
    system("pause");
```

```
    return 0;
}
```

思考问题：找出一个范围内所有指定数，这是一个常见的应用，通常使用穷举算法进行遍历。这道题又借鉴了判断素数的方法，请对这种标志位应用进行总结。

四、编程并上机调试

1．输入一个数 n，判断 1~n 范围内的所有完数。完数是指一个数恰好等于它的因子之和，如 6=1+2+3，6 就是完数。

2．著名意大利数学家 Fibonacci（斐波那契）曾提出一个问题：有一对小兔子，从出生后第 3 个月起每个月都生一对兔子。小兔子长到第 3 个月后每个月又生一对兔子。按此规律，假设没有兔子死亡，第一个月有一对刚出生的小兔子，问第 n 个月有多少对兔子？

每个月兔子的数量组成的数列即斐波那契数列。请输出斐波那契数列的第 n 项数。

3．用迭代法，求 a 的三次方根。

求三次方根的公式为 $x_n+1=(2/3)*x_n+a/(3*x_n*x_n)$，三次方根精度要求前、后项差的绝对值小于 10^{-4}。

习题 3

一、选择题

1．下列程序执行完后，x 的值是_____。

```
int x=0;
for (int k=0;k<90; k++)
{   if (k)    x++;}
```

A．0　　B．30　　C．89　　D．90

2．下列程序段循环次数是_____。

```
int x =−10;
while (++x)    cout<<x<<endl;
```

A．9　　B．10　　C．11　　D．无限

3．下面有关 for 循环的正确描述是_____。

A．for 循环只能用于循环次数已经确定的情况

B．for 循环是先执行循环体语句，后判定表达式

C．在 for 循环中，不能用 break 语句跳出循环体

D．for 循环体语句中，可以包含多条语句，但要用花括号括起来

4．以下关于循环体的描述中，_____是错误的。

A．循环体中可以出现 break 语句

B．循环体中还可以出现循环语句

C．循环体中不能出现 continue 语句

D．循环体中可以出现 switch 语句

5．下述关于 break 语句的描述中，_____是不正确的。

A．break 语句可用于循环体内，它将退出该重循环

B．break 语句可用于 switch 语句中，它将退出 switch 语句

C．break 语句可用于 if 体内，它将退出 if 语句

D．break 语句在一个循环体内可以出现多次

二、简答题

1．C/C++语言中 while 和 do-while 循环的主要区别是什么？

2．过程化程序有哪三种基本控制结构？

3．C/C++用于构成分支结构的语句有哪些？构成循环结构的语句有哪些？

三、填空题

1．用 for 循环打印 0 1 2 0 1 2 0 1 2;

```
for( i=1; i<=9; i++ )
printf(%2d " ,          );                    //等价于 cout<<;
```

2．下列程序段的输出为：

```
#include <stdio.h>                            //#include<iostream>
int main()
{
    int i=1;
    while (i <= −1)
        printf( " ### " ) ;                   //等价于 cout<< " ### " ;
    printf( " %d " , i ) ;                    //等价于 cout<<i;
    return 0;
}
```

3．下列程序段的输出为：

```
#include <stdio.h>                            //#include<iostream>
int main( )
{
        int i=5:
        do{
            i−−;
            printf( " ### " ) ;               //等价于 cout<< " ### " ;
        }while (i) ;
        printf(   " %d "   , i ) ;            //等价于 cout<<i;
        return 0;
}
```

4．下列程序求 $S_n=a+aa+aaa+\cdots+aa\cdots aa$（$n$ 个 a）的值，其中 a 是一个数字。当 $a=2$, $n=5$ 时，$S_5=2+22+222+2222+22222$，其值应为 24690。

```
#include <stdio.h>                            //#include<iostream>
int main( )
{
      int a, n, count =1, sn=0, tn=0;
      printf(   " Please input a and n: \n " ) ;
```

```
    scanf(   " %d%d " ,&a,&n);                  //等价于 cin>>a>>n;
    while (count <=n)
    {
        tn=tn+a;
        sn=_________;
        a=a*10;
        _________;
    }
    printf( " the Sn is: %d \n " , sn);       //等价于 cout<< " the Sn is: " <<sn<<endl;
    return 0;
}
```

四、编程题

1．输入一个整数 n，判断其能否同时被 5 和 7 整除，如能则输出“xx 能同时被 5 和 7 整除”，否则输出“xx 不能同时被 5 和 7 整除”。要求“xx”为输入的具体数据。

2．输入一个百分制的成绩，经判断后输出该成绩的对应等级。其中，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”。

3．某百货公司为了促销，采用购物打折的办法。1000 元以上者，按九五折优惠；2000 元以上者，按九折优惠；3000 元以上者，按八五折优惠；5000 元以上者，按八折优惠。编写程序，输入购物款数，计算并输出优惠价（要求用 switch 语句编写）。

4．编写一个求整数 n 阶乘（$n!$）的程序，要求显示的格式如下：

1:　　1　　2:　2　　3:　6

4:　　24　　5: 120　　6: 720

5．编写程序，求 1!+3!+5!+7!+9!。

6．编写程序，计算下列公式中 s 的值（n 是运行程序时输入的一个正整数）。

$s=1+(1+2)+(1+2+3)+\cdots+(1+2+3+\cdots+n)$

$s=12+22+32+\cdots+(10n+2)$

$s=1\times2-2\times3+3\times4-4\times5+\cdots+(-1)^{(n-1)}\times n\times(n+1)$

$$s=1+\frac{1}{1+2}+\frac{1}{1+2+3}+\cdots+\frac{1}{1+2+3+\cdots+n}$$

7.“百马百瓦问题”：有 100 匹马驮 100 块瓦，大马驮 3 块，小马驮 2 块，两个马驹驮 1 块。问大马、小马、马驹各有多少匹？

8．有一个数列，其前三项分别为 1、2、3，从第四项开始，每项均为其相邻前三项之和的 1/2，问：该数列从第几项开始，其数值超过 1200。

9．找出 1 与 100 之间的全部“同构数”。“同构数”是这样一种数，它出现在它的平方数的右端。例如，5 的平方是 25，5 是 25 中右端的数，5 就是同构数，25 也是一个同构数，它的平方是 625。

10．猴子吃桃问题。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个，第二天早上将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃前一天剩下的一半再加一个。到第 10 天早上想再吃时，发现只剩下一个桃子了。求第一天共摘了多少个桃子。