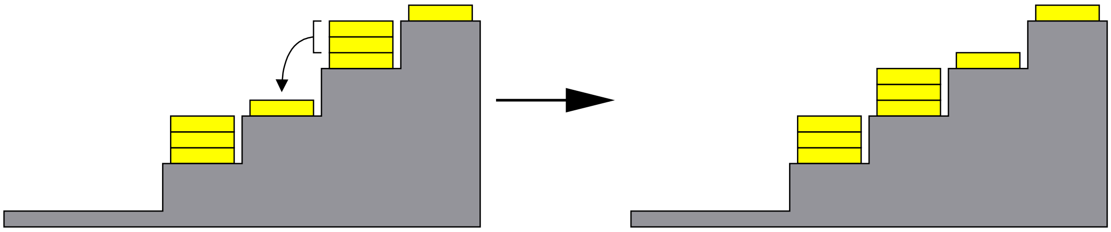
Submit:	`stairs.c` / `stairs.cpp` / `stairs.pas`		Time limit:	on the order of seconds
Input:	(none)		Memory limit:	32 MB
Output:	(none)		Points:	100

When the students of FMFI UK in Bratislava are forced to wait on the stairs, they usually spend the time by playing the following game: The game is designated for two players. At the beginning they put all the coins they have on the stairs. Then they move alternately. In each turn the player chooses one stair. Then he moves some coins from that stair to the stair immediately below it. The player who moves the last coin to the ground under the stairs wins and gets all the coins.



An example of one turn – the student moved 2 coins from the third stair.

Task: Write a program that will play this game. You are given a library playing as your opponent. You will communicate with the library by calling these functions:

Pascal:

You should add a line `uses uStairs;` at the beginning of your program.

```
procedure Init_game(var n:longint; var stairs:TARRAY)      TARRAY=array[0..1000] of longint
procedure Turn(from,coins:longint; var hisfrom,hiscoins:longint)
```

C:

You should `#include "uStairs.h"` in your program.

```
void Init_game(int *n, int *stairs)
void Turn(int from, int coins, int *hisfrom, int *hiscoins)
```

It is recommended to compile your program with the script `compile` which will take care of linking your program.

You are supposed to call the function `Init_game` exactly once – at the beginning of the game. This function will return the number of stairs in the current game (in the variable n) and the number of coins lying on each of the stairs (in the array *stairs*). The number of stairs will be between 1 and 1 000 inclusively. The stairs are numbered from zero (the ground) to n. The initial number of coins on a stair will be between 0 and 1 000 inclusively.

You make your turns by calling the function `Turn`. In the arguments *from* and *coins* you supply the number of the stair you selected and the number of coins you want to move to the lower stair (stair $from - 1$). The library will tell you its following move in the "output" arguments *hisfrom* and *hiscoins*.

You shouldn't end the program. The library does it for you when it realises that you won or lost.

Input and output: Your program should not read or write anything. It only plays the game by communicating with the library. You may assume that during evaluation the library will play optimally. In the test cases that can be won, your program is supposed to win. Otherwise it is supposed to lose correctly.

Example of a game: $n = 4$, $stairs = (47, 2, 1, 0, 1)$.

Player	Move	Position after the turn				
1.	1 coin from 4. stair	47	2	1	1	0
2.	1 coin from 2. stair	47	3	0	1	0
1.	1 coin from 3. stair	47	3	1	0	0
2.	2 coins from 1. stair	49	1	1	0	0
1.	1 coin from 1. stair	50	0	1	0	0
2.	1 coin from 2. stair	50	1	0	0	0
1.	1 coin from 1. stair	51	0	0	0	0

Player 1 won the game.