2012 Iverson Computing Science Competition
May 30, 2012

Answer key and marking guide.

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	Telephone alphabetics				
2	Chairlift				
3	Points on Polygon's Boundary				
4	Average Speed				
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. Parts b) and c) are 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. Parts b) and c) of each question require some more problem solving skills. 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 a programming language.**

Our primary interest is in your higher order thinking skills rather than your code wizardry. 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 you have to meet all problem specifications for the parts of the questions you attempt. 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 for the input to your programs.

2. Where feasible, sample executions of the desired program have been included. Review the samples carefully to make sure you understand the specifications. The samples can also give you 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, 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. 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 a relationship between their inputs and outputs.

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: Telephone alphabetics

A telephone set has many buttons as shown in the figure below. On certain occasions, for example, when texting or when the user wishes to add a new name to the private phone directory stored in the telephone set, some of the buttons are used in alphabetical entry mode.

In alphabetical entry mode, the user can enter a certain letter by pressing the corresponding digit key as many times as the position (in left-to-right order) of the desired letter on that particular key. For example, pressing the key 7 once means a `P`, whereas pressing the same key four times means an `S`. The `#` key is used to separate successive letters. However, pressing the `#` key between two letters may be omitted when it is clear where one letter ends and the next one begins.

Here are some examples with the details of encoding explained below.

	Key strokes—input	Text generated—output
Example 1	`44488833777#777766666`	`IVERSON`
Example 2	`3399#26`	`EXAM`
Example 3	`6#666#663399982555557777`	`MONEYTALKS`
Example 4	`3#33223366222999729997777`	`DECENCYPAYS`

- The input contains one line which contains one or more characters.
- The only visible characters that may appear in the input are the digits 2 through 9 and the pound character `#`. There will be no leading or trailing blank spaces or tabs in the input.
- The first character of every line will be a digit. The last visible character of every line will be also a digit.
- Each digit or `#` in the input will be interpreted by your program as a keystroke on the above telephone set in alphabetical entry mode.
- The output line consists of the alphabetical interpretation of the input line in upper case letters.
- The separator `#` between letters may be omitted between two digits that are different. For example, in the first line of the sample input above, the separator between `33` and `777` has been omitted and inserting a separator at this point would not affect the output. On the other hand, in the second input line, you see a redundant separator between `99` and `2`. Removing this separator will not affect the output.

- The separator `#` between letters may also be omitted in the case of a "long" stream of the same digit: a stream of more than four occurrences of `7`, more than four occurrences of `9`, or a stream of more than three occurrences of one of the other digits. For example, in the first line of the above input, there is a stream of five `6`'s. Inserting a separator after the first three `6`'s would not affect the output. Similarly, in the third line of the above input, removing the separator between the three `6`'s and the two `6`'s would not affect the output. More generally, in case of a long stream of `6`'s (or of any other digit except `7` or `9`), a separator could be inserted after every third `6` (starting from the left), without affecting the output. For example, `99996666666666666666` is equivalent to `9999666#666#666#666#666#6`, which would produce the output `ZOOOOOM`. The same rule applies to long streams of `7`'s or `9`'s, with "three" replaced by "four."
- The input will not contain two or more consecutive separators.

Question 1, Part a:

- Write down two different sequences of keystrokes which produce the following text.

  `THEGAMEISAFOOT`

  Answer. For example these two but there are many others as all pound signs are optional.

  ```
  844334263344477772333666666 8
  8#44#33#4#2#6#33#444#7777#2#333#666#666#8
  ```

- What is the text produced by the following keystrokes?

  `333444777#777787777666222557777844336677774466633777 7`

  Answer.

  ```
  FIRSTSOCKSTHENSHOES
  ```

- What is the text produced by the following keystrokes?

  `62667555266777746663555288 4#447777`

  Answer.

  ```
  MANPLANSGODLAUGHS
  ```

Question 1, Part b:

How many different sequences of input keystrokes can generate `IVERSON`?

Answer.

```
444 888 33 777#7777 666 66
  ?   ?  ?        ?   ?
```

There are 5 places where an optional `#` can be placed which results in $2^5 = 32$ different keystroke sequences that produce `IVERSON`.

Question 1, Part c:

Write a program which reads an input line and produces the desired output as described above.

Below is a solution by one of our teams in 2001.
Give some marks for any evidence of a reasonable idea expressed in whatever notation.

```
#include <stdio.h>

char line[10000];

int main() {
  char *alpha[]= { "ABC","DEF","GHI","JKL",
                   "MNO","PQRS","TUV","WXYZ"};
  int i, count;

  while(fgets(line,9000,stdin)) {
    for (i = 0; line[i] != '\0' && line[i] != '\n'; i+= count) {
      if (line[i]=='#') {
        count = 1;
        continue;
      }
      for (count = 1; line[count+i]==line[i];) count++;
      if (line[i]=='7' || line[i]=='9') {
        if (count > 4) count = 4;
      } else {
        if (count > 3) count = 3;
      }
      printf("%c",alpha[line[i]-'0'-2][count-1]);
    }
    printf("\n");
  }
  return 0;
}
```

Question 2: Chairlift

A chairlift is used to transport skiers up the mountain. The figure below shows the components of a chairlift, viewed from the air above the chairlift.

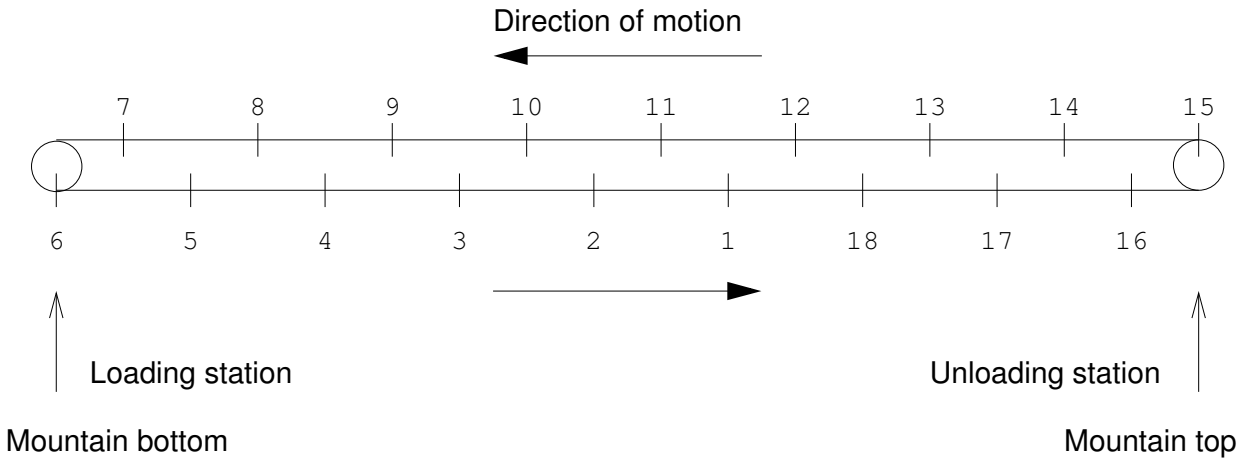

Figure 1: Components of a chairlift.

- A closed-loop steel cable is in continuous motion around two rotating wheels: one at the bottom, one at the top of the mountain.
- There are N equally spaced chairs suspended from the cable. N is an even integer, $18 \leq N \leq 300$. In Figure 1, $N = 18$. The chairs are shown as short dashes across the cable. You must understand that Figure 1 is a *snapshot*, i.e., that the chairs are in continuous motion along with the cable.
- The circumference of each wheel at the end of the chairlift is equal to the distance between adjacent chairs.
- Passengers ride up in the chairs on the *upward bound* side of the lift; the chairs come back down empty on the *downward bound* side.
- The loading station (see Figure 1) is located on the lift's upward bound side, at the point of tangency between the cable and the wheel at the bottom of the mountain. When a chair goes by the loading station, passengers will quickly board the chair; the chair doesn't stop. Each chair is actually a bench, it can hold up to four people, i.e. we have a quad-chair.
- The unloading station (see Figure 1) is also located on the lift's upward bound side, at the point of tangency between the cable and the wheel at the top of the mountain. When a chair goes by the unloading station, passengers will quickly get off the chair; the chair doesn't stop.
- The chairs are numbered in the order in which they appear at the loading station when the lift is first activated in the morning. That is, they are numbered starting at 1, increasing

sequentially in a direction opposite to the direction of motion. Figure 1 shows chair numbers for the case $N = 18$. Remember that the figure is just a *snapshot*, the chairs are in continuous motion.

- An upward bound chair and a downward bound chair *meet* when they are exactly opposite each other, i.e., both chairs are equidistant from the top of the mountain. At the instant shown in Figure 1, no two chairs meet, but very soon thereafter, chairs 6 and 7, as well as chairs 5 and 8, etc., would meet.

Question 2, Part a:

Consider a chairlift with 18 chairs as in Figure 1 at the instant when chair 1 meets chair 10. Complete the following table.

Chair	1	2	3	4	5	6	7	8	9
Meets chair	10	9	8	7	6	5	4	3	2

Does it matter which direction chair number 1 is going?

Answer No.

Question 2, Part b:

Two friends, whom we shall call Bob and Alice, are out for a day of skiing. They are just riding up in chair 56 and know that the lift has 126 chairs. Alice notes that their chair meets a downward bound empty chair number 71. Complete the following table.

Chair	56	1	15	33	87	92	111
Meets chair	71	126	112	94	40	35	16

Question 2, Part c:

Now that you are familiar with the chairlift operation, it is time to state the programming problem.

Two friends, whom we shall call Bob and Alice, are out for a day of skiing. They are just riding up in chair 56 and know that the lift has 126 chairs. They also know that the ride from the loading station to the unloading station takes exactly 225.0 seconds. Bob asks Alice: *How much longer will the ride take?* Alice doesn't look at her watch. Instead, she waits until their chair meets a downward bound empty chair, notes that the number of that chair is 71, and replies: *We have 198.9 more seconds to go.*

Your program will carry out Alice's computations.

The first line of the input file will contain N, followed by T. N is an even integer, $18 \leq N \leq 300$. T (the time it takes to ride from the loading station to the unloading station) is a floating point number, measured in seconds, $200.0 \leq T \leq 999.9$. The first line will be followed by one or more additional lines of input. On each of these additional lines, there will be exactly two integer values in the range $1..N$, representing two distinct chairs. The first of these chairs is the upward bound chair in which Bob and Alice are riding. The second chair is the downward bound chair they meet when Alice announces the remaining time.

After displaying N and T, your program will produce one line of output, containing the two chair numbers and the remaining time, for each pair of chairs in the input.

Sample Input

```
126 225
56 71
31 120
100 53
```

Sample Output

```
N = 126, T = 225.0
Chair  56 meets chair  71, remaining time = 198.9
Chair  31 meets chair 120, remaining time =  65.7
Chair 100 meets chair  53, remaining time =  83.7
```

(Space for answering Question 2, Part c.)

A solution by our first team of 2000.
Give some marks for any evidence of a reasonable idea expressed in whatever notation.

```
#include <stdio.h>
#include <stdlib.h>

double N,T;
double d;
int a,b;

int main() {

  scanf(" %lf %lf ",&N,&T);
  printf("N = %3.0f, T = %5.1f\n",N,T);
  while(scanf(" %d %d ",&a,&b) == 2) {
    if (a < b) {
      d = (N - (b-a) - 0.5)/2.0;
    } else {
      d = (a-b-0.5)/2.0;
    }
    printf("Chair %3d meets chair %3d, remaining time = %5.1f\n",
            a,b,T/((N-1)/2)*d);
  }
  return 0;
}
```

Question 3: Points on Polygon's Boundary

In geometry a *simple polygon* is defined as a flat shape, consisting of straight, non-intersecting, line segments that are joined pair-wise to form a closed path. If the sides intersect then the polygon is not simple. The qualifier "simple" is frequently omitted, with the above definition being understood to define a polygon in general.

Mathematicians typically use "polygon" to refer only to the shape made up by the line segments, not the enclosed region, however some may use "polygon" to refer to a plane figure that is bounded by a closed path, composed of a finite sequence of straight line segments (i.e., by a closed polygonal chain). The closed chain of adjacent line segments is then called *polygon's boundary*.

The definition given above ensures the following properties:

- Polygon's boundary encloses a region (called its interior) and the region always has a measurable area.
- The line segments that make-up a polygon (called sides or edges) meet only at their endpoints, called vertices (singular: vertex) or less formally "corners".
- Exactly two edges meet at every vertex.
- The number of edges always equals the number of vertices.

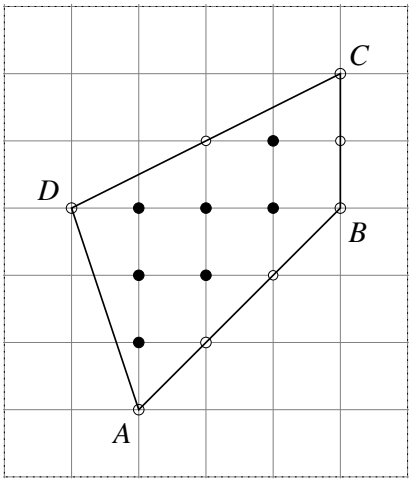

Figure 2: A sample polygon on a grid.

Given a simple polygon constructed on a grid of equal-distanced points (i.e., points with integer coordinates) such that all the polygon's vertices are grid points, Pick's theorem provides a simple formula for calculating the area A of this polygon in terms of the number i of grid points in the interior of the polygon and the number b of grid points placed on the polygon's boundary:

$$A = i + \frac{b}{2} - 1.$$

In the quadrilateral $ABCD$ of Figure 2 we have $i = 7, b = 8$ and by simple inspection you can check that $A = 10$.

Consider a problem which is of big interest to forresters: *how to find the number of grid points in the interior of a grid polygon?* We can solve the problem using Pick's theorem if we know the area of the polygon and the number of grid points on the polygon's boundary. It turns out, that the area of a simple polygon can be easily computed by methods of computational geometry but essentially we will not be concerned with computing areas of polygons here. We will focus on the remaining problem of how to find the number of grid points on the polygon's boundary.

Question 3, Part a:

The quadrilateral of Figure 2 can be seen as drawn on the grid with x, y-coordinates and the point $(0, 0)$ being the lowest left corner of the shown grid. We identify the labeled points with their planar coordinates and thus obtain $A = (2, 1)$, $B = (5, 4)$, $C = (5, 6))$, and $D = (1, 4)$.

Now imagine, that each coordinate of every point has been doubled and we obtain a new quadrilateral $A'B'C'D'$ with $A' = (4, 2)$, $B' = (10, 8)$, $C' = (10, 12)$, and $D' = (2, 8)$.

What is the area of the quadrilateral $A'B'C'D'$?

Answer. Since the linear dimensions of $ABCD$ increase by factor of 2, the area increases by factor of 2^2. The area of $A'B'C'D'$ is $2^2 \cdot 10 = 40$.

How many grid points are on the boundary of $A'B'C'D'$?

Answer. 16, which is twice as many as boundary points in $ABCD$.

How many grid points are in the interior of $A'B'C'D'$?

Answer. The answer is obtained from Pick's formula which for quadrilateral $A'B'C'D'$ with area 40 and $b = 16$ gives $40 = i + \frac{16}{2} - 1$ resulting in $i = 33$.

Question 3, Part b:

Given points $A = (x_a, y_a)$ and $B = (x_b, y_b)$ on an integer grid. Write a piece of code computing how many grid points are located on the line segment AB.

Answer. $1 + \gcd(|x_a - x_b|, |y_a - y_b|)$

Give marks for noticing that gcd enters the picture here. Students may not have met gcd yet but some of them can come up with the right idea.

Question 3, Part c:

Write a program for finding the number of grid points located on the boundary of a polygon whose vertices are located on grid points. Note that by definition, the vertices of such a polygon are located on the polygon's boundary.

The input consists of a description of a grid polygon, given as a sequence of $2*N+1$ numbers. The first number is N itself and then N pairs of numbers follow. Each pair of numbers gives the grid coordinates of a polygon vertex, and adjacent pairs of numbers define adjacent vertices. The last and the first vertex listed are adjacent.

The output of your program contains one number: the number of grid points on the polygon boundary. The sample input and output given below concerns the polygon of Figure 2.

Sample Input

```
4    2 1    5 4    5 6    1 4
```

Output for Sample Input

```
8
```

The solution: one loop through input vertices (note the need for wraparound) calling the solution to Part b at each iteration.

Give marks for whatever idea like the expected.

Question 4: Average Speed

You have bought a car in order to drive from Red Deer to a big city. The odometer on the car is broken, so you cannot measure distance. But the speedometer and cruise control both work, so the car can maintain a constant speed which can be adjusted from time to time in response to speed limits, traffic jams, and border queues. You have a stopwatch and note the elapsed time every time the speed changes. From time to time you wonder, *How far have I come?*. To solve this problem you must write a program to run on your laptop computer in the passenger seat.

The input contains several lines

- Each speed change is indicated by a line specifying the elapsed time since the beginning of the trip (hh:mm:ss in a 24 hour clock), followed by the new speed in km/h.
- Each query is indicated by a line containing only the elapsed time.

At the outset of the trip the car is stationary. Elapsed times are given in non-decreasing order and there is at most one speed change at any given time. Note the cycling nature of a 24 hour clock. The time elapsed between two consecutive time readings is shorter than 24 hours.

For each query in the input, you should print a line giving the time and the distance traveled, in the format below.

Sample Input

```
00:00:01 100
00:15:01
00:30:01
01:00:01 50
03:00:01
03:00:05 140
```

Output for Sample Input

```
00:15:01 25.00 km
00:30:01 50.00 km
03:00:01 200.00 km
```

Question 4, Part a:

What should be the output of the program for the following input?

```
16:00:00 80
17:15:00
17:30:00 60
18:00:00
```

Answer.

```
17:15:00 100.00 km
18:00:00 150.00 km
```

Question 4, Part b:

What should be the output of the program for the following input?

```
16:00:00 10
22:30:00
23:30:00 40
01:30:00
02:45:00
03:00:00 0
23:00:00 20
01:30:00
```

Answer.

```
22:30:00 65.00 km
01:30:00 155.00 km
02:45:00 205.00 km
01:30:00 265.00 km
```

Question 4, Part c:

Write a program that reads input and produces output as described above.

```
#include <stdio.h>

main(){
  int hh, mm, ss, speed=0, newspeed, i, j, k, n, time = 0, now, now1;
   char buf[10000];
   double dist = 0;

   while (gets(buf)) {
      n = sscanf(buf,"%d:%d:%d %d",&hh,&mm,&ss,&newspeed);
      now = hh*3600 + mm*60 + ss;
      now1 = now + (now < time ? 86400 : 0);
      dist += (now1 - time) / 3600.0 * speed;
      time = now;
      if (n == 3) printf("%02d:%02d:%02d %0.2lf km\n",hh,mm,ss,dist);
      else if (n == 4) speed = newspeed;
      else printf("oops!\n");
   }
}
```

Give some marks for any evidence of a reasonable idea expressed in whatever notation.