The CENTRE for EDUCATION
in MATHEMATICS and COMPUTING

2008 Canadian Computing Competition: Senior Division

Sponsor:

University of
Waterloo

Canadian Computing Competition
Student Instructions for the Senior Problems

1. You may only compete in one competition. If you wish to write the Junior paper, see the other problem set.

2. Be sure to indicate on your **Student Information Form** that you are competing in the **Senior** competition.

3. You have three (3) hours to complete this competition.

4. You should assume that

   - all input is from a file named `sX.in`, where `X` is the problem number ($1 \leq$ `X` ≤ 5).
   - 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 `s1.pas` or `s1.c` or `s1.java` (or some other appropriate extension) for Problem S1. 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, especially Problems 4 and 5. Be sure your code is as efficient (in terms of time) as possible.

11. Note that the top 2 Senior 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. If you finish in the top 20 competitors on this competition, you will be invited to participate in CCC Stage 2, held at the University of Waterloo in May 2008. Should you finish in the top 4 at Stage 2, you will be a member of the Canadian IOI team at IOI 2008, held in Egypt. Note that you will need to know C, C++ or Pascal if you are invited to Stage 2. But, first, do well on this contest!

13. 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 S1: It's Cold Here!

Problem Description

Canada is cold in winter, but some parts are colder than others. Your task is very simple, you need to find the coldest city in Canada. So, when given a list of cities and their temperatures, you are to determine which city in the list has the lowest temperature and is thus the coldest.

Input

The input is a sequence of city names and temperature values. Temperatures are integer, possibly preceded with a "minus" sign. There is a single space between the city name and the temperature. No city name contains any whitespace and is always less than 256 characters in length. There is at least one city in the list, no more than 10000 cities, and the last city is always Waterloo. You may assume that the temperature is not less than -273 and not more than 200.

Output

You are to output the name of the coldest city on a single line with no whitespace before or after the name. You may assume that there will not be more than one city which is the coldest.

Sample Input

```
Saskatoon -20
Toronto -2
Winnipeg -40
Vancouver 8
Halifax 0
Montreal -4
Waterloo -3
```

Output for Sample Input

```
Winnipeg
```

Problem S2: Pennies in the Ring

Problem Description

The game "Pennies in the Ring" is often played by bored computer programmers who have gotten tired of playing solitare. The objective is to see how many pennies can be put into a circle. The circle is drawn on a grid, with its center at the coordinate $(0, 0)$. A single penny is placed on every integer grid coordinate (e.g., $(1, 1), (1, 2),$ etc.) that lies within or on the circle. It's not a very exciting game, but it's very good for wasting time. Your goal is to calculate how many pennies are needed for a circle with a given radius.

Input

The input is a sequence of positive integer values, one per line, where each integer is the radius of a circle. You can assume the radius will be less than or equal to 25000. The last integer will be indicated by 0. You may assume that the grid is large enough for two pennies to be on adjacent integer coordinates and not touch.

Output

You are to output, each on its own line, the number of pennies needed for each circle. You do not need to output 0 for the last 0. You may assume that the number of possible pennies is less than 2 billion (which is only $20 million dollars: computer scientists have lots of money).

Sample Input

```
2
3
4
0
```

Output for Sample Input

```
13
29
49
```

Problem S3: Maze

Problem Description

In order to make a few dollars, you have decided to become part of a scientific experiment. You are fed lots of pizza, then more pizza and then you are asked to find your way across the city on a scooter powered only by pizza. Of course, the city has lots of intersections, and these intersections are very controlled. Some intersections are forbidden for you to enter; some only let you move north/south as you leave the intersection; others let you move only east/west as you leave the intersection; and the rest let you go in any compass direction (north, south, east or west).

Thankfully your scientific friends have given you a map of the city (on the back of a pizza box), with an arrangement of symbols indicating how you can move around the city. Specifically, there are 4 different symbols on the box:

- The symbol `+` indicates we can move in any direction (north/south/east/west) from this location.
- The symbol `-` indicates we can move only east or west from this location.
- The symbol `|` indicates we can move only north or south from this location.
- The symbol `*` indicates we cannot occupy this location.

Your task is to determine how many intersections you must pass through to move from the north-west corner of the city to the south-east corner of the city.

Input Specification

The input begins with a number t ($1 \leq t \leq 10$) on its own line, which indicates how many different cases are contained in this file. Each case begins with a number r on one line, followed by a number c on the next line ($1 \leq r, c \leq 20$). The next r lines contain c characters, where each character is one of {`+`, `*`, `-`, `|`}. You may assume the north-west corner of the city can be occupied (i.e., it will not be marked with `*`).

Output Specification

The output will be t lines long, with one integer per line. The integer on line i ($1 \leq i \leq t$) indicates the minimum number of intersections required to pass through as you move from the north-west corner of the city to the south-east corner of the city. If there is no way to get from the north-west corner to the south-east corner, output -1 for that test case.

Sample Input

```
3
2
2
-|
*+
3
5
+||*+
+++|+
**--+
2
3
+*+
+*+
```

Output for Sample Input

```
3
7
-1
```

Problem S4: Twenty-four

Problem Statement

Twenty-four is a popular card game designed to be played by four players. Each player is dealt a deck of cards, which are kept face down. On every turn, each of the four players turns over the top card of his or her deck, so that it is visible to all. The goal is to find an arithmetic expression using the values of the cards (with A meaning 1, J meaning 11, Q meaning 12, and K meaning 13) that evaluates to the number 24. For example, for the example in the illustration, one possible expression would be:

```
((A * K)- J) * Q
((1*13)-11) * 12
```

The first player to find such an expression wins the turn, and adds all four cards to the bottom of his or her deck.

Each valid arithmetic expression must use all four cards, and must combine their values using addition, subtraction, multiplication, or division. Parentheses are allowed to specify precedence of operations. Juxtaposing cards to make multiple-digit decimal numbers is not allowed (e.g. you cannot place the cards 2 and 4 beside each other to make 24). Non-integer quotients of division are also not allowed, even as a partial result (of a subexpression of the overall expression).

In some cases, the players may take a very long time to find an expression evaluating to 24. In fact, in some cases, no such expression exists. Your task is to determine, given four cards, an expression that evaluates to the largest number less than or equal to 24.

Input Specification

The first line contains an integer $1 \leq N \leq 5$ indicating the number of card hands that follow. Each hand consists of four lines. Each of these lines is an integer $1 \leq C \leq 13$ indicating the value of a card.

Output Specification

For each hand, output a line containing an integer n if the cards can be combined using arithmetic operators to evaluate to n. The value n should be the largest possible value amongst all possible arithmetic expressions using these 4 cards, so long as $n \leq 24$.

Sample Input

```
3
3
3
3
3
3
1
1
1
1
12
5
13
1
```

Output for Sample Input

```
24
4
21
```

Problem S5: 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 30 (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.