# 第5章 算法和编程

## [导语]

算法就是解决问题的一系列方法和步骤，最早可追溯到公元前2000年，古巴比伦的数学家提出了一元二次方程和解法。约公元前480年，中国人使用分配方法得到二次方程的正根。在计算机领域，算法通常指能在计算机上执行的一种解法，它接收一些值作为输入，产生一些值作为输出，算法若用某种编程语言实现就是程序，程序就是算法在计算机上的实现。

## [教学建议]

| 教学要点 | 建议课时 | 呼应的思政元素 |
| --- | --- | --- |
| 算法设计 | 2 | 引入中国古代的故事作为问题，让学生体会古人的智慧，增强民族自豪感 |
| 编程实例 | 2 | 用计算思维解决古代数学问题，让学生体会古人的智慧，培养科学自信 |


视频讲解

## 5.1 问题求解思想

人类在解决问题时一般遵循一定的步骤，如图5.1所示，从理解问题，制订解决问题的计划或方案，实施计划，到对完成情况进行评估，这一系列步骤就是问题求解过程。例如，假设某公司有员工100人，每位员工的工资和当月业绩有关，如果现在需要统计该公司当月工资总和，则可以按照以下步骤解决问题。

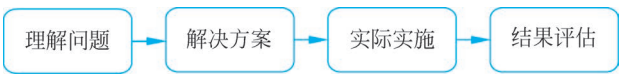



图5.1 人类解决问题的步骤

(1) 理解问题：数值求和问题。

(2) 解决方案：使用加法相加。

(3) 实际实施：找纸和笔，逐个加上每位职工当月工资。

(4) 结果评估：工资总和是否有异常。

和人类解决问题的步骤类似，利用计算机求解问题时，也包括几个阶段，但不同的是，计

算机用程序来实现算法，因此需要考虑算法实现过程中数据的存储、数据的处理、数据的输入/输出等问题，如图 5.2 所示。

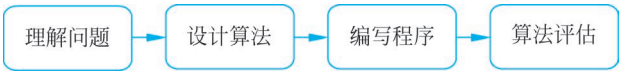



图 5.2　计算机解决问题的步骤

（1）理解问题：数值求和问题，输入 $n$ 个工资信息，输出 $n$ 个工资的和。

（2）设计算法：输入数据放在数组里，采用循环语句将数组里的数据逐个相加，并输出最终结果到终端。

（3）编写程序，代码如下：

```
#include <iostream>
using namespace std;
const int N = 100;
int main()
{
  int i;
  float salary[N];
  double total = 0;
  for(i = 0;i < 100;i++)
    {
    cin >> salary[i];
    total += salary[i];
    }
  cout << total << endl;
  return 0;
}
```

（4）算法评估：时间复杂度为 $O(n)$，空间复杂度为 $O(1)$。

## 5.2　算法设计

### 5.2.1　算法特性


视频讲解

算法指的是解决问题的一种方法或一个过程，是若干指令的有穷序列。算法具有以下 5 个重要特性。

（1）有穷性：算法必须在执行有穷步之后结束，且每一步都可以在有穷的时间内完成。

（2）确定性：算法的每条指令必须有确切的含义，确保无歧义。

（3）可行性：算法中描述的操作都可以通过已经实现的基本运算执行有限次来实现。

（4）输入：一个算法有 0 个或多个输入。

（5）输出：一个算法有 1 个或多个输出。

算法与程序的区别如下。

（1）算法是解决问题的方法、步骤。

（2）程序是算法的具体代码实现。

（3）算法是程序设计的核心，算法的好坏直接决定了程序的效率。


视频讲解

## 5.2.2 算法的表示

### 1. 算法的自然语言表示

用自然语言直接描述算法,简单、便于阅读,如例 5-1。

**【例 5-1】** 用自然语言描述:计算并输出矩形的面积。

**【解 5-1】** 算法主要分为输入、处理和输出三个步骤,具体如下:

(1) 输入矩形的长 $a$ 和宽 $b$。

(2) 计算 $a \times b$ 的值,并赋给变量 $s$。

(3) 输出 $s$。

### 2. 算法的伪代码表示

伪代码用类似自然语言的形式表达算法,结构清晰、代码简单、可读性好。

**【例 5-2】** 用伪代码描述:计算并输出矩形面积。

**【解 5-2】** 伪代码如下所示:

```
(1) Begin
(2) 定义 a,b,s
(3) 输入 a,b 的值
(4) s = a * b
(5) 输出 s
(6) End
```

### 3. 程序流程图表示

程序流程图用图的形式画出程序流向,是算法的一种图形化表示方法,具有直观、清晰、更易理解的特点。程序流程图由处理框、判断框、起止框、输入/输出框、连接点、流程线等构成,程序流程图的符号如表 5.1 所示。

表 5.1 程序流程图的符号

| 符 号 | 含 义 |
| --- | --- |
|  | 起止框用圆边框表示,代表算法的开始或结束 |
|  | 处理框用矩形框表示 |
|  | 输入/输出框用平行四边形表示 |
|  | 判断框用菱形表示 |
|  | 箭头表示流程线 |
|  | 圆圈表示连接点 |

**【例 5-3】** 用程序流程图描述:输入 x,y,计算 z=x/y,输出 z。

**【解 5-3】** 对应的程序流程图如图 5.3 所示。

### 4. N-S 图表示

N-S 图是美国学者 L. Nassi 和 B. Shneiderman 于 1973 年提出的一种新的流程图形式。N-S 图去掉了带箭头的流程线,全部算法写在一个矩形框内,算法的每个处理步骤用一个矩形框表示,且矩形框中可以嵌套另一个矩形框,或者说,由一些基本的框组成一个更大的框。N-S 图也被称为 N-S 结构化流程图。图 5.4 列出了 N-S 图的三种基本结构。

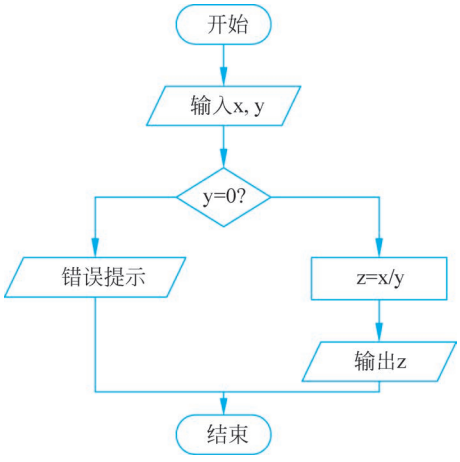
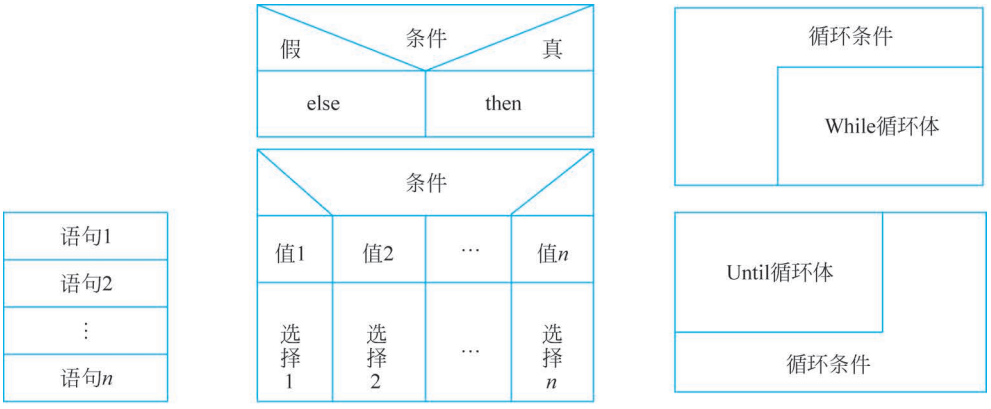



图 5.3　程序流程图




(a) 顺序结构　(b) 选择结构　(c) 循环结构

图 5.4　N-S 图

【例 5-4】　输入整数 m，判断它是否为素数。

【解 5-4】　其 N-S 图如图 5.5 所示。

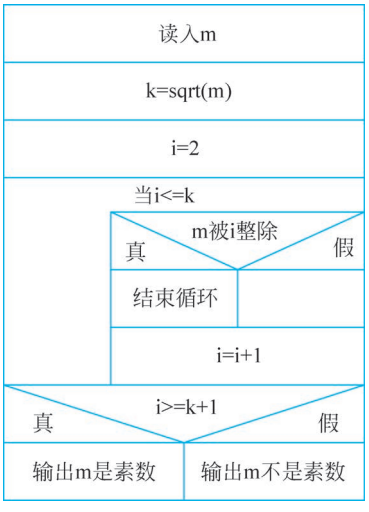



图 5.5　例 5-4 的 N-S 图


视频讲解

### 5.2.3 算法复杂度的度量标准

算法复杂度指的是算法运行时所需要的计算资源的量，主要包括时间资源和空间资源，分别对应时间复杂度和空间复杂度。算法效率的度量是评价算法优劣的重要依据。一个算法的复杂度的高低体现在运行该算法所需要的计算机资源的多少上面，所需的资源越多，算法的复杂度越高；反之，所需的资源越少，则该算法的复杂度越低。算法复杂度只依赖于算法要求解的问题的规模、算法的输入和算法本身。如果用 $C$ 表示复杂性，分别用 $N$、$I$ 和 $A$ 表示算法要求解的问题的规模、算法的输入和算法本身，那么应该有 $C=F(N,I,A)$，$F()$ 为关于 $N$、$I$、$A$ 的函数。

**1. 算法的时间复杂度**

算法的时间复杂度是定性描述该算法的运行时间。一个算法执行所耗费的时间，不仅和算法有关，和运行时的环境也有关，因此通过在计算机上运行程序获取某个算法的绝对运行时间，称为事后估计法。使用绝对运行时间衡量算法的效率是不合适的，如果不关注与计算机软硬件相关的因素，则可以认为一个算法的时间效率只依赖于问题的规模(通常用整数 $n$ 表示)，是问题规模的函数，通常用算法中基本操作重复执行的次数来表示，称为语句频度或时间频度，用 $T(n)$表示。若有某个辅助函数 $f(n)$，使得当 $n$ 趋近无穷大时，$T(n)/f(n)$的极限值为不等于零的常数，则称 $f(n)$是 $T(n)$的同数量级函数。记作 $T(n)=O(f(n))$，称 $O(f(n))$ 为算法的渐进时间复杂度，简称时间复杂度。时间复杂度常用 $O$ 表示，不包括这个函数的低阶项和首项系数。

在各种不同的算法中，若算法中语句执行次数为一个常数，则时间复杂度为 $O(1)$。另外，在时间频度不相同时，时间复杂度有可能相同，如 $T(n)=n^2+3n+4$ 与 $T(n)=4n^2+2n+1$ 的频度不同，但时间复杂度相同，都为 $O(n^2)$。

常见的时间复杂度量级：常数阶 $O(1)$<对数阶 $O(\log n)$<线性阶 $O(n)$<线性对数阶 $O(n\log n)$<平方阶 $O(n^2)$<立方阶 $O(n^3)$<$k$ 次方阶 $O(n^k)$<指数阶$(2^n)$，从前往后时间复杂度越来越大，执行的效率越来越低。下面我们举一些例子来讨论一下时间复杂度。

(1) 常数阶 $O(1)$。

无论代码执行了多少行，只要没有循环等复杂结构，那这个代码的时间复杂度就都是 $O(1)$，如下述程序段：

```
int i = 1;
int j = 2;
++i;
j++;
int m = i + j;
```

上述代码在执行的时候，它所耗费的时间并不随着某个变量的增长而增长，那么无论这类代码有多长，即使有几万、几十万行，都可以用 $O(1)$来表示它的时间复杂度。

(2) 线性阶 $O(n)$。

请看如下这段代码：

```
for(i=1; i<=n; ++i)
{
```

```
    j = i;
    j++;
}
```

这段代码，for 循环里面的代码会执行 $n$ 遍，因此它消耗的时间是随着 $n$ 的变化而变化的，因此这类代码都可以用 $O(n)$来表示它的时间复杂度。

(3) 对数阶 $O(\log n)$。

请看如下程序段：

```
int i = 1;
while(i<n)
{
    i = i * 2;
}
```

从上面的代码可以看到，在 while 循环里面，每次都将 i 乘以 2，乘完之后，i 距离 n 就越来越近了。我们试着求解一下，假设循环 x 次之后，i 就大于 $n$ 了，此时这个循环就退出了，也就是说 2 的 x 次方等于 n，那么 $x=\log_2 n$，也就是说当循环 $\log_2 n$ 次以后，这个代码就结束了。因此这段代码运行的时间复杂度为 $O(\log n)$。

(4) 线性对数阶 $O(n\log n)$。

时间复杂度为 $O(\log n)$的代码循环 $n$ 遍，它的时间复杂度就是 $nO(\log n)$，也就是 $O(n\log n)$。下面的程序段便是这种情况：

```
for(m=1; m<n; m++)
{
   i = 1;
   while(i<n)
   {
      i = i * 2;
   }
}
```

(5) 平方阶 $O(n^2)$。

如果把 $O(n)$ 的代码再嵌套循环一遍，它的时间复杂度就是 $O(n^2)$ 了，例如，以下程序片段：

```
for(x=1; i<=n; x++)
{
   for(i=1; i<=n; i++)
    {
       j = i;
       j++;
    }
}
```

这段代码嵌套了 2 层 for 循环，它的时间复杂度就是 $O(n\times n)$，即 $O(n^2)$，如果将其中一层循环的 n 改成 m，如下：

```
for(x = 1; i <= m; x++)
{
   for(i = 1; i <= n; i++)
    {
       j = i;
       j++;
    } }
```

此时程序段的时间复杂度就变成了 $O(m \times n)$，立方阶 $O(n^3)$、$k$ 次方阶 $O(n^k)$ 与以上类似。

**2. 算法的空间复杂度**

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的一个度量，包括：①程序本身所占空间；②输入数据所占空间；③辅助变量所占空间。输入数据所占空间只取决于问题本身，和算法无关，故只需分析除输入和程序之外的辅助变量所占的额外空间。空间复杂度一般也作为问题规模 $n$ 的函数，以数量级形式给出，记作 $S(n)=O(g(n))$。常见的 $S(n)$有 $O(1)$、$O(n)$、$O(n^2)$。

(1) 空间复杂度 $O(1)$。

如果算法执行所需要的临时空间不随着某个变量 $n$ 的大小而变化，即此算法空间复杂度为一个常量，记为 $O(1)$。举例如下：

```
int i = 1;
int j = 2;
++i;
j++;
int m = i + j;
```

代码中变量 i、j、m 所分配的空间都不随着处理数据量的变化而变化，因此它的空间复杂度为 $S(n)=O(1)$。

(2) 空间复杂度 $O(n)$。

举例如下：

```
1. int[] m = new int[n]
2. for(i = 1; i <= n; ++i)
3. {
4. j = i;
5. j++;
6. }
```

这段代码中，第一行动态申请了一个数组，数组的大小为 $n$，这段代码的 2～6 行，虽然有循环，但没有再分配新的空间，因此，这段代码的空间复杂度主要看第一行即可，即 $S(n)=O(n)$。


视频讲解

## 5.2.4 常见基础算法

**1. 穷举算法**

穷举算法的基本思想是根据题目的部分条件确定答案的大致范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。若某个情况验证符合题目的全部条件，则

为本问题的一个解；若验证完全部情况后都不符合题目的全部条件，则本题无解。穷举法也称为枚举法。例如，早在1500年前的《孙子算经》中，记载了一道有趣的数学题——鸡兔同笼问题。

**【例 5-5】** 鸡兔同笼问题，原文如下：

今有雉兔同笼，上有三十五头，下有九十四足，问雉兔各几何？

**【解 5-5】** 解决鸡兔同笼问题采用穷举思想：设鸡有 $x$ 只，又因为每只鸡有2条腿，所以鸡的数量小于或等于47只；因为鸡兔一共有35只头，所以兔的数量为 $35-x$，那么可以把鸡的数量一一列举出来，得到兔的数量，并验证是否满足题目条件。

具体的程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    for(x = 0;x <= 47;x++)
       if(2 * x + 4 * (35 - x) == 94)
       cout << x <<" "<< 35 - x << endl;
    return 0;      }
```

### 2. 贪心算法

贪心算法又称贪婪算法，是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，算法得到的是在某种意义上的局部最优解，贪心算法不是对所有问题都能得到整体最优解，贪心算法最重要的就是贪心策略的选择。例如西汉司马迁所著的《史记·孙子吴起列传》记载的历史上有名的田忌赛马的故事。

**【例 5-6】** 田忌赛马，原文描述如下：

齐使者如梁，孙膑以刑徒阴见，说齐使。齐使以为奇，窃载与之齐。齐将田忌善而客待之。忌数与齐诸公子驰逐重射。孙子见其马足不甚相远，马有上、中、下辈。于是孙子谓田忌曰："君弟重射，臣能令君胜。"

田忌信然之，与王及诸公子逐射千金。及临质，孙子曰："今以君之下驷彼上驷，取君上驷与彼中驷，取君中驷与彼下驷。"既驰三辈毕，而田忌一不胜而再胜，卒得王千金。于是忌进孙子于威王。威王问兵法，遂以为师。

**【解 5-6】** 将原文进行翻译，释义如下：

齐国使者到大梁来，孙膑以刑徒的身份秘密拜见，用言辞打动齐国使者。齐国使者觉得此人不同凡响，就偷偷地用车把他载回齐国。齐国将军田忌赏识他并像对待客人一样礼待他。田忌经常与齐国诸公子赛马，设重金赌注。孙膑发现他们的马脚力都差不多，可分为上、中、下三等。于是孙膑对田忌说："您只管下大赌注，我能让您取胜。"

田忌相信并答应了他，与齐王和诸公子用千金来赌胜。比赛即将开始，孙膑说："现在用您的下等马对付他们的上等马，拿您的上等马对付他们的中等马，拿您的中等马对付他们的下等马。"三场比赛完后，田忌一场不胜而两场胜，最终赢得齐王的千金赌注。于是田忌把孙膑推荐给齐威王。威王向他请教兵法后，就把他当作老师。

具体在解决田忌赛马问题时，采用贪心策略，即考虑孙膑的策略是充分利用每一匹马的

战斗力,若己方的最好马优于对方的最好马,则使己方的最好马战胜对方的最好马;若己方的最好马劣于对方的最好马,则必定要输一次,用己方最劣的一匹马消耗对方的最好马,从而使己方的最好马可以对战对方下一个等级的马,增大赢的概率。当双方的最好马实力相当时,为了尽可能多赢,检查双方最劣马的实力,若己方最劣马强于对方,则先用己方最劣马战胜对方最劣马,保证赢一局,若己方最劣马弱于对方,则该匹马注定要输,用其消耗对方最高战斗力。若双方最劣马实力相当,仍用己方最劣马消耗对方最好马,当只看最优和最劣两组数据时,这样最坏的结果是平局,然而可以增大中间数据赢的概率。

把三匹马较简单的情况扩展到 $n$ 匹马,则形成田忌赛马的一般问题,具体描述如下。

**【问题描述】** 如果 3 匹马变成 $n$ 匹($n \leqslant 100$),齐王仍然让他的马按照从优到劣的顺序出赛,田忌可以按任意顺序选择他的赛马出赛。赢一局,田忌可以得到 200 两银子。输一局,田忌就要输掉 200 两银子。已经知道齐王和田忌的所有马的奔跑速度,请设计一个算法,帮助田忌赢得最多的银子。

**【贪心策略】**

(1) 如果田忌的最快马快于齐王的最快马,则两者比。

(2) 如果田忌的最快马不快于齐王的最快马,则比较田忌的最慢马和齐王的最慢马。

① 若田忌的最慢马快于齐王的最慢马,两者比。

② 其他,则拿田忌的最慢马和齐王的最快马比。

具体的程序代码如下:

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

int count1 = 0;                                  //胜利场次
int count2 = 0;                                  //平局场次
int cnt = 0;                                     //败北场次

int tianRac(vector<long long>Tian, vector<long long>King, int n)
{
    int money = 0;                               //田忌赢的钱
    int tianh = 0, tiane = n-1, kingh = 0, kinge = n-1;
    //分别标记田忌队和齐王队的最快和最慢的马
    //共有 n 次比赛,每进行一次,就换下一匹马
    for (int i = 0; i<n; i++){
    //田忌快马比齐王快马快时,那就与其一较高下
        if (Tian[tianh]>King[kingh]){
            count1++;
            tianh++;                             //下一个
            kingh++;                             //下一个
        }
    //田忌快马不比齐王快马快时,比较慢马是否能赢
        else{
            //田忌的慢马比齐王的慢马快时
            if (Tian[tiane]>King[kinge]){
                count1++;
                tiane--;
```

```
                kinge--;
            }
            //田忌的慢马不比齐王的慢马快,就用慢马和他的快马比
            else if (Tian[tiane] > King[kingh]){
                cnt++;
                tiane--;
                kingh++;
            }
        }
    }
    count2 = n - count1 - cnt;
    money = count1 * 200 - cnt * 200;
    return money;                    //返回田忌赢的钱
}
int main()
{
    int n;                           //比赛双方马的数量
    cout << "公等马几何" << endl;
    cin >> n;
    vector < long long > tian;
    vector < long long > king;
    long long x;
    cout << "将军 马之疾" << endl;
    for(int i = 0; i < n; i++){
        cin >> x ;
        tian.push_back(x);
    }
    cout << "王 马之疾" << endl;
    for(int i = 0; i < n; i++){
        cin >> x ;
        king.push_back(x);
    }
    sort(tian.begin(),tian.end(),greater < long long >());
    sort(king.begin(),king.end(),greater < long long >());
    int result = tianRac(tian, king, n);
    if(result > 0){
        cout << "将军 胜" << result << "两" << endl;
    }
    else if (result == 0){
        cout << "和" << endl;
    }
    else{
        cout << "将军 输" << -result << "两" << endl;
    }
    return 0;
}
```

### 3. 迭代算法

迭代法也称“辗转法”,是一种不断用变量的老值递推新值的方法,常用于数值计算,例如,常见的累加求和、累乘求积都是迭代法的基础应用。

**【例 5-7】** 辗转相除法,也称欧几里得算法,是求最大公约数的算法。辗转相除法首次出现于欧几里得的《几何原本》中,而在中国则可以追溯至东汉出现的《九章算术》。

两个整数的最大公约数是能够同时整除它们的最大的正整数。辗转相除法基于如下原理：两个整数的最大公约数等于其中较小的数和两数的差的最大公约数。例如，252 和 105 的最大公约数是 21(252＝21×12，105＝21×5)；因为 252－105＝147，所以 147 和 105 的最大公约数也是 21。在这个过程中，较大的数缩小了，所以继续进行同样的计算可以不断缩小这两个数直至其中一个变成零。这时，所剩下的还没有变成零的数就是两数的最大公约数。

**【解 5-7】** 用辗转相除法求最大公约数的步骤如下。

(1) 输入两个非负整数 $a$、$b$。

(2) 较大的数为 $a$，否则交互 $a$、$b$ 的值。

(3) 把 $a$ 除以 $b$ 的余数赋值给 $b$，$a$ 取原来的 $b$ 值。

(4) 不断重复上述过程，直到余数为 0 为止，此时的除数即为 $a$、$b$ 的最大公约数。

用迭代法求最大公约数要比短除法效率高得多，即使手工求解也是一个快捷的方法，由此可见中国古人的智慧。

辗转相除法的程序代码如下：

```
#include <iostream>
using namespace std;
int main()
{
int m, n, r;
cin >> m >> n;
if(m < n)
{ int t = m;
      m = n;
      n = t;
}
    r = m % n;
    while( r )
    {
        m = n;
        n = r;
        r = m % n;
    }
    cout << n << endl;
    return 0; }
```

### 4. 递归算法

童年时，小孩央求大人讲故事，大人会讲这样的故事：从前有座山，山上有个庙，庙里有个老和尚和小和尚，老和尚给小和尚讲故事，讲的是：从前有座山，山上有个庙，……

这个故事隐含递归的思想，递归算法是把问题转换为规模缩小了的同类问题的子问题，然后递归调用自身函数来表示问题的解。能采用递归描述的算法通常有这样的特征：为求解规模为 $N$ 的问题，设法将它分解成规模较小的问题，然后从这些小问题的解构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。特别地，当规模 $N＝1$ 时，可以直接得到问题的解。

递归算法的两大要素分别为递归表达式和递归出口，递归表达式是问题分解的关键，递归出口是终止条件，如果没有终止条件，则递归永远无法结束，程序会变为死循环。相应的递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题（规模为 $n$）的求解推到比原问题简单一些的问题（规模小于 $n$）的求解。在回归阶段，当获得最简单情况的解后，逐级返回，依次得到稍复杂问题的解。例如著名的斐波那契数列问题。

**【例 5-8】** 求 $n$ 的阶乘 $n!=1\times2\times3\times\cdots\times n$。

**【解 5-8】** 求 $n!$可以采用迭代方法，程序如下：

```
int factorial(int n) {
  int s = 1;
   for(int i = 1;i <= n;i++)
     s * = i;
   return s;
}
```

也可以采用递归求解，首先给出递归的表达式：

$$n! = \begin{cases} 1, & n=1 \text{ 或 } 0 \\ (n-1)!\times n, & \text{其他} \end{cases}$$

上式中，当 $n=1$ 或 0 时，$n!=1$，这是递归的终止条件，即递归出口。递归函数一定要有终止条件，否则，它将会一直调用自己，进入死循环。$n!=(n-1)!\times n$ 为递归表达式，确定了递归的任务。递归出口和递归表达式为递归的两个要素，缺一不可。

```
int factorial(int n) {
  if(n == 1 || n == 0)
return 1;
   return factorial(n - 1) * n;
}
```

**【例 5-9】** 斐波那契数列（Fibonacci sequence）。

斐波那契数列又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1，1，2，3，5，8，13，21，34，……在数学上，斐波那契数列以如下递推的方法定义：

$$F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2) \quad (n \geqslant 2, n \in \mathbf{N})$$

根据上式，求解 $F(n)$，并将其进行推导，以求解 $F(n-1)$和 $F(n-2)$。

**【解 5-9】** 为计算 $F(n)$，必须先计算 $F(n-1)$和 $F(n-2)$，而计算 $F(n-1)$和 $F(n-2)$，又必须先计算 $F(n-3)$和 $F(n-4)$。以此类推，直至 $F(1)$和 $F(0)$，返回可得到 $F(2)$的结果，同样地，在得到了 $F(n-1)$和 $F(n-2)$后，返回可得到 $F(n)$的结果。

```
int fib(int n) {
    if(n == 1 || n == 2) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

**【例 5-10】** 最大公约数的递归法。

**【解 5-10】** 求两个正整数的最大公约数的辗转相除法可以表达如下：

$$\gcd(m,n)=\begin{cases} n, & m\%n=0 \\ \gcd(n,m\%n), & \text{否则} \end{cases}$$

其函数实现如下所示：

```
int gcd(int m, int n) {
    if(m % n == 0) return n;
    return gcd(n,m % n); }
```

### 5. 排序算法

排序是日常生活常见的操作之一。例如，在淘宝、京东等平台对商家按照不同规则排序，以帮助用户更快地搜到满足自己需求的商品。高校按照学生综合成绩排名，挑出最优秀的若干位学生颁发奖学金。再例如，飞机场一般都有几十个登机门，每天有几百架飞机降落和起飞，登机门的种类和大小是不同的，而飞机的机型和大小也是不同的。飞机按时刻表降落和起飞，当飞机占有登机门时，旅客可上下飞机，飞机也要接受加油、维护和装卸行李等服务。但也有意外发生，比如由于天气和机场的原因，飞机不能起飞，登机时间推迟。调度人员如何制订一个登机门的分配方案，使机场的利用率最高或晚点起飞的飞机最少。这一系列问题的求解都要以排序操作为基础。

所谓排序，就是使一串记录，按照其中某个或某些关键字的大小进行递增或递减排列的操作。排序算法就是使记录按照要求排列的方法。排序算法在很多领域具有很高的应用价值，尤其是在大量数据的处理方面。一个优秀的算法可以节省大量的资源。在各个领域中考虑到数据的各种限制和规范，要得到一个符合实际的优秀算法，得经过大量的推理和分析。

根据排序时数据能否全部放入内存，排序算法分为内部排序和外部排序。内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，不能一次容纳全部的排序记录，在排序过程中需要访问外存。常见的内部排序算法主要有交换类排序、插入类排序和选择类排序，对应的基础算法为冒泡排序、直接插入排序和选择排序。下面，我们一一进行介绍。

1) 冒泡排序(Bubble Sort)

冒泡排序的基本思想是重复地访问要排序的元素列，依次比较两个相邻的元素，如果元素逆序，就把它们交换过来。重复进行上述操作，直到没有相邻元素需要交换，也就是说该元素列已经排序完成。例如，图 5.6 中无序记录的初始关键字为{9,8,7,6,5,4,3,2,1,0}，

| 初始关键字 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| i=0 | 0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| i=1 | 0 | 1 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| i=2 | 0 | 1 | 2 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| i=3 | 0 | 1 | 2 | 3 | 9 | 8 | 7 | 6 | 5 | 4 |
| i=4 | 0 | 1 | 2 | 3 | 4 | 9 | 8 | 7 | 6 | 5 |
| i=5 | 0 | 1 | 2 | 3 | 4 | 5 | 9 | 8 | 7 | 6 |
| i=6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 9 | 8 | 7 |
| i=7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 8 |
| i=8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

图 5.6　冒泡排序

利用冒泡排序思想对初始记录按照从小到大的顺序排列。通过无序区中相邻记录关键字间的比较和位置的交换，使关键字最小的记录如气泡一般逐渐往上“漂浮”直至“水面”。整个算法是从最下面的记录开始，对每两个相邻的关键字进行比较，且使关键字较小的记录换至关键字较大的记录之上，使得经过一趟冒泡排序后，关键字最小的记录到达最上端，接着，再在剩下的记录中找关键字次小的记录，并把它换在第二个位置上。以此类推，一直到所有记录都有序。

这个算法名字的由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端(升序或降序排列)，就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名“冒泡排序”。

```
void BubbleSort(int R[],int n)
{       int i,j;
        int temp;
         for(i=0;i<n-1;i++)
         {     for(j=n-1;j>i;j--)
                     if(R[j]<R[j-1])
            {   temp=R[j];
             R[j]=R[j-1];
             R[j-1]=temp;
              }
         }
}
```

2) 直接插入排序(Insertion Sort)

直接插入排序的基本思想是：每次将一个待排序的记录，按其关键字大小插入前面已经排好序的子表中的适当位置，直到全部记录插入完成。假设待排序的记录存放在数组R[0..n−1]中，排序过程的某一中间时刻，R被划分成两个子区间R[0..i−1]和R[i..n−1]，其中，前一个子区间是已排好序的有序区，后一个子区间则是当前未排序的部分，不妨称其为无序区。直接插入排序的基本操作是将当前无序区的第1个记录R[i]插入有序区R[0..i−1]中适当的位置上，使R[0..i]变为新的有序区。重复这个过程，直到所有的无序记录都插入有序数组中。如图5.7所示，从第2个元素开始，依次将每个记录按照升序插入前面的有序序列中。

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 初始关键字 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| i=1 | [8 | 9] | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| i=2 | [7 | 8 | 9] | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| i=3 | [6 | 7 | 8 | 9] | 5 | 4 | 3 | 2 | 1 | 0 |
| i=4 | [5 | 6 | 7 | 8 | 9] | 4 | 3 | 2 | 1 | 0 |
| i=5 | [4 | 5 | 6 | 7 | 8 | 9] | 3 | 2 | 1 | 0 |
| i=6 | [3 | 4 | 5 | 6 | 7 | 8 | 9] | 2 | 1 | 0 |
| i=7 | [2 | 3 | 4 | 5 | 6 | 7 | 8 | 9] | 1 | 0 |
| i=8 | [1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9] | 0 |
| i=9 | [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9] |

图 5.7　插入排序

```
void InsertSort(int R[],int n)
{       int i,j;
         int temp;
```

```
        for(i = 1;i < n;i++)
        {   temp = R[i];
            j = i - 1;
            while(j > = 0 && temp < R[j])
            {     R[j + 1] = R[j];
                  j -- ;
            }
            R[j + 1] = temp;
        }
}
```

3) 选择排序(Selection Sort)

选择排序的基本思想是：每趟从待排序记录中选出排序码最小(最大)的记录，放在已排序记录序列的最前(后)，第 i 趟排序开始时，当前有序区和无序区分别为 R[0..i−1]和 R[i..n−1](0≤i<n−1)，该趟排序则是从当前无序区中选出关键字最小的记录 R[k]，将它与无序区的第 1 个记录 R[i]交换，使 R[0..i]和 R[i+1..n−1]分别变为新的有序区和新的无序区，具体例子如图 5.8 所示，第 1 趟排序，选择最小值 0，放到记录的第 1 个位置，第 2 趟排序，选择无序区最小值 1，放到记录的第 2 个位置，以此类推，直到所有元素都放到正确的位置。

| 初始关键字 | 6 | 8 | 7 | 9 | 0 | 1 | 3 | 2 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|
| i=0 | 0 | 8 | 7 | 9 | 6 | 1 | 3 | 2 | 4 | 5 |
| i=1 | 0 | 1 | 7 | 9 | 6 | 8 | 3 | 2 | 4 | 5 |
| i=2 | 0 | 1 | 2 | 9 | 6 | 8 | 3 | 7 | 4 | 5 |
| i=3 | 0 | 1 | 2 | 3 | 6 | 8 | 9 | 7 | 4 | 5 |
| i=4 | 0 | 1 | 2 | 3 | 4 | 8 | 9 | 7 | 6 | 5 |
| i=5 | 0 | 1 | 2 | 3 | 4 | 5 | 9 | 7 | 6 | 8 |
| i=6 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 8 |
| i=7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 8 |
| i=8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

图 5.8　选择排序

```
void SelectSort(int R[],int n)
{    int i,j,k;
      int temp;
      for(i = 0;i < n - 1;i++)
      {     k = i;
         for(j = i + 1;j < n;j++)
             if(R[j]< R[k])
                k = j;
             if(k!= i)
             { temp = R[i];
               R[i] = R[k];
               R[k] = temp;
             }
        }
}
```

## 5.3　编程语言

自然语言是人与人之间沟通的工具，编程语言则是人与计算机之间沟通交流的工具，我们知道计算机是基于0、1二进制序列工作的，而二进制序列表示的指令对于人类来说过于晦涩难懂，因此就创建了计算机编程语言，用接近自然语言的形式编写解决问题的源程序。

### 5.3.1　常见编程语言

程序语言的数量已超过上千种，不同程序语言解决的问题各不相同，没有一种语言可以解决所有的问题，当问题随着环境变化时，就需要创造新的程序语言来适用。如图5.9所示，TIOBE 2023年7月编程语言排行榜上显示，其中前5名和2022年7月相比几乎没有变化，仍然是Python、C、C++、Java和C#。Python在人工智能领域的带动下势不可挡，仍是第1名。自2021年10月登顶月度榜首之后，Python已牢牢占据该位置1年多，而且市场占有率继续稳步提升。JavaScript达到了第6位，创历史新高。排名前20的榜单中，MATLAB为第10位、Scratch为第12位、Rust为第17位，它们都也追平了各自的历史最高纪录。表5.2中给出了一些常见编程语言的简介、特点和目前市场需求。

| Jul 2023 | Jul 2022 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Python | 13.42% | -0.01% |
| 2 | 2 | | C | 11.56% | -1.57% |
| 3 | 4 | ^ | C++ | 10.80% | +0.79% |
| 4 | 3 | v | Java | 10.50% | -1.09% |
| 5 | 5 | | C# | 6.87% | +1.21% |
| 6 | 7 | ^ | JavaScript | 3.11% | +1.34% |
| 7 | 6 | v | Visual Basic | 2.90% | -2.07% |
| 8 | 9 | ^ | SQL | 1.48% | -0.16% |
| 9 | 11 | ^ | PHP | 1.41% | +0.21% |
| 10 | 20 | ^^ | MATLAB | 1.26% | +0.53% |
| 11 | 18 | ^^ | Fortran | 1.25% | +0.49% |
| 12 | 21 | ^^ | Scratch | 1.07% | +0.35% |
| 13 | 12 | v | Go | 1.07% | -0.07% |
| 14 | 8 | vv | Assembly language | 1.01% | -0.64% |
| 15 | 14 | v | Delphi/Object Pascal | 0.98% | -0.08% |
| 16 | 15 | v | Ruby | 0.91% | -0.08% |
| 17 | 29 | ^^ | Rust | 0.89% | +0.47% |
| 18 | 10 | vv | Swift | 0.88% | -0.39% |
| 19 | 19 | | R | 0.87% | +0.11% |
| 20 | 26 | ^^ | COBOL | 0.86% | +0.33% |

图5.9　TIOBE编程语言排行榜

表 5.2 常见编程语言简介

| 语　言 | 特　点 | 缺　点 | 职业前景 |
| --- | --- | --- | --- |
| Python | 人工智能、机器学习方向最佳的编程语言，具有广泛的库支持，为多种平台和系统提供支持，容易学习 | 由于 Python 是一种解释性编程语言，所以速度较慢 | 职位空缺最多，平均工资高 |
| Java | 服务器端最好的编程语言，广泛用于构建企业级 Web 应用程序 | 比 C 和 C++等本地编译的编程语言慢 | 很多大企业都在用 |
| C/C++ | 最通用的编程语言，C 和 C++在编程世界中占有重要地位，快速、稳定 | 比较底层，软件开发能力不足 | 很好 |
| C# | 微软开发的面向对象编程语言，广泛用于后端编程、构建游戏（使用 Unity）、构建 Windows 手机应用程序和许多其他用例 | 比较适合构建桌面应用程序 | 需求量不大 |
| Visual Basic | 微软开发的另一种编程语言，仍然非常流行，适合开发应用程序 | 不支持继承、无原生支持多线程、异常处理不完善 | 还不错 |
| JavaScript | Web 前端编程语言，客户端最常用的脚本语言，被广泛用于设计交互式前端应用程序，不需要编译，非常快 | 限于前端，适用范围比较小 | 很好 |
| Assembly Language | 一种面向机器的低级语言，通常是为特定的计算机专门设计的 | 面向机器，处于整个计算机语言层次结构的底层，缺乏可移植性 | 很好 |
| SQL | 结构查询语言，用于访问、操作、与数据库通信 | 难扩展，仅适合关系数据库 | 还不错 |
| Swift | iOS 端最高效的编程语言 | 有限的社区支持和资源，在编程场景中相对较新 | 非常好 |
| GO(Golang) | 是谷歌设计的一种编程语言，为多线程提供了出色的支持，语法简洁，更容易学习 | 没有 GUI 库支持 | 非常好 |
| PHP | 流行的后端编程语言之一。尽管 PHP 面临着来自 Python 和 JavaScript 的激烈竞争，但市场仍然需要大量的 PHP 开发人员 | 完全使用 PHP 开发网站要慢一些，缺乏安全性，错误处理能力差 | 非常好 |

## 5.3.2 编译方式

### 1. 编译器

编写的源代码在运行之前，编译器把源代码转换成机器代码，这一过程包括预处理、编译、链接三个步骤。预处理是对源文件进行一些文本方面的操作，比如文本替换、文件包含、删除部分代码等，常用于处理宏定义、文件包含、条件编译等操作。预处理命令要放在所有函数之外，而且一般都放在源文件的前面。例如 C 语言程序中，以“#”号开头的命令称为预处理命令。编译把预处理过的文件翻译为二进制机器语言，编译器的输出结果称为目标代码。在一些语言中，必须把目标代码链接在一起，生成一个真正的可执行文件。而在另一些语言中，目标代码本身就可以直接执行。程序员可以把可执行目标代码复制到类似的系统上，然后就可以运行程序了。经过编译以后，程序就是一个独立的可执行文件，每种编程语言都需要使用自己的编译器转换利用这种语言编写的代码。例如，编程语言 C++需要使用 C++编译器，Java 语言需要 Java 编译器。

集成开发环境(Integrated Development Environment,IDE)是用于提供程序开发环境的应用程序,一般包括代码编辑器、编译器、调试器和图形用户界面等工具。集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套。所有具备这一特性的软件或者软件套(组)都可以叫集成开发环境,如微软的 Visual Studio 系列,Borland 的 C++ Builder、Delphi 系列,JetBrains 公司的 IntelliJ IDEA 等。当然也可以采用命令方式,手动完成预处理、编译、链接等步骤,例如,采用 g++编译器编译 C++源程序的过程如下。

(1) 预处理命令。

```
g++  -E file.cpp  -o file.i
```

(2) 编译命令。

```
g++  -S file.i -o file.s
```

(3) 汇编命令。

```
g++  -c file.s -o file.o
```

(4) 链接命令。

```
g++  file.o -o file.exe
```

**2. 解释器**

解释程序(interpreter)也可以把源代码转换成机器代码。不过,解释程序不是创建可执行的目标代码文件,而是在转换后执行每行代码,每次执行一行。由于解释程序动态地转换代码,所以具有编译器所缺乏的某种灵活性。但是,由于每次运行代码时都必须解释,而且在使用代码的地方必须使用解释程序的副本,所以解释代码的运行速度比编译代码慢。常用的解释语言包括 Python、JavaScript、Visual Basic。

## 5.4　编程技术


视频讲解

编程技术主要有结构化编程技术、面向对象编程技术、基于模型的编程技术、智能化软件开发技术等,结构化编程技术和面向对象编程技术仍然是编程主流。

### 5.4.1　结构化程序设计

结构化程序设计(structured programing)思想最早是 E. W. Dijikstra 于 1965 年提出的,结构化程序设计采用自顶向下、逐步求精的设计方法,各个模块通过“顺序、选择、循环”的控制结构进行连接,并且只有一个入口、一个出口。

结构化程序设计的原则可表示为:程序=算法+数据结构。

结构化编程技术的目的是创建易于阅读的代码,主要有以下 3 种典型的基本结构。

(1) 顺序结构:表示程序中的各操作是按照它们出现的先后顺序执行的。

(2) 选择结构:表示程序的处理步骤出现了分支,它需要根据某一特定的条件选择其中的一个分支执行。选择结构有单选择、双选择和多选择 3 种形式。

(3) 循环结构:表示程序反复执行某个或某些操作,直到某条件为假(或为真)时才可终止循环。在循环结构中最主要的是:什么情况下执行循环?哪些操作需要循环执行?循环结构的基本形式有两种:当型循环和直到型循环。

① **当型循环**:表示先判断条件,当满足给定的条件时执行循环体,并且在循环终端处

流程自动返回到循环入口；如果条件不满足，则退出循环体直接到达流程出口处。因为是“当条件满足时执行循环”，即先判断后执行，所以称为当型循环。

② **直到型循环**：表示从结构入口处直接执行循环体，在循环终端处判断条件，如果条件不满足，返回入口处继续执行循环体，直到条件为真时再退出循环到达流程出口处，是先执行后判断。因为是“直到条件为真时为止”，所以称为直到型循环。

结构化程序设计以模块功能和处理过程设计为主，主要有以下 4 个原则。

(1) 自顶向下：程序设计时，应先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标。不要一开始就过多追求细节，应先从最上层总目标开始设计，逐步使问题具体化。

(2) 逐步求精：对复杂问题，应设计一些子目标作为过渡，逐步细化。

(3) 模块化：一个复杂问题肯定是由若干稍简单的问题构成的。模块化是把程序要解决的总目标分解为子目标，再进一步分解为具体的小目标，把每个小目标称为一个模块。

(4) 限制使用 goto 语句。

**【例 5-11】** 输出前 50 个素数，每行输出 10 个数。

**【解 5-11】** 根据结构化程序设计思想，该问题可以分为以下几个子问题。

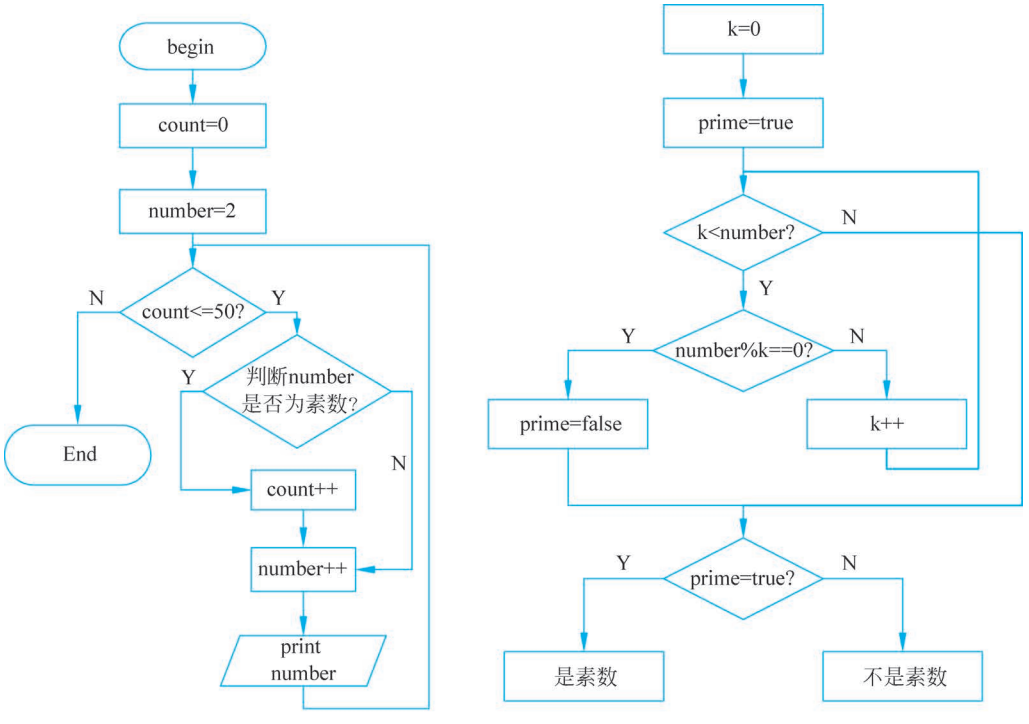
(1) 判断某一自然数是否为素数。

(2) 统计素数个数。

(3) 按照每行 10 个数输出素数。

(4) 重复以上过程，直到达到 50 个素数为止。

算法的流程图如图 5.10(a)所示，图 5.10(b)为判断素数的子模块的细化流程图。




(a) 输出前50个素数的流程图　　(b) 判断素数的细化的流程图

图 5.10　算法流程图

```
#include <iostream>
#include <iomanip>
using namespace std;
bool isPrime(int n)
{
    bool isPrime = true;
    for(i = 2; i <= n - 1; i++)
    {
        if(n % i == 0)
        {
            isPrime = false;
            break;
        }
    }
    if(!isPrime)
       return flase;
    else
        return true;
    return 0;
}
int main()
{
    int n = 2,count = 0;
    int i;
    while(count < 50)
    {
        if(isPrime(n))
        {
            count++;
            if(count % 10 == 0)
                cout << setw(4)<< n << endl;
            else
                cout << setw(4)<< n; }
        n++;
    }
    return 0; }
```

## 5.4.2　面向对象程序设计

20 世纪 80 年代，面向对象编程技术（Object-Oriented Programming，OOP）得以开发。OOP 的构件称为对象，是一种可重用的模块式组件，利用重用代码，可以快速、准确地构建程序。对象和类是面向对象编程技术中的基本概念，图书、计算机、灯、墙壁、植物、图片这些现实世界的实体，都可以用类描述。类包括属性和方法，属性描述实体的静态特征，方法描述实体的行为特征。例如，汽车类的属性，如颜色、尺寸、形状和最高时速等。汽车类的方法用于描述汽车的作用，即汽车的功能，如前进、后退、窗户打开等操作。把属性和功能结合起来以后，就可以定义一个对象。在 OOP 的语言中，每个对象都具有可以封装其他对象的属性和功能。如图 5.11 所示，类 Person 描述了实体人的属性和方法，属于基类，Employee 是派生类，从 Person 类继承，除拥有 name 和 age 属性之外，新增了属性 salary，其 C++描述如下所示。

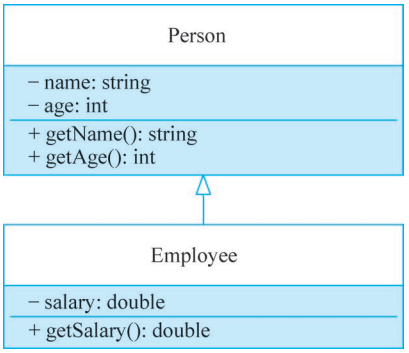



图 5.11 类图

```
class Person
{
    string name;
    int age;
public:
   Person(string name1,int age1)
  {
      name = name1;
      age = age1;
      cout << "Performs tasks for Person's constructor" << endl;
  }
   string getName()
  {
      return name;
  }
  int getAge()
  {
      return age;
  }
    ~Person()
    {
        cout << "Performs tasks for Person's destructor" << endl;
    }
};
class Employee: public Person
{
    double salary;
public:
  Employee(double s, string name1, int age1):Person( name1, age1)
  {
      salary = s;
      cout << "Performs tasks for Employee's constructor" << endl;
  }
   double getSalary()
   {
       return salary;
   }
    ~Employee()
    {
```

```
        cout << "Performs tasks for Employee's destructor" << endl;
    }
};
```

面向对象程序设计的三个基本特征是封装、继承和多态。

(1) 封装是指将某事物的属性和行为包装到对象中，这个对象只对外公布需要公开的属性和行为，而这个公布也可以有选择性地公布给其他对象。

(2) 继承是子对象可以继承父对象的属性和行为，亦即父对象拥有的属性和行为，其子对象也就拥有了这些属性和行为。

(3) 多态是指允许不同类的对象对同一消息做出响应。

面向对象的概念和应用已超越了程序设计和软件开发，扩展到如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD 技术、人工智能等领域。面向对象是一种对现实世界理解和抽象的方法，是计算机编程技术发展到一定阶段后的产物。面向对象方法把相关的数据和方法组织为一个整体来看待，从更高的层次来进行系统建模，更贴近事物的自然运行模式。

## 5.5　编程实例


**【例 5-12】** 秦九韶公式。我们都知道古希腊数学家海伦建立了利用三角形的三条边的边长直接求三角形面积的公式，其实我国南宋数学家秦九韶于 1247 年在他的《数书九章》里独立提出了三斜求积术，书中原文是这样记载的：

问沙田一段，有三斜，其小斜一十三里，中斜一十四里，大斜一十五里，里法三百步，欲知为田几何？

答曰："三百一十五顷"。

**【解 5-12】** 其解法为：以小斜幂，并大斜幂，减中斜幂，余半之，自乘于上；以小斜幂乘大斜幂，减上，余四约之，为实；一为从隅，开平方得积。

秦九韶把三角形的三条边分别称为小斜、中斜和大斜。"术"即方法。三斜求积术就是用小斜平方加上大斜平方，减中斜平方，取余数的一半的平方，而得一个数。小斜平方乘以大斜平方，减上面所得到的那个数。相减后余数被 4 除所得的数作为"实"，作 1 作为"隅"，开平方后即得面积。翻译过来公式如下：

$$S=\sqrt{\frac{1}{4}\left[c^2a^2-\left(\frac{c^2+a^2-b^2}{2}\right)^2\right]}$$

C++程序实现如下：

```
#include <iostream>
using namespace std
int main()
{
double a,b,c;
double s;
cin >> a >> b >> c;
```

```
s = sqrt(c * c * a * a - pow((c * c + a * a - b * b)/2,2)/4);
cout << s << endl;
return 0;
}
```

三斜求积术虽然与海伦公式在形式上有所不同,但它完全与其等价,它填补了中国数学史上的一个空白,从中可以看出中国古代已经具有很高的数学水平,其是我国数学史上的一颗明珠。

**【例 5-13】** 百钱买百鸡。中国古代五六世纪的《张丘建算经》中记载了一个很著名的数学问题,即"百鸡问题",原文如下:鸡翁一,值钱五,鸡母一,值钱三,鸡雏三,值钱一。百钱买百鸡,问鸡翁、鸡母、鸡雏各几何?这个问题的意思是说:一只公鸡,值五个钱;一只母鸡,值三个钱;三只小鸡,值一个钱。如果用一百个钱买一百只鸡,问公鸡、母鸡、小鸡各买多少只?

张丘建对此题给出了三组答案。第一组:公鸡四、母鸡十八,小鸡七十八;第二组:公鸡八、母鸡十一,小鸡八十一;第三组:公鸡十二、母鸡四、小鸡八十四。

**【解 5-13】** 本题目同样可以采用穷举法,一一列举公鸡、母鸡、小鸡可能的取值,并根据题意验证,具体程序如下:

```
#include<iostream>
using namespace std;
int main()
{
    int a = 0;                                    //表示公鸡数
    int b = 0;                                    //表示母鸡数
    int c = 0;                                    //表示小鸡数
    for(a = 0; a<=100/5; a++)
    {
        for(b = 0; b<=100/3; b++)
        {
            c = 100 - a - b;
            if(c % 3 == 0&&a * 5 + b * 3 + c/3 == 100)
            {
                cout <<"公鸡:"<< a <<" 母鸡:"<< b <<" 小鸡:"<< c << endl;
            }
        }
    }
    return 0;
}
```

这是世界上关于解不定方程组的最早表述,也是张丘建首创,开中国古代不定方程研究之先河。其影响一直持续到 19 世纪,比印度早 400 多年,比西方其他国家早 1000 多年,由此证明中国古代数学取得了辉煌的成就,我国一直都是世界上数学最为发达的国家。

**【例 5-14】** 杨辉三角形。杨辉所著的《详解九章算法》一书中,辑录了杨辉三角形,称为"开方作法本源"图,并说明此表引自 11 世纪中叶(约公元 1050 年)贾宪的《释锁算术》,并绘制了"古法七乘方图"。故此,杨辉三角又被称为"贾宪三角"。杨辉三角形是二项式系数在三角形中的一种几何排列,如图 5.12 所示。

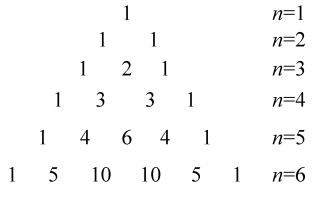



图 5.12 杨辉三角形

**【解 5-14】** 杨辉三角形问题的具体程序如下：

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    const int n=15;
    const int m =2*n-1;
    int arr[n + 1][m] = { 0 };
    for(int i=0; i<n; i++)
    {
        arr[i][n-i-1] = 1;
        arr[i][n+i-1] = 1;
    }
    for(int i=2; i<n; i++)
    {
        for(int j=n-i+1; j<n-2+i; j=j+2)
            arr[i][j] = arr[i-1][j-1] + arr[i-1][j+1];
    }
    int p;
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n-i-1; j++)
            cout<<"    ";
        p = 1;
        for(int j=n-i-1; p<i+2; j=j+2)
        {
            cout<<setw(4)<<arr[i][j]<<"    ";
            p = p + 1;
        }
        cout<<endl;
    }
    return 0; }
```

## 【思政 5-1】

杨辉三角形是中国古代数学的杰出研究成果之一，它把二项式系数图形化，把组合数内在的一些代数性质直观地从图形中体现出来，是一种离散型的数与形的结合。在欧洲，这个表叫作帕斯卡三角形。帕斯卡(1623—1662)是在 1654 年发现这一规律的，比杨辉要迟 393 年，比贾宪迟 600 年。