前言

我們長期從事資料結構、演算法的教學和程式設計競賽的訓練，教學和訓練的實際經驗讓我們產生對程式設計、資料結構和演算法的教學模式進行改革的想法。

1） 在課程中增加思維方式和解題策略的引導，引導學生思考各類資料結構、演算法的本質特徵是什麼，將「知識體系結構的掌握」與「思維方式的訓練」結合起來。

我們認為，評價一個人的專業能力要看以下兩個方面：①知識體系結構。他能用哪些知識解決問題，或者說，哪些是他所真正掌握並能應用而不僅僅是他學過的知識？②思維方式。在他面對問題，特別是不太標準化的問題時，要解決問題的策略是什麼？例如，面對的問題為什麼要採用這樣的資料結構表示，而不宜採用那樣的資料結構表示？當有多個資料結構可用時，怎樣權衡時間複雜度、空間複雜度、程式設計複雜度和思維複雜度四個因素，尋找最合適的資料結構...等等。

2） 在課程和訓練中全部採用程式設計競賽的試題作為案例進行教學。

傳統教學著重於理論教學和筆試，可能會使學生渾然不知程式設計語言、資料結構、演算法在現實生活中究竟有什麼用處，這也就失去了學習的意義。

陸遊詩云：「紙上得來終覺淺，絕知此事要躬行。」案例教學是透過模擬或者重現現實生活中的一些場景，讓學生把自己置入問題情境之中，通過思考、討論和上機程式設計來進行學習的方式：學生拿到試題後，先進行審題（What to do），然後考慮如何採用資料結構和演算法來解決問題（How to do），這無形中激發了學生的求知欲望，加深學生對知識的理解；在列出透過程式設計解決問題的方法後，學生還要經過程式設

計，將解決方法變為解決問題的程式，並進行除錯，在允許的時間和空間範圍內測試用例實作通過（Implementation）。這個「認識–實作–再認識–再實作」的過程，應視為知識理解上的一種提高，知識學習與應用能力間的一種轉變和昇華。

程式設計競賽是透過程式設計解決問題的比賽。從 20 世紀 80 年代末期到現在，各類大學生程式設計競賽、各種線上比賽及中學生資訊學奧林匹克競賽構成了浩如煙海的試題庫。這些來自全球各地且凝聚無數命題者的心血和智慧的試題，為相關知識創設了豐富有趣的問題背景，而且針對這些試題有許多線上測試平臺，學生可以透過這些平臺測試自己編寫的程式。因此，這些試題不僅可以用於程式設計競賽選手的訓練，而且可以用於程式設計語言、資料結構、演算法的教學和實作，能夠系統化、全面地提升學生透過程式設計解決問題的能力。

3）從程式設計的本質上說，程式設計是技術。

正因為程式設計是技術，所以，要磨煉學生的程式設計能力。首先是要求學生練習、練習、再練習；其次是要安排學生系統地練習。程式設計的知識體系可以概括為「演算法+資料結構=程式」這一項公式，也是電腦科學與技術的知識體系結構之核心。所謂系統地練習，就是在試題及其測試資料、解答程式及解題分析的引導之下，透過解題有系統地磨練學生應用演算法和資料結構各個知識點解決實際問題的能力，從而有效地掌握程式設計的知識體系。

基於上述想法，我們規劃出「大學程式設計課程與競賽訓練教材」系列，並於 2012 年出版該系列的第一本著作《資料結構程式設計實作：大學程式設計課程與競賽訓練教材》。隨後，在臺灣，由碁峰出版了該書的繁體版《提升程式設計的資料結構力：國際程式設計競賽之資料結構原理、題型、解題技巧與重點解析》；在美國，由 CRC Press 出版了該書的英文版《Data Structure Practice : for Collegiate Programming Contests and Education》，並在全球發行。目前，該書在世界各地廣受讀者的歡迎和好評。

在此基礎上，我們根據境內外的同學和同行在使用該書的過程中提出的意見及建議，以及電腦科學技術和程式設計競賽的發展，這幾年在阿曼、臺灣和香港、美國、馬來西亞、孟加拉等國家和地區的講學和訪學工作，對該書進行修訂增補和改進，最終出版了《提升程式設計的資料結構力：國際程式設計競賽之資料結構原理、題型、解題技巧與重點解析》第二版。

本書根據資料結構的知識體系結構，按照循序漸進的原則，分為四大篇（修練基本程式設計能力、線性資料結構的程式設計實作、樹的程式設計實作、圖的程式設計實作）共 15 章的內容。每一章在介紹了相關的資料結構知識後，會列出對應的實作範例，並在最後一節列出配合的相關題庫。

相對於本書的第 1 版，我們除了修正原有的小錯誤、筆誤及改進一些表述外，較大的改進如下：在第一篇的第 3 章中，由原來的「簡單遞迴的程式設計實作」改善為「遞迴與回溯的程式設計實作」。在第二篇的第 4 章中，將「在陣列中快速搜尋指定元素」和「透過陣列分塊技術優化演算法」的實作範例融合到原有的內容中；在第 5 章的佇列程式設計實作中，完整地列出三種佇列形式（順序佇列、優先佇列、雙端佇列）的實作範例。在第三篇的第 8 章中，加入「用四元樹求解二維空間問題」的實作範例；在第 10 章，在已有的二元排序樹和二元堆積的程式設計實作基礎上，列出兼具二元排序樹和二元堆積性質的樹堆之實作範例。在第四篇中，新增了第 15 章「應用狀態空間搜索程式設計」。

我們對浩如煙海的 ACM 程式設計競賽區預賽、總決賽、各大學程式設計競賽、線上程式設計競賽以及中學生資訊學奧林匹克競賽的試題進行分析和整理，從中精選出 227 道試題作為書中內容。其中 88 道試題作為實作範例，每道試題不僅有詳盡的解析，還提出帶有詳細注釋的參考程式；139 道作為題庫試題，所有試題都有清晰的提示，同時針對第 1 版讀者反映的題庫中某些「看了提示也很難寫出程式」的較難試題列出了帶有詳盡注解的解答示範程式（或者直接列出，或者作為電子版隨試題原版列出）。

書中每道題都註明了試題來源和線上測試網址，學生可以使用線上測試平臺測試自己編寫的程式。

本書的網站（www.hzbook.com）提供所有試題的英文原版及大部分試題的官方測試資料和解答程式。

本書既可以用於大學程式設計課程的教學和實作，又可以用於程式設計競賽的訓練。對於本書，我們的使用建議是：書中每章的實作範例可以用於程式設計語言、資料結構課程的教學、實作和上機作業，以及程式設計競賽選手掌握知識點的入門訓練；而在每章最後提供的「相關題庫」之試題則可以作為程式設計競賽選手的專項訓練試題，以及學生進一步提高程式設計能力的練習題。根據本書第 1 版的使用情況，本書也非常適合學生自學，在知識背景、試題、測試資料、解題分析或提示的支持之下，即使沒有老師、沒有同伴，同學們也能夠系統、全面地提高自己的程式設計能力。

本書是在復旦大學程式設計集訓隊長期活動的基礎上積累而成的。

Part 01
訓練基本的程式設計能力

程式設計是技術。正因為程式設計是技術，所以，程式設計、資料結構、演算法這樣的課程不是聽會的，也不是看會的，而是練會的，是讓學生在程式設計實作的過程中，逐步掌握透過程式設計解決實際問題的能力。資料結構的前導課程是程式語言，其教學目的是讓學生學會用程式語言編寫程式。因此，在學生進行資料結構程式設計實作之前，本篇先引領學生溫故知新，進行如下三個方面的程式設計實作：

1）簡單計算

2）簡單模擬

3）遞迴與回溯

這三類實作既是對程式設計語言課程的溫習，也是資料結構程式設計實作課程的入門和基礎。

Chapter 01
簡單計算的程式設計實作

所謂簡單計算題，指的是「輸入─處理─輸出」模式中，「處理」環節涉及的運算規則比較淺顯，程式設計的重心應放在如何正確地輸入、輸出或優化計算上。學生可以透過程式設計解簡單計算題的實作，掌握 C/C++或 Java 程式設計語言的基本語法，熟悉線上測試系統和程式設計環境，初步學會如何將一個自然語言描述的實際問題轉換成一個計算問題，列出計算過程，繼而程式設計實作計算過程，並將計算結果還原成對原來問題的解答。

雖然簡單計算題的運算相對簡單，但還是應該抱持著「舉輕若重」的科學態度。因為試題的輸 入輸出格式是多樣的，而計算精度和時效一般有嚴格的定義。「細節決定成敗」，一些程式設計細節若處理不好，則會導致整個程式「功虧一簣」。下面，我們將討論幾個相關問題：

- ◆ 改進程式書寫風格。
- ◆ 正確處理多組測試用例。
- ◆ 提高實數的計算精度。
- ◆ 用二分法提高計算效率。

一般來講，較複雜的問題大多由一些含簡單計算的子問題組合而成。「萬丈高樓平地起」，要提高程式設計能力，就需要從訓練簡單計算題的程式設計能力做起。

1.1 改進程式書寫風格的實作範例

如果一個程式具有良好的書寫風格，不僅在視覺上給人美感，而且也為程式的除錯和檢查帶來諸多方便。初看程式，往往可先從第一印象中判斷出程式設計者的思維清晰與否。但是，怎樣的程式書寫風格才算「好」呢？仁者見仁，智者見智。不過這並不意味著程式的書寫風格「無章可循」。

1.1.1 ▶ Financial Management

Larry 今年畢業，找到了工作，並賺了很多錢。但不知為何，Larry 總感覺錢不夠用。因此，Larry 想用財務報表來解決他的財務問題：他要計算自己能用多少錢。現在你可以透過 Larry 的銀行帳號看到他的財務狀況。請你幫 Larry 寫一個程式，根據他過去 12 個月的每月收入來計算，若要達到收支平衡，他每個月平均能用多少錢。

輸入

輸入 12 行，每一行是一個月的收入，收入的數字是正數，精確到分，不用美元的符號。

輸出

一個數字，它是這 12 個月收入的平均值。精確到分，前面加美元符號，後面加行結束符號。在輸出中沒有空格或其他字元。

樣例輸入：	樣例輸出：
100.00 489.12 12454.12 1234.10 823.05 109.20 5.27 1542.25 839.18 83.99 1295.01 1.75	$1581.42

試題來源：ACM Mid-Atlantic 2001

線上測試：POJ 1004，ZOJ 1048，UVA 2362

▼解題與分析

本題採用非常簡單的「輸入─處理─輸出」模式：

1） 先透過結構為 for(i=0; i<12; i++) 的迴圈輸入 12 個月的收入 a[0..11]。

2） 累計總收入sum = $\sum_{i=0}^{11} a[i]$；然後計算月均收入avg=$\frac{\text{sum}}{12}$。

3） 最後輸出 avg。

⬇參考程式

```
#include<iostream>                    //預編譯命令
using namespace std;                  //使用 C++標準程式庫中的所有識別字
int main()                            //主函數
{                                     //主函數開始
    double avg, sum=0.0, a[12]={0};   //定義雙精度實數變數 avg、sum 和實數陣列 a 的初始值
    int i;                            //宣告整型變數 i
    for(i=0; i<12; i++){                    //依次讀入每個月的收入，並累計年收入
          cin>>a[i];
          sum+=a[i];
    }
    avg=sum/12;                       //計算月平均收入
    printf("$%.2f",avg);              //輸出月平均收入
    return 0;
}
```

我們可從上述程式範例中得到 4 點啟示：

- 嚴格按照題意要求的格式輸入輸出。例如，題目要求輸入的月收入是精確到分的正數，因此程式中用提取運算子「 >> 」，將鍵盤輸入的月收入儲存到雙精度實數類型的陣列元素 a[i] 中；同樣地，程式中採用 printf("$%.2f ", avg) 語句使輸出的月均收入精確到分，且前有美元符號，後有行結束符號。程式執行於線上測試系統，關乎成敗的首要因素是程式的輸入輸出格式是否符合題意。如果沒有按照題目要求的格式輸入輸出，即便演算法正確，結果也是「Wrong Answer」。
- 同一結構程式段內的所有語句（包括說明語句）與本結構程式段的首行靠左對齊。
- 程式行按邏輯深度呈鋸齒狀排列。例如迴圈體縮進幾個字元，用內縮表示選擇結構等等。這種鋸齒型的編排格式能夠清晰地反映程式結構，提升易讀性。
- 在程式段前或開始位置加上描述程式段功能的注釋，對於變數及其變化也應該加上注釋。這樣做不僅是為了除錯工具和日後閱讀的方便，更重要的是培養一種團隊合作的精神。將來你進入職場後，可能加入一個研發團隊，大家一起合作開發程式、互相協助，這就更需要將注釋寫得清清楚楚，讓其他人看得明白。

1.2 正確處理多個測試用例的實作範例

【1.1.1 Financial Management】只列出了一個測試用例，該測試用例中資料的個數是已知的（12 個月的收入），且運算十分簡單（累加月收入，計算月平均值）。但在通常情況下，為了全面檢驗程式的正確性，大多試題都要求測試多個測試用例，只有通過所有測試用例才算程式正確。如果測試用例的個數或每個測試用例中資料的個數是預先確定的，則處理多個測試用例的迴圈結構比較簡單。但是，若測試用例的個數或每組測試用例中資料的個數未知，僅知道測試用例內資料的結束旗標和整個輸入的結束旗標，此時怎麼辦？在資料量較大、所有測試用例都採用同一運算且資料範圍已知的情況下，有無提高計算時效的辦法呢？在本節中，我們列出以下兩個實例。

1.2.1 ▶ Doubles

列出 2～15 個不同的正整數，計算在這些數裡面有多少資料對滿足一個數是另一個數的兩倍。例如列出

1 4 3 2 9 7 18 22

答案是 3，因為 2 是 1 的兩倍，4 是 2 的兩倍，18 是 9 的兩倍。

輸入

輸入包括多個測試用例。每個測試用例一行，列出 2～15 個兩兩不同且小於 100 的正整數。每一行最後一個數是 0，表示這一行的結束，這個數不屬於 2～15 個給定的正整數。輸入的最後一行僅列出整數 -1，該行表示測試用例的輸入結束，不用進行處理。

輸出

對每組輸入資料，輸出一行，列出有多少個數對滿足其中一個數是另一個數的兩倍。

樣例輸入：	樣例輸出：
1 4 3 2 9 7 18 22 0 2 4 8 10 0 7 5 11 13 1 3 0 -1	3 2 0

試題來源：ACM Mid-Central USA 2003

線上測試：POJ 1552，ZOJ 1760，UVA 2787

▼解題與分析

本題包含多組測試用例，因此需迴圈處理各組資料，迴圈的結束旗標是目前資料組的第一個數字是 -1。迴圈本體內做兩項工作：

- 透過一重迴圈讀入目前測試用例的陣列 a，並累計資料元素個數 n。目前測試用例的結束旗標是讀入資料 0。
- 透過兩重迴圈結構列舉 $a[]$ 的所有資料對 $a[i]$ 和 $a[j]$（$0 \le i < n\text{-}1$，$i\text{+}1 \le j < n$），判斷 $a[i]$ 和 $a[j]$ 是否呈兩倍關係（$a[i]*2==a[j] \,||\, a[j]*2==a[i]$）。

⬇參考程式

```
#include <iostream>                        //預編譯命令
using namespace std;                       //使用 C++標準程式庫中的所有識別字
int main()                                 //主函數
{                                          //主函數開始
    int i, j, n, count, a[20];             //宣告整型變數 i、j、n、count 和整型陣列 a
    cin>>a[0];                             //輸入第 1 組的第一個資料
    while(a[0]!=-1)                        //若輸入未結束，則輸入新一組資料
    {     n=1;                             //讀入目前資料組
          for( ; ; n++)
          {
                cin>>a[n];
                if (a[n]==0) break;
          }
          count=0;    //處理：計算目前資料組裡有多少資料對滿足一個數是另一個數的兩倍
          for (i=0; i<n-1; i++)            //列舉所有資料對
          {
                for (j=i+1; j<n; j++)
                {
                      //若目前資料對滿足 2 倍關係，則累計
                      if (a[i]*2==a[j] || a[j]*2==a[i])
                      count++;
                }
          }
          cout<<count<<endl;               //輸出目前測試用例中滿足兩倍關係的資料對數
          cin>>a[0];                       //輸入下一個測試用例的首個資料
    }
    return 0;
}
```

本題的測試用例數和測試用例長度都是未知的，求解程式採用雙重迴圈結構。

- 外迴圈：列舉各組測試用例，結束旗標為輸入結束符號（本題的輸入結束符號為-1）。
- 內迴圈：輸入和處理目前組的測試用例，輸入的結束旗標為資料組的結束符號（本題資料組的結束符號為 0）。

在處理多個測試用例的過程中，可能會遇到這樣一種情況：資料量較大，所有測試用例都採用同一運算，並且資料範圍已知。在這種情況下，為了提高計算時效，可以採用離線計算方法，即預先計算出指定範圍內的所有解，存入某個常數陣列；以後每測試一個測試用例，直接從常數陣列中引用相關資料就可以了，這樣做可避免重複運算。

1.2.2 ▶ Sum of Consecutive Prime Numbers

一些正整數能夠表示為一個或多個連續質數的和。列出一個正整數，會有多少個這樣的表示方式？例如，整數 53 有兩個表示，即 5+7+11+13+17 和 53；整數 41 有三個表示，即 2+3+5+7+11+13、11+13+17 和 41；整數 3 只有一個表示 3；整數 20 沒有這樣的表示。注意加法運算數必須是連續的質數，因此，對於整數 20、7+13 和 3+5+5+7 都不是有效的表示。

請寫一個程式，對於一個正整數列出其連續質數和的表示數。

輸入

輸入一個正整數序列，每個數一行，在 2～10000 之間取值。輸入結束以 0 表示。

輸出

輸出的每一行對應輸入的每一行，除了最後的 0。輸出的每一行，對於一個輸入的正整數，列出連續質數的和的表示數。輸出中沒有其他的字元。

樣例輸入：	樣例輸出：
2 3 17 41 20 666 12 53 0	1 1 2 3 0 0 1 2

試題來源：ACM Japan 2005

線上測試：POJ 2739、UVA 3399

▼解題與分析

由於每個測試用例都要計算質數，且質數上限為 10000，因此：首先，我們離線計算出 [2..10001] 內的所有質數，按照遞增順序存入陣列 prime[1.. total]。

然後，依次處理每個測試用例：

- 設目前測試用例的輸入為 *n*；連續質數的和為 cnt；cnt==*n* 的表示數為 ans。
- 採用雙重迴圈計算 *n* 的表示數 ans：
  - 外迴圈 *i*：列舉所有可能的最小質數 prime[i] (for (int *i*=0; *n*>=prime[*i*]; *i*++))；
  - 內迴圈 *j*：列舉由 prime[*i*]開始的連續質數的和 cnt，條件是所有質數在 prime[] 中且 cnt 不大於 *n* (for(int *j*=*i*; *j* < total && cnt<*n*; *j*++) cnt += prime[*j*])。內迴圈結束後，若 cnt==*n*，則 ans++。
  - 外迴圈結束後得出的 ans 即為問題解。

⬇參考程式

```
#include <iostream>                              //預編譯命令
using namespace std;                             //使用 C++ 標準程式庫中的所有識別字
const int maxp = 2000, n = 10000;                //設定質數表長和輸入值的上限
int prime[maxp], total = 0;                      //質數表和表長初始化為 0

bool isprime(int k)                              //判定 k 是否為質數
{
    for (int i = 0; i < total; i++)
          if (k % prime[i] == 0)
                return false;
    return true;
}

int main(void)                                   //主函數
{                                                //主函數開始
    for (int i = 2; i <= n; i++)                 //預先建立質數表
          if (isprime(i))
                prime[total++] = i;
          prime[total] = n + 1;

    int m;
    cin >> m;                                    //輸入第 1 個正整數
    while (m) {                                  //迴圈，直至輸入正整數 0 為止
          int ans = 0;                           //和初始化為 0
          for (int i = 0; m >= prime[i]; i++) {  //列舉最小質數
                int cnt = 0;                     //求連續質數的和
                for (int j = i; j < total && cnt < m; j++)
                      cnt += prime[j];
                if (cnt == m)                    //若和恰等於 m，則累計答案數
                      ++ans;
          }
          cout << ans << endl;                   //輸出答案數
          cin >> m;                              //輸入下一個正整數
    }
    return 0;
}
```

1.3 提高實數精度的實作範例

在有些情況下，試題的資料是實數且蘊含實數運算。例如，如何判斷兩個實數相等、對實數取整究竟是取大於該實數的最小整數還是取小於該實數的最大整數...等等。任何程式設計語言的實數精度都是有限的。若試題要求輸出實數值的誤差小於語言規定的精度範圍，怎麼辦呢？程式若不能處理好這些細節問題，即便計算步驟正確，也可能導致錯誤結果。我們不妨看一個實例。

1.3.1 ▶ I Think I Need a Houseboat

Fred Mapper 考慮在 Louisiana 購買一些土地，並在這些土地上建造他的家。在對土地的調查中他發現，由於 Mississippi 河的侵蝕，Louisiana 州的土地每年減少 50 平方英里（1 平方英里≈2.59 平方公里）。因為 Fred 準備在他所建的家中度過他的後半生，所以他要知道土地是否會因為河流的侵蝕而喪失。

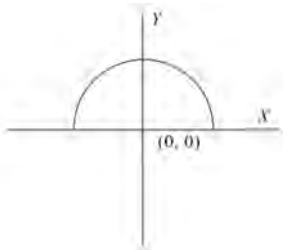

圖 1.3-1

在做了很多的研究後，Fred 發現正在失去的土地構成一個半圓形（半圓如圖 1.3-1 所示）。這一半圓形是一個圓的一部分，圓心在（0, 0），二等分這個圓的線是 *X* 軸，*X* 軸的下方是河水。在第 1 年開始的時候，這一半圓的面積為 0。

輸入

輸入的第一行是一個正整數，表示有多少個測試用例（*N*）。後面有 *N* 行，每行列出笛卡兒座標 *X* 和 *Y*，表示 Fred 考慮購買的土地的位置。這些數是浮點數，以英里為單位。*Y* 座標非負。不會列出（0, 0）。

輸出

對每個輸入的測試用例，輸出一行。這一行的形式為「Property *N*: This property will begin eroding in year *Z*.」，其中 *N* 是測試用例編號（從 1 開始記數），*Z* 表示他的土地在第 *Z* 年結束的時候要落到半圓形中（從 1 開始記數），*Z* 必須是一個整數。在最後一個測試用例後，輸出「END OF OUTPUT.」。

樣例輸入：	樣例輸出：
2 1.0 1.0 25.0 0.0	Property 1: This property will begin eroding in year 1. Property 2: This property will begin eroding in year 20. END OF OUTPUT.

說明：

1）購買的土地不會在半圓形邊界上：或者落在半圓形內，或者落在半圓形外。

2）這一問題被自動裁判，所以輸出要精確比對匹配，包括大小寫、標點符號和空格，以及到行末完整的那一個句子。

3）所有的地點以英里為單位。

試題來源：ACM Mid-Atlantic 2001

線上測試：POJ 1005、ZOJ 1049、UVA 2363

▼解題與分析

由於測試用例組數 n 預先確定，且每組測試用例僅為笛卡兒座標，因此可直接採用 for 迴圈處理所有測試用例組。第 i 個測試組（X_i, Y_i）與圓心（0, 0）構成的半圓面積即為土地被淹沒的範圍。由於每年減少 50 平方英里土地，而年份是整數，因此淹沒（X_i, Y_i）的年份應為大於 $\frac{\text{半圓面積}}{50}$ 的最小整數，這個取整數過程應使用向上取整數函數 ceil(x)。若使用向下取整數函數 floor(x)，則可能提前 1 年失去土地。

⬇參考程式

```
#include <stdio.h>                  //預編譯命令
#include <math.h>
#define M_PI 3.14159265
int num_props;                      //定義測試用例組數為整數
float x, y;                         //定義笛卡兒座標為單精確度實數
int i;
double calc;                        //定義「半圓面積/50」為雙精度實數
int years;                          //定義失去土地的年份為整數
void main(void)                     //主函數
{                                   //主函數開始
    scanf("%d", &num_props);        //輸入測試用例
    for (i = 1; i <= num_props; i++)
    {
          scanf("%f %f", &x, &y); //輸入第 i 個考慮購買的土地位置，
          //計算「半圓面積/50」(上取整後即為土地失去的年份)
          calc = (x*x + y*y)* M_PI / 2 / 50;
          years = ceil(calc);
          printf("Property %d: This property will begin eroding in year %d.\n",
                      i, years);  //輸出
    }
    printf("END OF OUTPUT.\n");
}
```

在實數運算中，經常需要判斷實數 x 和實數 y 是否相等，程式設計者往往把判斷條件簡單設成 y-x 是否等於 0。但這樣的做法可能會產生精度誤差。避免精度誤差的辦法是設一個精度常數 delta。若 y-x 的實數值與 0 之間的區間長度小於 delta，則認定 x 和 y 相等，如此一來就可將誤差控制在 delta 範圍內，如圖 1.3-2 所示。

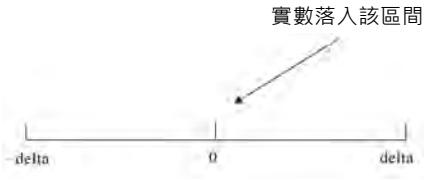

圖 1.3-2

顯然，判斷實數 x 和實數 y 是否相等的條件應設成 $|y\text{-}x| \leq$ delta。程式範例可查閱【1.4.1 Hangover】。

1.4 使用二分法提高計算時效的實作範例

在有些情況下，問題的所有資料為一個有序區間。二分法（或稱二分法）將這個區間等分成兩個子區間，根據計算要求決定下一步計算是在左子區間還是在右子區間進行；然後再根據計算要求等分所在區間，直至找到解為止。顯然，對一個規模為 O(n) 的問題，如果採用盲目列舉的辦法，則效率為 O(n)；若採用二分法，則計算效率可提高至 O($\log_2(n)$)。

許多演算法都採用了二分法。例如二分法搜尋、減半遞迴技術、快速排序、合併排序、最優二元樹、線段樹等。其中相對比較淺顯的演算法是二分法搜尋和減半遞迴技術，使用這兩種方法解簡單計算題，可以顯著提高計算時效。

二分法搜尋的基本思維：假設資料是按升冪排序的，對於待搜尋值 x，從序列的中間位置開始比較。

1） 若目前中間位置值等於 x，則搜尋成功。

2） 若 x 小於目前中間位置值，則在數列的左子區間（數列的前半段）中搜尋。

3） 若 x 大於目前中間位置值，則在右子區間（數列的後半段）中繼續搜尋。

以此類推，直至找到 x 在序列中的位置（搜尋成功）或子區間不存在（搜尋失敗）為止。若搜尋失敗，則目前子區間右指標所指的元素是序列中大於 x 的最小數。

1.4.1 ▶ Hangover

你能使一疊在桌子上的卡片向桌子外伸出多遠？如果是一張卡片，這張卡片向桌子外伸出卡片的一半長度（卡片以直角伸出桌子）。如果有兩張卡片，就讓上面一張卡片向外伸出下面那張卡片的一半長度，而下面的那張卡片向桌子外伸出卡片的三分之一長度，所以兩張卡片向桌子外延伸的總長度是 1/2 + 1/3 = 5/6 卡片長度。以此類推，n 張卡片向桌子外延伸的總長度是 1/2 + 1/3 + 1/4 + ... + 1/(n + 1) 卡片長度：最上面的卡片向外延伸 1/2，第二張卡片向外延伸 1/3 ，第三張卡片向外延伸 1/4..... 最下面一張卡片向桌子外延伸 1/(n + 1)，如圖 1.4-1 所示。

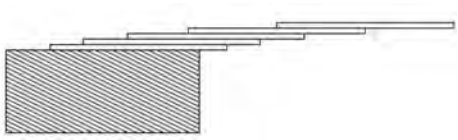

圖 1.4-1

輸入

輸入由一個或多個測試用例組成，最後一行用 0.00 表示輸入結束，每個測試用例佔據一行， 是一個 3 位數的正浮點數 c，最小值 0.01，最大值 5.20。

輸出

對每個測試用例 c，輸出要伸出卡片長度 c 所最少要用的卡片的數目，輸出形式見樣例輸出。

樣例輸入：	樣例輸出：
1.00 3.71 0.04 5.19 0.00	3 card(s) 61 card(s) 1 card(s) 273 card(s)

試題來源：ACM Mid-Central USA 2001

線上測試：POJ 1003，UVA 2294

▼解題與分析

由於資料範圍很小，因此先離線計算向桌子外延伸的卡片長度不超過 5.20 所需的最少卡片數。設 total 為卡片數，len[i] 為前 i 張卡片向桌子外延伸的長度，即 $\text{len}[i] = \text{len}[i\text{-}1] + \frac{1}{i+1}$，i ≥ 1，len[0]=0。顯然 len[] 為遞增序列。

請留意，由於 len 的元素和被搜尋的要伸出卡片長度 x 為實數，因此要嚴格控制精度誤差。設精度 delta=1e-8。zero(x) 是實數 x 為正負數和 0 的旗標。

$$\text{zero}(x)=\begin{cases} 1 & x > \text{delta}，\text{旗標 } x \text{ 是正實數} \\ -1 & x < \text{-delta}，\text{旗標 } x \text{ 是負實數} \\ 0 & \text{是否旗標 } x \text{ 是 } 0 \end{cases}$$

初始時 len[0]=0，透過結構為

```
for(total=1;zero(len[total-1]-5.20)<0;total++)
          len[total]=len[total-1]+1.0/double(total+1)
```

的迴圈，遞迴計算 len 序列。

在計算出 len 陣列後，先輸入第 1 個測試用例 x，並進入結構為 while (zero(x)) 的迴圈，對於每一次迴圈，使用二分法在 len 表中搜尋伸出卡片的長度 x 所最少要用的卡片數，並輸入下一個測試用例 x。這個迴圈過程直至輸入測試用例 x=0 .00 為止。

二元搜尋的過程如下：

初始時區間 $[l, r] = [l, \text{total}]$，區間的中間指標是$\min=\left\lceil\frac{l+r}{2}\right\rceil$。若 zero(len[mid] - x) < 0，則所需的卡片數在右區間（l = mid）；否則所需的卡片數在左區間（r = mid）。繼續二分區間 $[l, r]$，直至 $l+1 \geq r$ 為止。此時得出的 r 即為最少要用的卡片數。

⬇參考程式

```
#include <iostream>                  //預編譯命令
using namespace std;                 //使用 C++標準程式庫中的所有識別字
const int maxn = 300;                //len 陣列容量
const double delta = 1e-8;           //設定精度

int zero(double x)                   //在精度 delta 的範圍內，若 x 是小於 0 的負實數，則返回-1；
                                     //若 x 是大於 0 的正實數，則返回 1；若 x 為 0，則返回 0
{
    if (x < -delta)
          return -1;
    return x > delta;
}

int main(void)                                  //主函數
{                                               //主函數開始
    double len[maxn];                           //定義 len 陣列和陣列長度
    int total;
    len[0] = 0.0;                               //直接計算出截止長度不超過 5.20 所需的最少卡片數
    for (total = 1; zero(len[total - 1] - 5.20) < 0; total++)
          len[total] = len[total - 1] + 1.0 / double(total + 1);
    double x;
    cin >> x;                                   //輸入第 1 個測試用例 x
    while (zero(x)) {                           //用二分法在 len 表中搜尋不小於 x 的最少卡片數
          int l, r;
          l = 0;                                //設定搜尋區間的左右指標
          r = total;
          while (l + 1 < r) {                   //迴圈條件是尋找區間存在
                int mid = (l + r) / 2;          //計算搜尋區間的中間指標
                //若中間元素值小於 x，則在右區間搜尋；否則在左區間搜尋
                if (zero(len[mid] - x) < 0)
                      l = mid;
                else
                      r = mid;
          }
          cout << r << " card(s)" << endl;      //輸出至少伸出 x 長度最少要用的卡片數
          cin >> x;                             //讀下一個測試用例
    }
    return 0;
}
```

二分法不僅可用於資料搜尋，亦可用於函數計算。例如有變數 x_1、x_2、x_3，已知函數值是 x_1，x_2、x_3 是函數的引數，即函數關係式 $x_1=f(x_2, x_3)$。如何在函數值 x_1 和引數值 x_2 確定的情況下，求滿足函數的 x_3 值。下面介紹一種演算法—減半遞迴技術。

所謂「減半」是指將問題的規模（例如 x_3 的取值範圍）減半，而問題的性質（例如 $x_1 = f(x_2, x_3)$）不變。假設原問題的規模為 n，則可以採用與問題有關的特定方法將原問題化為 c（c 是與規模無關而只與問題有關的常數）個規模減半的問題，然後透過研究規模為 $n/2$ 的問題（顯然與原問題性質相同，只是規模不同而已）來解決問題。問題的規模減少了，會對解決問題帶來不少方便。但是，在規模減半過程中，勢必增加某些輔助工作，在分析工作量時必須予以考慮。所謂「遞迴」是指重複上述「減半」過程。因為規模為 $n/2$ 的問題又可以化為 c 個規模為 $n/4$ 的相同性質問題。依此類推，直至問題的規模減至最小，很方便地解決為止。

1.4.2 ▶ Humidex

濕熱指數（humidex）是加拿大氣象學家用來反映溫度和濕度的綜合影響的度量衡。它不同於美國採用露點（dew point）來表示酷熱指數而不是相對濕度（摘自 Wiki 百科）。

當溫度為 30˚C（86˚F）和露點為 15˚C（59˚F）時，濕熱指數是 34（注意濕熱指數是一個沒有度量單位的數，這個值表示大約的攝氏溫度值）。如果溫度保持在 30˚C 並且露點上升到 25˚C（77˚F），濕熱指數就上升到 42.3。

在相同的溫度和相對濕度情況下，濕熱指數往往比美國酷熱指數高。

目前用來確定濕熱指數的公式是在 1979 年由加拿大大氣環境服務局的 J. M. Masterton 和 F.A. Richardson 提出的。

根據加拿大氣象局的觀點，濕熱指數達到 40 會使人「非常不舒服」; 在 45 以上則是「危險」; 當濕熱指數達到 54，人就會馬上中暑。

在加拿大，濕熱指數的最高紀錄出現在 1953 年 6 月 20 日的加拿大安大略省溫莎市，當時達到 52.1（溫莎市的居民並不知道，因為當時濕熱指數尚未發明）。濕熱指數於 1995 年 7 月 14 日在溫莎和多倫多兩地達到 50。

濕熱指數的公式如下：

```
濕熱指數 = 溫度 + h
    h = (0.5555)×(e - 10.0)
    e = 6.11×exp [5417.7530×((1/273.16) - ( 1 / (露點 + 273.16)))]
```

其中，exp(x) 是以 2.718281828 為底，x 為指數。

由於濕熱指數只是一個數字，電臺播音員就像宣佈氣溫一樣宣佈它，例如：那裡氣溫 47 度，濕熱指數.....有時天氣預報列出溫度和露點，或者溫度和濕熱指數，但它們很少同時報告這三個測量值。請寫一個程式，列出其中兩個測量值，並計算第 3 個值。

本題設定，對於所有的輸入，溫度、露點和濕熱指數的範圍在 -100 ~ 100℃之間。

輸入

輸入包含了許多行，除了最後一行，每行由空格分開的 4 個項目組成：一個字母、一個數字、第二個字母、第二個數字。每個字母說明後面跟著的數字之含義：T 表示溫度，D 表示露點，H 表示濕熱指數。輸入結束行只有一個字母 E。

輸出

對於每行輸入，除了最後一行，產生一行輸出。輸出的形式如下。
T 數字 D 數字 H 數字
其中的 3 個數字列出溫度、露點以及濕熱指數。每個數字是十進位數字，精確到小數點後一位，所有溫度以攝氏表示。

樣例輸入：	樣例輸出：
T 30 D 15 T 30.0 D 25.0 E	T 30.0 D 15.0 H 34.0 T 30.0 D 25.0 H 42.3

試題來源：Waterloo Local Contest, 2007.7.14

線上測試：POJ 3299

▼解題與分析

由題目列出的濕熱指數公式可以看出，*h* 與露點呈正比關係。顯然，已知露點和溫度（或濕熱指數），則先由露點推出 *h* 值，再由「濕熱指數=溫度+*h*」公式得出濕熱指數。或者，由公式「溫度=濕熱指數-*h*」得出溫度。

若已知的兩個測試量是溫度和濕熱指數，則採取減半遞推技術計算露點值。

假設最初露點值為 0，然後進入迴圈。露點的增量值從 100 開始，每次迴圈減半。若根據公式得出的濕熱指數大於預報的濕熱指數，則露點值減少一個增量（即 *h*↘，使得公式得出的濕熱指數下逼預報值），否則露點值增加一個增量（即 *h*↗，使得公式得出的濕熱指數上逼預報值）。這個迴圈過程直至增量值 ≤ 0.0001 為止，此時得出應預報的露點值。

⬇參考程式

```
#include <stdio.h>                              //預編譯命令
#include <math.h>
#include <assert.h>
char a,b;                                       //定義兩個測試旗標字元
double A,B,temp,hum,dew;
double dohum(double tt,double dd){              //根據溫度 tt 和露點 dd 計算濕熱指數
```

```
    double e = 6.11 * exp (5417.7530 * ((1/273.16) - (1/(dd+273.16))));
    double h = (0.5555)*(e - 10.0);
    return tt + h;                              //返回濕熱指數
}

double dotemp(){                                //根據露點 dew 和濕熱指數 hum 計算溫度
    double e = 6.11 * exp (5417.7530 * ((1/273.16) - (1/(dew+273.16))));
    double h = (0.5555)*(e - 10.0);
    return hum - h;                             //返回溫度
}

double dodew(){                                 //根據溫度 temp 和濕熱指數 hum 計算露點
    double x = 0;                               //露點值及其增量值初始化
    double delta=100;
//迴圈：增量值從 100 開始，每次迴圈減少一半，若根據目前溫度 temp 和露點 x 得出的濕熱指數大於 hum，
//則露點 x 減少一個增量值，否則露點 x 增加一個增量值，這個迴圈過程直至增量值≤0.0001 為止
    for (delta=100;delta>.00001;delta *=.5) {
          if (dohum(temp,x)>hum) x -= delta;
          else x += delta;
    }
    return x;                                   //返回露點 x
}

main()          //主函數
{               //迴圈：依次輸入每次天氣預報的兩個測試量，直至結束旗標'E'

    while (4 == scanf(" %c %lf %c %lf", &a, &A, &b, &B) && a != 'E')
    {
          temp = hum = dew = -99999;     //溫度、濕熱指數和露點值初始化
          if (a == 'T') temp = A;        //預報的第 1 個測量值是溫度
          if (a == 'H') hum = A;         //預報的第 1 個測量值是濕熱指數
          if (a == 'D') dew = A;         //預報的第 1 個測量值是露點
          if (b == 'T') temp = B;        //預報的第 2 個測量值是溫度
          if (b == 'H') hum = B;         //預報的第 2 個測量值是濕熱指數
          if (b == 'D') dew = B;         //預報的第 2 個測量值是露點
          if (hum == -99999) hum = dohum(temp,dew);  //若缺失濕熱指數，則根據溫度和露點計算
          if (dew == -99999) dew=dodew();            //若缺失露點，則根據溫度和濕熱指數計算
          if (temp == -99999) temp = dotemp();       //若缺失溫度，則根據濕熱指數和露點計算

          printf("T %0.1lf D %0.1lf H %0.1lf\n", temp, dew, hum); //輸出溫度、露點和濕熱指數
    }
    assert(a == 'E');                   //遇到結束旗標'E'時使用 assert
}
```

1.5 相關題庫

1.5.1 ▶ Sum

請求出 1 到 n 之間所有整數的總和。

輸入

輸入為一個絕對值不大於 10000 的整數 n。

輸出

輸出一個整數，是所有在 1 到 n 之間整數的總和。

樣例輸入：	樣例輸出：
-3	-5

試題來源：ACM 2000 Northeastern European Regional Programming Contest (test tour)

線上測試：Ural 1068

提示

根據等差數列 $s = 1 + 2 + \cdots + n$ 的求和公式，如果 n 是大於 0 的正整數，則 $s=\frac{1+n}{2}\times n$；否則 $s=\frac{1-n}{2}\times n+1$。

1.5.2 ▶ Specialized Four-Digit Numbers

找到並列出所有具有這樣特性的十進位的 4 位數字：其 4 位數字的和等於這個數字以十六進位表示時的 4 位數字之和，也等於這個數字以十二進位表示時的 4 位數字之和。

例如，整數 2991（十進位）的 4 位數字之和是 2+9+9+1 = 21，因為 2991 = 1×1728 + 8×144 + 9×12 + 3，所以其十二進位表示為 1893_{12}，4 位數字之和也是 21。但是 2991 的十六進位表示為 BAF_{16}，並且 11+10+15 = 36，因此 2991 被程式排除了。

下一個數是 2992，3 種表示方式的各位數字之和都是 22（包括 $BB0_{16}$），因此 2992 要被列在輸出中。（本題不考慮少於 4 位數字的十進位數字—排除了前置字元為零—因此 2992 是第一個正確答案。）

輸入

本題沒有輸入。

輸出

輸出為 2992 和所有比 2992 大的滿足需求之 4 位數字（以嚴格的遞增序列），每個數字一行，數字前後不加空格，以行結束符號來結束。輸出沒有空行。輸出的前幾行如下所示。

樣例輸入：	樣例輸出：
本題無輸入	2992 2993 2994 2995 2996 2997 2998 2999 ...

試題來源：ACM Pacific Northwest 2004

線上測試：POJ 2196，ZOJ 2405，UVA 3199

提示

首先設計一個函數 calc(*k*, *b*)，計算和返回 *k* 轉換成 *b* 進位後的各位數字之和。然後列舉 [2992..9999] 內的每個數 *i*：若 calc(*i*, 10) == calc(*i*, 12)== calc(*i*, 16)，則輸出 *i*。

1.5.3 ▶ Quicksum

校驗是一個掃描資料封包並返回一個數字的演算法。校驗的思維是，如果資料封包發生變化了，校驗值也將隨著發生變化，所以校驗經常被用於檢測傳輸錯誤、驗證檔的內容，而且在許多其他情況下，用於檢測資料的不良變化。

本題請你實作一個名為 Quicksum 的校驗演算法。Quicksum 的資料封包只包含大寫字母和空格，以大寫字母開始和結束，空格和字母可以任何組合形式出現，可有連續的空格。

Quicksum 計算在資料封包中每個字元的位置與字元對應值的乘積之總和。空格的對應值為 0，字母的對應值是它們在字母表中的位置。A=1、B=2，以此類推 Z=26。例如 Quicksum 計算資料封包「ACM」和「MID CENTRAL」如下：

```
ACM: 1×1 + 2×3 + 3×13 = 46
MID CENTRAL: 1×13 + 2×9 + 3×4 + 4×0 + 5×3 + 6×5 + 7×14 + 8×20 + 9×18 + 10×1 + 11×12 = 650
```

輸入

輸入由一個或多個測試用例（資料封包）組成，輸入最後列出僅包含「#」的一行表示輸入結束。每個測試用例一行，開始和結束沒有空格，包含了 1～255 個字元。

輸出

對每個測試用例（資料封包），在一行中輸出其 Quicksum 的值。

樣例輸入：	樣例輸出：
ACM MID CENTRAL REGIONAL PROGRAMMING CONTEST ACN A C M ABC BBC #	46 650 4690 49 75 14 15

試題來源：ACM Mid-Central USA 2006

線上測試：POJ 3094、ZOJ 2812、UVA 3594

提示

設計一個函數 value(*c*)：若字元 *c*== '_'，則返回 0，否則返回字元 *c* 的對應值 *c*-'A'+1。

整個計算過程為一個迴圈，每次迴圈輸入目前測試用例並計算和輸出其 Quicksum 值：

字元位置和 Quicksum 值初始化為 0，目前測試用例所對應的字串 *s* 設空，然後反復讀入字元 *c*，並將 *c* 送入 *s*，直至 *c* 為檔案結束旗標（EOF）或行結束符號（'\n'）為止。字串 *s* 的 Quicksum 值可一邊輸入一邊計算，即 Quicksum=$\sum_{i=0}^{s.size-1}(i+1)\times value(s[i])$。

若 *s* 為輸入結束符號「#」，則退出程式。

1.5.4 ▶ A Contesting Decision

對程式設計競賽進行裁判是一項艱苦的工作，要面對要求嚴格的參賽選手、要做出乏味的決定，以及要進行著單調的工作。不過，這其中也可以有很多的樂趣。

對於程式設計競賽的裁判來說，用軟體使得評測過程自動化對裁判有很大的幫助，而一些比賽軟體存在的不可靠性也使得人們希望比賽軟體能夠更好、更可用。你現在是競賽管理軟體發展團隊中的一員，基於模組化設計原則，你所開發模組的功能是為參加程式設計競賽的隊伍計算分數並確定冠軍；列出參賽隊伍在比賽中的情況，確定比賽的冠軍。

記分規則：一支參賽隊的記分由兩個部分組成。第一部分是被解出的題數，第二部分是加罰時間，表示解題的總耗費時間和試題沒有被解出前錯誤的提交所另加的加罰時間。對於每個被正確解出的試題，加罰時間等於該問題被解出的時間加上每次錯誤提交的 20 分鐘加罰時間。在問題沒有被解出前不加罰時間。

因此，如果一支隊伍在比賽 20 分鐘的時候第二次提交解出的第 1 題，他們的加罰時間是 40 分鐘。如果他們提交第 2 題 3 次，但沒有解決這個問題，則沒有加罰時間。如果他們在 120 分鐘提交第 3 題並一次解出，則該題的加罰時間是 120 分。如此一來，該隊的成績是加罰時間 160 分，解出了兩道試題。

冠軍隊是解出最多試題的隊伍。如果兩隊在解題數上打成平手，那麼加罰時間少的隊伍是冠軍隊。

輸入

程式評判的程式設計競賽有 4 題。本題設定，在計算加罰時間後，不會導致隊與隊之間不分勝負的情況。
第 1 行為參賽隊數 *n*；
第 2 ~ *n*+1 行為每個隊的參賽情況。每行的格式為：

< Name > < *p*1Sub > < *p*1Time > < *p*2Sub > < *p*2Time > … < *p*4Time >

第一個元素是不含空格的隊名。後面是對於 4 道試題的解題情況（該隊對這一試題的提交次數和正確解出該題的時間（都是整數））。如果沒有解出該題，則解題時間為 0。如果一道試題被解出，提交次數至少是一次。

輸出

輸出一行。列出優勝隊的隊名、解出題目的數量及加罰時間。

樣例輸入：	樣例輸出：
4 Stars 2 20 5 0 4 190 3 220 Rockets 5 180 1 0 2 0 3 100 Penguins 1 15 3 120 1 300 4 0 Marsupials 9 0 3 100 2 220 3 80	Penguins 3 475

試題來源：ACM Mid-Atlantic 2003

線上測試：POJ 1581，ZOJ 1764，UVA 2832

提示

假設冠軍隊的隊名為 wname，解題數為 wsol，加罰時間為 wpt；目前隊的隊名為 name，解題數為 sol，加罰時間為 pt；目前題的提交次數為 sub，解題時間為 time。

我們依次讀入每個隊伍的隊名 name 和 4 道題的提交次數 sub，以及解題時間 time。

若該題成功解出（time>0），則累計該隊的解題數（++sol），統計加罰時間 pt（pt += (sub - 1) *20 + time）。

計算完 4 道題的解題情況後，若目前隊解題數最多，或雖同為目前最高解題數但罰時最少（sol > wsol || (sol == wsol && wpt > pt)），則將目前隊暫設為冠軍隊，記下隊名、解題數和加罰時間（wname = name; wsol = sol; wpt = pt）。

顯然，處理完 *n* 個參賽隊的資訊後，wname、wsol 和 wpt 就是問題的解。