Maximum Element

locked

by saikiran9194

Editorial by dcod5

The approach is pretty straight forward. We keep two stacks: One is for pushing and popping elements as they come and go (*elements stack*), and the other is for keeping track of the maximum element (*maximum elements stack*).

When the first element arrives, push it onto both of the stacks. As the elements may not be distinct, we need to keep track of their indices. So, declare a stack of $pair < int, int >$ or use a structure.

We maintain the maximum elements stack so that the maximum element till now is on the top. Whenever we push an element, it normally goes to the elements stack, but we also push it to the maximum elements stack if, and only if, it is greater than the current maximum element.

Statistics

Difficulty:	Easy
Time Complexity:	O(N) Required
Knowledge:	Stacks
Publish Date:	Dec 17 2015

Now, when an element is to be deleted, we pop it directly from the elements stack. We also check if that particular element is the maximum element or not. We do so by comparing the value and the index of the popped element with the element on top of the maximum elements stack. If they are equal, we pop that element as well.

Finally, to answer the maximum element query 3, we print the element on top of the maximum elements stack.

Tested by dcod5

Problem Tester's code :

```
#include <iostream>
#include <vector>
#include <cstring>
#include <stack>

using namespace std;

int main()
{
    stack<pair<int,int> > ele, maxi;
    int n, i = 1;
    cin>>n;
    while(i <= n)
    {
        int t, v;
        pair<int, int> p1, p2;
        cin>>t;
        if(t == 1)
        {
            cin>>v;
            ele.push(make_pair(v, i));
            if(maxi.size() == 0)
            {
                maxi.push(make_pair(v, i));
            }
            else
            {
                p1 = maxi.top();
                if(p1.first < v)
                {
                    maxi.push(make_pair(v, i));
                }
            }
        }
        else if(t == 2)
        {
            p1 = ele.top();
            ele.pop();
            p2 = maxi.top();
            if(p1.first == p2.first && p1.second == p2.second)
            {
                maxi.pop();
            }
        }
        else
        {
            p1 = maxi.top();
            cout<<p1.first<<endl;
        }
        i++;
    }
    return 0;
}
```

Equal Stacks

locked

by nabila_ahmed

You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.

Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they're all the same height, then print the height. The removals must be performed in such a way as to maximize the height.

Note: An empty stack is still a stack.

Input Format

The first line contains three space-separated integers, n_1, n_2, and n_3, describing the respective number of cylinders in stacks 1, 2, and 3. The subsequent lines describe the respective heights of each cylinder in a stack *from top to bottom*:

- The second line contains n_1 space-separated integers describing the cylinder heights in stack 1. The first element is the top of the stack.

- The third line contains n_2 space-separated integers describing the cylinder heights in stack 2. The first element is the top of the stack.
- The fourth line contains n_3 space-separated integers describing the cylinder heights in stack 3. The first element is the top of the stack.

Constraints

- $0 < n_1, n_2, n_3 \leq 10^5$
- $0 < \textit{height of any cylinder} \leq 100$

Output Format

Print a single integer denoting the maximum height at which all stacks will be of equal height.

Sample Input

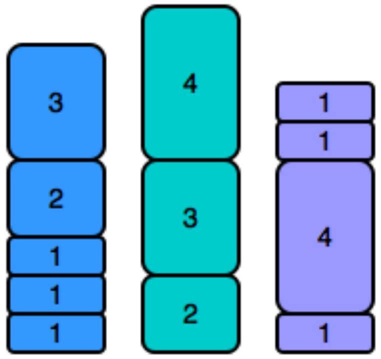
```
5 3 4
3 2 1 1 1
4 3 2
1 1 4 1
```

Sample Output

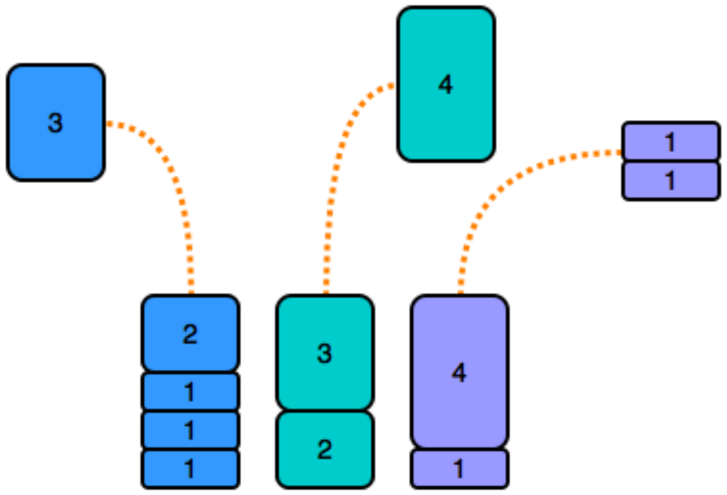
```
5
```

Explanation

Initially, the stacks look like this:



Observe that the three stacks are not all the same height. To make all stacks of equal height, we remove the first cylinder from stacks 1 and 2, and then remove the top two cylinders from stack 3 (shown below).



As a result, the stacks undergo the following change in height:

1. $8 - 3 = 5$
2. $9 - 4 = 5$
3. $7 - 1 - 1 = 5$

All three stacks now have $height = 5$. Thus, we print 5 as our answer.

Submissions: 22
Max Score: 25
Difficulty: Easy

Rate This Challenge:

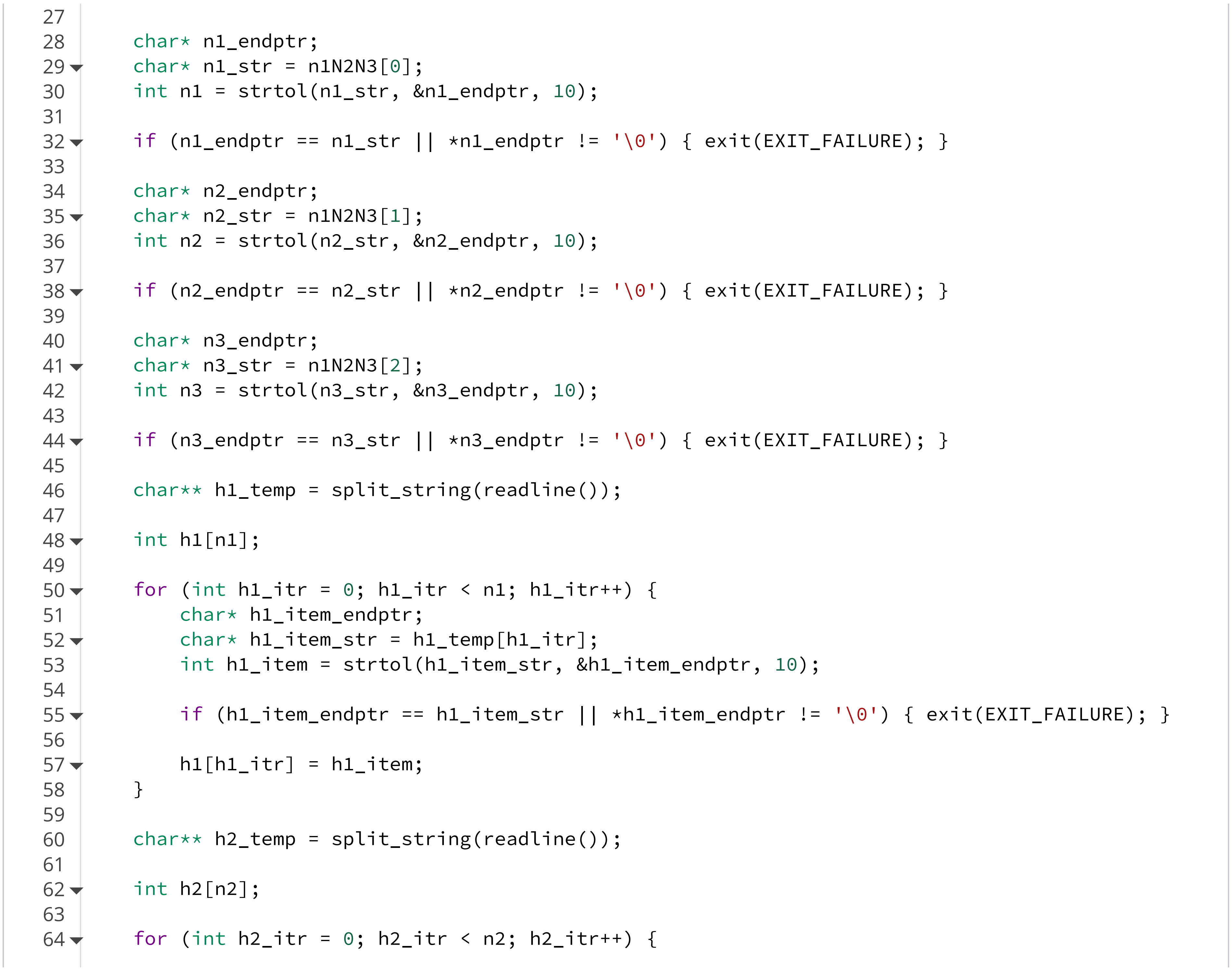
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

C

```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char** split_string(char*);

/*
 * Complete the equalStacks function below.
 */
int equalStacks(int h1_count, int* h1, int h2_count, int* h2, int h3_count, int* h3) {
    /*
     * Write your code here.
     */

}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char** n1N2N3 = split_string(readline());
```



```c

    char* n1_endptr;
    char* n1_str = n1N2N3[0];
    int n1 = strtol(n1_str, &n1_endptr, 10);

    if (n1_endptr == n1_str || *n1_endptr != '\0') { exit(EXIT_FAILURE); }

    char* n2_endptr;
    char* n2_str = n1N2N3[1];
    int n2 = strtol(n2_str, &n2_endptr, 10);

    if (n2_endptr == n2_str || *n2_endptr != '\0') { exit(EXIT_FAILURE); }

    char* n3_endptr;
    char* n3_str = n1N2N3[2];
    int n3 = strtol(n3_str, &n3_endptr, 10);

    if (n3_endptr == n3_str || *n3_endptr != '\0') { exit(EXIT_FAILURE); }

    char** h1_temp = split_string(readline());

    int h1[n1];

    for (int h1_itr = 0; h1_itr < n1; h1_itr++) {
        char* h1_item_endptr;
        char* h1_item_str = h1_temp[h1_itr];
        int h1_item = strtol(h1_item_str, &h1_item_endptr, 10);

        if (h1_item_endptr == h1_item_str || *h1_item_endptr != '\0') { exit(EXIT_FAILURE); }

        h1[h1_itr] = h1_item;
    }

    char** h2_temp = split_string(readline());

    int h2[n2];

    for (int h2_itr = 0; h2_itr < n2; h2_itr++) {
        char* h2_item_endptr;
        char* h2_item_str = h2_temp[h2_itr];
        int h2_item = strtol(h2_item_str, &h2_item_endptr, 10);

        if (h2_item_endptr == h2_item_str || *h2_item_endptr != '\0') { exit(EXIT_FAILURE); }

        h2[h2_itr] = h2_item;
    }

    char** h3_temp = split_string(readline());

    int h3[n3];

    for (int h3_itr = 0; h3_itr < n3; h3_itr++) {
        char* h3_item_endptr;
        char* h3_item_str = h3_temp[h3_itr];
        int h3_item = strtol(h3_item_str, &h3_item_endptr, 10);

        if (h3_item_endptr == h3_item_str || *h3_item_endptr != '\0') { exit(EXIT_FAILURE); }

        h3[h3_itr] = h3_item;
    }

    int result = equalStacks(h1_count, h1, h2_count, h2, h3_count, h3);

    fprintf(fptr, "%d\n", result);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) { break; }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') { break; }

        size_t new_length = alloc_length << 1;
        data = realloc(data, new_length);

        if (!data) { break; }

        alloc_length = new_length;
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }

    data = realloc(data, data_length);

    return data;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);
        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

```

Line: 1 Col: 1

Upload Code as File

☐ Test against custom input

Run Code

Submit Code

Balanced Brackets

locked

by saikiran9194

Problem Submissions Leaderboard Discussions Editorial

A bracket is considered to be any one of the following characters: `(`, `)`, `{`, `}`, `[`, or `]`.

Two brackets are considered to be a *matched pair* if the an opening bracket (i.e., `(`, `[`, or `{`) occurs to the left of a closing bracket (i.e., `)`, `]`, or `}`) *of the exact same type*. There are three types of matched pairs of brackets: `[]`, `{}`, and `()`.

A matching pair of brackets is *not balanced* if the set of brackets it encloses are not matched. For example, `{[(])}` is not balanced because the contents in between `{` and `}` are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, `(`, and the pair of parentheses encloses a single, unbalanced closing square bracket, `]`.

By this logic, we say a sequence of brackets is *balanced* if the following conditions are met:

- It contains no unmatched brackets.
- The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return `YES`. Otherwise, return `NO`.

Function Description

Complete the function *isBalanced* in the editor below. It must return a string: `YES` if the sequence is balanced or `NO` if it is not.

isBalanced has the following parameter(s):

- *s*: a string of brackets

Input Format

The first line contains a single integer n, the number of strings.
Each of the next n lines contains a single string s, a sequence of brackets.

Constraints

- $1 \leq n \leq 10^3$
- $1 \leq |s| \leq 10^3$, where $|s|$ is the length of the sequence.
- All chracters in the sequences ∈ { **{, }**, **(,)**, **[,]** }.

Output Format

For each string, return `YES` or `NO`.

Sample Input

```
3
{[()]}
{[(])}
{{[[(())]]}}
```

Sample Output

```
YES
NO
```

```
YES
```

Explanation

1. The string `{[()]}` meets both criteria for being a balanced string, so we print `YES` on a new line.
2. The string `{[(])}` is not balanced because the brackets enclosed by the matched pair `{` and `}` are not balanced: `[(])`.
3. The string `{{[[(())]]}}` meets both criteria for being a balanced string, so we print `YES` on a new line.

Submissions: 21
Max Score: 25
Difficulty: Medium

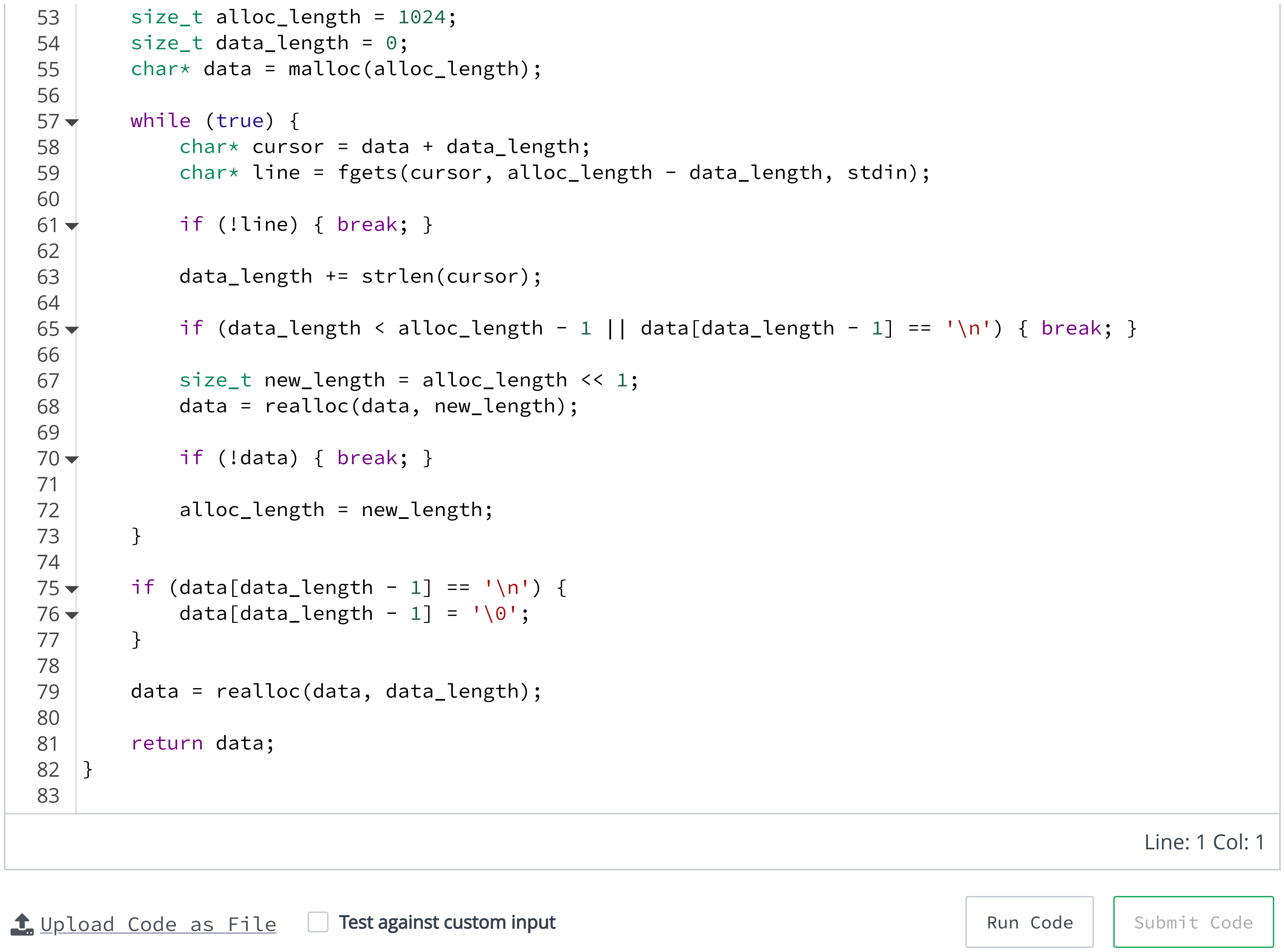
Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

C

```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

// Complete the isBalanced function below.

// Please either make the string static or allocate on the heap. For example,
// static char str[] = "hello world";
// return str;
//
// OR
//
// char* str = "hello world";
// return str;
//
char* isBalanced(char* s) {


}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* t_endptr;
    char* t_str = readline();
    int t = strtol(t_str, &t_endptr, 10);

    if (t_endptr == t_str || *t_endptr != '\0') { exit(EXIT_FAILURE); }

    for (int t_itr = 0; t_itr < t; t_itr++) {
        char* s = readline();

        char* result = isBalanced(s);

        fprintf(fptr, "%s\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
```



```c
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) { break; }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') { break; }

        size_t new_length = alloc_length << 1;
        data = realloc(data, new_length);

        if (!data) { break; }

        alloc_length = new_length;
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }

    data = realloc(data, data_length);

    return data;
}

```

Queue using Two Stacks

locked

by saikiran9194

Problem Submissions Leaderboard Discussions Editorial

A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a *First-In-First-Out* (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed.

A basic queue has the following operations:

- *Enqueue*: add a new element to the end of the queue.
- *Dequeue*: remove the element from the front of the queue and return it.

In this challenge, you must first implement a queue using *two stacks*. Then process q queries, where each query is one of the following 3 types:

1. `1 x`: Enqueue element x into the end of the queue.
2. `2`: Dequeue the element at the front of the queue.
3. `3`: Print the element at the front of the queue.

Input Format

The first line contains a single integer, q, denoting the number of queries.
Each line i of the q subsequent lines contains a single query in the form described in the problem statement above. All three queries start with an integer denoting the query $type$, but only query 1 is followed by an additional space-separated value, x, denoting the value to be enqueued.

Constraints

- $1 \leq q \leq 10^5$
- $1 \leq type \leq 3$
- $1 \leq |x| \leq 10^9$
- It is guaranteed that a valid answer always exists for each query of type 3.

Output Format

For each query of type 3, print the value of the element at the front of the queue on a new line.

Sample Input

```
10
1 42
2
1 14
3
1 28
3
1 60
1 78
2
2
```

Sample Output

```
14
14
```

Explanation

We perform the following sequence of actions:

1. Enqueue 42; $queue = \{42\}$.
2. Dequeue the value at the head of the queue, 42; $queue = \{\}$.
3. Enqueue 14; $queue = \{14\}$.
4. Print the value at the head of the queue, 14; $queue = \{14\}$.
5. Enqueue 28; $queue = \{14, 28\}$.
6. Print the value at the head of the queue, 14; $queue = \{14, 28\}$.
7. Enqueue 60; $queue = \{14, 28, 60\}$.
8. Enqueue 78; $queue = \{14, 28, 60, 78\}$.
9. Dequeue the value at the head of the queue, 14; $queue = \{28, 60, 78\}$.
10. Dequeue the value at the head of the queue, 28; $queue = \{60, 78\}$.

f 🐦 in

Submissions: 46
Max Score: 30
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

C

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main() {

    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File   Test against custom input   Run Code   Submit Code

Game of Two Stacks

locked

by Shafaet

Alexa has two stacks of non-negative integers, stack $A = [a_0, a_1, \ldots, a_{n-1}]$ and stack $B = [b_0, b_1, \ldots, b_{m-1}]$ where index 0 denotes the top of the stack. Alexa challenges Nick to play the following game:

- In each move, Nick can remove one integer from the top of either stack A or stack B.
- Nick keeps a running sum of the integers he removes from the two stacks.
- Nick is disqualified from the game if, at any point, his running sum becomes greater than some integer x given at the beginning of the game.
- Nick's *final score* is the total number of integers he has removed from the two stacks.

Given A, B, and x for g games, find the maximum possible score Nick can achieve (i.e., the maximum number of integers he can remove without being disqualified) during each game and print it on a new line.

Input Format

The first line contains an integer, g (the number of games). The $3 \cdot g$ subsequent lines describe each game in the following format:

1. The first line contains three space-separated integers describing the respective values of n (the number of integers in stack A), m (the number of integers in stack B), and x (the number that the sum of the integers removed from the two stacks cannot exceed).
2. The second line contains n space-separated integers describing the respective values of $a_0, a_1, \ldots, a_{n-1}$.
3. The third line contains m space-separated integers describing the respective values of $b_0, b_1, \ldots, b_{m-1}$.

Constraints

- $1 \leq g \leq 50$
- $1 \leq n, m \leq 10^5$
- $0 \leq a_i, b_j \leq 10^6$
- $1 \leq x \leq 10^9$

Subtasks

- $1 \leq n, m, \leq 100$ for 50% of the maximum score.

Output Format

For each of the g games, print an integer on a new line denoting the maximum possible score Nick can achieve without being disqualified.

Sample Input 0

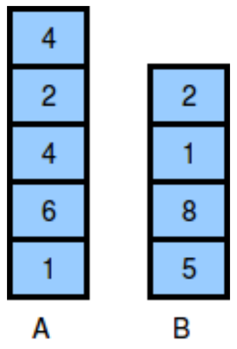
```
1
5 4 10
4 2 4 6 1
2 1 8 5
```

Sample Output 0

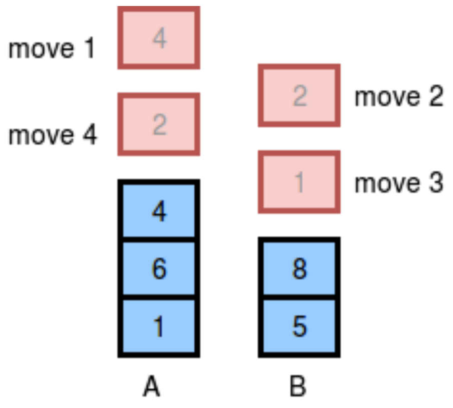
```
4
```

Explanation 0

The two stacks initially look like this:



The image below depicts the integers Nick should choose to remove from the stacks. We print 4 as our answer, because that is the maximum number of integers that can be removed from the two stacks without the sum exceeding $x = 10$.



(There can be multiple ways to remove the integers from the stack, the image shows just one of them.)

Submissions: 12
Max Score: 30

Difficulty: Medium

Rate This Challenge:

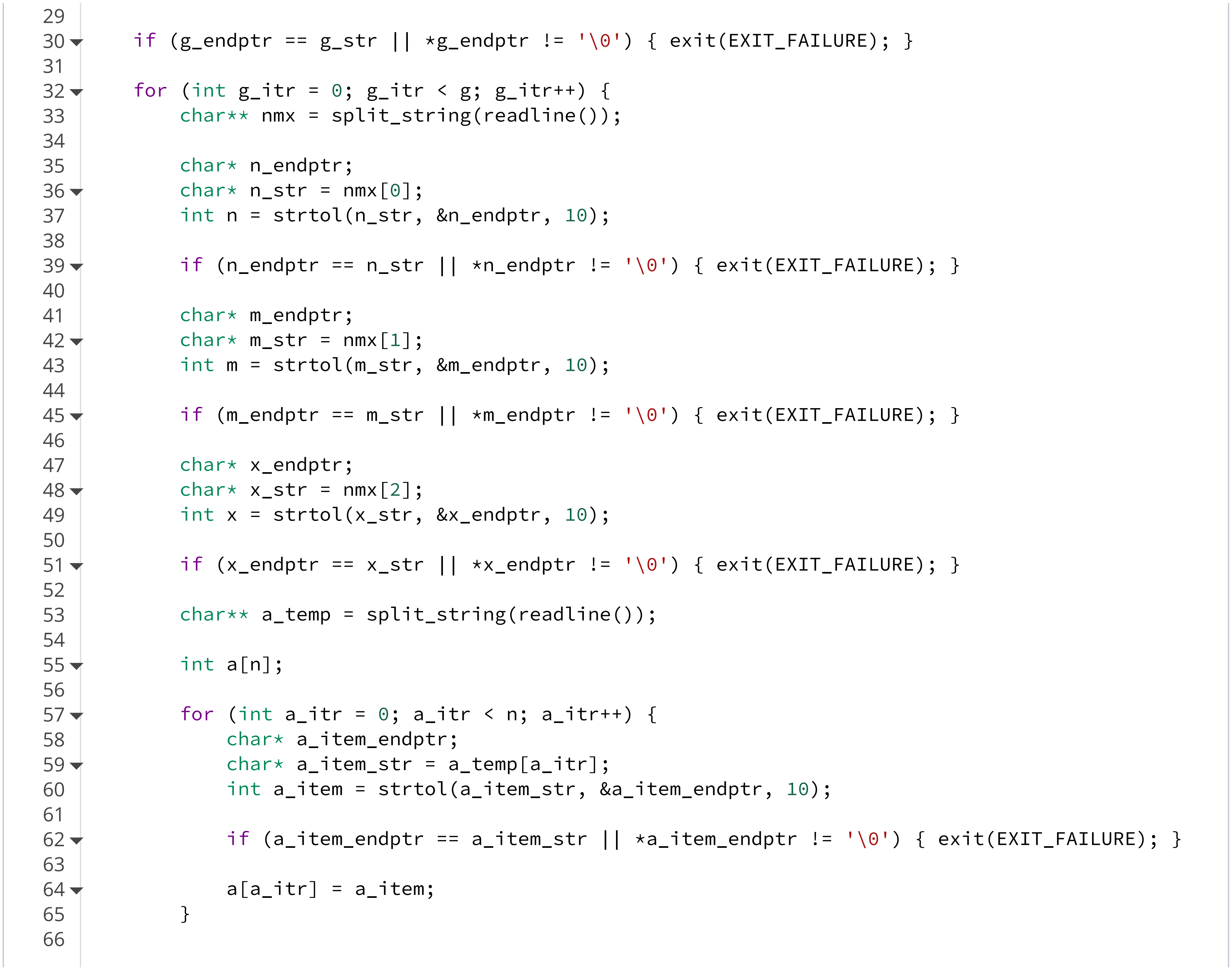
☆☆☆☆☆

More

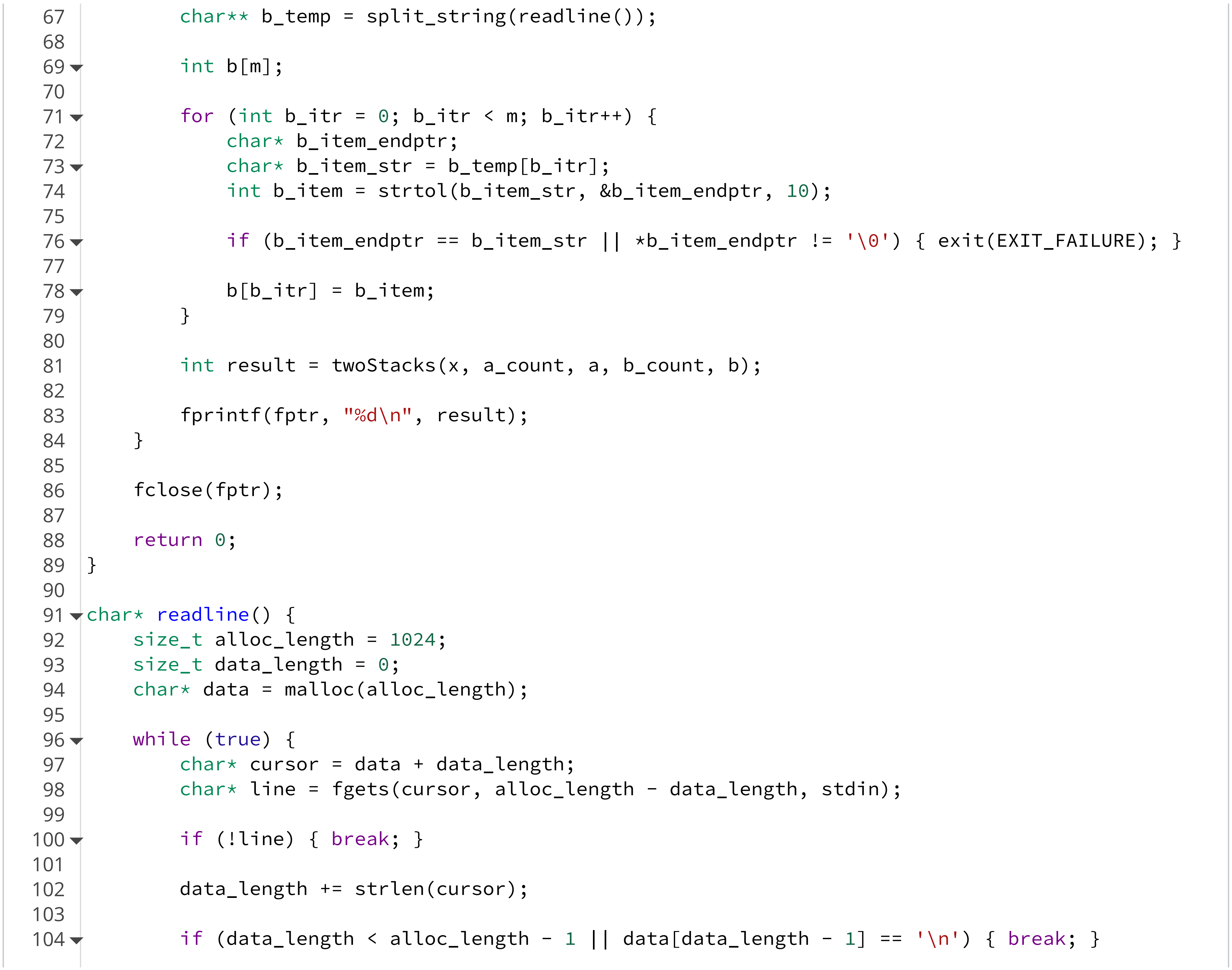
Current Buffer (saved locally, editable)

C

```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char** split_string(char*);

/*
 * Complete the twoStacks function below.
 */
int twoStacks(int x, int a_count, int* a, int b_count, int* b) {
    /*
     * Write your code here.
     */

}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* g_endptr;
    char* g_str = readline();
    int g = strtol(g_str, &g_endptr, 10);
```



```c

    if (g_endptr == g_str || *g_endptr != '\0') { exit(EXIT_FAILURE); }

    for (int g_itr = 0; g_itr < g; g_itr++) {
        char** nmx = split_string(readline());

        char* n_endptr;
        char* n_str = nmx[0];
        int n = strtol(n_str, &n_endptr, 10);

        if (n_endptr == n_str || *n_endptr != '\0') { exit(EXIT_FAILURE); }

        char* m_endptr;
        char* m_str = nmx[1];
        int m = strtol(m_str, &m_endptr, 10);

        if (m_endptr == m_str || *m_endptr != '\0') { exit(EXIT_FAILURE); }

        char* x_endptr;
        char* x_str = nmx[2];
        int x = strtol(x_str, &x_endptr, 10);

        if (x_endptr == x_str || *x_endptr != '\0') { exit(EXIT_FAILURE); }

        char** a_temp = split_string(readline());

        int a[n];

        for (int a_itr = 0; a_itr < n; a_itr++) {
            char* a_item_endptr;
            char* a_item_str = a_temp[a_itr];
            int a_item = strtol(a_item_str, &a_item_endptr, 10);

            if (a_item_endptr == a_item_str || *a_item_endptr != '\0') { exit(EXIT_FAILURE); }

            a[a_itr] = a_item;
        }

```



```
        char** b_temp = split_string(readline());

        int b[m];

        for (int b_itr = 0; b_itr < m; b_itr++) {
            char* b_item_endptr;
            char* b_item_str = b_temp[b_itr];
            int b_item = strtol(b_item_str, &b_item_endptr, 10);

            if (b_item_endptr == b_item_str || *b_item_endptr != '\0') { exit(EXIT_FAILURE); }

            b[b_itr] = b_item;
        }

        int result = twoStacks(x, a_count, a, b_count, b);

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) { break; }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') { break; }
```

```c

        size_t new_length = alloc_length << 1;
        data = realloc(data, new_length);

        if (!data) { break; }

        alloc_length = new_length;
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }

    data = realloc(data, data_length);

    return data;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);
        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

```

Down to Zero II

locked

by dcod5

Problem Submissions Leaderboard Discussions Editorial

You are given Q queries. Each query consists of a single number N. You can perform any of the 2 operations on N in each move:

1: If we take 2 integers a and b where $N = a \times b(a \neq 1, b \neq 1)$, then we can change $N = max(a, b)$

2: Decrease the value of N by 1.

Determine the minimum number of moves required to reduce the value of N to 0.

Input Format

The first line contains the integer Q.
The next Q lines each contain an integer, N.

Constraints

$1 \leq Q \leq 10^3$
$0 \leq N \leq 10^6$

Output Format

Output Q lines. Each line containing the minimum number of moves required to reduce the value of N to 0.

Sample Input

```
2
3
4
```

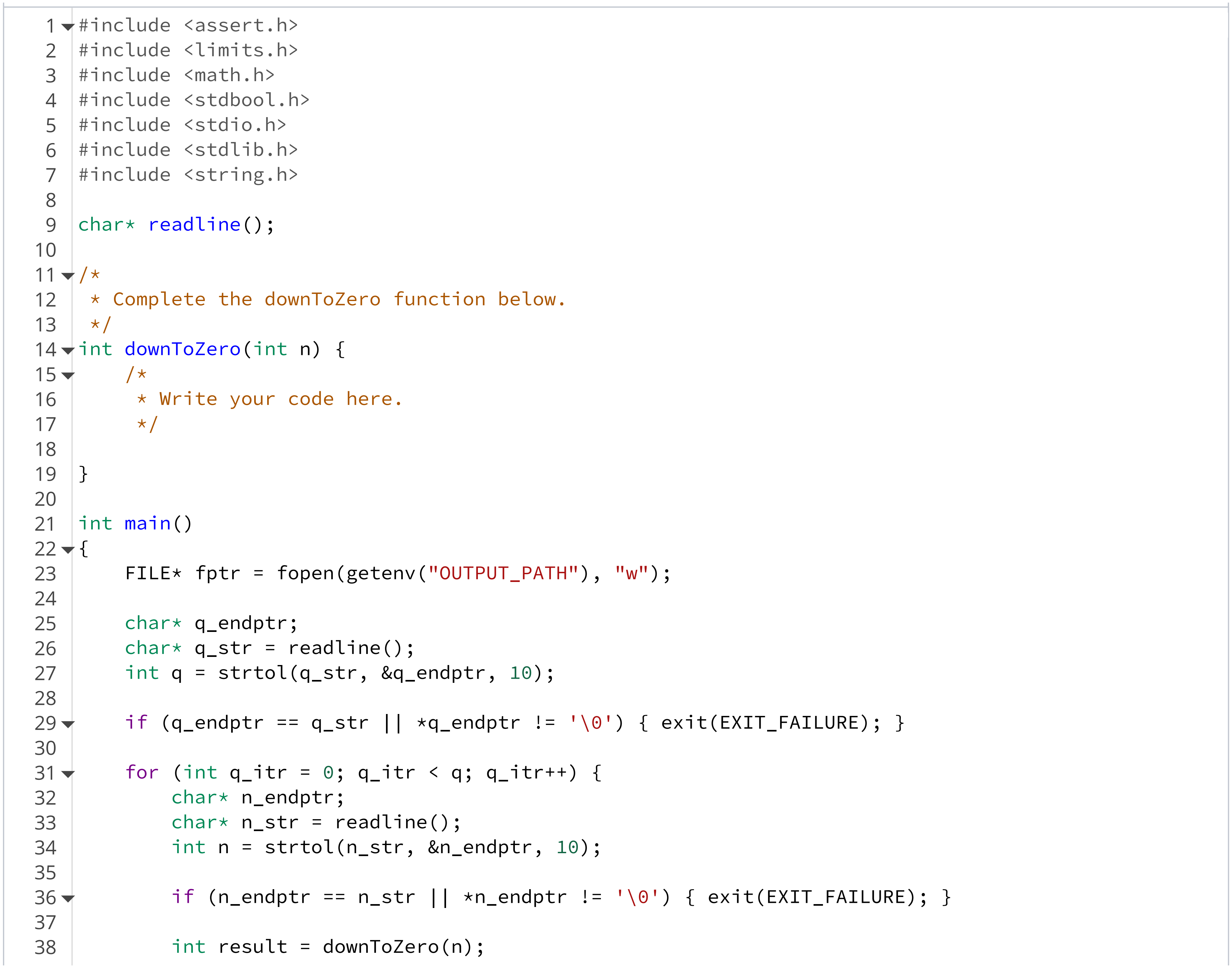
Sample Output

```
3
3
```

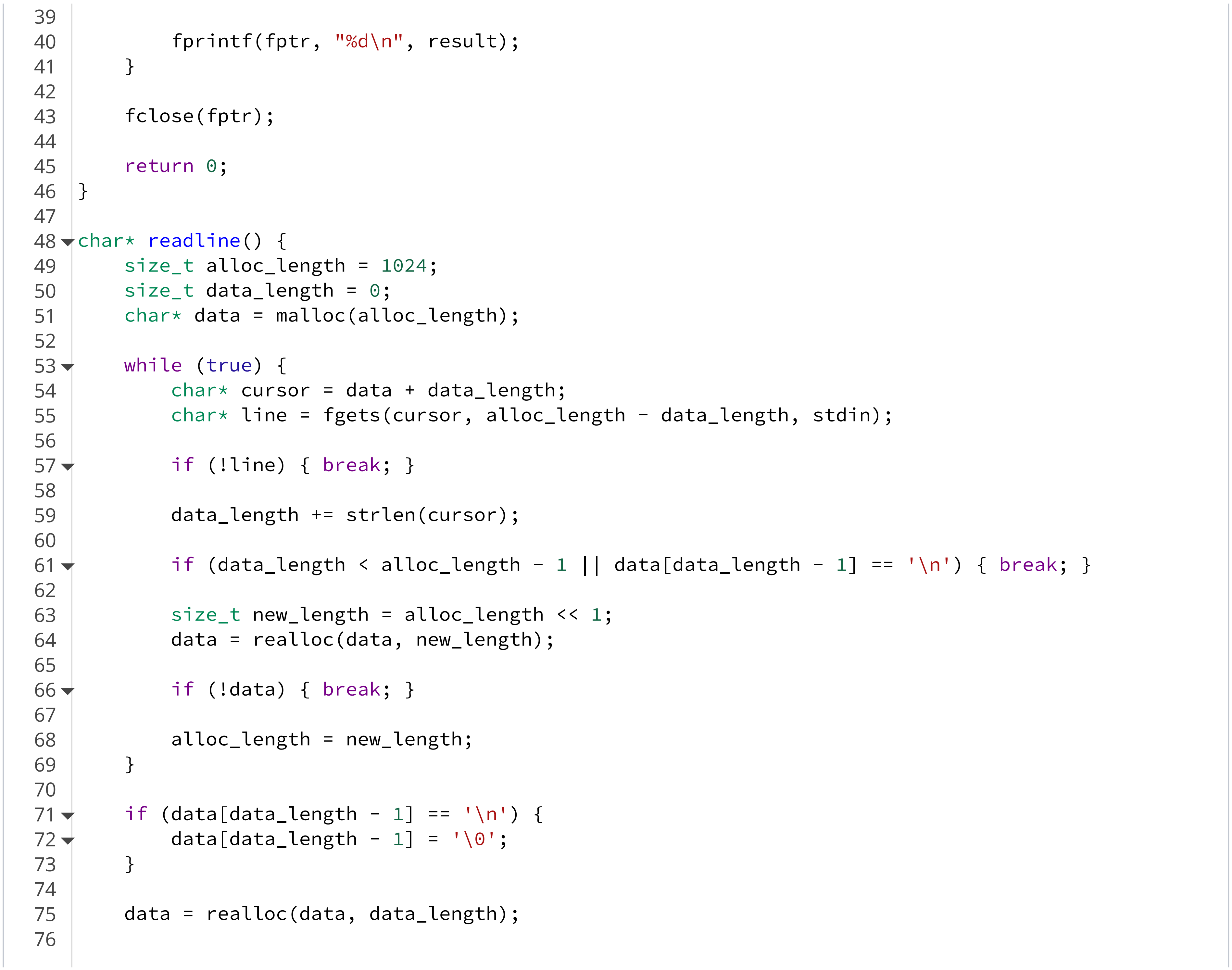
Explanation

For test case 1, We only have one option that gives the minimum number of moves.
Follow 3 -> 2 -> 1 -> 0. Hence, 3 moves.

For the case 2, we can either go 4 -> 3 -> 2 -> 1 -> 0 or 4 -> 2 -> 1 -> 0. The 2nd option is more optimal. Hence, 3 moves.



```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

/*
 * Complete the downToZero function below.
 */
int downToZero(int n) {
    /*
     * Write your code here.
     */

}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* q_endptr;
    char* q_str = readline();
    int q = strtol(q_str, &q_endptr, 10);

    if (q_endptr == q_str || *q_endptr != '\0') { exit(EXIT_FAILURE); }

    for (int q_itr = 0; q_itr < q; q_itr++) {
        char* n_endptr;
        char* n_str = readline();
        int n = strtol(n_str, &n_endptr, 10);

        if (n_endptr == n_str || *n_endptr != '\0') { exit(EXIT_FAILURE); }

        int result = downToZero(n);
```



```c

        fprintf(fptr, "%d\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) { break; }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') { break; }

        size_t new_length = alloc_length << 1;
        data = realloc(data, new_length);

        if (!data) { break; }

        alloc_length = new_length;
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';
    }

    data = realloc(data, data_length);

    return data;
}

```

Line: 1 Col: 1

Upload Code as File

Test against custom input

Run Code

Submit Code

Simple Text Editor

locked

by pkacprzak

In this challenge, you must implement a simple text editor. Initially, your editor contains an empty string, S. You must perform Q operations of the following 4 types:

1. *append*(W) - Append string W to the end of S.
2. *delete*(k) - Delete the last k characters of S.
3. *print*(k) - Print the k^{th} character of S.
4. *undo*$()$ - Undo the last (not previously undone) operation of type 1 or 2, reverting S to the state it was in prior to that operation.

Input Format

The first line contains an integer, Q, denoting the number of operations.
Each line i of the Q subsequent lines (where $0 \leq i < Q$) defines an operation to be performed. Each operation starts with a single integer, t (where $t \in \{1, 2, 3, 4\}$), denoting a type of operation as defined in the *Problem Statement* above. If the operation

requires an argument, t is followed by its space-separated argument. For example, if $t = 1$ and $W = \text{"abcd"}$, line i will be `1 abcd`.

Constraints

- $1 \leq Q \leq 10^6$
- $1 \leq k \leq |S|$
- The sum of the lengths of all W in the input $\leq 10^6$.
- The sum of k over all delete operations $\leq 2 \cdot 10^6$.
- All input characters are lowercase English letters.
- It is guaranteed that the sequence of operations given as input is possible to perform.

Output Format

Each operation of type 3 must print the k^{th} character on a new line.

Sample Input

```
8
1 abc
3 3
2 3
1 xy
3 2
4
4
3 1
```

Sample Output

```
c
y
a
```

Explanation

Initially, S is empty. The following sequence of 8 operations are described below:

1. $S = ""$. We append abc to S, so $S = "abc"$.
2. Print the 3^{rd} character on a new line. Currently, the 3^{rd} character is `c`.
3. Delete the last 3 characters in S (abc), so $S = ""$.
4. Append xy to S, so $S = "xy"$.
5. Print the 2^{nd} character on a new line. Currently, the 2^{nd} character is `y`.
6. Undo the last update to S, making S empty again (i.e., $S = ""$).
7. Undo the next to last update to S (the deletion of the last 3 characters), making $S = "\text{abc}"$.
8. Print the 1^{st} character on a new line. Currently, the 1^{st} character is `a`.

Submissions: 7
Max Score: 65
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

C

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main() {

    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}

```

Line: 1 Col: 1

Upload Code as File ☐ Test against custom input

Run Code Submit Code