Organization

Objective
Give practice with dynamic memory in C.

Story
You managed to round up all the animals. Now you need to structure your park. You and your employees need to be able to look up information quickly, so you will write a program to help assist you and your "assistant to the park managers". Again the park is composed of multiple sections, and each section will have multiple cages. You plan on doing the following,

- Changing the number of cages within particular park sections
- Adding and removing animals to different cages within particular sections
- Listing the animal within a particular cage of a particular park section

Problem
Given a list of commands determine at particular points what animal is contained in a requested cage.

Input
Input will begin with a line containing 1 integer, ***N*** ($1 \leq N \leq 100{,}000$), representing the number of park sections. Each park section initially contains 0 cages.

Following this line will be list of commands. The commands must be processed in the order they are given. Each command begins with a number and can be 1 of 4 types. The types are listed below

- `1 S C`
    - This command changes the number of cages in some particular section. The section that is modified is the section numbered `S` (1-indexed). The resulting number of cages in section `S` is the number `C`. This could increase or decrease the number of cages. Any animal that was in a cage with ID after `C` (1-indexed) is sent back to the warehouse. If the section number is invalid, no changes should be made.
- `2 S C A`
    - This command adds an animal to a cage. The section that is modified is the section numbered `S` (1-indexed). The resulting cage the animal is added to is `C` (1-indexed), if the cage is empty and the section and cage exists. If an animal is already present in cage `C` of section `S` or there is no section `S` or cage `C` of section `S` does not exist, then the animal is NOT added to the cage. The type of animal is `A` (a string of at most 1000 alphanumeric characters).
- `3 S C`
    - This command requests the animal type of the animal in cage `C` (1-indexed) of section `S`. If no animal is present or the section or cage number is invalid, the phrase "`No animal found.`" (quotes for clarity) should be printed instead.
- `4`
    - This command requests that the program ends (I guess we finished organizing all the exhibits).

Output

Output should contain a line for each command of type 3. The line should either contain the animal type if the request was valid or the string “`No animal found.`”

Sample Input	**Sample Output**

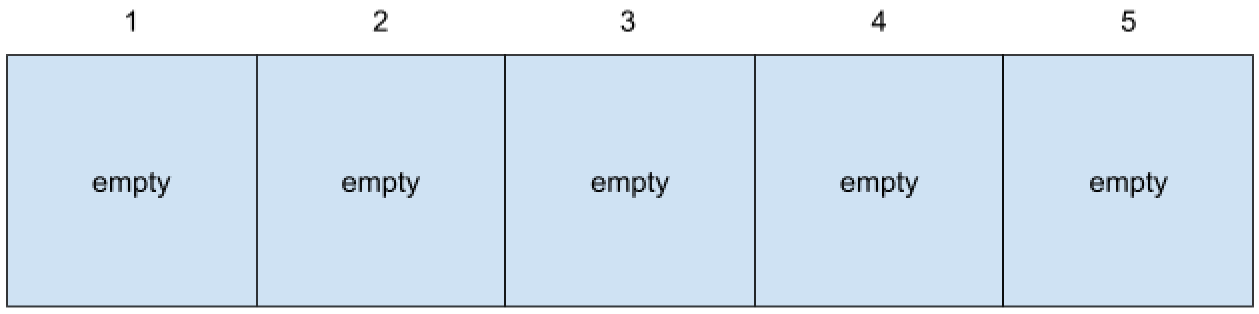
`5`
`1 5 10`
`2 5 8 cat`
`2 5 9 dog`
`3 5 9`
`3 5 8`
`1 5 8`
`3 5 8`
`3 5 9`
`4`
	`dog`
	`cat`
	`cat`
	`No animal found.`

`2`
`1 1 1`
`3 1 1`
`2 1 1 Qawous`
`3 1 1`
`4`
	`No animal found.`
	`Qawous`

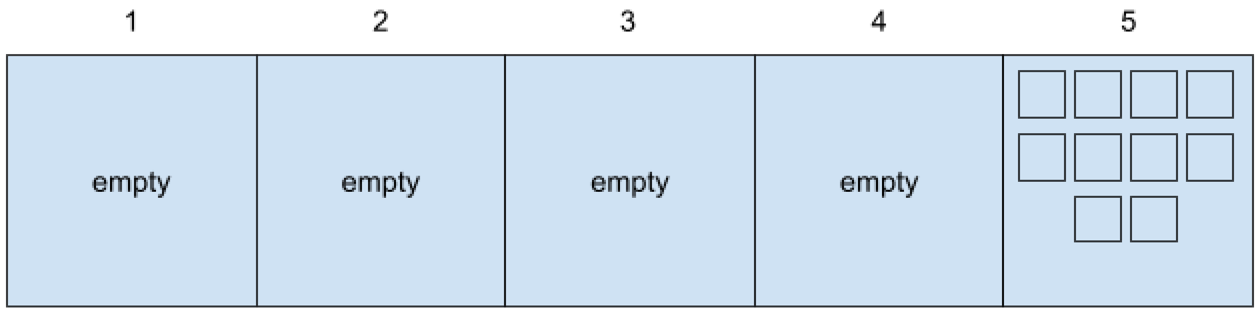
Explanation

Case 1

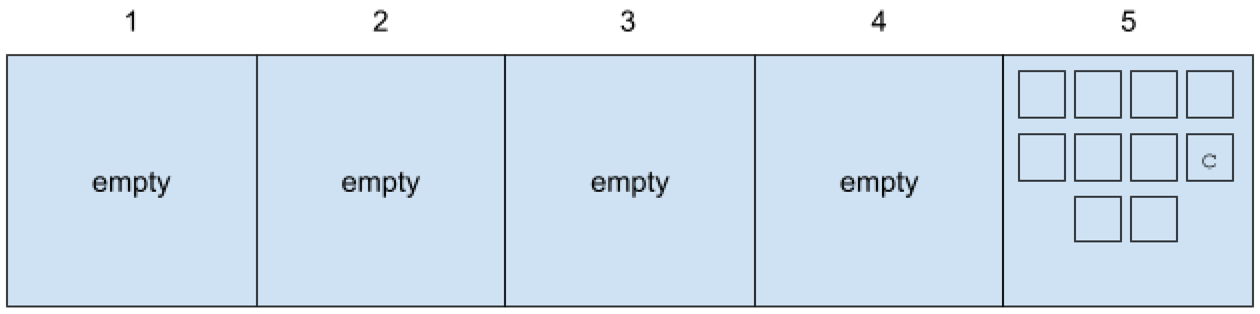
The first line says that we have 5 sections. Each section is empty.



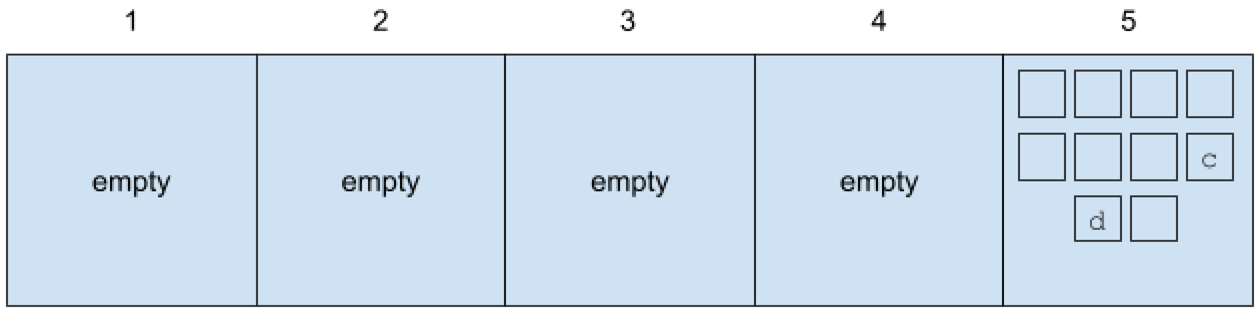
The second line (1 5 10) says that section 5 will now have 10 cages (all of which will be empty)



The next line (2 5 8 cat) adds a cat to the 8th cage of section 5.



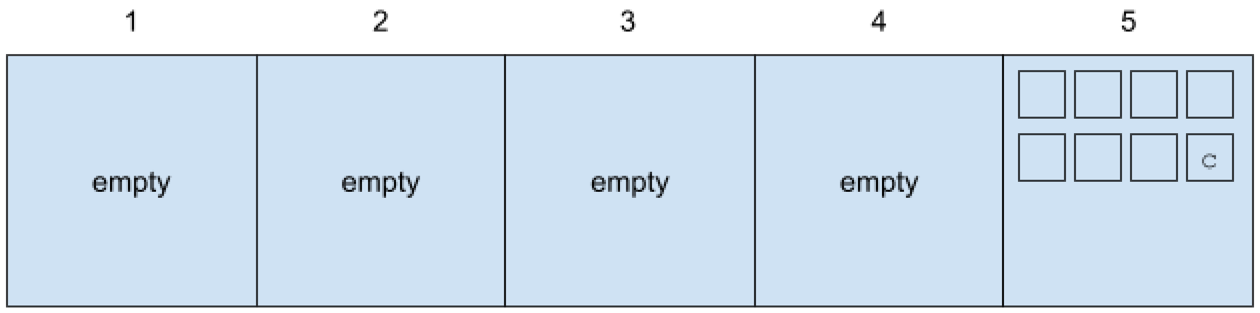
The next line (2 5 9 dog) adds a dog to the 9th cage of section 5.



The next line (3 5 9) looks up the animal in the 9th cage of section 5, which is a dog. Thus the output's first line is `dog`.

The next line (3 5 8) looks up the animal in the 8th cage of section 5, which is a cat. Thus the output's second line is `cat`.

The next line (1 5 8) reduces the number of cages in section 5 to 8. This removes the dog from section 5.



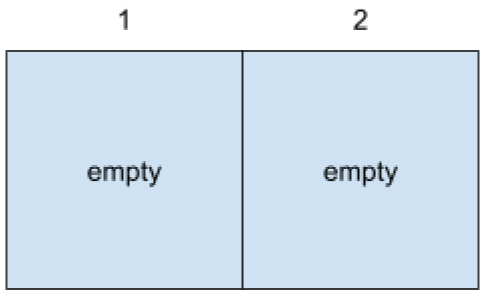
The next line (3 5 8) looks up the animal in the 8th cage of section 5, which is still a cat. Thus the output's third line is `cat`.

The next line (3 5 9) looks up the animal in the 9th cage of section 5, but there is no 9th cage in that section. Thus the output's fourth line is `No animal found.`

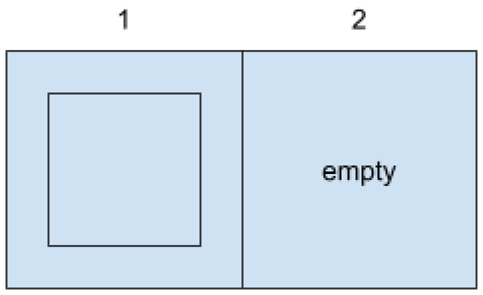
Note the period is contained in the output.

The last line of the first case (4) tells the program to stop running.

Case 2
The first line of input (2) tells us that there are going to be 2 sections. Both sections start empty.

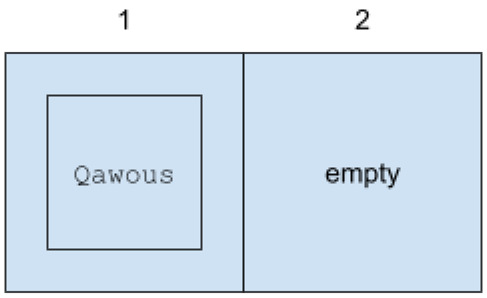


The second line of input (1 1 1) tells us that section 1 now has 1 cage.



The third line of input (3 1 1) requests the animal in the first cage of the first section, but since the cage is empty, we print `No animal found.`

The fourth line of input (2 1 1 Qawous) adds a Qawous to the first cage of the first section.



The fifth line of input (3 1 1) requests the animal in the first cage of the first section, which is a `Qawous`.

The last line of input (4) tells the program to end.

Helpful Hints/Recommendations

Reading First Line: I recommend reading in the first number using traditional scanf. If you want to you could instead read the first line using fgets or gets, but then you need to use a method to parse the integer from the string you extract. Additionally, it is not too difficult to read the remaining lines with scanf.

Multiple Following Lines: After reading the first line, you need to process all the remaining lines until a 4 command is reached. Since you will need to process multiple commands I recommend using some loop to accomplish this task.

Reading Command Types: You can read the type of command (the first number on a line) using a scanf. Then use a series of if statements or a switch statement to determine how (if at all) to read the remainder of the command. The rest of the command can be read in using some following scanf.

Null Terminator: Keep in mind that if you are using character arrays as strings in C you need to make enough space for the Null Terminator at the end of the string.

Storage: To avoid creating too much memory, you should use dynamic memory to keep track of **both** the cages and the names within each cage. I recommend using a struct for each section (using a struct is 5 points of the rubric). See the following code snippet for my recommendation.

```
struct Section
{
    struct Cage * cages;
    int numCages;
};
```

Your main function you would look like the following

```
int main()
{
    struct Section * allSections;

    // Read in the number of sections

    allSections = allocation size based on the number of sections;

    // Handle commands

    // Clean up remaining memory

    return 0;
}
```

Grading Criteria

- Read/Write from/to **standard** input/output (e.g. scanf/printf and no FILE *)
    - 10 points
- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
    - 10 points
- Use structs
    - 5 points
- Process a different number of tokens per line dependent on the first value of the line.
    - 10 points
- Programs will be tested on 10 cases
    - 5 points each

No points will be awarded to programs that do not compile using “`gcc -std=gnu11 -lm`*”.*

Sometimes a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem your solution must use dynamic memory. ***Without this, programs will earn at most 50 points!***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

No partial credit will be awarded for an incorrect case.