British Informatics Olympiad

Time allowed: 3 hours

The 2019 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 *1 second* 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: *Palindromic Numbers*

A *palindromic number* is one that is the same when its digits are reversed.

For example:
- 98789 is a palindrome;
- 12344321 is a palindrome;
- 12345 is not a palindrome as it becomes 54321 when reversed.

Except for 0, a palindromic number's leftmost digit must be non-zero.

1(a) [25 marks]

Write a program that reads in a positive integer of up to 20 digits.

You should output the smallest palindromic number that is higher than the input.

Sample run 1

```
17
22
```

Sample run 2

```
343
353
```

1(b) [2 marks]

What is the largest difference between a palindromic number (of up to 20 digits) and the next highest palindromic number?

1(c) [4 marks]

How many integers are there, between 1 and 99999 inclusive, that are *not* the sum of two palindromic numbers?

Question 2: *Trail*

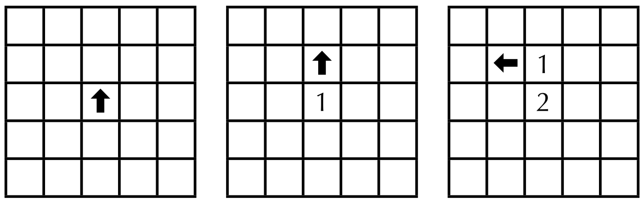
An *explorer* is moving across a rectangular grid of squares, leaving behind a *trail*. Keen to experience the unknown, they only move into squares that do not contain part of the trail. The trail fades over time and the explorer can re-enter a square once its trail has disappeared, although they will leave behind a new trail.

If the trail fades after *n* moves, it means that the trail in a square will disappear once the explorer has finished moving *n* times.

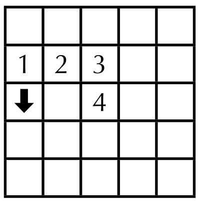
The explorer is following a set of instructions. If the instructions indicate *left* or *right* they will turn 90 degrees in the corresponding direction, otherwise they will stay facing *forward*. The explorer will then move to the adjacent square in the direction they are facing, so long as it does not contain the trail. If this is not possible, they will turn 90 degrees to the *right* and try to move. This will be repeated until they have moved or tried all four directions.

Each entry in the explorer's instructions is either `L` (*left*), `R` (*right*) or `F` (*forward*). The explorer works through the entries in order, restarting at the beginning once they have all been used.

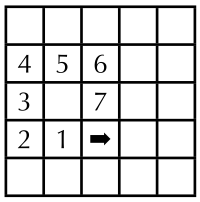
For example, suppose the trail fades after the explorer has moved 8 times and their instructions are `FL`. (In the diagrams, the arrow will indicate the explorer and the direction they are facing, with the numbers indicating the age of the trail.)



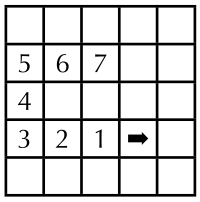
The explorer first moves forward, and then moves to their left. After this second move they are facing in a new direction and the trail covers two squares.



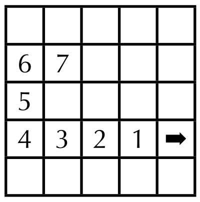
The explorer will now restart their instructions. Another move forward in the direction they are facing, followed by a move to the left.



After another three moves the explorer's next instruction is a *left*. They turn to the left but are facing a square that contains the oldest part of the trail. They therefore turn to the right …



… and move into the empty square. They have now moved 8 times since the oldest part of the trail was formed, so it disappears.



They continue with the next instruction, which is *forward*.

2(a) [24 marks]

Write a program that tracks the explorer.

Your program should first input an integer *t* (1 ≤ *t* ≤ 100) indicating the number of moves for the trail to disappear, followed by a word *i* of between 1 and 10 upper case letters (each `L`, `R` or `F`) indicating the explorer's instructions, followed by an integer *m* (1 ≤ *m* ≤ 10000) indicating how many moves the explorer makes.

If the explorer is ever unable to make a move they will stop and not attempt to make any more moves.

You should output the co-ordinates of the explorer after making their moves.

The explorer starts at (0,0) facing (0,1). To their left is (-1,0) and to their right is (1,0).

Sample run

```
8 FL 9
(2,-1)
```

2(b) [2 marks]

The introduction to this question gave the diagram after `8 FL 9`. Give the diagram after `8 FL 16`.

2(c) [3 marks]

If the trail never disappears and the explorer's instructions are `L`, how many moves are required for every square at (*x*,*y*) to be visited, where -10 ≤ *x* ≤ 10 and -10 ≤ *y* ≤ 10?

2(d) [5 marks]

The explorer's instructions are `LLRFFF` and they are able to move indefinitely. What is the largest possible number of moves before the trail disappears?

Question 3: *Block-chain*

A set of children's blocks, each illustrated with a single different letter, have been chained together in a line. They have been arranged so that it is not possible to find three (not necessarily adjacent) letters, from left to right, that are in alphabetical order.

For example, if there are four blocks (A, B, C and D) the possible block-chains are:

```
ADCB    BADC    BDAC    BDCA    CADB
CBAD    CBDA    CDAB    CDBA    DACB
DBAC    DBCA    DCAB    DCBA
```

3(a) [24 marks]

Write a program that enumerates block-chains.

Your program should input a single integer *l* (1 ≤ *l* ≤ 19) indicating that the blocks are illustrated with the first *l* letters of the alphabet, followed by a word *p* of between 1 and *l* uppercase letters indicating (in order) the leftmost letters of the block chain. *p* will only contain letters take from the first *l* letters of the alphabet and will not contain any duplicates.

You should output a single integer giving the number of possible block-chains that begin with *p*.

Sample run

```
4 CB
2
```

3(b) [2 marks]

List the valid block-chains containing the blocks `B`, `I` and `O`.

3(c) [4 marks]

A block-chain containing the first *n* (1 ≤ *n* ≤ 13) letters of the alphabet is attached to the left of a block-chain containing the last *m* (1 ≤ *m* ≤ 13) letters of the alphabet, forming a new block-chain. What were the original block-chains? Justify your answer.

3(d) [5 marks]

Suppose all the valid block-chains containing the first 19 letters of the alphabet are sorted into alphabetical order. Which one comes first? Which one comes 1,000,000,000th?

Total Marks: 100 End of BIO 2019 Round One paper