TASK OVERVIEW SHEET / DAY-1

TASK	Mobiles	Ioiwari	Twofive
Task material directory/Linux	~/mobiles	~/ioiwari	~/twofive
Task material directory/Win98	C:\IOI\mobiles	C:\IOI\ioiwari	C:\IOI\twofive
Time limit per test	1 seconds	1 seconds	0.02 seconds
Memory limit	5MB	32MB	32MB
Compiler options/C and C++	-O2 -static	-O2 –static	-O2 –static
Compiler options/Pascal	-So -O2 -XS	-So -O2 –XS	-So -O2 -XS
Number of Tests	20	25	20
Maximum points per test	5	4	5
Maximum total points	100	100	100

Program header comment when using Pascal	{
	PROG: mobiles
	LANG: PASCAL
	}
		{
		PROG: ioiwari
		LANG: PASCAL
		}
			{
			PROG: twofive
			LANG: PASCAL
			}

Program header comment when using C	/* PROG: mobiles LANG: C */	/* PROG: ioiwari LANG: C */	/* PROG: twofive LANG: C */
Program header comment when using C++	/* PROG: mobiles LANG: C++ */	/* PROG: ioiwari LANG: C++ */	/* PROG: twofive LANG: C++ */
Submission is accepted, if:	The example test case is solved.	Game is played according to the rules – win, lose, or a tie.	The example test case is solved.

The opponent program for reactive tasks runs as a separate process, whose run time is *not* added to the run time of your program.

Mobile phones

PROBLEM

Suppose that the fourth generation mobile phone base stations in the Tampere area operate as follows. The area is divided into squares. The squares form an $S \times S$ matrix with the rows and columns numbered from 0 to S-1. Each square contains a base station. The number of active mobile phones inside a square can change because a phone is moved from a square to another or a phone is switched on or off. At times, each base station reports the change in the number of active phones to the main base station along with the row and the column of the matrix.

Write a program, which receives these reports and answers queries about the current total number of active mobile phones in any rectangle-shaped area.

INPUT AND OUTPUT

The input is read from standard input as integers and the answers to the queries are written to standard output as integers. The input is encoded as follows. Each input comes on a separate line, and consists of one instruction integer and a number of parameter integers according to the following table.

Instruction	Parameters	Meaning
0	S	Initialize the matrix size to $S \times S$ containing all zeros. This instruction is given only once and it will be the first instruction.
1	$X\ Y\ A$	Add A to the number of active phones in table square (X, Y). A may be positive or negative.
2	$L\ B\ R\ T$	Query the current sum of numbers of active mobile phones in squares (X,Y), where $L \le X \le R$, $B \le Y \le T$
3		Terminate program. This instruction is given only once and it will be the last instruction.

The values will always be in range, so there is no need to check them. In particular, if A is negative, it can be assumed that it will not reduce the square value below zero. The indexing starts at 0, e.g. for a table of size 4×4, we have $0 \le X \le 3$ and $0 \le Y \le 3$.

Your program should not answer anything to lines with an instruction other than 2. If the instruction is 2, then your program is expected to answer the query by writing the answer as a single line containing a single integer to standard output.

PROGRAMMING INSTRUCTIONS

In the examples below, the integer `last` is the last one to be read from a line, and `answer` is the integer variable containing your answer.

If you program in C++ and use `iostreams`, you should use the following implementation for reading standard input and writing to standard output:

```
cin>>last;
cout<<answer<<endl<<flush;
```

If you program in C or C++ and use scanf and printf, you should use the following implementation for reading standard input and writing to standard output:

```
scanf ("%d", &last);
printf("%d\n",answer); fflush (stdout);
```

If you program in Pascal, you should use the following implementation of reading standard input and writing to standard output:

```
Read(last); ... Readln;
Writeln(answer);
```

EXAMPLE

stdin	stdout	explanation
`0 4`		Initialize table size to 4×4.
`1 1 2 3`		Update table at (1,2) with +3.
`2 0 0 2 2`		Query sum of rectangle $0 \le X \le 2,\ 0 \le Y \le 2$.
	`3`	Answer the query.
`1 1 1 2`		Update table at (1, 1) with +2.
`1 1 2 -1`		Update table at (1, 2) with -1.
`2 1 1 2 3`		Query sum of rectangle $1 \le X \le 2,\ 1 \le Y \le 3$.
	`4`	Answer the query.
`3`		Terminate program.

CONSTRAINTS

Table size	$S \times S$	$1 \times 1 \le S \times S \le 1024 \times 1024$
Cell value V at any time	V	$0 \le V \le 2^{15}-1$ (= 32767)
Update amount	A	$-2^{15} \le A \le 2^{15}-1$ (= 32767)
No of instructions in input	U	$3 \le U \le 60002$
Maximum number of phones in the whole table	M	$M = 2^{30}$

Out of the 20 inputs, 16 are such that the table size is at most 512×512.

NOTE: The web test facility feeds your input file to your program's standard input.

Ioiwari Game

PROBLEM

The Mancala family of games with beads and pits is among the oldest forms of human entertainment. This task introduces a version of the game especially developed for the IOI. The game is played by two players on a round board with seven pits around the edge. In addition, there is a bank for each player. The game begins by randomly distributing 20 beads into the pits so that each pit contains at least 2 and at most 4 beads. The two players move alternately. To move, the player chooses a non-empty pit and takes all beads out of the pit, and holds them in her hand. As long as there are beads in the player's hand, she considers the pits in clockwise order, starting one after the emptied one, and performs the following operations:

- More than one bead in your hand: If the current pit already contains 5 beads, then take one bead out of the current pit and place it into your bank, otherwise place one bead from your hand into the current pit.
- One bead in your hand: If the current pit contains at least one and at most four beads then move all beads from the pit and the one from your hand into your bank, otherwise (the pit contains 0 or 5 beads) place the bead in your hand into the opponent's bank.

The game is over when after a move all pits are empty and the winner is the player with most beads in her bank.

The starting player always has a winning strategy. You are to write a program, which plays Ioiwari as the starting player and wins. The evaluation opponent plays optimally, that is, once given a chance, it will win and your program will lose.

INPUT AND OUTPUT

Your program reads input from standard input and writes output to standard output. Your program is player 1, and the opponent is player 2. When your program is started, it must first read a line with 7 integers p_1,..p_7: the initial number of beads in pits 1,..7, respectively. The pits are labeled with integers from 1 to 7 in clockwise direction on the board. After this, the game starts with empty banks. Your program should play as follows:

- If it is your program's turn to move, then your program should write the label of the pit describing the move to standard output
- If it is your program's opponent's turn to move, then your program should read the label of the pit defining the move (the pit from which the beads are removed) from standard input.

TOOLS

You are given a program (`ioiwari2` on Linux, `ioiwari2.exe` on Windows), which plays from one initial game position optimally as Player 2. It will first write to

standard output the first line your program is supposed to read, describing the initial values of beads in that game: `4 3 2 4 2 3 2`

After this, the program will play the game, trying to read Player 1's moves from standard input and writing its own moves to standard output. You can run your program and ioiwari2 in separate windows and transfer the conversation manually to both programs. Ioiwari2 records the dialogue in the file `ioiwari.out`.

PROGRAMMING INSTRUCTIONS

In the examples below, you are reading the last integer of the input into variable `last` and the variable `mymove` contains your move.

If you program in C++ and use iostreams, you should use the following implementation for reading standard input and writing to standard output:

```
cout<<mymove<<endl<<flush;
cin>>last;
```

If you program in C or C++ and use scanf and printf, you should use the following implementation for reading standard input and writing to standard output:

```
printf("%d\n",mymove); fflush (stdout);
scanf ("%d", &last);
```

If you program in Pascal, you should use the following implementation of reading standard input and writing to standard output:

```
Writeln(mymove);
Readln(last);
```

EXAMPLE

Here is a correct sequence of 6 moves

	Pit and bank contents after the operation								
Operation/Pit label	1.	2.	3.	4.	5.	6.	7.	Bank1	Bank2
Initial situation	4	3	2	4	2	3	2	0	0
Player 1's move: 2	4	0	3	5	0	3	2	3	0
Player 2's move: 3	4	0	0	4	1	4	0	3	4
Player 1's move: 5	4	0	0	4	0	0	0	8	4
Player 2's move: 4	0	0	0	0	1	1	1	8	9
Player 1's move: 5	0	0	0	0	0	0	1	10	9
Player 2's move: 7	0	0	0	0	0	0	0	11	9

SCORING

If your program wins a test run, then you get 4 points for that test, a tie in a test gives you 2 points for that test, and otherwise you get 0 points for a test.

Twofive

PROBLEM

The secret messages between Santa Claus and his little helpers are usually encoded in the 25-language. The 25-alphabet is the same as the Latin alphabet with one exception - the letter 'Z' is absent, i.e. the 25-alphabet contains 25 Latin letters from 'A' through 'Y' in the same order as the Latin alphabet. Each word in the 25-language consists of exactly 25 different letters. A word can be written in a 5x5 table filling the rows first; for example, the word ADJPTBEKQUCGLRVFINSWHMOXY will be written as follows:

A	D	J	P	T
B	E	K	Q	U
C	G	L	R	V
F	I	N	S	W
H	M	O	X	Y

A valid word in the 25-language has its letters in each row as well as in each column written in ascending order. Thus, the word ADJPTBEKQUCGLRVFINSWHMOXY is a valid word, in contrast to the word ADJPTBEGQUCKLRVFINSWHMOXY (the ascending order is violated in the second column, and in the third column, too).

Santa Claus has a lexicon. His lexicon is the list of all valid 25-language words in ascending order (lexicographically) along with their ordinal numbers starting from 1. For example, in the lexicon ABCDEFGHIJKLMNOPQRSTUVWXY is the word number 1 and ABCDEFGHIJKLMNOPQRSUTVWXY is the word number 2. In word number 2, U and T are interchanged from their order in word number 1.

Unfortunately, this lexicon is huge. Write a program that determines the ordinal number of an arbitrary given word, and also the word corresponding to a given ordinal number. There are no more than 2^{31} words in the lexicon.

INPUT

The input file is named `twofive.in` and consists of two lines. The first line contains a string with one character: a 'W' or an 'N'. If the first line contains a 'W', then the second line contains a valid 25-language word, that is, a string with 25 characters. If the first line contains an 'N', then the second line contains the ordinal number of an existing 25-language word.

OUTPUT

The output file is named `twofive.out` and consists of one line. If the second line of the input file contains a 25-language word, then the line of the output file contains the ordinal number of that word. If the second line of the input file contains a number, then the line of the output file contains the 25-language word with that ordinal number.

EXAMPLE INPUTS AND OUTPUTS

twofive.in

```
W
ABCDEFGHIJKLMNOPQRSUTVWXY
```

twofive.out

```
2
```

twofive.in

```
N
2
```

twofive.out

```
ABCDEFGHIJKLMNOPQRSUTVWXY
```