Assignment 1

Due Date: Monday, February 6th 2017, 11:59pm

Objectives

The purpose of this assignment is twofold. First, I would like to evaluate your C abilities after leaving COP 3223. Second, I would like to challenge you to write a program in C that requires organization and planning.

View this assignment as a stepping stone to writing larger scale C programs. Throughout this course, we will be developing larger scale programs and designing algorithms to tackle problems more efficiently. To do that, we need to develop your implementation skills.

Deliverables

Submit two files.

The file named "`stamp.h`" is your header file. Place all function declarations in this header with comments on how you designed the header. You may use the header I provide or design your own. In either case, include a header.

The file named is "`stamp.c`" is your source file. Implement all functions defined in the header file. Include a main function in this file.

All input should be read from a file named "`stamp.txt`". The input specification will define precisely what is allowed for input to your program. You need not guard against cases not allowed in the input.

Scoring

I will run your program multiple times on multiple test files. Your grade will be awarded purely based on the number of tests you pass. Be sure to test your program on more than just the sample cases provided.

To guard against infinite loops and inefficient implementations, your program will be run for x seconds per test case. Upon grading, you will receive a "`results.txt`" file detailing which tests where passed and which failed. Here is a table of possible results:

Error Code	Meaning
AC	Accepted! :D All is wonderful in the world and your code worked correctly.
WA	Wrong Answer: The code executed but produced faulty output.
RTE	Runtime Error: The code crashed spewing memory and guts everywhere.
TLE	Time Limit Exceeded: I killed the program after x seconds. I feel betrayed.
CTE	Compile Time Error: Your code did not compile. ☹

Stamp

`filename: stamp`
`timelimit: 3 seconds`

In this problem, I will specify a stamp simulator. You will read in a database of stamps, followed by stamp commands. Following this declaration, will be a declaration of page size and a sequence of stamping commands. Your task is to print an ASCII art representation of the page after it has been stamped some number of times. Print '`.`' (a period) for any cell not containing any ink.

Input "stamp.txt"

The first line of the input file contains a single integer n ($1 \leq n \leq 100$) representing the number of stamps to process.

This is followed by n descriptions of stamps. The first line of a stamp description starts with a single integer t ($1 \leq t \leq 2$), representing the type of stamp being specified.

For a type 1 stamp, the line contains two more integers r and c ($1 \leq r, c \leq 100$), representing the size a stamp takes up. This is followed by r rows of c characters each. Each of these character will be uppercase or lowercase letters, a digit, or a symbol from "`!@#$%^&*(){}[];<>.,/~\|`". A period represents a cell that should not be stamped.

A type 2 stamp is an aggregate stamp. This is effectively the application of several stamps one after another, even other aggregate stamps! The first line additionally contains one more integer m ($1 \leq m \leq 100$). This is followed by m lines. The i^{th} line contains three integers $a_i\ r_i\ c_i$ where ($1 \leq a_i < k, 0 \leq r_i, c_i \leq 100$) The integer a_i represents the i^{th} stamp that should be applied in an aggregate stamp. The integer k is the id from $[1, n]$ representing which stamp has been read in the order of the input file. In other words, an aggregate stamp k is only composed of stamps specified before stamp k. The characters r_i and c_i represent the row and column offset the specified stamp must be applied from the upper left corner.

After the description of the stamps, follows a line with three integers $c\ r_p\ c_p$ ($1 \leq c \leq 100$, $1 \leq r_p \leq 1200$, $1 \leq c_p \leq 800$), representing the number of stamping commands and the size of the paper in rows and columns, respectively.

For the following c lines, i^{th} line contains three integers $a_i\ r_i\ c_i$ ($1 \leq a_i \leq n, -1200 \leq r_i \leq 1200, -800 \leq c_i \leq 800$), representing which stamp to apply, the row offset of the stamp and the column offset of the stamp relative to row 1 column 1.

Rows are specified from top to bottom. Columns are specified from left to right.

Output (stdout)

Output r_p lines each containing c_p characters. If a cell of the page remains unmarked, print the character '**.**'. For each line output, be sure to print the end-line character. If a stamping goes off the page, only print the characters present on the page.

Sample 1

Input	3 1 3 4 .#.@ #.#. .#.@ 1 2 2 ** ** 1 5 1 + + + + + 6 5 7 2 0 0 2 3 3 3 0 2 3 0 4 1 0 0 1 0 3
Output	*#+@#.@ #*##+#. .#+@#.@ ..+*+.. ..+*+..

Sample 2

Input	4 1 2 2 ** ** 1 2 3 +#+ .+. 2 2 2 0 0 2 1 0 2 3 1 0 0 1 2 2 1 0 4 6 9 9 4 0 0 4 4 0 4 8 0 3 0 7 3 4 4 3 6 6
Output	**..**.+# **..**.+# ..**....+ ..**..... **..+#+.. **..+#+.. ..**.++#+ ..**..+#+ **..**.+.

Tips

I recommend implementing a version of this program that only handles type 1 stamps correctly first. After you can get that version working, start implementing type 2 stamps. If you can only work with type 1 stamps, turn in that version of the code! Some of the tests will only contain type 1 stamps.

If something in the problem appears unclear, please come see me or send me an email. I'll do my best to clarify either the specification or provide further clarification in class or through WebCourses.