The First Annual Community College Programming Contest: The Problems Used and the Problems Not Used

Sylvester Tuohy
Bernice Houle
Seidenberg Inaugurates Yet Another "First"
Area Community Colleges Participate in Programming Contest

by Bernice Houle

Fourteen students comprising five teams from three area community colleges -- Orange, Rockland, and Westchester -- tested their programming skills and problem-solving abilities at the First Annual Community College Programming Contest held on the Pleasantville campus on Saturday, April 24, 2010. The teams had one and a half hours to solve eight problems contributed by faculty members from the Westchester Computer Science Department. An Orange County Community College team placed 1st followed by two teams from Rockland placing 2nd and 3rd.

The event was organized by Bernice Houle, Ph.D., Associate Dean of the Seidenberg School, and Sylvester Tuohy, Ph.D., Professor of Computer Science. It was supported by funding from a Verizon Foundation Thinkfinity Grant.

from the Spring 2010 eCommuniqué

Special recognition and thanks are due Thomas Reivik, Seidenberg Network and Systems Analyst, who set-up a special local area network, server, and system for accepting programs from the student teams, notifying judges as submissions came-in and making them available, accepting judges' scores and comments, displaying these to teams, and generally keeping track of things so that all participants could do likewise.
The First Annual Community College Programming Contest: The Problems Used and the Problems Not Used

Problems for the contest were contributed by CS faculty members and selected for use by Sylvester Tuohy. At the contest, under the direction of Professor Tuohy, submitted programs were judged by Prof. Lixin Tao, Prof. Mary Courtney, Prof. Mehdi Badii, Prof. Narayan Murthy, and Allen Stix.

This report archives the problems used for the contest as well as the problems considered but not used. Professor Tuohy requested that problem submitters supply sample solutions along with their problems. This was so that in addition to a subjective impression of the solution's complexity, he had a rough view of the programming constructs involved and the lines of code. Where available, the code is included.

First, the eight problems that were selected. They are identified as problems A through H.

Community College Programming Contest
(CCPC)
Sponsored by
The Seidenberg School of
Computer Science & Information Systems at
Pace University (Pleasantville Campus)

Saturday, April 24, 2010
Solve Problems:
A, B, \ldots H
In the shortest time
Problem A

Given the following array write the code to find a 2 X 2 pattern. The program should prompt for the two line input.

BHTCOMMUNITY
COLLEGEYWMB
PROGRAMMING
BCONTESTXZST
KVGA@CE@HDQ
VBERGENQDYAS
PDUurchesszlt
NMWFairfield
JMORANGEKLTR
ZMCPSAAICXY
ROCKLANDPAUV
WESTCHESTERY

For example

Input:

XY

UV // on two separate lines.

Output:

Row = 10   Column = 11

//written to the screen will be the upper left corner of the found pattern:

// starting with the first row and column equal to 1.

or the words ‘Not Found’.
Problem B

Character Frequencies

Input: The program accepts a string entered from the keyboard.

Output: The program will display the number of times each different character appears in the entered string.

Example:

Input: Welcome to the Pace University Programming Contest!

Output:

w = 1
e = 6
l = 1
c = 3
o = 4
m = 3
    = 6 ← This is the space
t = 5
h = 1
p = 2
a = 2
u = 1
n = 3
i = 3
v = 1
r = 3
s = 2
y = 1
g = 2
! = 1

Notice that the character tabulations are case insensitive. The 'c' in Welcome and the 'c' in Contest are treated as the same character.

Notice also that spaces and punctuation (the exclamation point) are counted.
import java.io.InputStreamReader;
import java.io.BufferedReader;

class CharacterFrequencies  //Problem B
{
    public static void main(String[] args) throws Exception
    {
        String inputString = getInputStream();  //get input from keyboard
        String scratchStr = inputString.toLowerCase(); //for case-insensitivity
        while(scratchStr.length() > 0)  //while chars remain to be counted
        {
            char ch = scratchStr.charAt(0);
            System.out.print(ch + "=");

            int frequency = 0; //

            int i = 0; //index of the current character (left end of scratch string)
            //locate, count, and remove copies of the char from the left-end
            while (i < scratchStr.length())
            {
                if(scratchStr.charAt(i)==ch)
                {
                    frequency = frequency + 1;

                    if(i==0)  //if i is at the left-end
                    {
                        scratchStr = scratchStr.substring(1, scratchStr.length());
                    }
                    else
                    {
                        if(i == scratchStr.length()-1)  //if i is at the right-end
                        {
                            scratchStr = scratchStr.substring(0, scratchStr.length()-1);
                        }
                        else  //if i is in the interior
                        {
                            scratchStr = scratchStr.substring(0, i) +
                                        scratchStr.substring(i+1, scratchStr.length());
                        }
                    }
                    //string has been contracted, don't move ahead
                }
                else
                {
                    i = i + 1;
                }
            }
            System.out.println(frequency);
        }
    }

    static String getInputStream() throws Exception
    {
        System.out.println();
        System.out.print("ENTER INPUT: ");
        InputStreamReader source = new InputStreamReader(System.in);
        BufferedReader keyboard = new BufferedReader(source);
        String input = keyboard.readLine();
        return input;
    }
}
Problem C

Computing \( \pi \) (3.14159...)

\( \pi \) can be expressed as the sum of the following convergent infinite series:

\[
\pi = 4 \times (1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 - 1/15 \ldots)
\]

Write a program that outputs the value of \( \pi \) based on

exactly \( N \) terms of this series,

Example:

INPUT:

1000 // Your Program should ask for a single input value

//Which can be 1<=N<=10,000

OUTPUT:

\( \pi = 3.140592653839794 \)
import java.util.Scanner;

class pi_calculation //Problem C
{
    public static void main(String[] args)
    {
        System.out.print("number of terms (1..10000): ");
        Scanner keyboard = new Scanner(System.in);
        int numberOfTerms = keyboard.nextInt();

        if (numberOfTerms >= 1 && numberOfTerms <= 10000)
        {
            double pi = calculatePi(numberOfTerms);
            System.out.print("pi based on " + numberOfTerms + " terms = ");
            System.out.println(pi);
        }
    }

    static double calculatePi(int numOfTerms)
    {
        int termNumber = 1;
        double sum = 0;

        boolean oddTerm = true;

        while (termNumber <= numOfTerms)
        {
            int denominator = (2*termNumber) - 1;

            if (oddTerm)
                sum = sum + 1.0/denominator;
            else
                sum = sum - 1.0/denominator;

            termNumber = termNumber + 1; //termNumber for next iteration

            if (oddTerm) oddTerm = false;
            else oddTerm = true;
        }

        return sum * 4; //this is pi
    }
}
Encryption with a Transposition Cipher

A transposition cipher changes the order of the plaintext characters, but not the characters themselves.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>

For instance if this were the entered plaintext:

and this were transposition instructions:

this would be the ciphertext:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>f</td>
<td>i</td>
<td>c</td>
<td>g</td>
<td>e</td>
<td>b</td>
<td>j</td>
<td>a</td>
<td>h</td>
</tr>
</tbody>
</table>

The program works with a block of plaintext ten characters long.

The transposition instructions consist of the ten numerals, 0..9. Each number specifies the "destination subscript" for the corresponding plaintext character.

In this example directly above, 'a' is the plaintext character in "slot 0."

8 is the instruction numeral in "slot 0"

This means that the ciphertext destination for 'a' is "slot 8."

Each instruction numeral is used once, and only once because two different plaintext characters cannot both go into the same ciphertext "slot."

Input: The program must query the user for the transposition instructions.

The program must query the user for a ten-character block of plaintext.

Output: The program must output the encrypted plaintext (that is, the ciphertext)

Sample run:

ENTER PERMUTATION ACTION  >  9876543210
ENTER PLAINTEXT (10 CHAR)  >  university

Plaintext: university
Ciphertext: ytisrevinu

This will spell the plaintext backwards.
import java.io.InputStreamReader;
import java.io.BufferedReader;

class TranspositionCipher // Problem D
{
    public static void main(String[] args) throws Exception
    {
        // String permutationAction = "8630514927";
        String permutationAction = getPermutationAction(); // a permutation of the ten numerals

        // set-up the array of indirect reference
        int[] at = new int[permutationAction.length()];

        for (int i = 0; i < permutationAction.length(); i++)
        {
            at[i] = Character.getNumericValue(permutationAction.charAt(i));
        }

        // String plaintext = "abcdefghij";
        String plaintext = getPlaintext();

        char[] ciphertextArray = new char[permutationAction.length()];

        for (int i = 0; i < permutationAction.length(); i++)
        {
            ciphertextArray[ at[i] ] = plaintext.charAt(i); // indirect reference
        }

        String ciphertext = "";

        for (int i = 0; i < permutationAction.length(); i++)
        {
            ciphertext = ciphertext + ciphertextArray[i];
        }

        System.out.println();
        System.out.println("Plaintext: " + plaintext);
        System.out.println("Ciphertext: " + ciphertext);
    }

    static String getPermutationAction() throws Exception
    {
        System.out.println();
        System.out.print("ENTER PERMUTATION ACTION > ");

        InputStreamReader source = new InputStreamReader( System.in );
        BufferedReader keyboard = new BufferedReader( source );
        String input = keyboard.readLine();

        return input;
    }

    static String getPlaintext() throws Exception
    {
        System.out.print("ENTER PLAINTEXT (10 CHARs) > ");

        InputStreamReader source = new InputStreamReader( System.in );
        BufferedReader keyboard = new BufferedReader( source );
        String input = keyboard.readLine();

        return input;
    }
}
Problem E

The game of NIM is played with two players and match sticks in any number of rows. A player can take 1 to as many as the whole row of match sticks on each turn. The winner is the player who goes last. The game resembles the Chinese game of "Jianshizi" ("picking stones"). A winning strategy is to have the XOR of all the columns be zero (even count) after each move. For example:

<table>
<thead>
<tr>
<th>Matches</th>
<th>Binary</th>
<th>Matches</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1011</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>15</td>
<td>1111</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>1011</td>
</tr>
</tbody>
</table>

XOR of Columns = 0110 → Win Strategy = -6 from any row.

XOR = 0000 → no Win Strategy

Note: XOR is equivalent to MOD 2 on the Sum of the column.

Your problem is to write a program that accepts 5 numerical inputs (one input per line from 1 to 15) representing the matches. The output will be a winning strategy or "no winning strategy".

Input:
11
4
7
13
3

Output:
Take 6 matches from any row.
Problem F

The Fibonacci sequence of numbers has \( n_1 = 1, n_2 = 1, n_i = n_{i-1} + n_{i-2} \). The first 10 terms of this sequence are 1 1 2 3 5 8 13 21 34 55 ... . If we use mod 7 on the sequence we get the first 25 terms:

1 1 2 3 5 1 6 0 6 5 4 2 6 1 0 1 1 2 3 5 1 6 0 6 ... .

And you can see the sequence is repeating.

The repeating sequence has 16 values in the sequence that will repeat forever.

Your program will read in an integer \( N \) which will be the Mod for the Fibonacci sequence (2 \( \leq N \leq 10 \)) and your program should output the repeating sequence.

For Example:

Input:

7

Output:

The sequence for Mod 7 is (1 1 2 3 5 1 6 0 6 5 4 2 6 1 0)
Problem G

In GIS (Geographic Information Systems), maps are drawn with polygons that are generated by connecting points. An important question is "Do 2 lines cross?" which reduces to, is a point inside of a polygon. The Programming question: Is a point inside or outside of a triangle.

Given the triangle vertices (1,2), (3,5), (6,0) and the point in question = (4,1) The answer is Inside.

Assume that we will only work in the first quadrant. No test point will be on an edge. The program should input 3 vertices and a point in question from the keyboard.

Input:
1 2 //1 vertex per line separated by a space.
3 5
6 0
4 1 //point in question

Output:
(4,1) is inside the triangle (1,2), (3,5) and (6,0). //or outside as the case may be.
Problem H

There is a Canadian Network that destroys the packets if they exceed the maximum number of hops.

So when a packet is sent the maximum number of hops is placed in a time-to-live field in the packet. This hop count is decremented once for each hop that the packet takes. When the count is zero the packet is destroyed, thus preventing packets from flooding the network. Write the program to find this maximum hop count given that the network has no cycles. Your program should allow the input to come from the keyboard.

The graph:

```
   A
    |
    C-------D
    |  ^    |
    |   ^   |
    |       |
    G-------H
    |       |
    |  v    |
    I-------J
```

Note only the number of hops is to be printed - not the path.

Input :

```
N //The number of nodes. They will start at 'A' and continue to the Nth letter of the alphabet.

//In the above example N = 10.
AC  BC  CD  DE  EF  EG  GH  GI  IJ  //Hops (or Links) are enter as pairs one per line
```

Output:

Maximum number of hops = 6
Three Unused Problems
Newton's Method

Isaac Newton, in the Seventeenth Century, devised an algorithm for computing the $n^{th}$ root of any number to any degree of precision -- for instance, the square root of 2 correct to ten places or the fifth root of 765 correct to eight places.

Newton's method works through the formula:

$$\text{estimate} = \frac{\text{number} + (\text{root} - 1) \times (\text{approximation}^{\text{root}})}{\text{root} \times (\text{approximation}^{\text{root}-1})}$$

where: \textbf{estimate} represents the current computes estimate of the root

\textbf{root} represents the root being sought (for the fifth root, root equals 5)

\textbf{number} represents the number whose root is sought

\textbf{approximation} represents the latest estimate of the root thus far computed

For the initial iteration, $\text{approximation} = \frac{\text{number}}{\text{root}}$

Implement Newton's Method. Assume the number and the root are whole, positive numbers.
Continue the computation until the absolute value of $(\text{number} - \text{estimate}^{\text{root}})$ is less than 0.0000000001.

\textbf{INPUT}: the \textbf{number} and the \textbf{root} to be computed
\textbf{OUTPUT}: the computed result

Sample runs: number = 1024, root = 10, computed result = 2.0
number = 2, root = 2, computed result = 1.4142135623746899
number = 27, root = 3, computed result = 3.0000000000000018
number = 765, root = 5, computed result = 3.773395151070657
import java.util.Scanner;

class NewtonsMethod
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        System.out.print("number: ");
        int number = keyboard.nextInt();

        System.out.print("root: ");
        int root = keyboard.nextInt();

        double approximation = (double)number/root;
        double estimate = approximation;

        while (Math.abs(number - Math.pow(estimate, root)) >= 0.00000000001)
        {
            estimate = (number + (root-1)*Math.pow(approximation,root)) /
                (root*Math.pow(approximation, root-1));

            approximation = estimate;
            System.out.println(estimate);
        }

        System.out.println(estimate);
    }
}
Collating Sequence

The integers from 32 through 126, when cast to char, represent both the ASCII collating sequence and the ISO 8859 Latin-1 codesets, which are

If you printed the integers and characters down the page, you'd see what is shown to the right. (32 represents the character from the space bar. There is only room enough on the page to get through character number 110.)

TO DO: Write a program that displays the collating sequence from character 32 through 126 in the number columns read-in as input.

Each column should hold the same number of entries, except for the rightmost column which may hold several less (but no less than number_of_columns - 1)

Character numbers should increase sequentially down each column.

Input: number of columns

If the input were 4, the output should appear as shown below:

| 32 | 56 | 8 | 80 | P | 104 | h |
| 33 | ! | 57 | 9 | 81 | Q | 105 | i |
| 34 | * | 58 | : | 82 | R | 106 | j |
| 35 | # | 59 | ; | 83 | S | 107 | k |
| 36 | $ | 60 | < | 84 | T | 108 | l |
| 37 | % | 61 | = | 85 | U | 109 | m |
| 38 | & | 62 | > | 86 | V | 110 | n |
| 39 | ` | 63 | ? | 87 | W | 111 | o |
| 40 | ( | 64 | @ | 88 | X | 112 | p |
| 41 | ) | 65 | A | 89 | Y | 113 | q |
| 42 | * | 66 | B | 90 | Z | 114 | r |
| 43 | + | 67 | C | 91 | [ | 115 | s |
| 44 | , | 68 | D | 92 | \ | 116 | t |
| 45 | - | 69 | E | 93 | ] | 117 | u |
| 46 | . | 70 | F | 94 | ^ | 118 | v |
| 47 | / | 71 | G | 95 | | | 119 | w |
| 48 | 0 | 72 | H | 96 | ` | | 120 | x |
| 49 | 1 | 73 | I | 97 | a | | 121 | y |
| 50 | 2 | 74 | J | 98 | b | | 122 | z |
| 51 | 3 | 75 | K | 99 | c | | 123 | |
| 52 | 4 | 76 | L | 100 | d | | 124 | |
| 53 | 5 | 77 | M | 101 | e | | 125 | }
| 54 | 6 | 78 | N | 102 | f | | 126 | ~
| 55 | 7 | 79 | O | 103 | g | |
import java.util.Scanner;

class CollatingSequence {
    public static void main(String[] args) {
        System.out.print("number of columns: ");
        Scanner keyboard = new Scanner(System.in);
        int numCols = keyboard.nextInt();

        int firstVal = 32; //value in the collating sequence
        int lastVal = 126;

        int numberOfChars = lastVal - firstVal + 1;
        int charsPerCol = numberOfChars / numCols;

        if (numberOfChars % numCols != 0)
            charsPerCol = charsPerCol + 1;

        for (int row = 0; row < charsPerCol; row++) {
            for (int col = 0; col < numCols; col++) {
                int ASCII_value = (firstVal + row + charsPerCol * col);
                if (ASCII_value <= lastVal)
                    System.out.print(ASCII_value + " " + (char)ASCII_value + "\t");
            }
            System.out.println();
        }
    }
}
Simulation

There is a story about a statistics professor who told the class to record the results of tossing a fair coin 200 times. As he collected the papers, he amazed the class by declaring each data set to be authentic or faked. The telltale indicator was a run of six consecutive heads or a run of six consecutive tails. The odds are very high that 200 tosses of an actual coin will contain at least one such run. On the other hand, the odds are also high that fudged data will not contain such a run because most people are unaware of this counterintuitive probabilistic fact.

The aim of this program is to calculate an approximation of the probability of obtaining either a run of six heads or a run of six tails in 200 tosses of a fair coin.

** The program's centerpiece will be a simulated a trial of 200 tosses and a test to see whether or not a 6-head run or a 6-tail run appeared.

The "trials with a run" counter is incremented by 1 if the particular trial has one or more runs.

** The program will conduct 1000 of these trials and display the number of these in which the runs were found.

INPUT: none

OUTPUT: 960 of the 1000 trials had a run of six heads or a run of six tails

Due to its stochastic nature, this number will vary from run to run.
class Simulation
{
    public static void main(String[] args)
    {
        final int NUMBER_OF_TRIALS = 1000; // 1000 replications of the 200 coin flips
        int trialsWithRun = 0; // accumulator

        for (int trialNum = 1; trialNum <= NUMBER_OF_TRIALS; trialNum++)
        {
            // initializations for the current trial of 200 tosses
            char previousToss = ' ';     // char currentToss = ' ';
            int runLength = 0;
            int tossNumber = 0;

            // toss the coin two hundred times or until a "run of 6" is found
            while (tossNumber < 200 && runLength < 6)
            {
                tossNumber++;

                if (Math.random() < .5) currentToss = 'h';
                else currentToss = 't';

                if (currentToss == previousToss)
                {
                    runLength++;
                    if (runLength == 6) trialsWithRun++;
                }
                else
                {
                    runLength = 1;
                    previousToss = currentToss;
                }
            }

            System.out.println( trialsWithRun + " of the " + NUMBER_OF_TRIALS + 
                                 " trials had a run of six heads or a run of six tails.");
        }
    }
}
Community College Programming Contest (CCPC)

Participate in a programming contest between area community colleges. Teams will compete on Saturday, April 24, 2010 at Pace University, Pleasantville Campus.

**Place:** Goldstein Academic Center, Pleasantville Campus, Pace University

**Time:** 9:00 – 2:00.
- 9:00  Registration, coffee, juice (Goldstein Academic Center Room 100)
- 9:30  Rules and method of submission (Goldstein Academic Center Room 300)
- 10:00 Contest (Goldstein Academic Center Room 300)
- 3:00  Awards and reception

**Languages:** C++ or Java

**Compilers:** Eclipse or Plain Text Editor

**Operating System:** MS Windows XP

**Eligibility:** Students must be enrolled with at least a half time load. This will be verified by the School Team Advisor (Faculty or Staff)

**Teams:** Three students per team; at most three teams per college

**Fee:** $25 per team

**Problems:** 6 to 8 problems posed in English. String, Mathematical, Graph, Sort/Search, Compression/Encryption, etc.

**Computers:** Two per team supplied by Pace. USB's, Disks etc. will not be allowed. There will be no access to the Internet. Cell phones and other electronic devices will not be allowed in the room.

**Allowed:** Books, program listings, manuals

**Scoring:** Five points for each problem solved. One point penalty for each incorrect submission of that problem. Maximum of 5 penalties per problem submitted

**Winner:** Highest scoring team wins. Tie breaker will be submissions with the lowest total time

**Rejected Submissions:** Syntax errors, run-time error, time-limit error, wrong answer, failed test case, too little/too much output. **Note:** Only the first observed error will be noted

**Collaboration:** No collaboration with other teams. Questions directed to the Contest Staff. If there is a problem with ambiguity, all teams will be notified

**Prizes:** First, second and third place prizes
The Seidenberg School of Computer Science and Information Systems
Pace University
Technical Report Series

EDITORIAL BOARD

Editor:
Allen Stix, Computer Science, Pace--Westchester

Associate Editors:
Constance A. Knapp, Interim Dean, The Seidenberg School of Computer Science and Information Systems -- Pace University
Susan M. Merritt, Computer Science, Pace--Westchester

Members:
Howard S. Blum, Computer Science, Pace--New York
Mary F. Courtney, Computer Science, Pace--Westchester
Nicholas J. De Lillo, Mathematics and Computer Science, Manhattan College
Daniel Farkas, Information Systems, Pace--Westchester
Fred Grossman, Information Systems; Doctor of Professional Studies, Pace--New York and White Plains
Fran Goertz, Information Systems, Pace--Westchester
Joseph F. Malerba, Computer Science, Pace--Westchester
John S. Mallozzi, Computer Information Sciences, Iona College
John C. Molluzzo, Information Systems, Pace--New York
Pauline Mosley, Technology Systems, Pace--New York
Narayan S. Murthy, Computer Science, Pace--New York
Catherine Riccardo, Computer Information Sciences, Iona College
Judith E. Sullivan, CSIS Alumna, MS in CS from Pace--Westchester
Sylvester Tuohy, Computer Science, Pace--Westchester

The Seidenberg School of Computer Science and Information Systems, through the Technical Report Series, provides members of the community an opportunity to disseminate the results of their research by publishing monographs, working papers, and tutorials. Technical Reports is a place where scholarly striving is respected.

All preprints and recent reprints are requested and accepted. New manuscripts are read by two members of the editorial board; the editor decides upon publication. Statements of policy and mission may be found in issues #29 (April 1990) and #34 (September 1990).

Please direct submissions as well as requests for single copies to:

Allen Stix
The Seidenberg School of Computer Science and Information Systems
Goldstein Academic Center
Pace University
861 Bedford Road
Pleasantville, NY 10570-2799