British Informatics Olympiad

Time allowed: 3 hours

The 2007 British Informatics Olympiad

LIONHEAD STUDIOS

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 a blank floppy disk (or other storage device) on which to save your programs. You must not use any other material such as disks, 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. 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: ***Cards***

A card game is played with a deck of forty cards, containing each of the numbers from 1 to 10 exactly four times. The game is scored according to the following two rules: a point is given for each pair of cards with identical numerical values and for any group of cards whose numerical values sum to 15.

For example, the set of cards *8, 8 and 8* is worth three points since there are three different pairs of cards with identical numerical values. The set *10, 5, 2 and 3* is worth two points since there are two groups of cards whose numerical values sum to 15.

1(a) [24 marks]

Write a program which inputs 5 numbers (each of which will between 1 and 10 inclusive) indicating the numerical values on 5 different cards. Your program should print out the number of points these 5 cards are worth and then terminate.

Sample run

```
3 3 3 2 10
6
```

For the following parts remember that you are allowed to use each number from 1 to 10 at most *four* ***times.***

1(b) [2 marks]

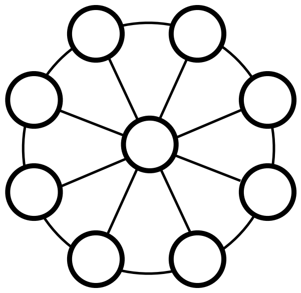
Give a set of five cards that score 0 points.

1(c) [4 marks]

How many different sets of five cards are there where the sum of the values of all five cards is exactly 15? [You should assume that order does not matter and cards are only different if they contain different values; e.g. *1 2 3 4 5* is the same as *5 4 3 2 1*.]

Question 2: ***Mu Torere***

The Maori strategy game of *Mu Torere* is played on a circular board with eight positions (called *kawai)* on the circumference and one position (called *putahi*) on the centre. It is a two-player game, where each player has four markers. Each position is either empty, or contains a single marker.



On a player's turn they can move one of their markers to an adjacent position. Adjacent positions are those joined directly by a line in the above diagram; i.e. the *putahi* is adjacent to all the *kawai*, and each *kawai* is also adjacent to the *kawai* immediately clockwise and anti-clockwise. The only exception is that a player cannot move their marker from a *kawai* if the two adjacent *kawai* also contain markers that belong to them.

Play alternates between the two players. A player loses the game if they are unable to move on their turn.

We will represent this board by a string of 9 characters; the first will represent the *putahi* and the last eight will represent the *kawai* (starting at the left of the two top positions and running clockwise around the board). The character `O` will represent a position containing a marker belonging to the first player, `X` will represent a position containing a marker belonging to the second player and `E` will represent an empty position.

For example, traditionally the game starts with both players having four adjacent *kawai*. We can represent this starting layout by the string `EOOOOXXXX`.

A simple strategy is for a player to choose their moves based on the following rules (where *leftmost* means *leftmost on our board representation*):

1. If there is a move which means my opponent will then lose, this move is played. If several such moves exist, choose the one that uses the leftmost of my markers.
2. If the first rule does not indicate which move to take and there are moves, after which my opponent will be able to make a move meaning that I will then lose, those moves are to be avoided (by moving the leftmost of my markers that avoids these moves). If it is not possible to avoid such a move, move the leftmost of my markers.
3. If the previous rules do not indicate which move to take, move the leftmost of my markers.

There are marks available for programs which are only able to deal with rules 1 and 3.

2(a) [24 marks]

Write a program that plays Mu Torere using the simple strategy for both players.

Your program should first read in a single line, containing 9 characters representing a starting layout. This will be a valid layout where each player will have exactly four markers.

Until your program terminates you should repeatedly wait for input, and then:

- If you receive the letter `n`, you should play the next turn of the game unless the current player has lost because there are no valid moves.
- If you receive the letter `r`, you should try to play the rest of the game, until one player has won or you believe that the game will never finish.
- Ignore any other input.

After the `n` command you should output the board layout. If the game has been finished you should also output **`Player 1 wins`** (for a first player win) or **`Player 2 wins`** (for a second player win) and then terminate. You should do the same thing after the `r` command, unless you believe the game will never finish, in which case you do not need to output a board layout and should just output **`Draw`** and then terminate.

Sample run

```
EOOOOXXXX
n

OEOOOXXXX

r

OXOEOXXXO
Player 1 wins
```

2(b) [3 marks]

If the board layout is `XEOXXXOOO` and it is the first player's turn, what possible moves can be made? You should show the board layouts after the valid moves. Which of these moves will be made if the simple strategy is followed?

2(c) [3 marks]

Do any layouts exist which, depending on whose turn it is to play, are winning layouts for both players? Justify your answer.

2(d) [5 marks]

Our board representation made two choices which do not affect the moves available to players during the game. It chose the starting position on the board (the left of the top two positions) and the direction (clockwise).

We can treat `EOOOOXXXX` and `EXOOOOXXX` as equivalent layouts, since the same board could lead to the two strings if we just change our starting position. We can also treat `EOOOXOXXX` and `EOXXXOXOO` as equivalent layouts, since the same board could lead to the two strings if we just change our direction from clockwise to anti-clockwise.

If we say two layouts are the same if, by choosing the starting position and direction, they could produce the same string, how many different layouts exist? [You might find it easier to consider the number of different layouts in the cases where the *putahi* is empty, contains one of the first player's markers or contains one belonging to the second player.]

Question 3: ***String rewriting***

A *string rewriting rule* is a simple instruction for changing a string. It says that every occurrence of a given character should be changed into one or more other characters. In this question we will use five rules:

- `A` should be changed into `B`
- `B` should be changed into `AB`
- `C` should be changed into `CD`
- `D` should be changed into `DC`
- `E` should be changed into `EE`

In a single step, we apply these rules simultaneously and on every character in a string. For example, suppose we have the string `DECADE`, after one step we have the string `DCEECDBDCEE`, and after a second step we have the string `DCCDEEEECDDCABDCCDEEEE`.

3(a) [25 marks]

Write a program to determine the number of each type of character in part of a string after a given number of steps.

Your program should input two lines, the first containing a three letter string (where each letter will be `A`, `B`, `C`, `D` or `E`), and the second containing two integers *s* ($1 \leq s \leq 29$) followed by *p* ($1 \leq p \leq 2^{31}$). The first integer *s* indicating the number of rewriting steps which should take place and the second *p* indicating a position.

Sample run

```
DEC
2 10
0 0 3 3 4
```

You should output 5 numbers, the number of `A`s (then `B`s, `C`s, `D`s and `E`s) occurring in the first *p* (inclusive) positions of the string after *s* string rewriting steps.

The value *p* will never be larger than the number of characters in the string after *s* steps.

3(b) [2 marks]

How many letters will there be if you start from the string `C` and apply 8 steps? How about if you start from the string A?

3(c) [5 marks]

Given any starting position and number of steps (>1) is it ever possible to have three adjacent `C`s in the resulting string? Justify your answer.

3(d) [3 marks]

For some strings we can go backwards; i.e. we can find a previous string which leads to it after applying a step. With the rules given in this question, when we are able to move backwards there is always a *single* possible previous string.

For some sets of rules the situation might be ambiguous; when we try to move backwards there many be more than one possible previous string.

Find the smallest set of rules (both in terms of the number of rules and the number of characters used in those rules) which makes things ambiguous when moving backwards. Your rules must allow growth, so given any starting string it must be possible to apply enough steps to make the length of that string increase.

Total Marks: 100 End of BIO 2007 Round One paper