Problem 5 – Bittris

Internal memo from Phobos Research Station's Chief Science Officer.
2 weeks before the explosion.

Now that we have achieved a working AI, we are faced with an interesting challenge - keeping it entertained. Although the artificial brain is extremely efficient in solving quantum physics problems, it seems to get bored quite quickly. We have received reports that it has hacked into personal computers, changed wallpapers to pictures of cats, sent indecent emails to staff members and even tried to compose bad poetry in Microsoft Word. This cannot be tolerated.

After consulting with the station psychologist, I have come to the conclusion that the best way to entertain the AI would be a simple logic game based on the classic Tetris. I have entrusted its development to our lead programmer, in whom I have complete confidence. They say he was a real software ninja back on Earth.

--Dr. K. Sayonara

Problem description

You are tasked with implementing **Bittris**, a slightly modified version of Tetris. It is played on a **4x8 field**, with the 4 rows represented by 4 8-bit integers, whose '0' bits represent **free cells**, and '1' bits - **occupied cells**. Pieces appear on the **top row** and fall down until they reach the **bottom row** or until they **cannot move any lower** without crashing into occupied cells.

The pieces are just sequences of '1' bits, represented as 32 bit integers. Their **low 8 bits** represent the shape of the piece; the **high 24 bits** do not influence the piece's shape, but are important for the scoring (see below). The pieces cannot be rotated, but they can be **shifted left or right** before every move. Note that the **shifting does not affect the high 24 bits**, which means the score for a shape is the same, regardless of whether it has been shifted or not. The shift direction is controlled by the player and occurs immediately **before** the piece tries to move downward.

If the shift direction is such that part of the piece would **leave the playing field**, the command is ignored and the piece simply moves downward. Note that the pieces cannot be shifted on the final row - they land in the position in which they were when they got there. Note also that not all commands need to be used – if the piece has landed before that, they are ignored and the game continues with the next piece.

When a piece lands (reaches its final position), several things can occur. Normally, the player simply **gains points** equal to the number of '1' bits in the piece's input integer (including any leading bits that have no relation to its shape). However, if the row in which it ended up has been filled completely (it contains only '1' bits), then the row must be cleared - all rows above it are **shifted down** by one position, and the score for the move is **multiplied by 10.**

If after the piece has landed, the top row contains any '1' bits, **the game ends**; otherwise the game ends after all pieces in the input have been **exhausted**.
Note that it is completely legal to have a piece that covers an entire row; if it lands on the final row, the row will be cleared normally and the game will continue.

Full example

(">>" represents a line of input, the steps are numbered in the top left of each cell *(1)*, *(2)*, …):

(1)
`>> 19`
 	
`00000000`
`00000000`
`00000000`
`00000000`
 	
score: 0
	(5)
	`>> D`
	
	`00000000`
	`00000000`
	`00000000`
	`01110000`
	
	`score: 0`
		(9)
		`>> L`
		
		`00000000`
		`00000000`
		`00011110`
		`01110000`
		
		`score: 7`
			(13)
			`>> 14`
			
			`00001110`
			`11111000`
			`00011110`
			`01110000`
			
			`score: 12`
				(17)
				`>> D`
				
				`00000000`
				`11111000`
				`00011110`
				`01110000`
				
				`score: 42`

(2)
`>> 112`
 	
`01110000`
`00000000`
`00000000`
`00000000`
 	
`score: 0`
	(6)
	`>> 30`
	
	`00011110`
	`00000000`
	`00000000`
	`01110000`
	
	`score: 3`
		(10)
		`>> 248`
		
		`11111000`
		`00000000`
		`00011110`
		`01110000`
		
		`score: 7`
			(14)
			`>> R`
			
			`00000000`
			`11111111`
			`00011110`
			`01110000`
			
			`score: 12`
				(18)
				`>> D`
				
				`00000000`
				`11111000`
				`00011110`
				`01110000`
				
				`score: 47`

(3)
`>> D`
 	
`00000000`
`01110000`
`00000000`
`00000000`
 	
`score: 0`
	(7)
	`>> D`
	
	`00000000`
	`00011110`
	`00000000`
	`01110000`
	
	`score: 3`
		(11)
		`>> D`
		
		`00000000`
		`11111000`
		`00011110`
		`01110000`
		
		`score: 7`
			(15)
			`>> D`
			
			`00000000`
			`00000000`
			`00011110`
			`01110000`
			
			`score: 42`
				(19)
				`>> 15`
				
				`00001111`
				`11111000`
				`00011110`
				`01110000`
				
				`score: 47`

(4)
`>> D`
 	
`00000000`
`00000000`
`01110000`
`00000000`
 	
`score: 0`
	(8)
	`>> R`
	
	`00000000`
	`00000000`
	`00001111`
	`01110000`
	
	`score: 3`
		(12)
		`>> D`
		
		`00000000`
		`11111000`
		`00011110`
		`01110000`
		
		`score: 12`
			(16)
			`>> 248`
			
			`11111000`
			`00000000`
			`00011110`
			`01110000`
			
			`score: 42`
				(20)
				`>> D`
				
				`00001111`
				`11111000`
				`00011110`
				`01110000`
				
				`score: 51`

The game ends with a score of 51.

Notice that in the first move of the piece represented by the integer "14," the piece is blocked by the piece below it, but manages to avoid it by shifting to the right. In the next move, it lands on a row that gets completely filled and the move after that is ignored because it can't go any further down. The row it lands on is cleared and if the rows above had any occupied cells, they would have been shifted down.

Input

The input data should be read from the console.

On the first line of the standard input, there will be the number **N** – the total number of commands (creating and shifting pieces) which will be entered.

For each next group of 4 lines, the following will be true:

- The first line will be an integer number – describing the current falling piece
- Each of the next 3 lines will contain a single capital letter – 'L', 'D' or 'R', operating on the current falling piece
    - 'L' means "shift left", 'R' means "shift right" and 'D' means just continue down without shifting

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console. On the only output row, you should print the final score of the game.

Constraints

- All input integers are non-negative and will fit into 4 bytes.
- N is divisible by 4 (N%4==0 is always true)
- Allowed work time for your program: 0.1 seconds. Allowed memory: 16 MB.

Example

Example input	Example output
24 112 D D D 30 D R L 248 D D R 14 R D L 248 D D R 15 D R D	51