The CENTRE for EDUCATION
in MATHEMATICS and COMPUTING

2006 Canadian Computing Competition: Junior Division

Sponsor:

University of
Waterloo

Canadian Computing Competition
Student Instructions for the Junior Problems

1. You may only compete in one competition. If you wish to write the Senior paper, see the other problem set.
2. Be sure to indicate on your **Student Information Form** that you are competing in the **Junior** competition.
3. You have three (3) hours to complete this competition.
4. You should assume that
   - all input is from the keyboard
   - all output is to the screen

   For some problems, you may be asked for prompting: please provide this for the user. If no prompting is required, you do not need to provide any. Be sure your output matches the output in terms of order, spacing, etc. IT SHOULD MATCH EXACTLY!
5. Do your own work. Cheating will be dealt with harshly.
6. Do not use any features that the judge (your teacher) will not be able to use while evaluating your programs.
7. Books and written materials are allowed. Any machine-readable materials (like other programs which you have written) are *not* allowed. However, you are allowed to use "standard" libraries for your programming languages; for example, the STL for C++, java.util.*, java.io.*, etc. for Java, and so on.
8. Applications other than editors, compilers, debuggers or other standard programming tools are **not** allowed. Any use of other applications will lead to disqualification.
9. Please use file names that are unique to each problem: for example, please use `j1.pas` or `j1.c` or `j1.java` (or some other appropriate extension) for Problem J1. This will make the evaluator's task a little easier.
10. Your program will be run against test cases other than the sample ones. Be sure you test your program on other test cases.
11. Note that the top 2 Junior competitors in each region of the country will get a plaque and $100, and the schools of these competitors will also get a plaque. The regions are:
    - West (BC to Manitoba)
    - Ontario North and East
    - Metro

- Ontario Central and West
- Quebec and Atlantic

12. Check the CCC website at the end of March to see how you did on this contest, how the problems were meant to be solved, and to see who the prize winners are. The CCC website is:

    `www.cemc.uwaterloo.ca/ccc`

Problem J1: The New CCC (Canadian Calorie Counting)

Problem Description

At Chip's Fast Food emporium there is a very simple menu. Each food item is selected by entering a digit choice.

Here are the three burger choices: 1 – Cheeseburger (461 Calories) 2 – Fish Burger (431 Calories) 3 – Veggie Burger (420 Calories) 4 – no burger	Here are the three drink choices: 1 – Soft Drink (130 Calories) 2 – Orange Juice (160 Calories) 3 – Milk (118 Calories) 4 – no drink
Here are the three side order choices: 1 – Fries (100 Calories) 2 – Baked Potato (57 Calories) 3 – Chef Salad (70 Calories) 4 – no side order	Here are the three dessert choices: 1 – Apple Pie (167 Calories) 2 – Sundae (266 Calories) 3 – Fruit Cup (75 Calories) 4 – no dessert

Write a program that will compute the total Calories of a meal.

Input Specifications

The program should prompt the user for a number for each type of item then calculate and display the Calorie total. You may assume that each input will be a number from 1 to 4. That is, each customer has to pick exactly one number from each of the four options out of each of the four categories.

Output Specifications

The program prints out on the screen the total Calories of the selected meal, and stops executing after this output.

Sample Prompting and User Input (user input in *italics*)

```
Welcome to Chip’s Fast Food Emporium
Please enter a burger choice: 2
Please enter a side order choice: 1
Please enter a drink choice: 3
Please enter a dessert choice: 4
```

Output for the Sample

```
Your total Calorie count is 649.
```

Problem J2: Roll the Dice

Problem Description

Diana is playing a game with two dice. One die has m sides labelled 1, 2, 3, ..., m. The other die has n sides labelled 1, 2, 3, ..., n. Write a program to determine how many ways can she roll the dice to get the sum 10.

For example, when the first die has 6 sides and the second die has 8 sides, there are 5 ways to get the sum 10:

$$\begin{aligned} 2+8 &= 10 \\ 3+7 &= 10 \\ 4+6 &= 10 \\ 5+5 &= 10 \\ 6+4 &= 10 \end{aligned}$$

Input Specifications

The input is given as two integers. First, the user is prompted for and must enter in the number m $(1 \leq m \leq 1000)$. Second, the user is prompted for and must enter the number n $(1 \leq n \leq 1000)$.

Output Specifications

The program prints out the number of ways 10 may be rolled on these two dice. Note that in the output, the word "way" should be used if there is only one way to achieve the sum of 10; otherwise, the word "ways" should be used in the output. That is, if there is only one way to get the sum 10, the output should be:

```
There is 1 way to get the sum 10.
```

Sample Prompting and User Input 1 (user input in *italics*)

```
Enter m: 6
Enter n: 8
```

Output for Sample 1

```
There are 5 ways to get the sum 10.
```

Sample Prompting and User Input 2 (user input in *italics*)

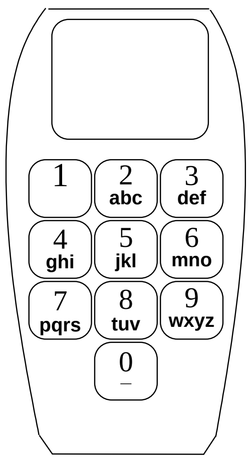
```
Enter m: 12
Enter n: 4
```

Output for Sample 2

```
There are 4 ways to get the sum 10.
```

Problem J3: Cell-Phone Messaging

Problem Description



Joe Coder has just received a cell phone for his birthday. At first he was not so excited about it, since he does not like to talk that much, nor listen for that matter, and he hates being interrupted by phone calls while coding or playing his favourite computer game. But, Joe learned that he can talk to his friends and also send e-mails. That made the phone cool.

In order to fit the 26 letters of the alphabet onto the keys of a cell phone, several letters are assigned to each key, as shown on the diagram.
To write a text message, we have to choose a letter from a set assigned to a key. The first letter on a key is chosen by pressing the key once, the second letter by pressing the key twice, third letter by pressing the key three times, and so on.
For example, to write "a" we press the key '2' once and we are done; to write "dada" we press 3232—four key presses; and to write "bob" we press 2266622. An obvious issue is how to write two consecutive letters on the same key, for example in 'abba' or 'cell'. The problem is solved by introducing a time-out feature: a letter currently displayed is chosen when another key is pressed, but also after a pause, i.e., a time out. Hence for example, to write 'abba' we press 2-pause-22-pause-22-pause-2; to write 'cell' we press 22233555-pause-555; or to write 'www' we press '9-pause-9-pause-9'.

This kind of typing takes some time, and Joe is working on a program to calculate how much time is needed to type certain words. His assumption is that he spends one second per press, and whenever he makes a pause he loses an additional two seconds. You are to help him to calculate the minimal time needed to type a message, under the above assumptions.

Input Specifications

Each line of input contains a word consisting only of lowercase letters. Words have at most 20 characters. Input will be given from the keyboard, and the program should stop reading input when the word `halt` has been entered.

Output Specifications

For each input word (excluding the word `halt`), print (on the screen) the minimal number of seconds needed to type in the word, with one number of output per line.

Sample Input

```
a
dada
bob
abba
cell
www
halt
```

Output for Sample Input

```
1
4
7
12
13
7
```

Problem J4: It’s tough being a teen!

Problem Description

There is always a list of things to be done!

Here is a list for you left just this morning by your parental figure.

1. Do your Math homework.
2. Call your grandma.
3. Call me at work.
4. Call your friend.
5. Feed the dog.
6. Let the dog out.
7. Watch television.

(It is nice that your parental figure makes sure you watch television, and not just use the internet all day long.)

As well, your parental figure has given you constraints on these tasks:

- Do your Math homework BEFORE you watch Television.
- Do your Math homework BEFORE you call your friend.
- Call your grandma BEFORE you do your Math homework.
- Call me at work BEFORE you call your friend.
- Feed the dog AFTER you call me at work.

Your “to do” list (above) can now be abbreviated to:

```
1,7
1,4
2,1
3,4
3,5
```

where x, y indicates that the task numbered x should be done before the task numbered y.

Unfortunately, during the day additional instructions are emailed to you by your parental figure. Write a program to use your original "to do" list and the additional instructions to output a list of your jobs in the order you must do them, or alternately, if you cannot complete them, you should output that there is no way to complete these tasks, and you are just going to go to bed.

Input Specifications

You will be given pairs of numbers, one number per line, to represent the additional instructions to be included with your original "to do" list given above. The data terminates with the input pair 0 and 0. You can assume that there will be at most 10 additional constraints.

Output Specifications

You should output a list of tasks in the order that they should be performed, or an error message saying that the tasks cannot be completed. If there are multiple orders in which the tasks may be completed, the following tie-breaking rule must be used:

> If there is a set of tasks which may be performed at the same time during the process, the smallest numbered task should be performed first.

Sample Input 1

```
6
2
5
4
0
0
```

Output for Sample Input 1

```
3 5 6 2 1 4 7
```

Explanation for Sample Output 1

The input data tells us that task 6 must be performed before task 2, and task 5 before task 4. The only tasks with no prerequisites are 3 and 6, so we choose 3 because it has the lower number. Then 5 and 6 are possible; 5 is chosen, then 6. Next must come 2, followed by 1. Then both 4 and 7 are possible; the lower one is chosen first.

Sample Input 2

```
7
2
4
5
0
0
```

Output for Sample Input 2

```
Cannot complete these tasks.  Going to bed.
```

Explanation for Sample Output 2

Notice that task 2 must be done before task 1, which must be done before task 7, which must be done before task 2. This forms a contradiction, and we cannot perform the tasks in the order prescribed.

Problem J5: CCC Othello

Problem Description

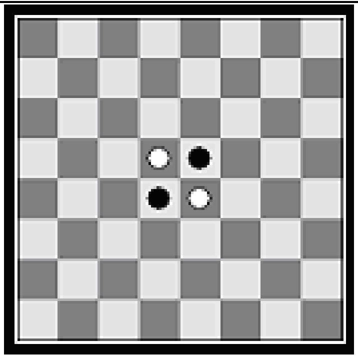
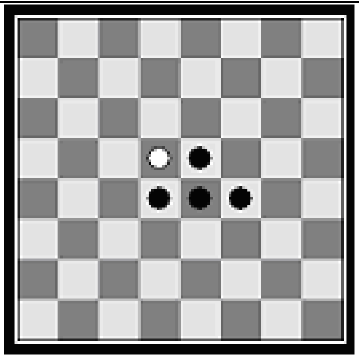
Othello (also known as Reversi or Black & White Chess) is a game played on an 8x8 board, similar to a checker board. For the purposes of describing this question, the spaces on the board are referred to by their row and column position, with the top row referred to as row 1 and the left hand column referred to as column 1. The game involves placing circular discs, one at a time, on the board. The discs are coloured black on one side and white on the other. One player places his/her discs with the white side up and the other player places his/her discs with the black side up.

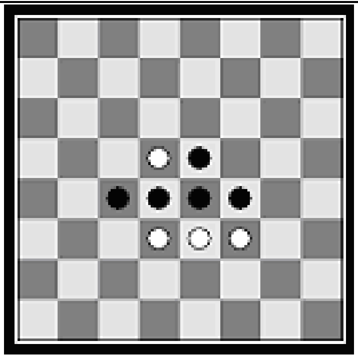
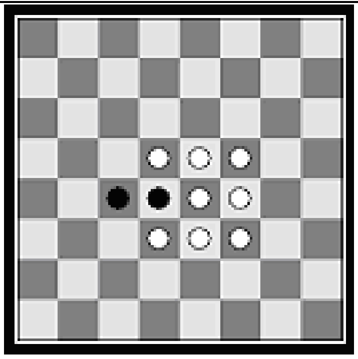
The game starts with some discs already placed on the board.

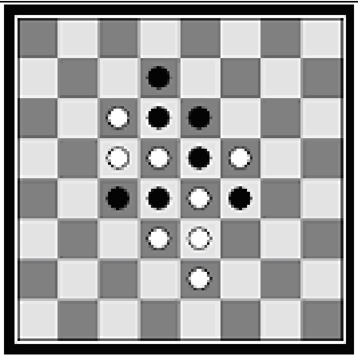
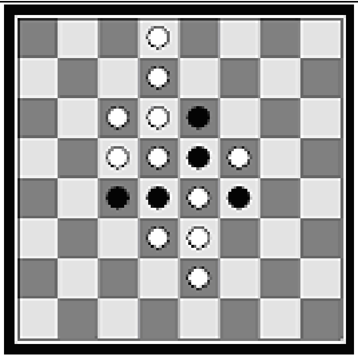
A move is *valid* if the following two conditions are satisfied:

1. The piece placed on the board is adjacent to a piece on the board (i.e, beside a piece either horizontally, vertically or diagonally).
2. At least one of your opponents' discs must be "flipped." If you are playing black pieces, you flip your opponents' (white) pieces (to black) so long as your opponents' pieces are on a line (either horizontally, vertically or diagonally) between the latest piece placed on the board and another black piece, with no other black pieces or empty squares in between the most recently placed black piece and the given white piece. The same rule applies if the player is placing white pieces.

Here are a couple of sample valid moves, starting from various configurations of the board:

Board at the beginning of the game	Board after the black player has placed a disc in row 5 and column 6.
	

Board after several plays by each player	Board after the white player has placed a disc in row 4 and column 6
	

Board after several plays by each player	Board after the white player has placed a disc in row 1 and column 4.
	

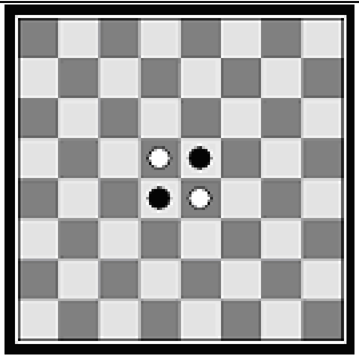
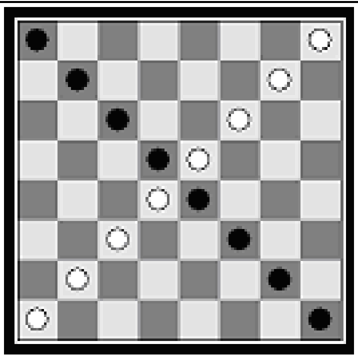
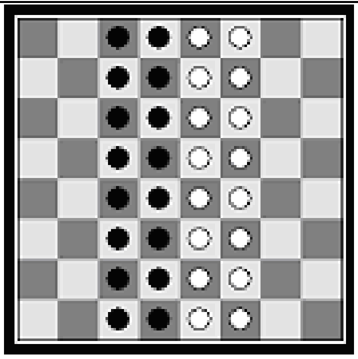
In the CCC version of Othello, the board may start with one of 3 configurations. Play will always start with the black player taking the first turn, and then alternating white to black for the rest of the turns. You must write a program to simulate taking turns in an Othello game, and at the end, report how many pieces of each colour are on the board.

Input Specifications

The user will enter three components of input (via the keyboard).

First the user will enter a letter representing the configuration of the initial board (either a, b or c).

Here are the initial configurations for the board.

Configuration ’a’	Configuration ’b’	Configuration ’c’
		

The second component of input will be an integer n, where $0 \leq n \leq 30$ which indicates the number of moves to be made in the simulation. The third component of input is n pairs of integers (R, C), where $1 \leq R \leq 8$ and $1 \leq C \leq 8$, and R represents the row and C represents the column of the next move.

Remember that the first move will be made by the black player, the next move will be made by the white player, then the black player, then the white player, and so on. You may also assume that all moves (R, C) will be valid moves on empty spaces on the board.

Output Specifications

The program will output the number of black discs showing followed by the number of white discs showing on the board after the moves have been made. You are not responsible for displaying a picture of the board during the game. However, during your own testing, this may be useful.

Sample Input 1

```
a 1 5 6
```

Output for Sample Input 1

```
4 1
```

Sample Input 2

```
b 0
```

Output for Sample Input 2

```
8 8
```

Sample Input 3

```
c 3 1 7 2 2 2 1
```

Sample Output for Sample Input 3

```
22 13
```