CMPUT 274/5 - Tangible Computing **Winter 2020**
Morning Problem: Pointer Simulator

Description

Thank you for your help with pointer chasing! Unfortunately, I have run into another confusing problem.

This time, I have been writing some programs. In each program, I allocate a region of memory and set it all to zero. As the program runs, I store values in memory and create pointers to those values, and sometimes pointers to those pointers. Then, I choose any pointer and "chase" it, dereferencing as many times as needed to find the value I am looking for. I might chase many different addresses, and I want to know the result of following each one.

However, my programs sometimes cause cycles, where some pointers point back to addresses that have already been seen. What is worse, sometimes I accidentally store addresses that are outside the bounds of the allocated memory region. Then, when chasing, I get the most peculiar error: `segmentation fault`.

Can you help me debug my programs by telling me which chase instructions lead to cycles and which ones lead to an out of bounds memory access?

Input

The first line of input will contain three space-separated integers S, A, N satisfying $0 \leq S < 2^{31}$, $1 \leq A \leq 1000$, and $1 \leq N \leq 1000$ and, additionally, $S + A \leq 2^{31}$.

- S: the starting address of the allocated region of memory
- A: the size of (number of elements in) the allocated region in memory
- N: the number of instructions in the program

The following N lines will each contain a single instruction, in one of three forms:

(a) `S addr val`
Store the negative integer value `val` at the address `addr`.

(b) `P addr1 addr2`
Store `addr1` point to `addr2`. That is, store `addr2` into the position represented by `addr1`.

(c) `C addr`
Chase the pointer `addr`. Starting at the memory location `addr`, dereference the pointer to check the value stored at its location in memory. If the value is also a memory address, dereference again and repeat until you either find a value that is negative (*i.e.*, not a pointer) or a value that causes you to index out of bounds of the allocated region of memory.

At the start of your program, the array has 0s in every position.

You are also guaranteed that all `P` and `S` operations have the first operand being a valid

address within the bounds of the the array.

Output

Your output should contain c lines, one for each pointer chasing (`C`) instruction. Each one should result in one of the following messages being printed:

1. the negative value v found at the end of the chase
2. `There was a cycle` - if, naturally, the chase does not terminate
3. `Out of bounds` - if the chase ends up indexing outside of the array

You should not print any output for the storing (`S` and `P`) instructions.

Comment: It is ok to print out the message after processing a chase before reading in other instructions.

Sample Input 1

```
8000 6 3
P 8000 8004
S 8004 -1
C 8000
```

Sample Output 1

```
-1
```

Explanation: The initial array in memory looks like this:

$$[0, 0, 0, 0, 0, 0]$$

After the first two instructions, it becomes:

$$[8004, 0, 0, 0, -1, 0]$$

Because the address 8004 is stored at address 8000 (the first position), and the negative value -1 is then stored at address 8004. Therefore, when chasing the address 8000, we jump to 8004 and return the value -1.

Sample Input 2

```
8000 6 8
P 8005 8001
P 8001 8005
C 8001
P 8003 8004
S 8001 -5
P 8004 8000
P 8000 8006
C 8003
```

Sample Output 2

```
There was a cycle
Out of bounds
```

Explanation: At the point when the first chase instruction is executed, the array in memory stores the following:

$$[0, 8005, 0, 0, 0, 8001]$$

The chase beginning at address 8001 enters an infinite cycle.

When the second chase instruction is read, the memory is:

$$[8006, -5, 0, 8004, 8000, 8001]$$

When the chase is called from address 8003, we go to position 8004, which stores address 8000. At 8000 we attempt to go to address 8006, but this is out of bounds (valid addresses are 8000 to 8005).

Sample Input 3

```
8000 6 8
P 8005 8003
P 8003 8001
S 8001 -5
C 8005
P 8002 8008
S 8003 -2
C 8002
C 8005
```

Sample Output 3

```
-5
Out of bounds
-2
```

Explanation: When the first chase instruction is executed, the array contains:

$$[0, -5, 0, 8001, 0, 8003]$$

and chasing from address 8005 leads to the negative value -5.

At the second and third chase instructions, the array contains:

$$[0, -5, 8008, -2, 0, 8003]$$

Chasing 8002 results in an out of bounds memory access because 8008 is not within valid range. We chase 8005 again, but this time it leads to address 8003 which now stores the value -2, which is printed.