

2010 Iverson Computing Science Competition

May 25, 2010

Name: ANSWER - edited by P. Rudnicki (May 28, 2010)
+ Marking Guide

School: _____

City: _____

Grade: _____

CS Teacher: _____

Are you taking AP or IB Computer Science? (Yes/No) _____

Have you taken Advanced Level courses? These are courses at the 3000 level such as CSE3110: Iterative Algorithms 1.

(Yes/No/Taking one or more now) _____

Please write clearly! If we cannot read your writing, we cannot give you marks.

Question	Question Name	Part a	Part b	Part c	Total
		Marks out of			
		2	3	5	10
1	Strings				
2	The Trip				
3	Dick and Jane				
4	Polylops				
TOTAL					

Exam Format:

This is a traditional 2 hour paper and pencil exam. It consists of four questions with each question consisting of three parts. Part a) of each question typically asks you to solve a specific instance of the problem by hand, and it is recommended for everyone. Part b) is designed to be relatively easy and Part c) is more challenging.

Even students who have just started their work in Computing Science should be able to do reasonably well on part a) of each question. Part b) of each question requires some more of problem solving skills. More knowledgeable students should be able to do reasonably well on the Part c) of each question as well. There are no required questions. Solve as many parts of as many questions as you can.

Programming Language:

The parts of the questions that require programming can be answered using any programming language you wish (for example, VB, C/C++, Java, or Perl). You can also use pseudo-code. Be sure to provide an adequate amount of detail so the markers can determine if your solution is correct. **Pseudo-code should be detailed enough to allow for a near direct translation into an appropriate programming language.**

Our primary interest is in your higher order thinking skills rather than on your code wizardry. So a demonstration of logical thinking and systematic problem solving approaches will count for more than a mastery of syntax of one particular language. Still, you will have to use some type of programming language or pseudo-code to demonstrate what you can do. Minor syntactical errors will be ignored. Add comments where needed to clarify your code.

Suggestions:

1. Read the problem descriptions carefully. To get full marks all problem specifications for the parts of the questions you attempt have to be met. You can assume that only valid input will be entered by the user. You do not need to include out-of-range or data type error checks in the input to your programs.
2. Where feasible, sample executions of the desired program have been included for each problem. Review them carefully to make sure you haven't missed any specifications and to get hints as to how to proceed.
3. We strongly recommend that you do some design work prior to writing any code. Use pseudo-code, diagrams, screen displays, tables or any other aid to help you plan your code. We will be looking at your rough work and you can get marks for it. Remember we are looking for the key computing ideas, not specific coding details. In particular you can

invent your own “built-in” functions for subtasks such as reading the next number, or the next character in a string, or loading an array. Just make sure you specify those functions by giving the relationship between inputs and outputs. Use pre- and post-conditions if you know what these are.

4. Take a look at all of the questions before deciding which ones to attempt, and in which order. Start with the easiest parts of each question. It is OK to do the questions out of the presented order.
5. Make sure to include English language comments to explain non-obvious or “clever” parts of your solution.

Question 1: Recognizing String Literals

In almost all programming languages, a sequence of characters enclosed in double quotes, such as "Hello World!" in C, is called a *character string* or *string literal*.

Your job in this question is to develop a finite state machine that reads input text one character at a time and recognizes whether string literals in the input text are properly formed.

String literals are allowed to contain the double quote character. However, to avoid ambiguity, a double quote character within a string has to be preceded by a backslash (\) character. For example, a string literal containing the following characters

```
Pat said: "let's play!" in a loud voice
```

has to be written as

```
"Pat said: \"let's play!\" in a loud voice"
```

Note how the whole string is enclosed in double quotes, but the double quotes inside the string are preceded by a backslash. Usually, some other characters in strings can also be preceded by a backslash to mean something special, but while allowing this to happen we will not be concerned with this issue here.

A backslash cannot appear in a valid input text outside of a string literal.

In order to answer Part b) and Part c) of this question, you will need some

Basic terminology for finite state machines

A *finite-state machine* (FSM) is a kind of computation device. Although you don't usually encounter them directly, finite-state machines exist inside almost any device that does something interesting albeit quite mechanical: kitchen appliances, remote control toys, simple phones (the complex ones have full computers), and so on.

There are three primitive concepts associated with an FSM, each associated with part of an FSM diagram.

State is written as a circle that indicates the settings of the parts of the machine. Typically the state has a label that tells what it means for the FSM to be in the state.

Transition is written as an arrow and indicates a potential change from one state to another or possibly the same state. The transition is labeled with the events that cause the transition to occur.

Event is a cause of a transition. For example, pushing a button, or reading a character from the input, or the ticking of a clock are events.

An FSM does a computation by starting in a specific start state (marked with a *), and then waiting for events to occur. Each event causes the machine to change state by following the transitions that match the event. The computation stops when there are no more events; or an event occurs that does not have a matching transition. In the latter case, we say that the FSM gets stuck.

A state of an FSM is called accepting or final, written as a double circle in the diagram, if the state corresponds to some desired property of the input text read so far.

A good physical intuition for these notions is: a state is a possible location that you can be at on a map. When you are at one location a transition is a road that can take you to another location. An event causes you to choose a particular road to travel. A computation is a road-trip, or path, between a start location and an end location.

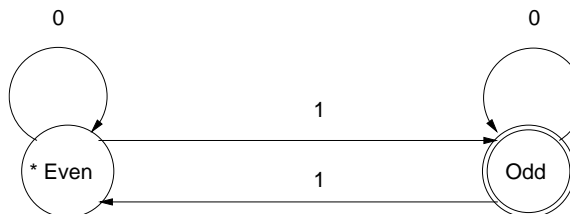
When you trace an FSM computation, think of putting a marker (like a pebble) on the current state, and when an event occurs, moving the marker to the next state along the transition that matches the incoming event.

Example Here is an FSM that reads characters from input and depending whether it reads a 0 or reads a 1 it switches between states.



This FSM starts in state **Even** (the * means start in this state). Every time a 1 is read, the FSM makes a transition to the other state. But every time a 0 is read the FSM makes a transition back to the same state. Thus, the name of the current state tells you whether the number of 1's that have arrived so far is odd (state **Odd**) or even (state **Even**). None of the states is marked as the accepting state.

The following machine is similar and accepts inputs with odd numbers of 1's. Note the accepting state **Odd**.



Question 1 Part a)

As stated above, the following sequence of characters

Pat said: "let's play!" in a loud voice

is made the contents of a string literal as follows.

"Pat said: \"let's play!\" in a loud voice"

Write down string literals containing the following sequences of characters.

1. Ted yelled: "got you!!!" and ran away

Answer: "Ted yelled: \"got you!!!\" and ran away"

2. "Use quotes? Why???"

Answer: "\"Use quotes? Why???\""

3. but:" Quotes within "quotes?" ridiculous!" she said.

Answer: "but:\" Quotes within \"quotes?\" ridiculous!\" she said."

4. """"""

Answer: "\"\"\"\"\"\"\"\"\"\""

Marking: deduct 1 mark for each error but give 1 mark if there is some other reason.

Question 1 Part b)

Define a finite state machine (FSM) for accepting valid input texts in which string literals are properly formed. Your FSM should read one character at a time from left to right. For simplicity, assume that only a, b and " will appear as input characters. Note: **for this part assume that no other characters can appear in a valid input text.**

The FSM should check whether each input text is properly built with respect to double quotes at both ends of a string. The FSM should reach an accepting state for such texts and only for such texts.

Empty input text is valid.

Examples:

1. a"aba"b is a valid input text.
2. a"b is not a valid input text. The quote starts a string which is not closed.
3. "ab"aaa"bb"ab"a"" is valid. A string can be empty and in such case the closing quote directly follows the opening one.
4. a"b"a""a"a is not valid. The last string is not closed.

Answer

Marking: Essentially no partial marks but use your own judgement. It is OK to have extra states for invalid input.

Question 1 Part c)

Define a finite state machine (FSM) for accepting valid input texts in which string literals are properly formed. Your FSM should read one character at a time from left to right. Assume that only the following four characters

a b \ "

occur in valid input texts. An input text containing any other character is invalid.

The FSM should check whether each input text is properly built with respect to double quotes at both ends of a string. Your FSM must account for double quotes within a string which are preceded by \. Those escaped double quotes characters, i.e. \", do **not** need to be balanced. The character \ is only allowed within a string and its occurrence outside of a string renders the input text invalid.

The FSM should reach an accepting state for such texts and only for such texts which satisfy the conditions described above.

Empty input text is valid.

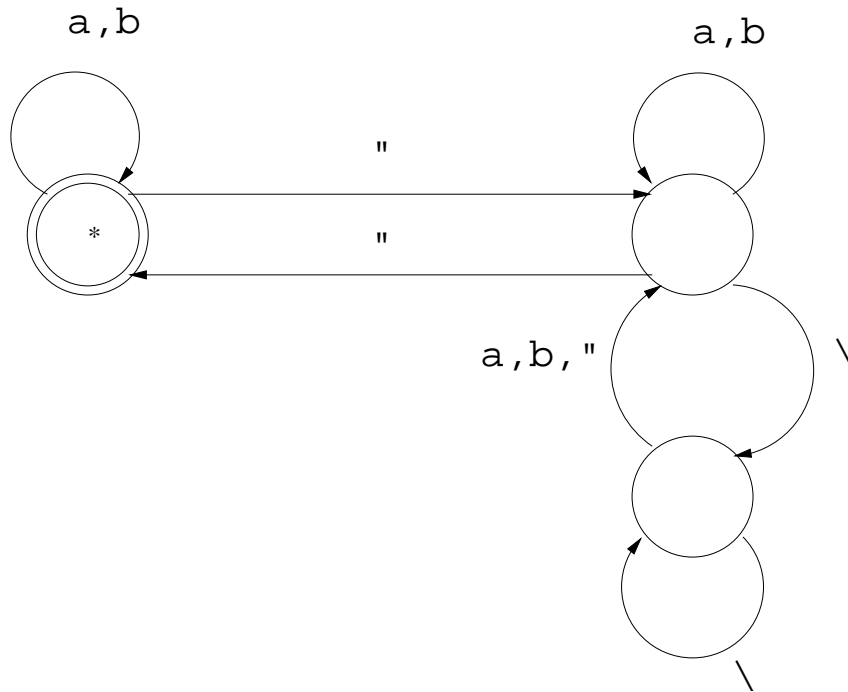
The examples below should clarify remaining doubt.

Examples:

1. All valid input texts from part b) are still valid.
2. "a\"b" is a valid input text. (The single quote in the middle is preceded by backslash.)
3. "ab"aaa\"bb"ab" is not valid (the \" is not within a string).
4. a"b\"a\""a"a" is valid.
5. "\"\"\"\"\"\"\"\"\"\"\"\" is valid.

Space for answering Question 1 Part c.

Answer



Note that the FSM does not treat `\` as a special character if it is not followed by `"`. The FSM also does not consume the entire input if the input is invalid.

Marking: Essentially no partial marks but use your own judgement. It is OK to have extra states for invalid input. Deduct 1 mark for wrong treatment of `\\` in a string.

Question 2: The Trip

A number of students are members of a club that travels annually to exotic locations. Their destinations in the past have included Indianapolis, Phoenix, Nashville, Philadelphia, San Jose, Atlanta, and more recently to Stockholm and Harbin. The next spring they are planning a trip to Cairo.

The group agrees in advance to share expenses equally, but it is not practical to have them share every expense as it occurs. So individuals in the group pay for particular things, like meals, hotels, taxi rides, plane tickets, etc. After the trip, each student's expenses are tallied and money is exchanged so that the net cost to each is the same, to within one cent. In the past, this money exchange has been tedious and time consuming. Your job is to compute, from a list of expenses, the minimum amount of money that must change hands in order to equalize (within a cent) all the students' costs.

Input

Standard input will contain the information for several trips. The information for each trip consists of a line containing a positive integer, n , the number of students on the trip, followed by n lines of input, each containing the amount, in dollars and cents, spent by a student. There are no more than 1000 students and no student spent more than \$10,000.00. A single line containing 0 follows the information for the last trip.

Output

For each trip, output a line containing one number stating the minimum total amount of money, in dollars and cents, that must be exchanged to equalize the students' costs.

Sample Input and Output

Sample Input

```
3
10.00
20.00
30.00
4
15.00
15.01
3.00
3.01
0
```

Output for Sample Input

```
$10.00
$11.99
```

Question 2 Part a)

Solve this problem by hand for the following input and write down the expected output.

3
5.00
10.00
30.00
4
140.13
9.01
31.96
76.67
0

Answer

\$15.00
\$87.91

Marking: 1 mark for each correct answer. (An output off even by \$0.01 is wrong).

In the first case, the sum of all expenditures is \$45.00 and the average is \$15.00. \$15.00 are transferred by the third student: \$10.00 are given to the first student and \$5.00 to the second.

In the second case, the sum is \$257.77 and the average is \$64.4425 which means that some students will end up with \$64.44 and some with \$64.45. The transferred money comes from the first student ($\$140.13 - \$64.45 = \$75.68$) and from the last student ($\$76.67 - \$64.45 = \$12.22$); total \$87.91.

Question 2 Part b)

Write a program that solves this problem for the case where there are exactly two students, $n = 2$.

Answer

See Part c for the general solution.

When there are only two students the problem is substantially simplified and can be solved in the following steps.

1. Compute the sum $S = s_1 + s_2$ of student expenditures.
2. Compute the average $a = S/2$ using integer division.
3. If $a > s_1$ then $a - s_1$ must be transferred, else if $a > s_2$ then $a - s_2$ must be transferred, else no transfer of money is needed.

Marking: deduct marks for using real numbers or if they compute some average of some differences.

Question 2 Part c)

Write a program for this problem for any number n of students, $0 \leq n \leq 1000$.

The following program is written in C and we hope that you can follow the program without additional explanations.

```
#include <stdio.h>

int main () {
    int n, A[1010];
    int sum, ave, i, d1, d2;
    int sabove, sbelow;

    while (1) {
        scanf ("%d", &n);
        if (!n) break;
        sum = 0;
        for (i = 0; i < n; i++) {
            scanf ("%d.%d", &d1, &d2);
            A[i] = d1*100+d2;
            sum += A[i];
        }

        ave = sum / n;
        sabove = sbelow = 0;
        for (i = 0; i < n; i++) {
            if (A[i] > ave+1) sabove += A[i] - (ave+1);
            if (A[i] < ave) sbelow += ave - A[i];
        }

        printf("$%.2f\n", ((sabove > sbelow) ? sabove : sbelow)/100.0);
    }
}
```

Marking: The program must compute the average in a loop (1 mark). If only the sum of above average or only the sum of below average is given as an answer then add 2 marks.

(It is unlikely that any student got it entirely right.)

Question 3: Dick and Jane

Dick is 12 years old. When we say this, we mean that it is at least twelve and not yet thirteen years since Dick was born.

Dick and Jane have three pets: Spot the dog, Puff the Cat, and Yertle the Turtle. Spot was s years old when Puff was born; Puff was p years old when Yertle was born; Spot was y years old when Yertle was born. The sum of Spot's age, Puff's age, and Yertle's age equals the sum of Dick's age and Jane's age (j). How old are Spot, Puff, and Yertle?

Each input line contains four non-negative integers: s, p, y, j . For each input line, print a line containing three integers: Spot's age, Puff's age, and Yertle's age. Ages are given in years, as described in the first paragraph.

Sample Input and Output

Sample Input	Output for Sample Input
5 5 10 9	12 7 2
5 5 10 10	13 7 2
5 5 11 10	13 7 2

Question 3 Part a)

Explain how the first sample solution was computed.

Answer

The sum of the ages of all three pets is 21 ($12 + 9$).

A rather naive explanation consists of just trying possible age of Yertle in years as, in this case, Yertle is the youngest of the three pets.

If Yertle is 0 years old at the current moment, then Spot is no more than 11 years old and Puff is no more than 6 years old. This cannot be as the sum of pet's ages is then 17.

If Yertle is 1 year old, then Spot is no more than 12 years old and Puff is no more than 7 years old. The sum of pet's ages is then 20 which is not enough.

If Yertle is 2 years old, then it is easy to see that Spot can be 12 years old, Puff can be 7 years old, which makes the total sum of pet's ages 21 as desired.

This approach would lead to some complications when applied to the 2nd and 3rd sample inputs. For proper solution, see Part b and Part c.

Marking: 2 marks if they get something like written above, even if details are missing.

Question 3 Part b)

Compute the missing entries in the following table.

s	p	y	j	Spot's age	Puff's age	Yertle's age
4	4	8	3	9	5	1
4	4	9	13	13	8	4
4	5	10	3	10	5	0
4	6	10	10	12	8	2
5	4	10	10	13	7	2
6	4	10	5	11	5	1

Marking: deduct 1 mark for each error but give 1 mark if at least 3 are OK.

Question 3 Part c)

Write a program that solves this problem. There is a solution which is very short, yet getting a correct solution (not even the short one) requires some careful thought.

Answer

In our solution we will use quotient ($/$ or `div`) and remainder ($\%$ or `mod`) of integer division. Given positive integers a and r , we have $a = (a/r) * r + a\%r$ with $0 \leq a\%r < r$.

The following property of integer division is crucial for the solution of our problem: Given positive integers a, b and r , we have

$$\begin{aligned}(a + b)/r &= ((a/r) * r + a\%r + (b/r) * r + b\%r)/r \\ &= a/r + b/r + (a\%r + b\%r)/r\end{aligned}$$

where $(a\%r + b\%r)/r < 2$ since $0 \leq a\%r < r$ and $0 \leq b\%r < r$. In other words, $(a\%r + b\%r)/r$ is either 0 or 1.

Let s, p, y, j be as in the problem statement. Let *Spot*, *Puff* and *Yertle* be the sought for answers.

For the explanation of the solution let S, P, Y, J be time intervals measured in days that correspond to s, p, y, j which are measured in years. Let the length of the year be r days; the actual value of r is irrelevant.

Let T be the number of days that passed since Yertle was born. We have $Spot = (Y + T)/r$, $Puff = (P + T)/r$, and $Yertle = t = T/r$. We have

$$\begin{aligned}12 + j &= (Y + T)/r + (P + T)/r + T/r \\ &= y + t + u + p + t + v + t \\ &= y + p + u + v + 3t\end{aligned}$$

where both u and v are integers, each either 0 or 1. If we set $d = (12 + j - y - p)$, then $u + v = d\%3$ and $t = d/3$.

- If $u = v = 0$, then $Spot = y + t$, $Puff = p + t$ and $Yertle = t$.
- If $u = v = 1$, then $Spot = y + t + 1$, $Puff = p + t + 1$ and $Yertle = t$.
- If $u + v = 1$, then we have to sort out whether $u = 1$ or $v = 1$. Note that we have $S + P = Y$ and therefore either $s + p = y$ or $s + p + 1 = y$. If $s + p = y$, then $P\%r \leq (S + P)\%r$ and therefore $u = 1$, otherwise $v = 1$.

Marking: Must have a loop (1 point). If the answer does not take into account the adjustments for extra years then 2 points maximum.

(I do not think anybody got it right.)

The complete program in *C* is as follows.

```
#include <stdio.h>
int main(){
    unsigned s,p,y,j, Spot, Puff, Yertle;
    int d;
    while(scanf("%d %d %d %d", &s,&p,&y,&j)==4)
    {
        d = (12+j) - (y+p);
        Spot = y + d/3;
        Puff = p + d/3;
        Yertle = d/3;
        switch (d%3) {
        case 0:
            break;
        case 1:
            if(s+p == y)
                Spot++;
            else
                Puff++;
            break;
        case 2:
            Spot++; Puff++;
            break;
        };
        printf("%d %d %d\n", Spot, Puff, Yertle);
    };
};
```

Question 4: Polylops

Given the vertices of a non-degenerate polygon (no 180-degree angles, zero-length sides, or self-intersection - but not necessarily convex), you must determine how many distinct lines of symmetry exist for that polygon. A line of symmetry is one on which the polygon, when reflected on that line, maps to itself.

Input

Input consists of a description of several polygons.

Each polygon description consists of two lines. The first contains the integer n , with $3 \leq n \leq 1000$, which gives the number of vertices on the polygon. The second contains n pairs of numbers (an x- and a y-value), describing the subsequent vertices of the polygon in some order. All coordinates are integers from -1000 to 1000.

Input terminates on a polygon with 0 vertices.

Output

For every polygon described, print out a line saying

Polygon # x has y symmetry line(s).

where x is the number of the polygon (starting numbering from 1), and y is the number of distinct symmetry lines on that polygon.

Sample Input and Output

Sample Input

```
4
-1 0 0 -1 1 0 0 2
3
-666 -42 57 -84 19 282
3
-241 -50 307 43 -334 498
0
```

Output for Sample Input

```
Polygon #1 has 1 symmetry line(s).
Polygon #2 has 0 symmetry line(s).
Polygon #3 has 1 symmetry line(s).
```

Question 4 Part a)

Give examples of polygons with 2 and 4 lines of symmetry. List the coordinates of the vertices of the polygons as above.

(We have good reasons for not asking for 3 lines of symmetry :-).

Answer

An example of a polygon with two lines of symmetry is a rectangle which is not a square.

An example of a polygon with four lines of symmetry is a square.

(An equilateral triangle has three lines of symmetry. Unfortunately, we cannot have an equilateral triangle whose vertices are at integer coordinates.)

Marking: 1 mark for each correct answer.

Question 4 Part b)

Solve the following programming problem. Given a polygon p as above and a line L , determine whether L is a symmetry line for p . Let L be given by the coordinates a_x, a_y, b_x, b_y of two (different) points a and b on L .

Answer

Please have a look at the solution for Part c which includes some guiding comments.

Thanks to Alfred Ye from Strathcona for suggesting a simple way of checking that two points are a reflection of each other wrt to a given line. His suggestion is used in the solution to Part c. (My original solution included perpendicularity checking with dot product.)

Marking: Marks for checking that every point has a corresponding reflection point in order of the the given points. Use your own judgment and give marks even if the solution is technically wrong but there is some thought in it.

Question 4 Part c)

Solve the original programming problem above.

Roughly, what is the running time of your algorithm for a polygon with n vertices in terms of n ? (Rough estimate of running time is fine.)

Marking: Use your judgement and give marks if there is any reason for.

(If you see a correct solution please let me know asap: I have seen one.)

The complete solution in C.

```
#include <stdio.h>
typedef struct {
    int x, y;
} Point;
Point p[3000];

int distance(Point p1, Point p2) { // square of dist
    return (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
}

main() {
    int i, j, z, n, prob=1;
    for(;;) {
        scanf( " %d", &n );
        if( !n ) break;
        // double the number of points to include midpoints
        n *= 2;
        for( i = 0; i < n; i += 2 ) { // the vertices located at even i's
            scanf( " %d %d", &p[i].x, &p[i].y );
            p[i].x *= 4; p[i].y *= 4; // midpoints' coordinates are also integers
        }
        for( i = 1; i < n; i += 2 ) { // midpoints of the sides at odd i's
            p[i].x = (p[i-1].x+p[(i+1)%n].x)/2;
            p[i].y = (p[i-1].y+p[(i+1)%n].y)/2;
        }
        z = 0; // counter for lines of symmetry
        for( i = 0; i < n; i++ ) {
            // Try that p[i] and p[(i+n/2)%n] are on a line of symmetry.
            // p[(i+j)%n] and p[(i+n-j)%n] must be reflections wrt to the line.
            for( j = 1; j < n/2; j++ ) {
                if ( ! (distance(p[i], p[(i+j)%n]) == distance(p[i], p[(i+n-j)%n]) &&
                    distance(p[(i+n/2)%n], p[(i+j)%n])
                    == distance(p[(i+n/2)%n], p[(i+n-j)%n])) )
                    break;
            }
            if( j < n/2 ) continue; // did not manage to check all
            z++; // Every line will be counted twice
        }
        printf( "Polygon #%d has %d symmetry line(s).\n", prob++, z/2 );
    }
}
```

(It doesn't look good but it fits into the page.)

The running time of the algorithm is proportional to n^2 because of the two nested loops when trying all possible lines of symmetry.

_____ Extra space _____