

LC 1

Code

```
struct sNode {
    char data;
    struct sNode* next;
};

// Function to push an item to stack
void push(struct sNode** top_ref, int new_data);

// Function to pop an item from stack
int pop(struct sNode** top_ref);

// Returns 1 if character1 and character2 are matching left
// and right Brackets
bool isMatchingPair(char character1, char character2)
{
    if (character1 == '(' && character2 == ')')
        return 1;
    else if (character1 == '{' && character2 == '}')
        return 1;
    else if (character1 == '[' && character2 == ']')
        return 1;
    else
```

```

        return 0;
    }

bool isValid(char* exp)
{
    int i = 0;

    struct sNode* stack = NULL;

    while (exp[i]) {

        if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
            push(&stack, exp[i]);

        if (exp[i] == '}' || exp[i] == ')'
            || exp[i] == ']') {

            if (stack == NULL)
                return 0;

            // Pop the top element from stack, if it is not
            // a pair bracket of character then there is a
            // mismatch.
            // his happens for expressions like {}
            else if (!isMatchingPair(pop(&stack), exp[i]))

```

```
        return 0;
    }
    i++;
}
```

```
// If there is something left in expression then there
// is a starting bracket without a closing
// bracket
if (stack == NULL)
    return 1; // balanced
else
    return 0; // not balanced
}
```

```
// Function to push an item to stack
void push(struct sNode** top_ref, int new_data)
{
    // allocate node
    struct sNode* new_node
        = (struct sNode*)malloc(sizeof(struct sNode));

    if (new_node == NULL) {
        printf("Stack overflow n");
        getchar();
        exit(0);
    }
}
```

```
}

// put in the data
new_node->data = new_data;

// link the old list of the new node
new_node->next = (*top_ref);

// move the head to point to the new node
(*top_ref) = new_node;
}
```

```
// Function to pop an item from stack
```

```
int pop(struct sNode** top_ref)
```

```
{
    char res;
    struct sNode* top;

    // If stack is empty then error
    if (*top_ref == NULL) {
        printf("Stack overflow n");
        getchar();
        exit(0);
    }
    else {
```

```

    top = *top_ref;

    res = top->data;

    *top_ref = top->next;

    free(top);

    return res;

}

}

```

Output

The screenshot displays a coding platform interface with the following components:

- Problem List:** A tab at the top left showing the current problem.
- Submissions:** A section showing the submission status as "Accepted" for user "samraatd" on Mar 07, 2024 at 22:00.
- Performance Metrics:**
 - Runtime:** 3 ms, Beats 38.47% of users with C.
 - Memory:** 5.51 MB, Beats 87.87% of users with C.
- Code Editor:** Contains the following C++ code:

```

1 struct sNode {
2     char data;
3     struct sNode* next;
4 };
5
6 // Function to push an item to stack
7 void push(struct sNode** top_ref, int new_data) {
8
9     // Function to pop an item from stack
10    int pop(struct sNode** top_ref);
11
12    // Returns 1 if character1 and character2 are equal and 0 otherwise
13    // and right Brackets
14    return 1;
15 }

```
- Testcase:** A section showing the test result for Case 1, Case 2, and Case 3. The input is "()" and the output is "s =".
- Samraatd Profile:** A sidebar menu on the right with the following options:
 - My Lists
 - Notebook
 - Submissions
 - Progress
 - Points
 - Session
 - Try New Features
 - Orders
 - My Playgrounds
 - Revert to old version
 - Appearance
 - Sign Out

LC 2

Code

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseList(struct ListNode* head) {
    struct ListNode* temp = head;
    struct ListNode* curr = temp;
    struct ListNode* prev = NULL;
    struct ListNode* nextOne = NULL;

    while(curr != NULL) {
        nextOne = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextOne;
    }
    return prev;
}
```

Output

The screenshot displays a coding platform interface with the following components:

- Problem List:** Located at the top left, showing the current problem and navigation options.
- Submissions:** A section on the left showing the submission status as "Accepted" for user "samraatd" on Mar 07, 2024. It includes buttons for "Editorial" and "Solution".
- Runtime/Memory Graph:** A bar chart showing performance metrics. The runtime is 0 ms, and memory is 6.34 MB. The graph indicates that the solution beats 100.00% of users with C and 44.20% of users with C++.
- Code Editor:** The main area for writing code, showing a C++ implementation of a singly-linked list reversal algorithm. The code defines a `ListNode` struct and a `reverseList` function.
- Testcase:** A section on the right showing the test case input: `head = [1,2,3,4,5]`.
- Submission Panel:** A sidebar on the right containing user information for "samraatd", a premium subscription notice, and various navigation icons (My Lists, Notebook, Submissions, Progress, Points, Session).

```
1 /**  
2  * Definition for singly-linked list.  
3  * struct ListNode {  
4  *     int val;  
5  *     struct ListNode *next;  
6  * };  
7  */  
8 struct ListNode* reverseList(struct ListNode* head) {  
9     struct ListNode* temp = head;  
10    struct ListNode* curr = temp;  
11    struct ListNode* prev = NULL;  
12    struct ListNode* nextOne = NULL;  
13    ...  
14 }
```

LC 3

Code

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

```
char* readline();
```

```
typedef struct SinglyLinkedListNode SinglyLinkedListNode;
typedef struct SinglyLinkedList SinglyLinkedList;
```

```
struct SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode* next;
};
```

```
struct SinglyLinkedList {
    SinglyLinkedListNode* head;
    SinglyLinkedListNode* tail;
};
```



```
SinglyLinkedListNode* create_singly_linked_list_node(int node_data) {  
    SinglyLinkedListNode* node = malloc(sizeof(SinglyLinkedListNode));  
  
    node->data = node_data;  
    node->next = NULL;  
  
    return node;  
}
```

```
void insert_node_into_singly_linked_list(SinglyLinkedList** singly_linked_list, int  
node_data) {  
    SinglyLinkedListNode* node = create_singly_linked_list_node(node_data);  
  
    if (!(*singly_linked_list)->head) {  
        (*singly_linked_list)->head = node;  
    } else {  
        (*singly_linked_list)->tail->next = node;  
    }  
  
    (*singly_linked_list)->tail = node;  
}
```

```
void print_singly_linked_list(SinglyLinkedListNode* node, char* sep, FILE* fptr) {  
    while (node) {
```

```

    fprintf(fp_ptr, "%d", node->data);

    node = node->next;

    if (node) {
        fprintf(fp_ptr, "%s", sep);
    }
}

}

void free_singly_linked_list(SinglyLinkedListNode* node) {
    while (node) {
        SinglyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}

```

// Complete the compare_lists function below.

```

/*
 * For your reference:
 *
 * SinglyLinkedListNode {

```

```

*   int data;
*   SinglyLinkedListNode* next;
* };
*
*/

```

```

bool compare_lists(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {
    struct SinglyLinkedListNode *t1;
    struct SinglyLinkedListNode *t2;
    t1=head1;t2 = head2;
    if(t1==NULL && t2==NULL)
        return 1;
    if(t1 != NULL && t2 == NULL)
        return 0;
    if(t1 == NULL && t2 != NULL)
        return 0;
    else
    {
        while(t1->next != NULL && t2->next != NULL)
        {
            if(t1->data == t2->data)
            {
                t1 = t1->next;
                t2 = t2->next;
            }
            else return 0;
        }
    }
}

```

```

    }
    if(t1->next == NULL && t2->next == NULL)
        return 1;
    else return 0;
}

}

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

    char* tests_endptr;
    char* tests_str = readline();
    int tests = strtol(tests_str, &tests_endptr, 10);

    if (tests_endptr == tests_str || *tests_endptr != '\0') { exit(EXIT_FAILURE); }

    for (int tests_itr = 0; tests_itr < tests; tests_itr++) {
        SinglyLinkedList* llist1 = malloc(sizeof(SinglyLinkedList));
        llist1->head = NULL;
        llist1->tail = NULL;

        char* llist1_count_endptr;
        char* llist1_count_str = readline();

```

```

int llist1_count = strtol(llist1_count_str, &llist1_count_endptr, 10);

if (llist1_count_endptr == llist1_count_str || *llist1_count_endptr != '\0') {
    exit(EXIT_FAILURE); }

for (int i = 0; i < llist1_count; i++) {
    char* llist1_item_endptr;
    char* llist1_item_str = readline();
    int llist1_item = strtol(llist1_item_str, &llist1_item_endptr, 10);

    if (llist1_item_endptr == llist1_item_str || *llist1_item_endptr != '\0') {
        exit(EXIT_FAILURE); }

    insert_node_into_singly_linked_list(&llist1, llist1_item);
}

SinglyLinkedList* llist2 = malloc(sizeof(SinglyLinkedList));
llist2->head = NULL;
llist2->tail = NULL;

char* llist2_count_endptr;
char* llist2_count_str = readline();
int llist2_count = strtol(llist2_count_str, &llist2_count_endptr, 10);

```

```
    if (llist2_count_endptr == llist2_count_str || *llist2_count_endptr != '\0') {  
exit(EXIT_FAILURE); }
```

```
for (int i = 0; i < llist2_count; i++) {
```

```
    char* llist2_item_endptr;
```

```
    char* llist2_item_str = readline();
```

```
    int llist2_item = strtol(llist2_item_str, &llist2_item_endptr, 10);
```

```
    if (llist2_item_endptr == llist2_item_str || *llist2_item_endptr != '\0') {  
exit(EXIT_FAILURE); }
```

```
    insert_node_into_singly_linked_list(&llist2, llist2_item);
```

```
}
```

```
bool result = compare_lists(llist1->head, llist2->head);
```

```
fprintf(fp, "%d\n", result);
```

```
}
```

```
fclose(fp);
```

```
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;
```

```
}
```

Output

HackerRank

Prepare > Data Structures > Linked Lists > Compare two linked lists

Exit Full Screen View

Problem

Submissions

Leaderboard

Discussions

This challenge is part of a tutorial track by [MyCodeSchool](#)

You're given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. If all data attributes are equal and the lists are the same length, return 1. Otherwise, return 0.

Example

l1 = 1 → 2 → 3 → NULL
l2 = 1 → 2 → 3 → 4 → NULL

The two lists have equal data attributes for the first 3 nodes. *l2* is longer, though, so the lists are not equal. Return 0.

Function Description

Complete the `compare_lists` function in the editor below.

`compare_lists` has the following parameters:

- `SinglyLinkedListNode l1`: a reference to the head of a list
- `SinglyLinkedListNode l2`: a reference to the head of a list

Returns

- `int`: return 1 if the lists are equal, or 0 otherwise

Input Format

The first line contains an integer *t*, the number of test cases.

Each of the test cases has the following format:

The first line contains an integer *n*, the number of nodes in the first linked list.

Each of the next *n* lines contains an integer, each a value for a data attribute.

The next line contains an integer *m*, the number of nodes in the second linked list.

Each of the next *m* lines contains an integer, each a value for a data attribute.

```
102 }
103
104
105 Name: SAMRAAT DABOLAY
106 USN: 1BM22CS236
```

Line: 105 Col: 22

Upload Code as File




☐ Test against custom input

Run Code

Submit Code

Congratulations

You solved this challenge. Would you like to challenge your friends?



Next Challenge

Test case 0

Test case 1

Test case 2

Test case 3

Test case 4

Compiler Message

Success

Input (stdin)

1	2
2	2
3	1
4	2
5	1

Download

LC 4

Code

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

void findLeaves(struct TreeNode* node, int** leafValues, int* size, int* capacity) {
    if (node == NULL) {
        return;
    }

    if (node->left == NULL && node->right == NULL) {
        if (*size >= *capacity) {
            *capacity *= 2;
            *leafValues = (int*) realloc(*leafValues, *capacity * sizeof(int));
        }
        (*leafValues)[(*size)++] = node->val;
    }
}
```

```
    findLeaves(node->left, leafValues, size, capacity);
    findLeaves(node->right, leafValues, size, capacity);
}
```

```
bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
```

```
    int *leaves1 = (int*) malloc(sizeof(int) * 10);
```

```
    int size1 = 0, capacity1 = 10;
```

```
    int *leaves2 = (int*) malloc(sizeof(int) * 10);
```

```
    int size2 = 0, capacity2 = 10;
```

```
    findLeaves(root1, &leaves1, &size1, &capacity1);
```

```
    findLeaves(root2, &leaves2, &size2, &capacity2);
```

```
    if (size1 != size2) {
```

```
        free(leaves1);
```

```
        free(leaves2);
```

```
        return false;
```

```
    }
```

```
    for (int i = 0; i < size1; i++) {
```

```
        if (leaves1[i] != leaves2[i]) {
```

```
            free(leaves1);
```

```
            free(leaves2);
```

```
            return false;
```

```

    }
}

free(leaves1);
free(leaves2);
return true;
}

```

Output

The screenshot displays the LeetCode submission interface for a problem involving binary trees. The submission is marked as "Accepted" and was submitted by user "samraatd" on March 07, 2024, at 22:09. The performance metrics show a runtime of 0 ms, which beats 100.00% of users with C, and a memory usage of 6.18 MB, which beats 95.75% of users with C. A performance graph is visible, showing the submission's performance relative to other users. The code is written in C++ and defines a binary tree node structure and a function to find leaves. The test case input is [3, 5, 1, 6, 2, 9, 8, null, null, 7, 4], and the output is a binary tree diagram.

Code:

```

1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     struct TreeNode *left;
6  *     struct TreeNode *right;
7  * };
8  */
9
10 void findLeaves(struct TreeNode* node, int**
11                if (node == NULL) {
12                    return;
13                }
14                ...

```

Testcase:

Case 1: [3, 5, 1, 6, 2, 9, 8, null, null, 7, 4]

Output:

```

graph TD
    3((3)) --- 5((5))
    3 --- 1((1))
    5 --- 6((6))
    5 --- 2((2))
    1 --- 9((9))
    1 --- 8((8))
    2 --- 7((7))
    2 --- 4((4))

```