

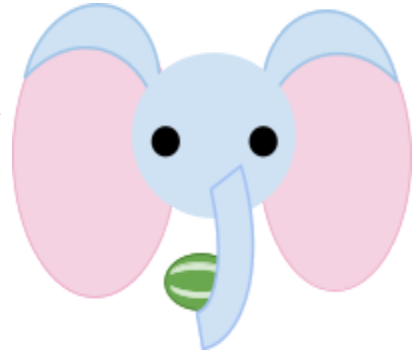
Elephant Extraction

Objective

Give additional practice with Dynamic Memory Allocation in C.
Give practice with Array Lists in C.
Give practice with Stacks in C.
Give practice with grids in C.

Story

It looks like the wrong bait was used to attract the raccoons. The bait used was watermelons and now there are a bunch of elephants running amok around town. You have offered your service to help determine the location of the elephants.



The town can be modeled as a 2 dimensional grid of R by C intersections. You know of the initial location of every elephant in town as required of your job as the IT specialist of the KAT pet shop. The employees not wanting to get in any further trouble have finally given you the full information as to where and when the baits were placed.

The elephants are lazy but greedy and at each hour they will move to the adjacent intersection with the most number of watermelons. If there are more watermelons in their current location than any adjacent intersection, then the elephant will stay where it is located. Once all the elephants have reached their desired intersection, they will eat all the watermelons present if any. If no location has watermelons they begin to walk back to their starting location, but only by retracing their steps.

Elephants have a really good memory and they will remember all the directions they took to get to where they are currently located. However, elephants are not the brightest, and will retrace their steps even if it will take them eventually to the same location they are currently located. For example if an elephant moves North to eat some watermelons, and in the next hour moves South to eat some watermelons, when the elephant decides to return home, they will move North in the first hour and South in the second hour ending up in their initial location.

Your bosses are concerned that some of the elephants might have consumed too many watermelons too quickly. To help them monitor the watermelon intake you want to print how much watermelon was consumed in each hour.

Problem

You will be given the initial locations of all the elephants, and a series of commands representing the occurrence of the change of an hour or the placement of bait. You need to determine the amount of bait consumed at each hour and the ending locations for all the elephants.

Input

The first line of input will contain a single integer, N ($1 \leq N \leq 10,000$), representing the number of elephants to track. The following N lines will each contain the initial location of the corresponding elephant as 2 integers, R and C ($1 \leq R, C \leq 500$), representing the row and column of the location in the grid.

Following this will be a series of command in one of the following 3 formats,

- `BAIT R C A`
 - Which represents A additional watermelons placed at the R -th row and the C -th column in the grid. ($1 \leq R, C \leq 500$; $1 \leq A \leq 1,000$)
- `HOUR`
 - Which represents the passing of an hour.
- `QUIT`
 - Which represents the end of the commands.

Output

For each `HOUR` command your program should print a line that contains the number of watermelons consumed with the passing of the hour.

After the `QUIT` command your program should print the resulting locations of the elephants in the order they were given in input.

Sample Input	Sample Output
3 1 4 7 3 10 8 BAIT 2 1 10 BAIT 1 3 5 BAIT 7 1 1 HOUR BAIT 1 4 2 HOUR HOUR QUIT	5 2 0 1 3 7 3 10 8
2 1 2 2 4 BAIT 2 2 2 BAIT 2 3 3 BAIT 2 4 2 BAIT 2 1 2 HOUR HOUR QUIT	4 3 2 3 2 3
Continued on the next page.	Continued on the next page.

1	0
10 10	0
HOUR	10 10
HOUR	
QUIT	

Sample Explanation

FOR CASE 1

There are 3 elephants. Below is a picture representing their initial locations


							e		
		e							
			e						

3 baits are placed before the end of the first hour. The locations of the baits looks like the following,

							e		
1		e							
10									
		5	e						


When the first hour ends the elephant at location (1, 4) moves to location (1, 3) to eat the bait. No other elephant will move. A total of 5 watermelons are consumed. Below is the resulting grid after the movement,

							e		
1		e							
10									
		e							



A new bait of 2 melons is placed at location. The result looks like the following,

							e		
1		e							
10									
		e	2						



When the next hour progresses the bottom elephant moves to the 2 melons. On the next page is the resulting grid,

							e		
1		e							
10									

Note that the elephant in row 1 will memorize the path that took them to this location rather than realize that it is back where it started. When the next (and final) hour progresses the bottom elephant moves towards the west once. Below is a picture of the final positions of the elephants.

							e		
1		e							
10									

The final locations for the elephants are (1,3), (7,3) and (10, 8)

FOR CASE 2

Below is a picture of what grid looks like just before the end of the first hour,

2 wm	2 wm	3 wm	e_2 2 wm	
	e_1			

The next hour looks like the following,

2 wm	e_1 0 wm	3 wm	e_2 0 wm	

The end of the next hour looks like the following,

2 wm	0 wm	$e_1 e_2$ 0 wm	0 wm	

The program ends in this arrangement.

FOR CASE 3

There are no baits placed for the 1 elephant, so they stay in place for 2 hours. No watermelons are consumed for the 2 hours, and their resulting location is the same as the original (10, 10).

Hints

2D Array: The 2D grid should be stored in a 2D integer array. This is achievable because the maximum value of the x and y are relatively small. I won't give cases that put enough melons in one spot to overflow the `int` type.

Stack Usage: A stack should be used to hold onto all the locations (or paths) an elephant has taken to leave their initial location. When the elephant backtracks the previous location should be removed from the stack. When the elephant travels to a location, a new location should be pushed onto the stack. If the elephant eats melons at their location or if the elephant is at the initial location, then the stack should not be modified.

NO PROMPTS: Do NOT prompt for input. Do not print messages like, "Please enter the number of cats:" or the like.

NO LABELS: Do NOT label the output. Do not print messages like, "The cat in cozy cage 1 is...".

Struct Recommendations: I recommend using the following structs,

```
struct Node {
    int r, c;
    Node * next;
};

struct Stack {
    Node * head;
};

struct Elephant {
    Stack memory;
};
```

Function Recommendations: Below is a list of the prototypes of the functions used by the instructor's solution,

```
Node * createNode(int row, int col);
void push(Stack * stk, int row, int col);
void pop(Stack * stk);
void top(Stack * stk, int * row_ptr, int * col_ptr);
void addBait(int ** grid, int row, int col, int amt);
int eatBait(int ** grid, int row, int col);
void move(Elephant * ele_ptr, int ** grid);
int eat(Elephant * ele_ptr, int ** grid);
int progress_hour(Elephant * ele_arr, int num_ele, int ** grid);
int main();
```

Input Output Syncing: You DO NOT need to sync the input and output. Below is an example of what the execution of the program could look like,

```
~$ ./a.out
2
1 2
2 4
BAIT 2 2 2
BAIT 2 3 3
BAIT 2 4 2
BAIT 2 1 2
HOUR
4
HOUR
3
QUIT
2 3
2 3
~$
```


Grading Criteria

- Read/Write from/to **standard** input/output (e.g. scanf/printf and no FILE *)
 - 5 points
- Good comments, whitespace, and variable names
 - 15 points
- No extra input output (e.g. input prompts, "Please enter the number of cats:")
 - 5 points
- Loops to the end input
 - 5 points
- Track the locations in a stack
 - 5 points
- Uses the command type to determine if more values should be read from the user
 - 5 points
- Moves all elephants before letting any elephant eat at each hour
 - 5 points
- Increments the bait placed at a location rather than setting it to new given value
 - 5 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 use a stack. **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.