British Informatics Olympiad

Time allowed: 3 hours

LIONHEAD STUDIOS

The 2009 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 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: *Digit Words*

A *digit word* is a word where, after possibly removing some letters, you are left with one of the single digits: ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT or NINE.

For example:
- ~~B~~O~~U~~N~~C~~E and ~~ANN~~O~~U~~N~~C~~E are digit words, since they contain the digit ONE.
- ENCODE is not a digit word, even though it contains an O, N and E, since they are not in order.

1(a) [24 marks]

Write a program which reads in a single upper-case word (with at most fifteen letters) and determines if it is a *digit word*.

If the word is not a digit word you should output the word **NO**. If the word is a digit word you should output the digit it contains, as a number.

You will not be given any words which contain more than one digit.

Sample run 1

```
BOUNCE
1
```

Sample run 2

```
ENCODE
NO
```

1(b) [2 marks]

In how many different ways can letters be removed from TWOTWOTWO to leave the digit TWO?

1(c) [4 marks]

The made-up digit word TWFOUR contains the digit TWO and the digit FOUR. What is the length of the shortest made-up word which contains all the digits ONE to FIVE? How about ONE to NINE?

Question 2: ***Puzzle Game***

In some *puzzle games* points are scored by joining together pieces of the same colour. These pieces are then removed, new pieces are added, and the game continues until no more pieces can be removed. Your task in this question is to model a simple game.

Our game will be played on a 4 by 4 grid. The pieces, each of which fits in a single space on the grid, will be red, green or blue - to be indicated by `R`, `G` and `B` respectively.

When two or more adjacent pieces have the same colour they form a *block*. Pieces are adjacent if they touch horizontally or vertically, but not diagonally. All blocks are removed simultaneously from the grid and all the pieces that remain in the grid drop down to fill in any empty space.

```
RRGB              GB             G 
GRBG  -------->  G BG  -------->  B 
RRGB               G            G GB
GBRB             GBR            GBRG
```

The number of points scored is equal to the product of the sizes of the removed blocks. In the above example, a red block of 5 pieces and a blue block of 2 pieces were removed. This scores 10 points.

New pieces then drop down from above the grid to fill in any empty spaces. A *round* in the game is completed when these new pieces have appeared on the grid.

In our simple game, we will assume that the store of pieces above the grid is a repetition of the pattern in the original grid.

The following example shows the first three rounds in a game. The pieces above the grid show the next few pieces that are due to drop down from above to fill in the empty spaces. Observe that we ignore pieces above the grid when finding blocks of adjacent pieces; i.e. they only count when they have dropped down onto the grid.

```
RRGB             RRGB             RRRB
GRBG             GRBB             GRGB
RRGB             RBGB             RBBG
GBRB             GRRG             GRGB
RRGB             RRGB             RRRB
GRBG             GRBB             GRGB
RRGB             RBGB             RBBG
GBRB             GRRG             GRGB

RRGB             RRGB             RRRB
GRBG  -------->  GRBB  -------->  GRGB
RRGB             GBGB             RBGG
GBRB             GBRG             GRRG
```

2(a) [24 marks]

Write a program to play the puzzle game.

Your program should first read in four lines, giving the grid's rows in order; the first line being the top row of the grid. Each line will consist of four characters, each an `R`, `G` or `B`, giving a row from left to right.

Until your program terminates you should repeatedly input an integer, and then:

- If you receive a positive integer *n* (1 ≤ *n* ≤ 100) you should play *n* rounds of the game.
- If you receive the number 0 your program should terminate immediately.
- Ignore any other input.

After each positive integer you should output the board along with the total number of points that have been scored so far. If, while playing a round, there are no blocks to be removed, you should just output **`GAME OVER`** along with the final score, and then terminate.

Sample run

```
RRGB
GRBG
RRGB
GBRB
2

RRRB
GRGB
RBGG
GRRG

82

0
```

2(b) [3 marks]

Draw an example grid which will score 105 points in one round.

2(c) [4 marks]

What is the maximum score that can be achieved in a single round?

2(d) [4 marks]

Suppose that you are given the shape and the location of all the blocks on the grid, and you now need to place pieces on the grid to match those blocks. Will it always be possible to place the pieces? Justify your answer.

Question 3: ***Child's Play***

A toy set contains blocks showing the numbers from 1 to 9. There are plenty of blocks showing each number and blocks showing the same number are otherwise indistinguishable. We can consider the number of different ways of arranging the blocks so that the displayed numbers add up to a fixed sum.

For example, suppose the sum is 4. There are 8 different arrangements:

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

These have been listed in *dictionary* order. Just like in a dictionary, where all the words beginning with an 'a' come before those that begin with a 'b', and all those beginning 'aa' come before those beginning 'ab', we have listed the arrangements beginning with a 1 before those with a 2, and those beginning with a 1 then another 1, before those beginning with a 1 then a 2.

3(a) [23 marks]

Write a program to determine the n^{th} way of arranging the blocks to make a fixed sum.

Your program should input integers s ($1 \leq s \leq 32$) then n ($1 \leq n < 2^{31}$). You should output the n^{th} way of arranging the blocks, in dictionary order, to make the sum s.

You will not be given an n that is greater than the number of ways to arrange the blocks.

Sample run

```
4 5
2 1 1
```

3(b) [2 marks]

How many blocks in total need to be in the toy set to ensure that any of the arrangements summing up to 32 can be made? Blocks may only be used to show a single number; e.g. a block showing 6 cannot be turned upside-down to show a 9.

3(c) [5 marks]

Rather than listing the arrangements in dictionary order we could list them in *increasing numeric* order. In this case, the list for a sum of 4 would have been 4, 13, 22, 31, 112, 121, 211 and 1111. In general, what is the connection between the last entry in the dictionary list, and the first entry in the numeric list? Justify your answer.

3(d) [5 marks]

A *palindromic* arrangement of blocks is one where the numbers appear in the same order whether they are read from the left or from the right. E.g. 121 is a palindromic arrangement. How many palindromic arrangements sum to 8? Of the 540,142,091,243,007 different ways to arrange the blocks so that they sum to 50, how many are palindromic?

Total Marks: 100 End of BIO 2009 Round One paper