British Informatics Olympiad

Time allowed: 3 hours

The 2014 British Informatics Olympiad

Instructions

You should write a program for part (a) of each question, and produce written answers to the remaining parts. Programs may be used to help produce the answers to these written questions but are not always necessary.

You may use a calculator and the on-line help that your programming language provides. You should have a pen, some blank paper, and an empty USB stick (or other storage device) on which to save your programs. You must not use any other material such as files on a computer network, books or other written information. You may not communicate with anyone, other than the person invigilating this paper.

Mark the first page of your written answers with your name, age in years and school/college. Number all pages in order if you use more than one sheet. All of your computer programs should display your name and school/college when they are run, and the storage device you use to submit the programs should also show your name and school/college.

For your programs to be marked, the source code must be saved, along with executables if your language includes a compiler; this includes programs used to help answer written questions. You must clearly indicate the name given to each program on your answer sheet(s).

Sample runs are given for parts 1(a), 2(a) and 3(a). **`Bold text`** indicates output from the program, and `normal text` shows data that has been entered. Where multiple items of input appear on the same line they should be separated by a single space. The output format of your programs should follow the 'sample run' examples. Your programs should take less than *2 seconds* of processing time for each test.

Attempt as many questions as you can. Do not worry if you are unable to finish this paper in the time available. Marks allocated to each part of a question are shown in square brackets next to the questions. Partial solutions (such as programs that only get some of the test cases correct within the time limit, or partly completed written answers) may get partial marks. Questions can be answered in any order, and you may answer the written questions without attempting the programming parts.

Hints

- If you can only see how to solve part of a problem it is worth writing a program that solves that part. We want to give you marks and questions are evaluated using multiple tests of differing difficulty. ***Remember, partial solutions may get partial marks.***
- Question 2 is an implementation challenge and question 3 is a problem solving challenge.
- Most written questions can be solved by hand without solving the programming parts.
- Do not forget to indicate the name given to your programs on your answer sheet(s).

Question 1: ***Lucky Numbers***

The *lucky numbers* are produced by taking a list of all the odd numbers and systematically removing some of the numbers. Our first lucky number is 1. We now repeatedly take the next highest number *n* that is still in the list, mark it as a lucky number and then remove every *n*th number from the list.

For example, with lucky numbers underlined as we progress:

- Initially we have: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, ...
- The next lucky number is 3, giving us: 1, 3, 7, 9, 13, 15, 19, 21, 25, 27, 31, ...
- Next is 7: 1, 3, 7, 9, 13, 15, 21, 25, 27, 31, ...
- Next is 9: 1, 3, 7, 9, 13, 15, 21, 25, 31, ...

1(a) [25 marks]

Write a program which reads in a single integer between 2 and 10,000 inclusive.

You should output two numbers. Firstly the closest lucky number that is *less* than the input, followed by the closest lucky number that is *greater* than the input.

Sample run

```
5
3 7
```

1(b) [2 marks]

How many numbers less than 100 are lucky?

1(c) [3 marks]

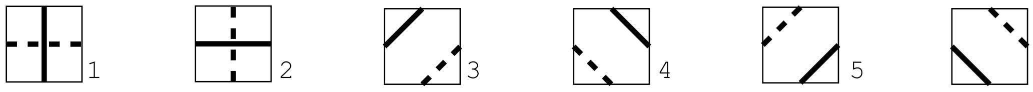
Suppose that you are told the 1,000,000,000th lucky number is *X*. A program that *correctly* solves 1(a) (on inputs of any size >2) is run on all numbers from 2 to *X* inclusive. How many *different* results will the program produce?

(NB: A result is a pair of numbers produced by the program; e.g. **3 7** is a single result.)

Question 2: ***Loops***

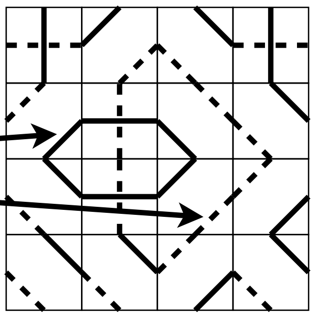
Red and Green are playing a game which consists of placing square tiles on a grid and trying to form *loops* of their own colour. Each tile incorporates a red line and a green line, touching different edges of the tile.

There are six different types of tile that are used in the game. Red and green lines are shown by solid and dashed lines respectively. The numbers by the tiles will be used when inputting grids.



Tiles are placed adjacent to each other on the grid. A line meeting another line of the same colour — across a shared edge between tiles — is treated as a single continuous line across multiple tiles. If the lines are different colours the lines are treated as being separate. A loop is a line that begins on one tile and continues along additional tiles until it rejoins itself on the first tile.

At the end of the game each player scores a single point for each tile that contains part of a line forming a loop of their colour. A tile can contribute towards the score of both players.



In the example to the right there is a single loop; the numbers indicate how the grid will be input.

- This is a red (solid) loop covering six tiles.
- This is not a loop as the green (dashed) line does not rejoin itself.

The red player has scored 6 points and the green player 0.

```
1341
5264
4235
4454
```

2(a) [25 marks]

Write a program that reads in a grid and outputs the score for each player.

Your program should first read in a single integer ($2 \leq n \leq 6$) indicating the size of the (square) grid. You should then read in n lines representing the rows (starting at the top), each of which will contain n digits representing the tiles on the grid in order.

You should output two values: the score for the red played followed by the score for the green player.

Sample run

```
4
1 3 4 1
5 2 6 4
4 2 3 5
4 4 5 4

6 0
```

2(b) [2 marks]

A single tile is changed on the example grid so that Red no longer has more points than Green. How many possible changed grids are there?

2(c) [3 marks]

In how many ways can Red score 16 points on a 4 × 4 grid?

2(d) [5 marks]

Red and Green have just played the game on a 500 × 500 grid. Is it possible for Red to have scored just 1 point more than Green? Justify your answer.

Question 3: ***Increasing Passwords***

A password scheme accepts passwords that contain combinations of upper-case letters and digits. The scheme restricts passwords to ones in which letters appear in alphabetical order. The digits 0, ..., 9 are treated as coming after Z in alphabetical order; lower digits are treated as coming before higher digits alphabetically. Passwords must contain at least one character (letter or digit) and no duplicate characters are allowed.

For example:
- `BIO14` is a valid password;
- `OLYMPIAD` is not a valid password (letters are not in alphabetical order).

The passwords have been put into an ordered list. If two passwords contain a different number of characters, the password with the fewer characters comes first. If they both have the same number of characters then they are sorted alphabetically.

The ordered list of passwords looks like this: `A`, `B`, ..., `Z`, `0`, `1`, ... `9`, `AB`, `AC`, ..., `A9`, `BC`, ...

3(a) [25 marks]

Write a program to determine the n^{th} password in the ordered list.

Your program should read in a single integer, ($1 \leq n \leq 1,000,000,000$). You should output the string which represents the n^{th} password.

Sample run

```
37
AB
```

3(b) [2 marks]

In what order do the following passwords appear in the list: `BIO`, `BIO14`, `NTU`, `14`, `ABCDE` ?

3(c) [3 marks]

How many passwords does the scheme accept?

3(d) [5 marks]

How many *pairs* of adjacent passwords (in the ordered list) contain all 36 characters between them, with each character appearing in only one of the passwords? Justify your answer.

Total Marks: 100 End of BIO 2014 Round One paper