The CENTRE for EDUCATION
in MATHEMATICS and COMPUTING

2008 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 MUST 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. Inefficient solutions may lose marks for some problems. Be sure your code is as efficient (in terms of time) as possible.

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 Toronto area
- 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: Body Mass Index

Problem Description

The Body Mass Index (BMI) is one of the calculations used by doctors to assess an adult's health. The doctor measures the patient's height (in metres) and weight (in kilograms), then calculates the BMI using the formula

$$\text{BMI} = \frac{\text{weight}}{\text{height} \times \text{height}}.$$

Write a program which prompts for the patient's height and weight, calculates the BMI, and displays the corresponding message from the table below.

BMI Category	**Message**
More than 25	Overweight
Between 18.5 and 25.0 (inclusive)	Normal weight
Less than 18.5	Underweight

Sample Input 1 (user input is in *italics*)

```
Enter weight: 69
Enter height: 1.73
```

Output for Sample Input 1

```
Normal weight
```

Explanation for Output in Sample Input 1

The BMI is $69/(1.73 \times 1.73)$, which is approximately 23.0545. According to the table, this is a "Normal weight".

Sample Input 2 (user input is in *italics*)

```
Enter weight: 84.5
Enter height: 1.8
```

Output for Sample Input 2

```
Overweight
```

Explanation for Output in Sample Input 2

The BMI is $84.5/(1.8 \times 1.8)$, which is approximately 26.0802. According to the table, this is "Overweight".

Problem J2: Do the Shuffle

Problem Description

Those tiny music machines that play your digital music are really computers that keep track of and play music files. The *CCC music player* (C^{3}MP) is currently in development and will be hitting the stores soon! In this problem, you have to simulate a C^{3}MP.

The C^{3}MP music player will hold 5 songs in memory, whose titles will always be “A”, “B”, “C”, “D” and “E”. The C^{3}MP also keeps track of a *playlist*, which is an ordering of all the songs. The C^{3}MP has 4 buttons that the user will press to rearrange the playlist and play the songs.

Initially, the C^{3}MP playist is “A, B, C, D, E”. The 4 control buttons do the following:

- Button 1: move the first song of the playlist to the end of the playlist.
  For example: “A, B, C, D, E” will change to “B, C, D, E, A”.
- Button 2: move the last song of the playlist to the start of the playlist.
  For example, “A, B, C, D, E” will change to “E, A, B, C, D”.
- Button 3: swap the first two songs of the playlist.
  For example, “A, B, C, D, E” will change to “B, A, C, D, E”.
- Button 4: stop rearranging songs and output the playlist.

You need to write a program to simulate a CCC music player. Your program should repeatedly ask for two positive integers b and n. Here b represents the button number that the user wants to press, $1 \leq b \leq 4$, and n represents the number of times that the user wants to press button b. You can assume that n always satisfies $1 \leq n \leq 10$.

The input will always finish with the pair of inputs ($b = 4$, $n = 1$) when this happens, you should print the order of songs in the current playlist and your program should end. You can assume that the user will only ever press button 4 once.

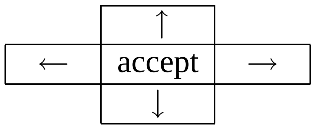
Sample session (user input in *italics*)	**Explanation**
	(initial playlist is “A, B, C, D, E”)
`Button number:` *2*	
`Number of presses:` *1*	($b = 2$, $n = 1$ so “A, B, C, D, E” changed to “E, A, B, C, D”)
`Button number:` *3*	
`Number of presses:` *1*	($b = 3$, $n = 1$, so “E, A, B, C, D” changed to “A, E, B, C, D”)
`Button number:` *2*	
`Number of presses:` *3*	($b = 2$, $n = 3$, so “A, E, B, C, D” changed to “B, C, D, A, E”)
`Button number:` *4*	
`Number of presses:` *1*	($b = 4$, $n = 1$) When this happens, you should output the playlist.

Output for Sample Input Session

```
B C D A E
```

Problem J3: GPS Text Entry

Problem Description
For her birthday, Sandy got a Global Positioning System (GPS) unit, which is an electronic device she can use to track the local hiking trails. Along the way Sandy can mark waypoints that can be recorded on a map when she gets home. A description of each waypoint can be entered in the unit, however the device does not have a keypad. Instead it has four cursor buttons, up, down, left, and right, and a button to accept the letter. The keypad looks like the following:



The screen displays a grid of the letters and symbols that can be used to "type out" the description. Here is the layout of the grid:

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	Z	space	-	.	enter

When you enter the name of the waypoint, the cursor starts at the A. You must move the cursor to the location of the next letter or symbol and then accept that letter. The cursor can only move to squares which are adjacent horizontally or vertically (not diagonally). Once you have entered all the letters in the description, you need to move the cursor to "enter" and accept the entire phrase.

You are to write a program that will calculate the number of cursor movements it takes to "type in" a phrase. For example, to enter the word "GPS", starting from the "A" position, you would move down 1 to select "G", then move right 3 and down 1 to select "P", then move down 1 and left 3 to select "S" and finally move down 1 and right 5 to select "enter". This is a total of 15 cursor movements. Note that the total number of cursor movements does not change if you choose to move down and then across or across and then down. Also note that you cannot move beyond the boundaries of the grid (e.g., you cannot move off the grid nor "wrap-around" the grid).

Input Specification
The input for your program will be a string of at most 40 characters. You may assume that all characters in the string are contained in the grid.

Output Specification
The output for your program will be an integer that is the total number of cursor movements needed to enter the string using the grid layout given.

Sample Input 1

```
GPS
```

Output for Sample Input 1

```
15
```

Sample Input 2

```
ECHO ROCK
```

Output for Sample Input 2

```
29
```

Problem J4: From Prefix to Postfix

Problem Description

Prefix notation is a non-conventional notation for writing arithmetic expressions. The standard way of writing arithmetic expressions, also known as *infix notation*, positions a binary operator between the operands, e.g., $3 + 4$, while in prefix notation the operator is positioned before the operands, e.g., $+3\ 4$. Similarly, the prefix notation for $5 - 2$ is $-5\ 2$. A nice property of prefix expressions with binary operators is that parentheses are not required since there is no ambiguity about the order of operations. For example, the prefix representation of $5 - (4 - 2)$ is $-5 - 4\ 2$, while the prefix representation of $(5 - 4) - 2$ is $- - 5\ 4\ 2$. The prefix notation is also known as *Polish notation*, due to Jan Łukasiewicz, a Polish logician, who invented it around 1920.

Similarly, in *postfix notation*, or *reverse Polish notation*, the operator is positioned after the operands. For example, postfix representation of the infix expression $(5 - 4) - 2$ is $5\ 4 - 2-$.

Your task is to write a program that translates a prefix arithmetic expression into a postfix arithmetic expression.

Input Specification

Each line contains an arithmetic prefix expression. The operators are `+` and `-`, and numbers are all single-digit decimal numbers. The operators and numbers are separated by exactly one space with no leading spaces on the line. The end of input is marked by 0 on a single line. You can assume that each input line contains a valid prefix expression with less than 20 operators.

Output Specification

Translate each expression into postfix notation and produce it on a separate line. The numbers and operators are separated by at least one space. The final 0 is not translated.

Sample Input and Output

Sample Input	Sample Output
`1`	`1`
`+ 1 2`	`1 2 +`
`- 2 2`	`2 2 -`
`+ 2 - 2 1`	`2 2 1 - +`
`- - 3 + 2 1 9`	`3 2 1 + - 9 -`
`0`	

Problem J5: Nukit

Problem Description
Canada's top two nuclear scientists, Patrick and Roland, have just completed the construction of the world's first nuclear fission reactor. Now it is their job to sit and operate the reactor all day, every day. Naturally they got a little bored after doing this for a while and as a result, two things have happened. First, they can now control the individual reactions that happen inside the reactor. Second, to pass the time, they have invented a new game called Nukit.

At the beginning of Nukit, a number of particles are put in the reactor. The players take alternating turns, with Patrick always going first. When it is a player's turn to move, they must select some of the remaining particles to form one of the possible reactions. Then those particles are destroyed. Eventually there will be so few particles that none of the reactions can be formed; at this point, the first person who is unable to form a reaction on their turn loses.

In our universe you can assume that there are only 4 types of particles: A, B, C, D. Each reaction is a list of particles that can be destroyed on a single turn. The five reactions are:

1. AABDD
2. ABCD
3. CCD
4. BBB
5. AD

For example, the first reaction "AABDD" says that it is allowable to destroy two A, one B, and two D particles all at the same time on a turn.

It turns out that, no matter how many particles start off in the reactor, exactly one of Patrick or Roland has a *perfect winning strategy*. By *player X has a perfect winning strategy*, we mean that no matter what the other player does, player X can always win by carefully choosing reactions. For example, if the reactor starts off with one A, five B, and three D particles then Roland has the following perfect winning strategy: "if Patrick forms reaction BBB initially, then form reaction AD afterward; if Patrick forms reaction AD initially, then form reaction BBB afterward." (The strategy works because either way, on Patrick's second turn, there are not enough particles left to form any reactions.)

Given the number of each type of particle initially in the reactor, can you figure out who has a perfect winning strategy?

Input Specification
The first line of input contains n, the number of test cases ($1 \leq n < 100$). Each test case consists

of 4 integers separated by spaces on a single line; they represent the initial number of A, B, C and D particles. You can assume that there are initially between 0 and 8 (inclusive) of each type of particle.

Output Specification
For each test case, output the player who has a perfect winning strategy, either “Roland” or “Patrick”.

Sample Input
```
6
0 2 0 2
1 3 1 3
1 5 0 3
3 3 3 3
8 8 6 7
8 8 8 8
```

Output for Sample Input
```
Roland
Patrick
Roland
Roland
Roland
Patrick
```

Partial Explanation for Sample Output
The first output occurs since Patrick loses immediately, since he cannot form *any* reaction. (Roland’s perfect winning strategy is “do nothing.”)

The second output occurs since Patrick has the perfect winning strategy “form reaction ABCD,” which makes Roland lose on his first turn.

The third output is explained in the problem statement.