A. Penguin Bashing

In a well-known online game you play a yeti whose goal it is to hit a cute and cuddly penguin in such a way that it flies as far as possible. The game is set in sunny Antarctica, where the ground is littered with obstacles left by human and robotic expeditions. The most dangerous obstacles are sharp, pointy, metal objects, known as spikes. These spikes are most definitely not safe for penguins and children under three years of age.

Besides the occasional expedition, there have also been many wars in Antarctica. The result is that besides spikes, penguins (and yetis), one is also likely to encounter mines on the Antarctic surface. While they are not good for the health of a penguin, the explosions do improve the distance he will travel.

The best way to get a high score in this game is by figuring out the optimal strategy in advance. To be able to do this you need to know the following facts:

- The yeti always bats at an integral angle between -90 and $+90$ degrees, $-90°$ is straight down, $0°$ is to the right and $+90°$ is straight up.
- The penguin gets a total velocity of $25\,{}^{\mathrm{m}}/_{\mathrm{s}}$ regardless of the angle.
- The yeti hits the 3 kg penguin at exactly one meter above the ground.
- Upon hitting the ground the penguin will slide along the ground, while undergoing friction. This will slow it down by $5\,{}^{\mathrm{m}}/_{\mathrm{s}}$ for every meter traveled.
- Upon hitting a mine the penguin will be flung 2 m forward through the air, and also undergo an instant increase in velocity of $4\,{}^{\mathrm{m}}/_{\mathrm{s}}$ forward.
- If the penguin hits a spike, the game ends immediately with a score of -100. Otherwise the score will be the horizontal distance traveled in centimeters, rounded to the nearest integer.
- Standard Earth gravity ($9.81\,{}^{\mathrm{m}}/_{\mathrm{s}^2}$) is used and drag is ignored.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with a single integer $n < 10^5$, the number of obstacles.
- n lines, each containing:
  - A single floating point number x_i, the position of the obstacle in meters from the start.
  - A string describing the obstacle, either "`mine`" or "`spike`".

Output

For each test case:

- One line containing two integers, the highest possible score, and the angle in degrees at which the yeti needs to hit the penguin to achieve that score. In case of a tie output the lowest angle.

Example

Input	Output
3 0 1 64.7 spike 2 63.9 mine 64.7 spike	6829 44 6466 34 7022 39

B. Gearbox

Jake decided to take the gearbox of his car apart and put it back together for 'learning purposes'. A gearbox is a box containing many gears, cogs and sprockets (as well as several springs and pinions). All these parts are connected together in intricate ways. Consequently, he is not completely sure if every part is in the correct position again.

All gears are placed on metal rods; when one gear on a rod turns, they all turn by an equal angle. The rods are connected by springs and plastic bars, but Jake is pretty sure he has that part right. The cogs and sprockets are not connected to anything, they are free to rattle around in the box. This may be a bit strange, but a quick Google search reveals that this is customary for gearboxes of this type.

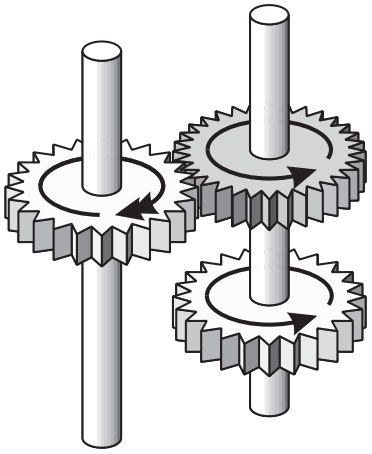


Figure 1: Two rods with three gears. The top two gears are interlocked, the two rods turn in opposite directions. If the dark gear has 36 teeth and the light gears have 24 teeth, then the left rod turns 1.5 times faster.

The only problem is that some gears are interlocked, which could cause the main drive shaft to jam. The gearbox uses modern InfiniTeeth$^{\text{TM}}$ gears, which means that it is impossible to count the number of teeth on a gear. All gears have a type number, and gears of the same type have the same number of teeth. According to the manual (which Jake should have read before he started this mess) all rods should be able to turn, regardless of the number of teeth on each type of gear. So Jake concludes that if this is true for his gearbox it is definitely assembled correctly.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with three integers n_g, n_r and n_i (all $< 10^5$), the number of gears, rods and interlockings.
- n_g lines, each containing two integers $0 < t_i < 100$ and $0 \leq r_i < n_r$, the type number of gear i and the index of the rod it is on respectively.
- n_i lines, each containing two integers $0 \leq a_j < b_j < n_g$, indicating that gears a_j and b_j are interlocked.

Output

For each test case:

- One line containing the text "`ok`" if the gearbox is definitely assembled correctly, and "`jammed`" otherwise.

Example

Input	Output
4 2 2 1 1 0 2 1 0 1 8 4 4 20 0 10 1 20 2 10 3 30 0 30 1 40 2 40 3 0 1 2 3 4 6 5 7 8 4 4 20 0 10 1 20 2 10 3 30 0 40 1 40 2 30 3 0 1 2 3 4 6 5 7 3 3 3 1 0 1 1 1 2 0 1 0 2 1 2	ok ok jammed jammed

C. Escape from the Minefield

After a failed parachute drop lieutenant Jones finds himself in a precarious situation. Instead of landing at the target coordinates, he and his platoon are trapped in the middle of an enemy minefield!

Fortunately, as a special-ops soldier, lieutenant Jones has come prepared for all situations. Among other things, he carries an accurate map that shows the locations of all the mines in the area. Intelligence briefings further suggest that the type of anti-personnel mine used by the enemy has a blast- and detection-radius of 2 meters; if you get any closer, you die.

For the moment Jones' platoon is safe, hiding beneath a bush. To keep things simple they plan to make a run for it while staying in a circular formation[1], starting centered at the bush. In this formation each soldier needs an area of 1 square meter on average. For the sake of morale the entire formation should stay outside the blast-radius of any mines.

Of course, lieutenant Jones wants to escape with as many soldiers as possible. But in order to ensure that no one steps on a mine he may have to leave some men behind to be captured by the enemy.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with a single positive integer $n < 10^5$, the number of mines in the minefield.
- n lines, each containing two integers, the x and y coordinates (in meters) of each mine relative to Jones' current location. $|x|, |y| < 10^5$. No two mines have the exact same coordinates.

Output

For each test case:

- One line containing a single integer, the maximum number of soldiers that can escape.

[1] In a circular formation the distance of each soldier to the circle's center is less than the circle's radius. The total area of the circle should be enough to accommodate all soldiers.

Example

Input	Output
3 3 2 2 2 -2 -2 -2 8 4 2 4 -1 1 4 -2 4 -4 2 -4 -2 -1 -4 2 -4 7 -10 -1 -4 3 -4 -4 -1 -6 2 4 3 -4 7 0	2 0 7

D. Baby's Blocks

Mikey is playing with his favorite toy blocks, each depicting one letter of the alphabet. He is trying to make words using all his blocks, but as he does not know valid words from invalid ones, he goes by all possible orderings of the letters in alphabetical order and asks Albert, his genius brother, if the word he made is a valid one. Mikey can make one word (including asking a question and getting an answer) every sixty seconds, and he never makes the same word twice.

Albert is delighted about Mikey's activity, but would rather not teach Mikey certain words. Help Albert by predicting when Mikey will start making a certain forbidden word, so he can set the alarm clock indicating Mikey's bedtime to just before this moment.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing the forbidden word to look for, consisting of at most 20 upper case characters. This word can be formed exactly using all of Mikey's blocks.

Output

For each test case:

- One line containing the number of minutes it will take until the forbidden word is reached, assuming Mikey has just started making the first possible word.

Example

Input	Output
4 ABC FSCK OMGWFTBBQ BANANA	0 10 112293 34

E. The Pharaoh's Curse

Most of the contestants managed to arrive at the Benelux Algorithm Programming Contest on time. All except those unfortunate souls who chose to use a cheap car navigation system. A couple of wrong turns led them completely off course, and now they have ended up all over the world.

Consider the case of contestant J.-P. B. (who has asked to remain anonymous). After driving for countless hours he found himself beneath three miles of water in the middle of the Atlantic Ocean. He was so focused on his navigation system that he failed to look outside until he noticed that the air was getting very moist. Fortunately for this contestant there was a spare submarine under his seat, which allowed him to escape this perilous situation.

Tragic as this story seems, it is not even the worst. Another contestant, whom we will simply call S, wisely decided to travel by train. Sadly, she still listened to the advice of her evil car navigation system, and this morning she took a southbound train instead of heading north. After some more bad directions she found herself locked inside a labyrinth, in the pyramid of the Egyptian pharaoh Sok-O-Ban. As any experienced adventurer knows, these ancient tombs are often riddled with traps and other mechanisms. Sok-O-Ban's final resting place was no exception.

During his lifetime this pharaoh was known for his sadistic behavior, and he liked to give impossible challenges to random strangers. It is therefore no coincidence that the cavern where contestant S ended up was completely closed off, surrounded by solid rock on every side.

After looking around, our protagonist S managed to draw a map of the tomb. Fortunately she found no skeletons, mummies or spiked death traps. She did discover some buttons embedded in the floor tiles. If she could press them all at the same time, then the hidden exit would open. But as soon as she would release one of the buttons, the door would close again.

Besides the buttons, the walls and a few dusty floor tiles, there were also two sarcophagi filled with rocks. To accommodate the small stature of the ancient Egyptians, the sarcophagi were cubes with sides measuring 1 meter, the same size as the floor tiles. It seemed that these sarcophagi were the key to escaping: by pushing them onto the buttons the exit could be kept open. Only one small problem remained: the sarcophagi were far too heavy to move by hand and too large to climb over.

Fortunately for S, the pharaoh did not anticipate the portable sarcophagus transporter she had conveniently stashed in her backpack. Attaching this reusable system to her arms allowed her to effortlessly push a sarcophagus exactly one meter straight ahead. This corresponds nicely to the 1 meter steps S used when marking out the grid in her map.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing two positive integers $h, w \leq 50$, the height and width of the maze.
- h lines of w characters each, the map of the cavern our protagonist made, in which:
  - '`#`' is a wall or otherwise impassible space.
  - '`.`' is an empty space, of which there are at most 100.

- '`S`' is our protagonist.
- '`X`' is one of the heavy sarcophagi. There are at most two of these.
- '`B`' is a button.
- '`E`' is the exit. There is exactly one exit, on the edge of the map.

The edge of the map contains only walls and the exit.

Output

For each test case:

- One line containing the minimum number of steps[2] it will take S to escape the tomb, or the text "`impossible`" if she cannot escape.

Example

Input	Output
3 7 8 ######## #..S...# #.####.# #.#.XB.# #.####.# #......E ######## 7 8 ######## #..S...# #.####.# #.#.BX.# #.####.# #......E ######## 4 8 ##E##### #...#### #SX.XBB# ########	impossible 10 19

[2]S always takes one meter steps along the grid lines, possibly pushing a sarcophagus.

F. Dimensional Warp Drive

In the year 3133 space travel has become a common sight for humanity. With the invention of the Dimensional Warp Drive it has become possible to travel between distant planets almost instantaneously.

The Dimensional Warp Drive functions according to the laws of Relativistic Curved Quantized Space theory. In RCQS theory space has 11 dimensions, each of which is curved in on itself. That means that if you travel far enough in one direction you end where you began. Space is also quantized in this theory, in each dimension there are only 11 possible positions, 0 up to 10. Position 11 would be equivalent to position 0.

In 3075 the first *warp trajectory* was discovered. The effect of following such a trajectory is that a spacecraft jumps to the position that is the sum of its current position and the coordinates of the warp trajectory (in each dimension). After being discovered, a warp trajectory can be used multiple times.

In the following years many more warp trajectories were discovered, allowing humanity to colonize space at an exponential rate. By 3106 integrated navigation computers were commonplace. These machines determine how to reach a destination using a combination of the known warp trajectories.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with a single integer $n < 1000$, the number of warp trajectories.
- A line containing 11 integers (between 0 and 10), the coordinates of the starting point of a journey.
- A line containing 11 integers, the coordinates of the destination, which is different from the starting point.
- n lines, each containing 12 integers, the coordinates of each warp trajectory followed by the year in which that trajectory was discovered. The year will be between 3075 and 100000.

Output

For each test case:

- One line containing a single integer, the year in which it became possible to reach the destination from the starting point, or "`unreachable`" if it cannot be reached.

Example

Input	Output
3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 3 0 0 3 0 0 0 0 1 0 0 1 0 0 1 0 0 3075 0 0 1 0 0 2 0 1 0 0 0 3082 2 0 0 0 0 0 0 0 9 0 0 0 0 0 4 0 0 6 0 0 2 0 0 0 0 1 0 0 1 0 0 1 0 0 3075 0 0 1 0 0 2 0 1 0 0 0 3082 3 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 6 0 0 3 0 0 0 0 1 0 0 1 0 0 1 0 0 3075 0 0 1 0 0 2 0 1 0 0 0 3082 1 2 3 4 5 6 7 5 4 3 2 3093	3075 3082 unreachable

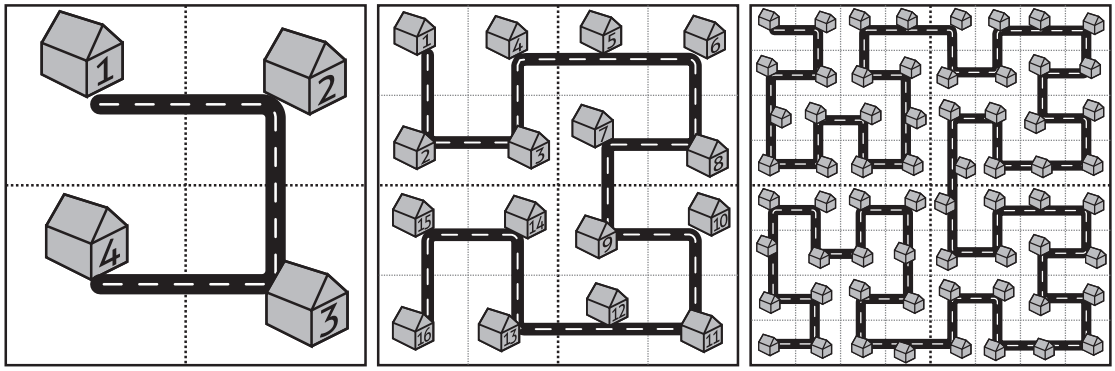
G. Fractal Streets

With a growing desire for modernization in our increasingly larger cities comes a need for new street designs. Chris is one of the unfortunate city planners responsible for these designs. Each year the demands keep increasing, and this year he has even been asked to design a completely new city.

More work is not something Chris needs right now, since like any good bureaucrat, he is extremely lazy. Given that this is a character trait he has in common with most computer scientists it should come as no surprise that one of his closest friends, Paul, *is* in fact a computer scientist. And it was Paul who suggested the brilliant idea that has made Chris a hero among his peers: *Fractal Streets*! By using a Hilbert curve, he can easily fill in rectangular plots of arbitrary size with very little work.

A Hilbert curve of order 1 consists of one "cup". In a Hilbert curve of order 2 that cup is replaced by four smaller but identical cups and three connecting roads. In a Hilbert curve of order 3 those four cups are in turn replaced by four identical but still smaller cups and three connecting roads, etc. At each corner of a cup a driveway (with mailbox) is placed for a house, with a simple successive numbering. The house in the top left corner has number 1, and the distance between two adjacent houses is 10 m.

The situation is shown graphically in figure 2. As you can see the Fractal Streets concept successfully eliminates the need for boring street grids, while still requiring very little effort from our bureaucrats.



(a) Order 1 (b) Order 2 (c) Order 3

Figure 2: Hilbert curves of order 1, 2 and 3, with the location of the houses indicated.

As a token of their gratitude, several mayors have offered Chris a house in one of the many new neighborhoods built with his own new scheme. Chris would now like to know which of these offerings will get him a house closest to the local city planning office (of course each of these new neighborhoods has one). Luckily he will not have to actually drive along the street, because his new company "car" is one of those new flying cars. This high-tech vehicle allows him to travel in a straight line from his driveway to the driveway of his new office. Can you write a program to determine the distance he will have to fly for each offer (excluding the vertical distance at takeoff and landing)?

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing a three positive integers, $n < 16$ and $h, o < 2^{31}$, specifying the order of the Hilbert curve, and the house numbers of the offered house and the local city planning office.

Output

For each test case:

- One line containing the distance Chris will have to fly to his work in meters, rounded to the nearest integer.

Example

Input	Output
3 1 1 2 2 16 1 3 4 33	10 30 50

H. No Smoking, Please

The new anti-smoking laws have been in effect for some time now, but some restaurant owners still have problems coping with the new rules. Some say that it is detrimental to their business, others are relieved that their clothes no longer smell of cigarette smoke after each service. However, whether they like it or not, nearly every restaurant has had to make at least some changes to its interior, especially if both smoking and non-smoking customers are to be seated in different parts of the restaurant.

Some entrepreneurs have made various ingenious devices to cope with the new rules, like the infamous "aquarium". This device creates overpressure behind the bar to ensure that the personnel is working in a smoke-free environment. However, for many restaurants this is obviously not feasible.

This is why Johann, proprietor of a classy restaurant in downtown Groningen, has decided to simply divide his restaurant into two zones, with air locks to connect the zones (and a hatch to pass food, as the personnel cannot come into the smoking zone). His idea is that the smoking zone should simply be directly connected to the outside, this will save on heating and supply fresh air. On the other hand, the non-smoking zone should obviously be connected to the kitchen to allow the personnel to serve food.

As the restaurant consists of many small rooms, the division into two connected zones is relatively easy. Johann can simply put air locks in a number of the passages that connect two rooms. However, Johann has a bit of a shock when he sees the price of an air lock and the associated hatch! An air lock costs a thousand euros per square meter of the passage and the hatches to pass food also cost a thousand euros each, as they have to satisfy numerous rules and regulations. Walls without a passage do not need to be replaced by air locks nor fitted with hatches.

Obviously Johann would like to minimize the cost of this operation, and he would like you to help him. His restaurant is completely rectangular, and consists of equally-sized square rooms that can each be connected to their (at most four) neighbors.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with two positive integers $n_r, n_c < 1000$, the number of rows and columns of rooms in the restaurant.
- A line with two integers $0 \leq e_r < n_r$ and $0 \leq e_c < n_c$, the row and column of the room which is connected to the entrance.
- A line with two integers $0 \leq k_r < n_r$ and $0 \leq k_c < n_c$, the row and column of the room which is connected to the kitchen (via an elevator).
- n_r lines of $n_c - 1$ non-negative integers < 100. The integer at position j of line i is the area in square meters of the passage between room (i, j) and room $(i, j + 1)$.
- $n_r - 1$ lines of n_c non-negative integers < 100. The integer at position j of line i is the area in square meters of the passage between room (i, j) and room $(i + 1, j)$.

Output

For each test case:

- The minimum cost for dividing the restaurant into two zones such that the kitchen-connected room is in the non-smoking zone and the entrance-connected room is in the smoking zone.

Example

Input	Output
2 1 2 0 0 0 1 1 2 2 0 0 1 1 1 1 3 2	2000 4000

I. Arctic Polar Explorer

The latest expedition to the North Pole has encountered a problem. The autonomous Arctic Polar Explorer robot (APE) needs to reach a higher cliff, but it was never designed for that task. The scientists at mission control needed to improvise and they came up with an ingenious solution: the robot can climb up the cliff by building its own staircase.

The APE's mission takes place in the Arctic Circle, where the landscape is littered with rocks, icebergs and the occasional lost penguin. The APE has collected a bunch of rocks of different sizes, and its goal is now to move the rocks in such a way that they become ordered by increasing size. The size of a rock is proportional to its weight. The rocks lie in a straight line, with one rock per meter. The APE is currently standing next to the first rock, while the cliff is just after the last rock. The robot can move left and right, parallel to the line of rocks. A possible situation is illustrated in figure 3.

The only way of manipulating the outside world is through two arms with grippers that can be used to pick up rocks. Unfortunately, there is no way of measuring the weight of rocks. After another lengthy brainstorm session the scientists at mission control have determined that the weight of the rocks in the grippers can be compared by measuring the tilt of the robot. If the robot tilts to the left then the left gripper contains a heavier rock, and if it tilts to the right then the right gripper contains a heavier rock.

Finally the APE has a very limited amount of memory, making writing programs for it rather difficult. All software for the robot is written in a specialized language, APECODE.

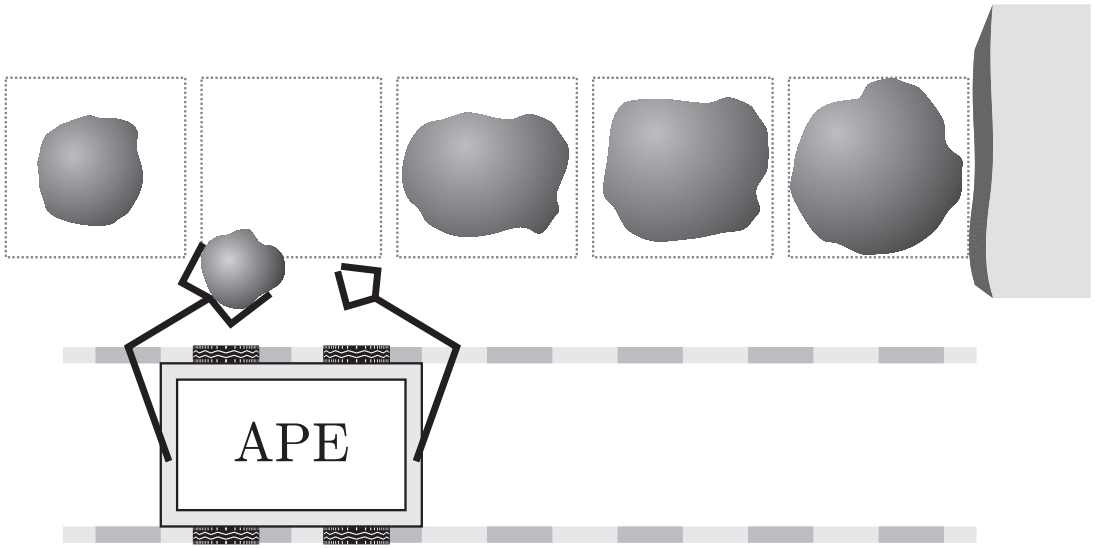


Figure 3: A sketch of a possible situation for the Arctic Polar Explorer. Here it holds a rock in the left gripper, while the right gripper is empty. There are five rocks, and the cliff is to the right.

The Language

Robots programmed in APECODE operate as a stack/state machine. The robot sequentially executes the statements in the current state, and when it reaches the end of a state it starts again from the beginning. APECODE has the following statements:

- With the "`call`" statement the robot jumps into a new state.
- With the "`return`" statement the robot returns to the calling state. (To the point just after the corresponding "`call`" statement.)
- The "`then`" statement executes the following body if and only if the last "`call`" returned "`true`", otherwise the "`else`" branch is executed (if there is any).

Programs must be given as a ⟨PROGRAM⟩ from the following grammar:

$$
\begin{aligned}
\langle \text{COMMENT} \rangle &::= \text{“//”}\ \langle \text{CHARACTER} \rangle^*\ \langle \text{NEWLINE} \rangle \\
&\mid \text{“/*”}\ \langle \text{CHARACTER} \rangle^*\ \text{“*/”} \\
\langle \text{LETTER} \rangle &::= \text{“\_”} \mid \text{“a”} \mid \cdots \mid \text{“z”} \mid \text{“A”} \mid \cdots \mid \text{“Z”} \\
\langle \text{LETTER\_OR\_DIGIT} \rangle &::= \langle \text{LETTER} \rangle \mid \text{“0”} \mid \cdots \mid \text{“9”} \\
\langle \text{NAME} \rangle &::= \langle \text{LETTER} \rangle \langle \text{LETTER\_OR\_DIGIT} \rangle^* \\
\langle \text{PROGRAM} \rangle &::= \langle \text{STATE} \rangle^* \\
\langle \text{STATE} \rangle &::= \text{“state”}\ \langle \text{NAME} \rangle\ \langle \text{STMTS} \rangle \\
\langle \text{STMTS} \rangle &::= \text{“\{”}\ \langle \text{STMT} \rangle^*\ \text{“\}”} \\
\langle \text{STMT} \rangle &::= \text{“call”}\ \langle \text{NAME} \rangle\ \text{“;”} \\
&\mid \text{“return”}\ \langle \text{BOOL} \rangle\ \text{“;”} \\
&\mid \text{“then”}\ \langle \text{STMTS} \rangle \\
&\mid \text{“then”}\ \langle \text{STMTS} \rangle\ \text{“else”}\ \langle \text{STMTS} \rangle \\
\langle \text{BOOL} \rangle &::= \text{“true”} \mid \text{“false”}
\end{aligned}
$$

Comments function as in C: ⟨CHARACTER⟩ is any character, while ⟨NEWLINE⟩ is the end of a line, comments do not nest. Just as in C, whitespace is ignored and names are case sensitive.

The Library

There is a built-in library of states for manipulating the robot. After completing the action, control is returned to the calling state.

- The states `move_left` and `move_right` move the robot one meter left or right respectively.
- The states `pick_up_left` and `pick_up_right` pick up a rock in one of the robot's grippers. It is a fatal error if that gripper already holds something.
- The states `put_down_left` and `put_down_right` place the contents of one of the robot's grippers on the ground. It is a fatal error if the ground around the robot is not clear of rocks.
- The states `if_empty_left` and `if_empty_right` return `true` if and only if the left/right gripper does not hold anything.

- The states `if_tilt_left` and `if_tilt_right` return `true` if and only if the robot is currently tilted to the left/right.
- The state `remember` remembers the return value of the last call, the state `recall` returns the last remembered value.
- The state `trace` can be used for debugging, it outputs a schematic representation of the current situation.

The Compiler

An APECODE compiler is provided for you. This command line program can be invoked with the command

`apecc <program_name>.ape`

This produces an executable named `<program_name>` that *simulates* the Arctic Polar Explorer's behavior.

The Simulator

The simulator produced by the compiler reads from the standard input and writes to the standard output.

On the first line of the simulator's input is a positive integer, the number of test cases. Then for each test case:

- The simulator reads a line containing a single positive integer $n < 5000$, the number of rocks, followed by a line with n positive integers, the weights of each rock in kilograms.
- The robot is placed next to the first rock with both grippers empty.
- The program starts in the state `main` and is run until that state returns.
- The simulator outputs a line containing the weights of the rocks after executing the APECODE program. If a certain place contains no rock, then the character '`-`' is printed.

Example

Input	Output
1 9 7 1 6 3 4 9 2 5 8	1 2 3 4 5 6 7 8 9

Note: You must write a program in APECODE, other programming languages do not work on the Arctic Polar Explorer.