British Informatics Olympiad

Time allowed: 3 hours

Sponsored by Microsoft Studios

The 2013 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: *Watching the Clock*

Two *clocks*, which show the time in hours and minutes using the 24 hour clock, are running at different speeds. Each clock is an exact number of minutes per hour fast. Both clocks start showing the same time (00:00) and are checked regularly every hour (starting after one hour) according to an accurate timekeeper. What time will the two clocks show on the first occasion when they are checked and show the same time?

NB: For this question we *only* care about the clocks matching when they are checked.

For example, suppose the first clock runs 1 minute fast (per hour) and the second clock runs 31 minutes fast (per hour).

- When the clocks are first checked after one hour, the first clock will show 01:01 and the second clock will show 01:31;
- When the clocks are checked after two hours, they will show 02:02 and 03:02;
- After 48 hours the clocks will both show 00:48.

1(a) [25 marks]

Write a program which reads in a two integers, each between 0 and 50,000 inclusive, indicating the number of minutes fast (per hour) of the first and second clock respectively.

You should output the time shown on the clocks when they first match. Both the hour and the minutes should be displayed with two digits.

Sample run 1

```
1 31
00:48
```

1(b) [3 marks]

Suppose the first clock is accurate and the clocks do *not* show 00:00 when they first match. The second clock is less than 20 minutes fast (per hour). How many minutes fast is the second clock? Write out all the possible answers.

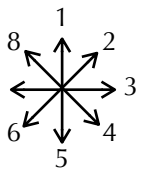
1(c) [4 marks]

Suppose the two clocks can be *any* number of full minutes fast per hour. What is the largest number of hours that can pass before the clocks first match?

Question 2: *Neutron*

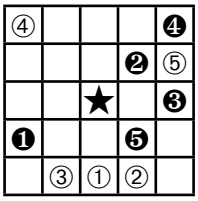
The game of *neutron*[1] is played on a 5×5 board between two players. The players have five pieces each and there is an additional neutron piece on the board. Each space on the board is either empty or contains a single piece. The object of the game is to manoeuvre the neutron to a space on either the top or bottom row of the board.

For convenience we will denote directions on the board as follows:



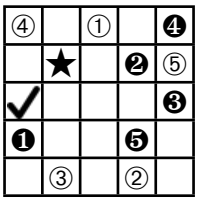
Players take alternate turns, each of which consists of first moving the neutron piece and then moving one of their own pieces. A piece is moved by sliding it *as far as possible* either horizontally, vertically or diagonally. If the neutron is moved to the bottom row the first player wins, if it reaches the top row the second player wins; it does not matter on whose turn the neutron was moved. If a player is unable to complete their turn (by being unable to move the neutron or subsequently one of their own pieces) the other player wins.

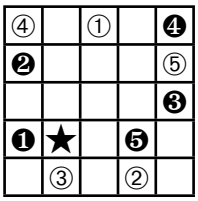
For example, consider the board on the right. The first player's pieces are denoted in white and the second player's pieces in black. The neutron is denoted by a star.



Suppose it is the first player's turn. The neutron has to be moved first. It can move in direction 1 (in which case it reaches the top of the board and the game ends with the second player winning) or direction 6 (reaching the bottom and winning the game for the first player) and cannot move in directions 2 or 4 (being blocked by ❷ and ❺ respectively). In can also move in directions 3, 5, 7 and 8 in which case the first player then needs to move one of their pieces.

If the first player decided to move the neutron in direction 8, followed by ① in direction 1, the board would look as it does on the left. If the second player then moved the neutron in direction 5, followed by the ❷ in direction 7, the board would look as it does in the right. Note that, in each case, pieces are moved as far as possible.



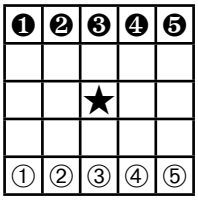


For this question each player will use the following strategy:

- Each player has a fixed order in which they play their pieces (as the game progress). Once the last piece in their order has been played, they start again at the beginning of the order. E.g. if the player's order is 5, 4, 3, 2, 1 they will use (their) piece 5 on their first turn, 4 on their second, …, 5 on their sixth turn, etc…
- At the start of a turn, if they are able to move the neutron and immediately win the game *for themselves* the neutron is moved in the first such direction (from 1 to 8).
- If the only possible moves for the neutron will immediately win the game for their opponent, the neutron is moved in the first such direction (from 1 to 8).
- If the previous two conditions are not met, the player finds a way to play the next piece in their order. First the neutron is moved in the first direction (from 1 to 8) which does not end the game or prevent their chosen piece from moving, then the chosen piece is moved in the first direction (from 1 to 8) in which it can move.

Your program will not be asked to deal with any cases where there is no valid turn for the neutron and player's chosen piece.

At the start of the game the neutron is in the centre of the board. The first player's pieces occupy the bottom row and the second player's pieces occupy the top row. Pieces are in increasing order from left to right.



[1] The actual game of *neutron* is slightly different to the description given here; you should only implement the rules detailed in the question.

2(a) [26 marks]

Write a program that plays neutron according to the given strategy.

Your program should read in two lines of input. The first line of input will contain the digits `1`, `2`, `3`, `4` and `5` in an order that represents the first player's fixed order. The second line of input will contain, in a similar format, the second player's fixed order.

You should output the layout of the board at three moments during the game. The first layout should be after the first player's first turn; the second layout after the second player's first turn; the final layout after the game has terminated. All possible inputs to this question lead to a game that finishes.

Your board should use an `F` for pieces belonging to the first player, an `S` for pieces belonging to the second player, a `*` for the neutron and a `.` for an unoccupied space.

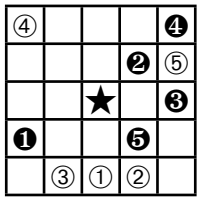
The three board layouts are marked independently and you can score marks if only one or two of your layouts are correct.

Sample run 1

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

SSSSS
F.*..
.....
.....
.FFFF

.SSSS
F...*
.....
...S.
.FFFF

..SSS
FF...
.....
...SS
.*FFF
```

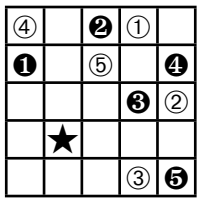
2(b) [3 marks]

Suppose the layout of the board is as follows and that it is the start of the second (Black) player's turn. Ignoring the strategy, in how many ways can the player take their turn?



2(c) [4 marks]

Suppose the layout of the board is as follows at the beginning of a turn and that you do not know whose turn it is. Ignoring the strategy, how many different layouts could the board have been in at the start of the previous turn?



Question 3: *Unlock*

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V	W	X	Y

A security system consists of a 5×5 keypad of buttons. Each button has a light which can be either *off*, *dim* or *bright*. When a button is pressed its light moves to the next state (*off*→*dim*, *dim*→*bright* and *bright*→*off*) as do the lights on any buttons touching it horizontally and vertically. The system is *unlocked* when all the lights are *off*.

The current lighting can be represented by a string, in alphabetical order, where a lowercase letter indicates the corresponding button is *dim* and an uppercase letter indicates it is *bright*. Lights that are *off* do not have a letter in the string.

For example, if all the buttons are *off* and button R is pressed, the lighting is `mqrsw`. If T is now pressed it becomes `moqrStwy`. Pressing S will make it `mnoqRTwxy`.

3(a) [24 marks]

Write a program to determine a sequence of button presses to *unlock* the system.

Your program should read in a string of between 1 and 12 *distinct* letters (inclusive) in alphabetical order, representing the current lighting of the security system.

You should output a string (in alphabetical order) that indicates a sequence of buttons which can be pressed to unlock the system, or output `IMPOSSIBLE` if it cannot be unlocked. A button can be pressed at most twice; a lowercase letter in your string will indicate that the corresponding button should be pressed once and an uppercase letter indicating that it should be pressed twice.

If there are multiple solutions you are only required to print out a single solution.

Sample run

```
mnoqRTwxy
RST
```

Alternative answer

```
mnoqRTwxy
aCeFhJprSUwY
```

3(b) [2 marks]

What is the lighting of the system if the system is unlocked and then B is pressed, followed by I and O? (Each button being pressed once.)

3(c) [4 marks]

Starting from an unlocked system, in how many ways can three different buttons be pressed once each, in alphabetical order, so that no light becomes bright?

3(d) [5 marks]

Suppose that the system has a configuration of lighting and that a particular sequence of button presses will unlock the system. Is it possible for the same sequence of button presses to unlock the system when it is in a different configuration? Justify your answer.

Total Marks: 100 End of BIO 2013 Round One paper