第3章 算法与控制结构

为了使用计算机解决实际问题，需要在分析研究的基础上制定相应的算法，然后使用某种程序设计语言（如C++）来编写并运行程序，以便得到所期望的结果。解决同一个问题往往可以找到多种不同的算法，根据这些算法编写的程序自然会有高下之分。

程序中，需要将实现算法的一连串语句编排成顺序结构、选择结构、循环结构或者它们套叠而成的复杂结构，其中选择结构和循环结构都需要专门的语句来实现。

3.1 基本知识

算法是程序设计的依据。算法可以用自然语言、伪代码、流程图等多种方式表现出来，其中流程图有多种形式：框形流程图、N-S 流程图、PAD 图等。按照结构化程序设计的思想，使用 3 种基本结构(顺序结构、选择结构、循环结构)或者由 3 种基本结构套叠而成的复杂结构可以表示任何算法。

C++程序中，使用 if 语句和 switch 语句实现选择结构，使用 while 语句、do-while 语句和 for 语句实现循环结构，还可以使用这些语句套叠而成更为复杂的结构。

3.1.1 算法的概念与表示

一般来说，计算机科学中的算法是由一套规则组成的一个过程。过程中包含一系列编排了顺序的操作，按既定的顺序执行这些操作就可以得到某种问题的解答。因此，算法实际上是一种抽象的解决问题的方案。

1. 算法的特性

通常认为一个算法必须具备有穷性、确定性、可行性以及数据输入和信息输出这 5 个基本特征。

(1) 有穷性：任何情况下，一个算法都应该在执行有穷步操作之后宣告结束，且其执行时间不应长于实际可容忍的限度。

(2) 确定性：算法中的每一步都必须是精确定义的，不能模棱两可。即每一步应该执行哪种动作必须是清楚的，无歧义的。否则，这样的算法是无法执行的。

(3) 可行性：算法中的任何一步操作都必须是可执行的基本操作，换句话说，每一种运算至少在原理上可由人用纸和笔在有限的时间内完成。

(4) 数据输入：一个算法可以有一个或多个输入，也可以没有输入。

(5) 信息输出：一个算法至少有一个已获得的有效信息输出。算法的输出可以是数字、文字、图形、图像、声音、视频信息，以及具有控制作用的电信号等多种信息形式。

2. 算法的表示

为了描述算法，可以采用多种不同的工具，如自然语言、伪代码、流程图等。如果一个算法是采用计算机能够理解和执行的语言来描述的，它就是程序。设计这样的算法的过程就叫做程序设计。

(1) 自然语言：使用人们日常生活中使用的语言来描述算法，容易做到通俗易懂，但其含义往往不太严格、容易出现歧义，也不便描述包含分支部分和循环部分的算法。

(2) 伪代码：是为了表示算法而专门制定的语言，可以将算法表示得非常清楚。伪代码既可由自然语言改造而成，也可将某种程序设计语言简化而得到。但是，由于难于

找到一种大家普遍接受的伪代码，因而限制了它的使用。

（3）流程图：是用于描述算法的特殊图形，它使用各种形状不同的带有说明性文字的图框分别表示不同种类的操作，用流程线或图框之间的相对位置来表示各种操作之间的执行顺序。流程图可以形象地描述算法中各步操作的具体内容、相互联系和执行顺序，直观地表明算法的逻辑结构，是使用最多的算法表示法。

3. 算法的流程图表示

常用的流程图有传统的框形流程图和 N-S 结构化流程图之分。其中传统流程图的主要构件如图 3-1(a)所示。

例 3-5 输入 10 个数，挑选出最小的数并输出该数。

可采用类似于“擂台赛”的方法来找出最小数：先输入一个数，把它作为最小数；再输入下一个数并与前一个数比较，较小的数成为新的最小数；……一直进行到输入了 10 个数并比较了 9 次为止，最后保留的就是最小的数。该算法的流程图如图 3-1(b)所示。

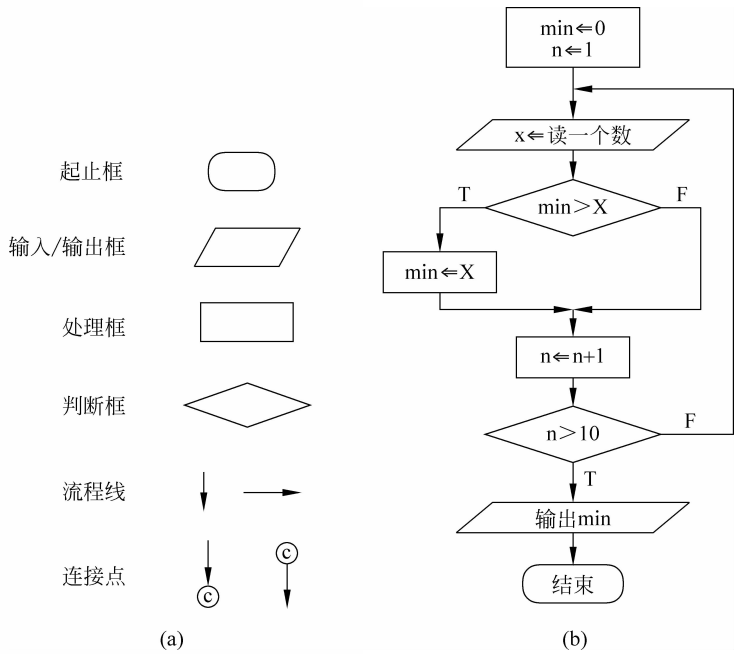

图 3-1 框形流程图的构件与求最小数的流程图

3.1.2 算法的 3 种基本结构

1. 结构化程序设计思想

计算机问世之初，受计算机性能的限制，程序设计方法的研究重点是如何运用一些技巧来节省内存空间，提高运算速度，研制出来的软件产品存在着错误多、可靠性差、维

护和修改困难等弊端或隐患。

随着计算机技术及其应用的发展，程序的可靠性和可维护性成为重要的追求目标，产生了结构化程序设计方法。这种方法的基本思路是，功能分解、逐步求精。即当要解决的问题比较复杂时，将其拆分成一些规模较小、易于理解或实现且互相独立的功能模块，每个模块还可以继续拆分为更小的模块，直到所有自完备的模块都易于理解或实现为止。

在每个模块内部，包含由各种操作构成的顺序结构、选择结构或者循环结构的功能模块。将按照这种原则和方法形成的一系列模块用高级语言表示出来，就是结构化的程序。这样设计出来的程序结构清晰，容易阅读、修改和维护，提高了程序的可靠性和可维护性。

结构化程序设计方法将数据和操纵数据的过程(C++中的函数)分别构建为相互独立的实体，编写程序时必须随时考虑所要处理的数据的格式，不便于实现代码的重复使用，也难以准确地描述实际问题。这种缺点可以通过面向对象程序设计方法加以解决。

2. 算法的3种基本结构

结构化程序设计方法规定，程序中只允许包含3种基本结构：顺序结构、选择结构和循环结构。

(1) 顺序结构：是最基本、最常见的结构。在这种结构中，各操作块按它们出现的先后顺序逐个执行。

注：一个操作块可以是一个操作、一组操作或一个基本结构等。

(2) 选择结构：在算法中，常要根据某一给定的条件是否成立来决定执行几个操作块中的哪一个。具有这种性质的结构称为选择结构。选择结构又分为双分支结构和单分支结构。双分支结构在条件成立时执行一个操作块，条件不成立时执行另一个操作块，单分支结构在条件不成立时不执行任何操作。

(3) 循环结构：算法中，经常需要在一个地方反复执行一连串的操作，这种情况称为循环结构。需要反复执行的操作块称为循环体。按是否循环的条件，可将循环结构分为两类：

① 当型循环结构：当给定条件成立时，反复执行循环体；条件不成立时终止执行。如果刚开始时条件就不成立，则一次也不执行循环体。

② 直到型循环结构：反复执行循环体，一直执行到给定条件成立时，终止执行。无论条件是否成立，至少执行一次循环体。

一般来说，同样一个问题，既可以用当型循环来解决，也可以用直到型循环来解决，也就是说，这两种循环可以互相转换。

3. N-S结构化流程图

传统的框型流程图是非结构化的。例如，各个判断分支经常不是汇集在一点；各个循环有时也不止一个入口；分支和循环经常交错在一起，这些都不符合结构化的原则。

而 N-S 结构化流程图是一种符合结构化程序设计原则的描述工具。这种流程图的顺序结构、选择结构、当型循环结构和直到型循环结构分别如图 3-2(a)、(b)、(c)和(d)所示。

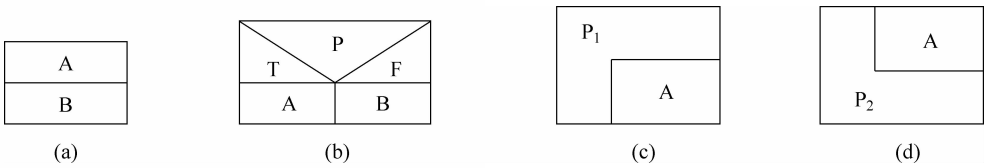

图 3-2　N-S 结构化流程图的 4 种基本结构

3.1.3　C++中实现选择结构和循环结构的语句

C++语言提供了以下语句来实现选择结构和循环结构：

- 使用 if 语句、switch 语句实现选择结构。
- 使用 while 语句、do-while 语句和 for 语句实现循环结构。
- 使用 break 语句跳出当前结构，使用 continue 语句缩短循环结构。另外，也可以使用 goto 语句实现流程的任意转向。

1. If 语句

if 语句用于实现程序中的选择结构，常用的 if 语句大体上有 3 种形式：

(1) 实现单边选择结构的 if 语句

```
if(表达式)
{    语句组
}
```

当语句组中只有一条语句时，可以省略大括号。

(2) 实现双边选择结构的 if 语句

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

(3) 实现多重选择结构 if 语句

```
if(表达式 1)
{    语句组 1
}else if(表达式 2)
{    语句组 2
}
else if(表达式 3)
{    语句组 3
```

```
}
…
}else
{    语句组 n+1
}
```

2. switch 语句

switch 语句用于实现多重选择结构。

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

该语句的执行过程为,求“表达式”的值,并逐个比较各分支中“常量表达式”的值。如果两值相等,则先执行相应分支的“语句组”,然后执行其后的其他“语句组”;当遇到 break 语句时,跳出该语句;当找到不相等的值时,执行 default 分支的语句。

3. while 语句

while 语句用于实现当型循环结构。

```
while(表达式)
{    语句组
}
```

while 语句构造的循环结构中,可以使用 if 语句与 break 语句来跳出循环,也可使用 continue 语句来中止本次循环而直接进入下一次循环。

4. do-while 语句

do-while 语句用于实现直到型循环结构。

```
do
{    语句组;
}while(表达式);
```

其中“表达式”是继续循环的条件,条件不满足时退出循环。可以使用 if 语句与 break 语句跳出循环,也可使用 continue 语句来中止本次循环而直接进入下一次循环。

5. for 语句

for 语句用于实现当型尤其是计数型循环结构。for 语句的功能很强,可以构造出灵活多样的循环结构。

```
for(表达式 1; 表达式 2; 表达式 3)
{    语句组
}
```

可以使用 if 语句与 break 语句跳出循环，也可使用 continue 语句来中止本次循环而直接进入下一次循环。

3.2 程序解析

本章中解析的 5 个程序分别用于：用海伦公式求三角形面积；使用多分支结构确定指定年份中指定月份的天数；输出指定范围内能够同时被两个数整除的所有数；穷举法求组合数；迭代法求累加和。

通过这几个程序的阅读和调试，可以较好地理解程序的 3 种基本结构，认知几种常用算法的程序实现方法并进一步体验程序设计的一般方法。

程序 3-1 求三角形的面积

本程序的功能为，已知三角形三条边的长度，按海伦公式

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

计算并输出三角形的面积。其中 s 等于三角形三条边长度之和的一半。

1. 算法分析

(1) 输入三角形三条边的长度并分别赋予变量 a、b、c。

(2) 判断：任意两边长度之和是否一定大于另一边的长度？

如果有例外，则输出“不能构成三角形”并转到(7)。

(3) 按 $s=\frac{1}{2}(a+b+c)$ 计算三角形的三条边长度之和的一半并将其值赋予 s 变量。

(4) 按海伦公式 $A=\sqrt{s(s-a)(s-b)(s-c)}$ 计算三角形的面积并将其值赋予 A 变量。

(5) 输出三角形的面积。

(6) 算法结束。

用 N-S 图表示的算法如图 3-3 所示。

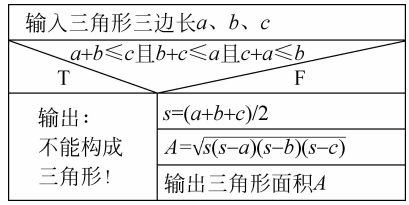

图 3-3 求三角形面积的算法

2. 程序

按照给定的算法,可编写如下程序:

```
//程序 3-1_已知 3 边长求三角形面积
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   double a,b,c;                          //双精度变量_分别表示 3 条边
    double s,A;                            //双精度变量_分别表示边之和一半及三角形面积
    cout<<"三角形 3 条边 a? b? c? ";       //输入提示
    cin>>a>>b>>c;                          //输入 3 条边
    if(a+b<=c || b+c<=a || c+a<=b)
        cout<<"不能构成三角形!"<<endl;
    else
    {   s=(a+b+c)/2.0;                     //计算 3 条边之和的一半
        A=sqrt(s*(s-a)*(s-b)*(s-c));//计算三角形面积
        cout<<"三角形面积=" <<A<<endl;     //输出三角形面积
    }
    return 0;
}
```

3. 程序运行结果

这里给出本程序的 3 次运行结果。

第 1 次:

```
三角形 3 条边 a? b? c? 2 3 5
不能构成三角形!
```

第 2 次:

```
三角形 3 条边 a? b? c? 1 2 3
不能构成三角形!
```

第 3 次:

```
三角形 3 条边 a? b? c? 3 4 5
三角形面积=6
```

请读者自行分析这几次运行的结果。

程序 3-2 确定某年某月的天数

本程序的功能为,按照用户输入的年份和月份,求解并输出该月的天数。

1. 算法分析

(1) 输入年份和月份。
(2) 分别按以下几种情况确定该年该月的天数：

- 1、3、5、7、8、10 和 12 月为 31 天。
- 4、6、9 和 11 月为 30 天。
- 闰年的 2 月为 29 天，正常年份的 2 月为 28 天，判断闰年的方法是，年份值能被 4 整除但不能同时被 100 整除或者能被 400 整除。

(3) 输出该年该月的天数。
(4) 算法结束。

2. 程序

按照给定的算法，可编写如下程序：

```
//程序 3-2_ 查询某年某月的天数
#include<iostream>
using namespace std;
int main()
{   int year,month,days;
    cout<<"查询 year 年 month 月的天数: year?month?";
    cin>>year>>month;
    switch(month)
    {   case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:days=31;
                break;
        case 4:
        case 6:
        case 9:
        case 11:days=30;
                break;
        case 2: if (year%4==0 && year%100!=0 || year%400==0)
                    days=28;
                else
                    days=29;
                break ;
        default:days=-1;                    //月份不在 1～12 之间时的处理
    }
    if(days==-1)
        cout<<"月份应在 1～12 之间!";
    else
```

```
        cout << year <<"年"<< month <<" 月有"<< days <<" 天"<< endl;
}
```

3. 程序运行结果

本程序的一次运行结果如下：

```
查询 year 年 month 月的天数: year? month? 2010 9
2010 年 9 月有 30 天
```

程序 3-3　输出 100 以内能同时被 3 和 5 整除的数

本程序的功能为,输出 100 以内能同时被 3 和 5 整除的自然数。

1. 算法分析

本程序所依据的算法如图 3-4 所示。

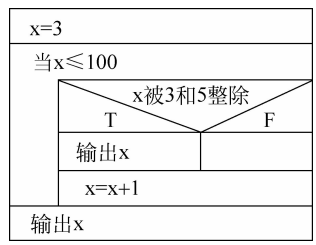

图 3-4　输出 0～100 内能被 3 和 5 整除的自然数的算法

2. 程序

按照给定的算法,可编写如下程序：

```
//程序 3-3_100 之内能同时被 3 和 5 整除的自然数
#include <iostream>
using namespace std;
//主函数
int main()
{   cout <<"100 以内可同时被 3 和 5 整除的自然数: " endl;
    for(int x = 3;x <= 100;x++)
        if(x % 3 == 0 && x % 5 == 0)
            cout << x <<'\t';
    cout << endl;
    return 0;
}
```

3. 程序的运行结果

100 以内可同时被 3 和 5 整除的自然数：

15　　30　　45　　60　　75　　90

4. 修改程序

如果数的范围扩大到 300 以内而且要求每行只显示 5 个数字，则可增设一个统计符合条件的数字个数的变量。修改后的程序如下：

```
//程序 3-3 改_300 之内能同时被 3 和 5 整除的数
#include <iostream>
using namespace std;
//主函数
int main()
{   int x,n = 0;
    cout <<"300 以内可同时被 3 和 5 整除的自然数: " endl;
    for(x = 3;x <= 300;x++)
    {   if(x % 3 == 0 && x % 5 == 0)
        {   cout << x <<'\t';
            n++;
            if(n % 5 == 0)
                cout << endl;
        }
    }
    cout << endl;
    return 0;
}
```

5. 范围扩大且限制每行数字个数的程序的运行结果

300 以内可同时被 3 和 5 整除的自然数：

15　　30　　45　　60　　75
90　　105　　120　　135　　150
165　　180　　195　　210　　225
240　　255　　270　　285　　300

6. 再次修改程序

为了增强本程序的通用性，可将程序改为运行时由用户输入自然数范围的上限以及需要试除的两个整数。修改后的程序如下：

```
//程序 3-3 再改_ 0～m 之间能被 a 和 b 整除的自然数
#include <iostream>
using namespace std;
//主函数
```

```
int main()
{   int m,a,b,n = 0;
    cout <<"0～m 之间能被 a 和 b 整除的自然数: m? a? b? ";
    cin >> m >> a >> b;
    for(int x = 3;x < = m;x++)
    {   if(x % a == 0 && x % b == 0)
        {   cout << x <<'\t';
            n++;
            if(n % 5 == 0)
                cout << endl;
        }
    }
    cout << endl;
    return 0;
}
```

7. 通用程序的运行结果

```
0～m 之间能被 a 和 b 整除的自然数: m? a? b? 1000 5 7
35      70      105     140     175
210     245     280     315     350
385     420     455     490     525
560     595     630     665     700
735     770     805     840     875
910     945     980
```

程序 3-4　穷举法求组合数

张女士有 5 本好书，分别借给 Li、Ma、Wu 3 位朋友，假定每人每次只能借一本，请编写并运行程序，输出各种不同的借法。

1. 算法分析

本程序将采用穷举法，逐个列举、判断并给出 3 个人各借一本书的所有可能性。算法中包含 3 层循环，分别用于

- 列举 Li 借 5 本书中 1 本的全部情况。
- 列举 Ma 借 5 本书中 1 本的全部情况。
- 列举 Li 和 Ma 借了不同的书时，Wu 借 5 本书中 1 本的全部情况。

当 Wu 与 Li、Ma 两人借的书都不同时，输出 3 人所借书的序号。

2. 程序

按照给定的算法，可编写如下程序：

```
//程序 3-4_ 穷举法求组合数
#include <iostream>
using namespace std;
int main()
{   int Li,Ma,Wu,nn = 0;
    for(Li = 1;Li <= 5;Li++)
        for(Ma = 1;Ma <= 5;Ma++)
            for(Wu = 1;Li!= Ma && Wu <= 5;Wu++)
                if(Wu!= Li && Wu!= Ma)
                    cout <<"第"<<++nn <<"种借法："<< Li <<"  "<< Ma <<"  "<< Wu << endl;
    return 0;
}
```

3. 程序运行结果

本程序的运行结果如下：

第 1 种借法：1 2 3
第 2 种借法：1 2 4
第 3 种借法：1 2 5
第 4 种借法：1 3 2
第 5 种借法：1 3 4
…… ……
第 20 种借法：2 4 3
第 21 种借法：2 4 5
第 22 种借法：2 5 1
…… ……
第 58 种借法：5 4 1
第 59 种借法：5 4 2
第 60 种借法：5 4 3

程序 3-5　计算 sinx 函数的值

本程序的功能为：按照等式

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

通过逐个计算当前项以及累加和的方式得出正弦函数的值，并在当前项的绝对值小于 10^{-7}（10 的－7 次方）时终止计算，然后通过与 C++标准函数 sin(x)求值结果的比较来确定计算得到的函数值是否精确。

1. 算法分析

(1) 输入自变量 x 的值。

(2) 赋初值：项数 n＝1，当前项 u＝x，累加和 sum＝x。
(3) 求当前项：u＝－u/(2＊n)/(2＊n＋1)＊x＊x。
(4) 求累加和：sum＝sum＋u。
(5) 项数增值：n＝n＋1。
(6) 如果当前项 u 的绝对值不小于 10 的－7 次方，则转向(3)。
(7) 输出累加和 sum 作为 sinx 的值。
(8) 调用 C++标准函数 sin(x)求解 sinx 的值。
(9) 判断：sum 与 sin(x)之差的绝对值是否小于 10 的－7 次方。
　　是则输出“精确!”；
　　否则输出“误差大!”。
(10) 算法结束。

2. 程序

按照给定的算法，可编写如下程序：

```
//程序 3-5_计算正弦函数的值
#include<iostream>
#include<cmath>
using namespace std;
int main()
{   double sum,u,x;
    int n=1;
    cout<<"求 sin(x): x?";
    cin>>x;
    sum=u=x;
    while(fabs(u)>1.0e-7)
    {   u=-u/(2*n)/(2*n+1)*x*x;
        sum=sum+u;
        n=n+1;
    }
    cout<<"求累加和得: sin("<<x<<")="<<sum<<endl;
    double sinx=sin(x);
    cout<<"标准函数求得: sin("<<x<<")="<<sinx<<endl;
    if(fabs(sinx-sum)<1e-7)
        cout<<"求累加和得到的 sin(x)值精确!"<<endl;
    else
        cout<<"求累加和得到的 sin(x)值误差太大!"<<endl;
    return 0;
}
```

3. 程序的运行结果

本程序的一次运行结果如下：

```
求 sin(x): x? 0.9
```

```
求累加和得: sin(0.9) = 0.783327
标准函数求得: sin(0.9) = 0.783327
求累加和得到的 sin(x)值精确!
```

3.3 实验指导

本章安排两个实验：

第 1 个实验侧重于 3 种基本结构的认知与使用。通过本实验，可以掌握使用 3 种基本结构编写程序来实现算法的一般方法。

第 2 个实验侧重于常用算法的程序实现。通过 3 个程序的编写和运行，可以体验编程序实现典型算法的一般方法。

实验 3-1 3 种基本结构

本实验中，需要编写 3 个程序：计算已知半径的圆面积、球体表面积和体积；一个正整数的自加和自乘；求多个数中的最大数。

1. 计算圆面积、球体积和表面积

【程序的功能】

输入一个数，计算以它为半径的圆面积、球体积和表面积。

【算法分析】

本程序按以下步骤完成任务：

(1) 输入半径 r。

(2) 计算圆面积：Area＝3.141 592 65 * r * r。

(3) 计算圆球表面积：Surface＝4 * 3.141 592 65 * r * r。

(4) 计算圆球体积：Volume＝3.141 592 65 * r * r * r * 4/3。

(5) 输出 Area、Surface 和 Volume。

(6) 算法结束。

【程序设计步骤】

依据给定的步骤，创建一个控制台工程，编写并运行程序。

2. 一个正整数自加或自乘

【程序的功能】

输入一个 100 以内的正整数，若为奇数则自加并输出结果，若为偶数则自乘并输出结果。

【算法分析】

本程序按以下步骤完成任务：

(1) 输入一个 100 以内的正整数 x。

(2) 判断：(x%2)！＝0？

是则 x＋＋并输出；

否则 x＝x＊x 并输出。

(3) 算法结束。

【程序设计步骤】

依据给定的步骤，创建一个控制台工程，编写并运行程序。

3. 输入一批数并找出最大数

【程序的功能】

输入 100 个数，找出其中最大的数并输出它。

【算法分析】

本程序所依据的算法如图 3-5 所示。

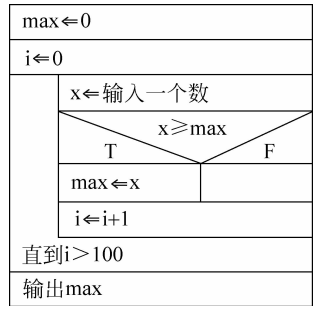

图 3-5　输入 100 个数并求最大数的算法

【程序设计步骤】

(1) 依据给定的步骤，补全下面的程序：

```
//实验 3－1－3_ 100 个数中的最大数
#include <iostream>
using namespace std;
int main()
{   double x,max;
    int i = 0;
    max = 0
    do
    {   cout <<"请输入一个数: ";
        cin >> x;
        if( ①  )
            ②
    }while( ③  )
    cout <<"100 个数中最大的是: "<< max << endl;
}
```

（2）创建一个控制台工程，编写并运行程序。

（3）将程序中的 do-while 循环改写为 for 循环，重新运行程序。

实验 3-2　迭代法与穷举法

本实验中，需要编写 3 个程序：使用迭代法求高次方程的根；使用辗转相减法（也属于迭代法）求两个数的最大公约数；使用穷举法求解不定方程。

1. 迭代法求方程的根

【程序的功能】

使用迭代法求解方程 $x^3-x-3=0$ 在 $x=1.671$ 附近的一个根。求解的基本思想是：

- 将方程改写为迭代算式 $x=\sqrt[3]{x+3}$。
- 给定初始近似值 $x_0=1.671$（双精度数字）。
- 代入算式右端，得 $X_1=\sqrt[3]{1.671+3}=1.671\,62$。

再用 x_1 作为新的 x_0，代入算式右端，得……

重复以上步骤，直到两次迭代结果之差小于 10^{-5}（1e－5）时，停止计算并输出结果。

【算法分析】

按照给定的迭代法思想，可以设计出便于程序实现的算法：

（1）输入方程的根的初始近似值：x0＝1.671（双精度数字）。

（2）定义统计迭代次数的变量：n＝0。

（3）将 x0 代入等式右端，得：x1＝(x0＋3)^(1/3)。

（4）将 x1 作为新的 x0：x1＝x0。

（5）迭代次数加 1：n＋＋。

（6）输出本次迭代得到的近似根。

（7）判断：两次迭代结果之差是否不小于 10 的负 5 次方（1e－5）？是则转向（3）。

（8）算法结束。

【程序设计步骤】

（1）依据给定的算法，补全下面程序：

```
//实验 3－2－1_ 迭代法求方程 x^3－x－3＝0 的根
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   double x0,x1;
    cout <<"x0＝? ";
    cin >> x0;
    int n＝0;
    do
```

```
    {   n++;
        ①
        cout << n <<"次迭代后的近似根 x = "<< x0 << endl;
    }while( ② );
    return 0;
}
```

(2) 创建一个控制台工程,编写并运行程序。

(3) 将程序中的 do-while 循环改写为 for 循环或 while,重新运行程序。

2. 辗转相减法求两个整数的最大公约数

【程序的功能】

使用辗转相减法求整数 189 和 81 的最大公约数。辗转相减法的基本思想是:当两个数不相等时,从大数中减去较小的数;如果较小的数不等于差,则将它作为大数,并将差作为较小的数,再从大数中减去较小的数;……如此反复执行,直到较小的数与差相等为止。

【算法分析】

按照这种思想,可以写出求两个整数最大公约数的算法:

(1) 输入两个整数,分别作为减数和被减数。

(2) 如果减数比被减数小,则交换两个变量的值(用变量 t 作中介)。

(3) 当减数不等于被减数时,从减数中减去被减数求得差。

(4) 如果被减数等于差,则差即为求得的"等数",即最大公约数(可赋予减数变量,准备输出);否则被减数作为新的减数,差作为新的被减数,转向(3)。

(5) 输出等数(可以是减数变量)。

(6) 算法结束。

【程序设计步骤】

(1) 依据给定的算法,补全下面的程序:

```
#include <iostream>
using namespace std;
int main()
{   int a,b,t;
    cout <<"两个整数?";
    cin >> a >> b;
    if(a < b)
    {
        ①
    }
    while(a!= b)
    {   t = ② ;
        if(b > t)
        {
            ③
```

```
        }else
            ④
    }
    cout<<"两个整数的最大公约数："<< ⑤ <<endl;
    return 0;
}
```

（2）创建一个控制台工程，编写并运行程序。

3. 穷举法求解不定方程

【程序的功能】

解答问题：

男职工、女职工和他们的孩子一起动手搬走10个桌子和100凳子。搬桌子时，男职工3人1个，女职工4人一个，孩子5人一个；搬凳子时，男职工每人2个，女职工每人3个，孩子每人4个。问各有多少男职工、女职工和孩子？

【算法分析】

【提示】 假设男职工x人，女职工y人，孩子z人，则得方程组：

$$\begin{cases} x/3+y/4+z/5=10 \\ 2x+3y+4z=100 \end{cases}$$

可估算出：男职工不会超过30人、女职工不会超过40人。故可令x从0循环到30，y从0循环到到40，分别计算z，然后判断是否满足条件。

【程序设计步骤】

（1）依据给定的算法，补全下面程序：

```
//实验3-2-3_ 穷举法求解不定方程
#include<iostream>
using namespace std;
int main()
{   double x,y,z;
    cout<<"男职工、女职工、孩子各有："<<endl;
    for(x=0;x<=30;x+=3)
        for( ① )
        {   z= ② )/4.0;
            if( ③ &&z>0)
                cout<<x<<"人、"<<y<<"人、"<<z<<"人。"<<endl;
        }
}
```

（2）创建一个控制台工程，编写并运行程序。