

Exercise 2: Design patterns

Issued: **May 10, 2007**. Return for corrections: **May 24, 2007**.

The task is to write an undo-redo mechanism for a simple system. The example is a Square Manipulation System (SMS).

Input

The program reads its data from the standard input. That data is in the form of successive lines, each of one of the following forms, where i , j and k are non-negative integers:

C i j

T i j

M i j k

S i j

U

R

P

Your program may assume that the input follows this syntax; in other words, error handling is not requested.

In addition, for the C, T, M and S cases your program may ignore any line for which i is greater than 1000. (HINT: this facilitates the choice of data structure and avoids having to perform a sorting step for the P command, see below.)

Semantics

Note that no graphical display is required. All information about squares will be output textually (see the P command).

The C command (Create) creates a square numbered i , of side length j , centered at the origin of coordinates, with sides horizontal and vertical. If there already was a square numbered i , it is no longer considered part of the system; the new one takes over its number. As noted above, you may ignore any C command for which i is greater than 1000.

The T command (Turn) rotates square numbered i by j degrees. If there is no square numbered i , the command has no effect.

The M command (Move) moves the square numbered i by j pixels horizontally and k pixels vertically. If there is no square numbered i , the command has no effect.

The S command (Scale) multiplies the size of the square numbered i by j . If there is no square numbered i , the command has no effect.

The U command (Undo) cancels the last not-yet-undone C, T, M or S command. If none remains to be undone, it has no effect.

The R command (Redo) is applicable only if the last executed command was U or R. In that case it re-executes the most recent D, T or M command undone and not yet redone. If not, it has no effect.

(HINT: U and R behave as undo and redo as present in most modern interactive systems, often through shortcuts such as CTRL-Z for undo and CTRL-Y for redo.)

The P command (Print) prints on the standard output the current details of all points created so far,

in the order of their numbers, one per line. Each line should have the following exact format:

`i j k l`

where *i* is the square number, *j* its horizontal coordinate, *k* its vertical coordinate, and *l* its angle from the horizontal axis. All are integers; in addition, *l* should be between 0 and 359 inclusive. The values should all be separated by a single space.

Reading input

To read the input, you may use the following standard library mechanisms: `STD_FILES` and `FILE`

Hint

The standard solution for undo-redo, which you are encouraged to use, is documented in both of the textbooks recommended for this course: Meyer ("Undo-Redo") and Gamma et al. ("Command pattern"). The idea is that the basic abstraction is the notion of undoable command. You should have a class `COMMAND` whose instances each represent the result of executing an undoable command (here one other than U, R or P); for each command kind, define a descendant class of `COMMAND`, for example `ROTATE`, with attributes (fields) containing just enough information to undo and redo one execution of the command. For example, in a text editor, a `COMMAND` object for a "delete a line" command would contain attributes defining the content of a line and its position in the file. Then your program should contain a "history list" - a list of `COMMAND` objects that can be traversed back and forth to process a succession of U and R.

Note that there is no limit on the number of input lines. We will test the output of your program on various sample inputs; to facilitate testing, it is important that you follow the exact output format mentioned above.