Experiment No: 1.3

Student Name: Lipakshi	**UID: 20BCS5082**
Branch: CSE	**Section/Group: 607-B**
Semester:5th	**Date of Performance:05-09-2022**
Subject Name: Competitive Coding lab	**Subject Code:20CSP-314**

1. Aim/Overview of the practical:

Q1. 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 . Otherwise, return .

Example

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

Function Description

Complete the *compare_lists* function in the editor below.

compare_lists has the following parameters:

- *SinglyLinkedListNode llist1:* a reference to the head of a list
- *SinglyLinkedListNode llist2:* 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 , the number of test cases.

Each of the test cases has the following format:

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

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

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

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

Constraints

-
-
-

Output Format

Compare the two linked lists and return 1 if the lists are equal. Otherwise, return 0. Do NOT print anything to stdout/console.

The output is handled by the code in the editor and it is as follows:

For each test case, in a new line, print if the two lists are equal, else print .

Sample Input

```
2
2
1
2
1
1
2
1
2
2
1
2
```

Sample Output

```
0
1
```

Explanation

There are test cases, each with a pair of linked lists.

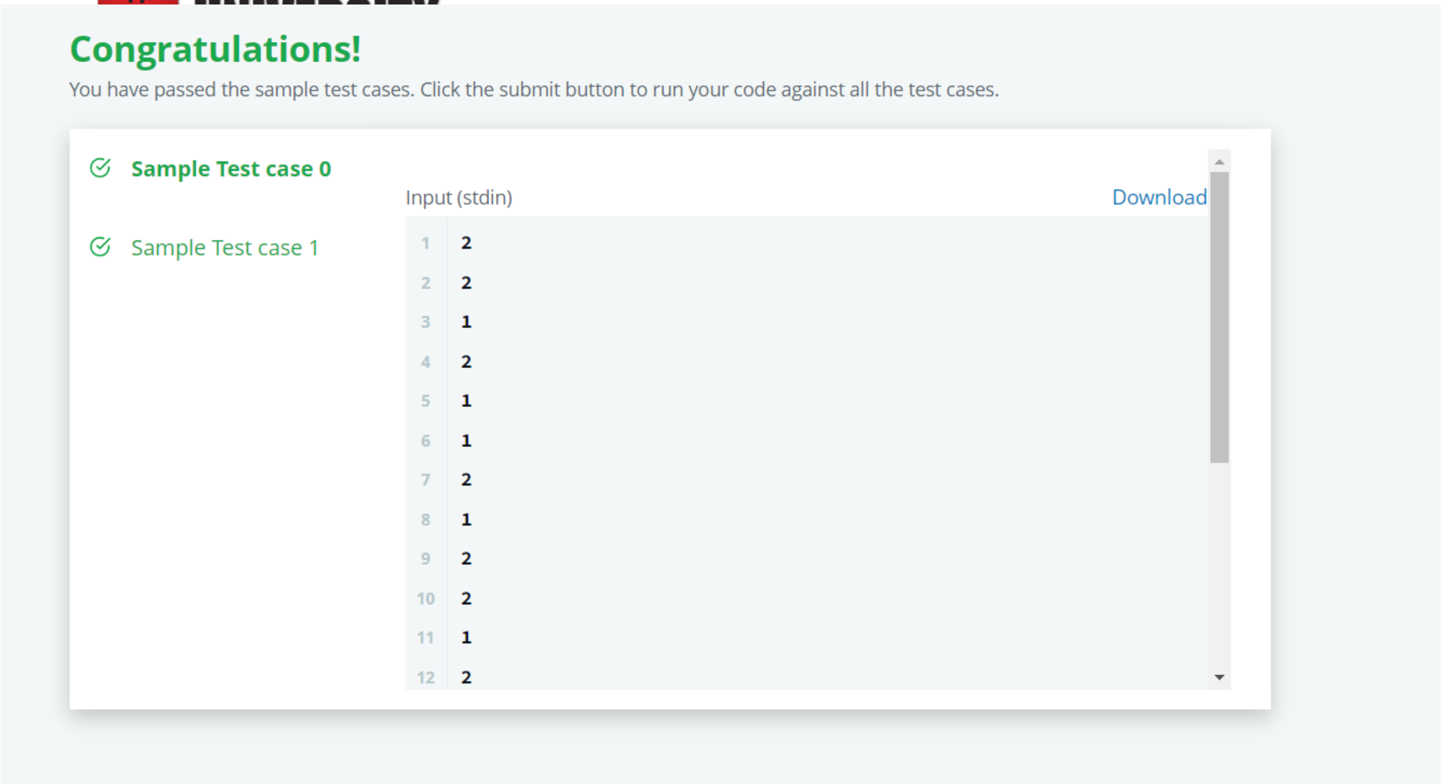
- In the first case, linked lists are: 1 -> 2 -> NULL and 1 -> NULL
- In the second case, linked lists are: 1 -> 2 -> NULL and 1 -> 2 -> NULL

2. Code:

```
#include <bits/stdc++.h>

using namespace std;

class SinglyLinkedListNode {
    public:
        int data;
        SinglyLinkedListNode *next;

        SinglyLinkedListNode(int node_data) {
            this->data = node_data;
            this->next = nullptr;
        }
};

class SinglyLinkedList {
    public:
        SinglyLinkedListNode *head;
        SinglyLinkedListNode *tail;

        SinglyLinkedList() {
            this->head = nullptr;
            this->tail = nullptr;
        }

        void insert_node(int node_data) {
            SinglyLinkedListNode* node = new SinglyLinkedListNode(node_data);

            if (!this->head) {
                this->head = node;
            } else {
                this->tail->next = node;
            }

            this->tail = node;
        }
```

```
};

void print_singly_linked_list(SinglyLinkedListNode* node, string sep, ofstream& fout)
 {
    while (node) {
        fout << node->data;

        node = node->next;

        if (node) {
            fout << sep;
        }
    }
}

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

        free(temp);
    }
}

bool compare_lists(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {

int res=1;
    while(head1 != NULL || head2 != NULL){
        if(head1 == NULL) {res=0; break;}
        if(head2 == NULL) {res=0; break;}
        if(head1->data != head2->data){res=0;break;}
        head1=head1->next;
        head2=head2->next;
    }
    return res;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    int tests;
    cin >> tests;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int tests_itr = 0; tests_itr < tests; tests_itr++) {
        SinglyLinkedList* llist1 = new SinglyLinkedList();
        int llist1_count;
        cin >> llist1_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist1_count; i++) {
            int llist1_item;
            cin >> llist1_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist1->insert_node(llist1_item);
        }

        SinglyLinkedList* llist2 = new SinglyLinkedList();

        int llist2_count;
        cin >> llist2_count;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        for (int i = 0; i < llist2_count; i++) {
            int llist2_item;
            cin >> llist2_item;
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            llist2->insert_node(llist2_item);
        }

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

        fout << result << "\n";
    }

    fout.close();

    return 0;
}
```

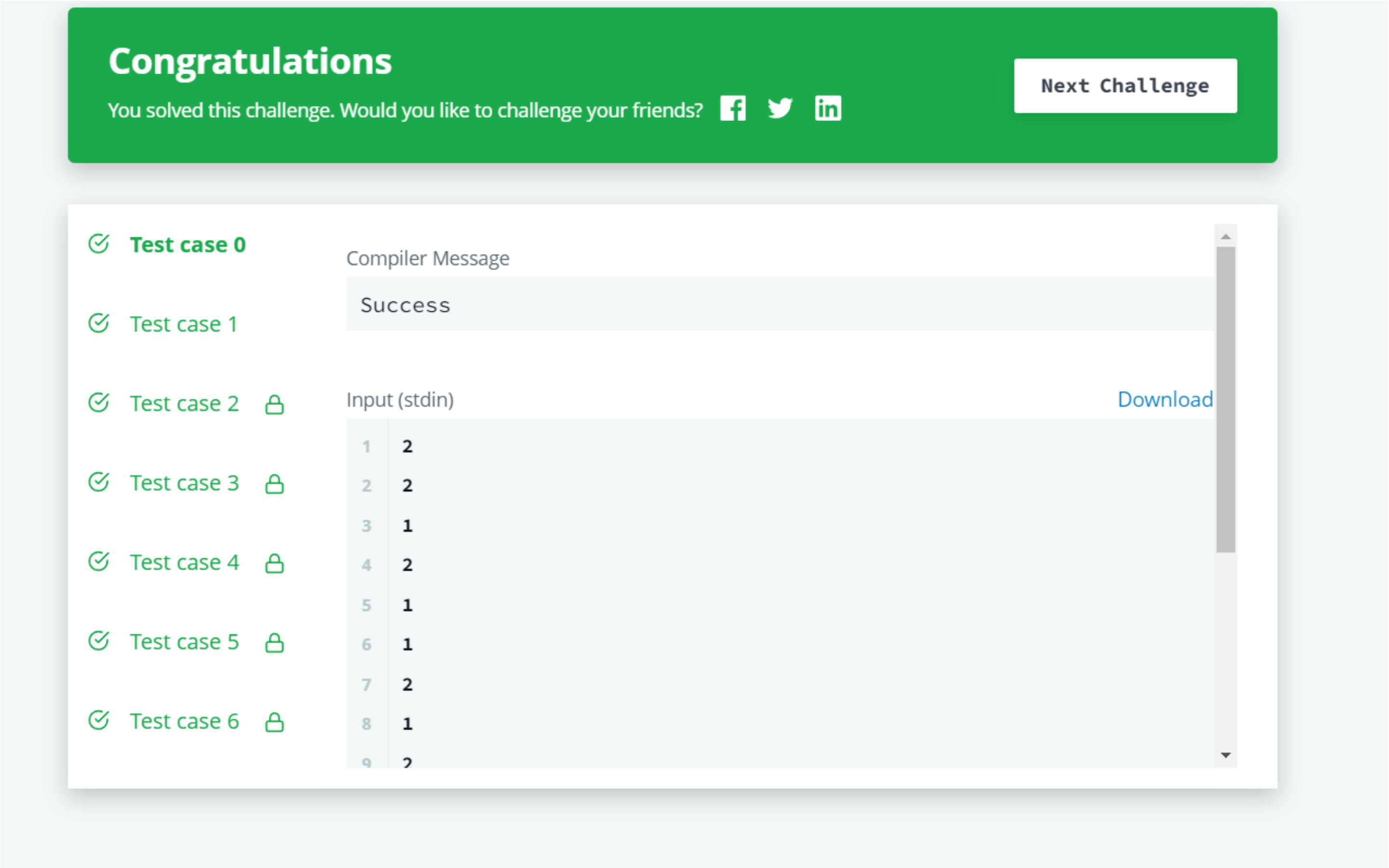
Result:



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

```
2
2
1
2
1
1
2
1
2
2
1
2
```



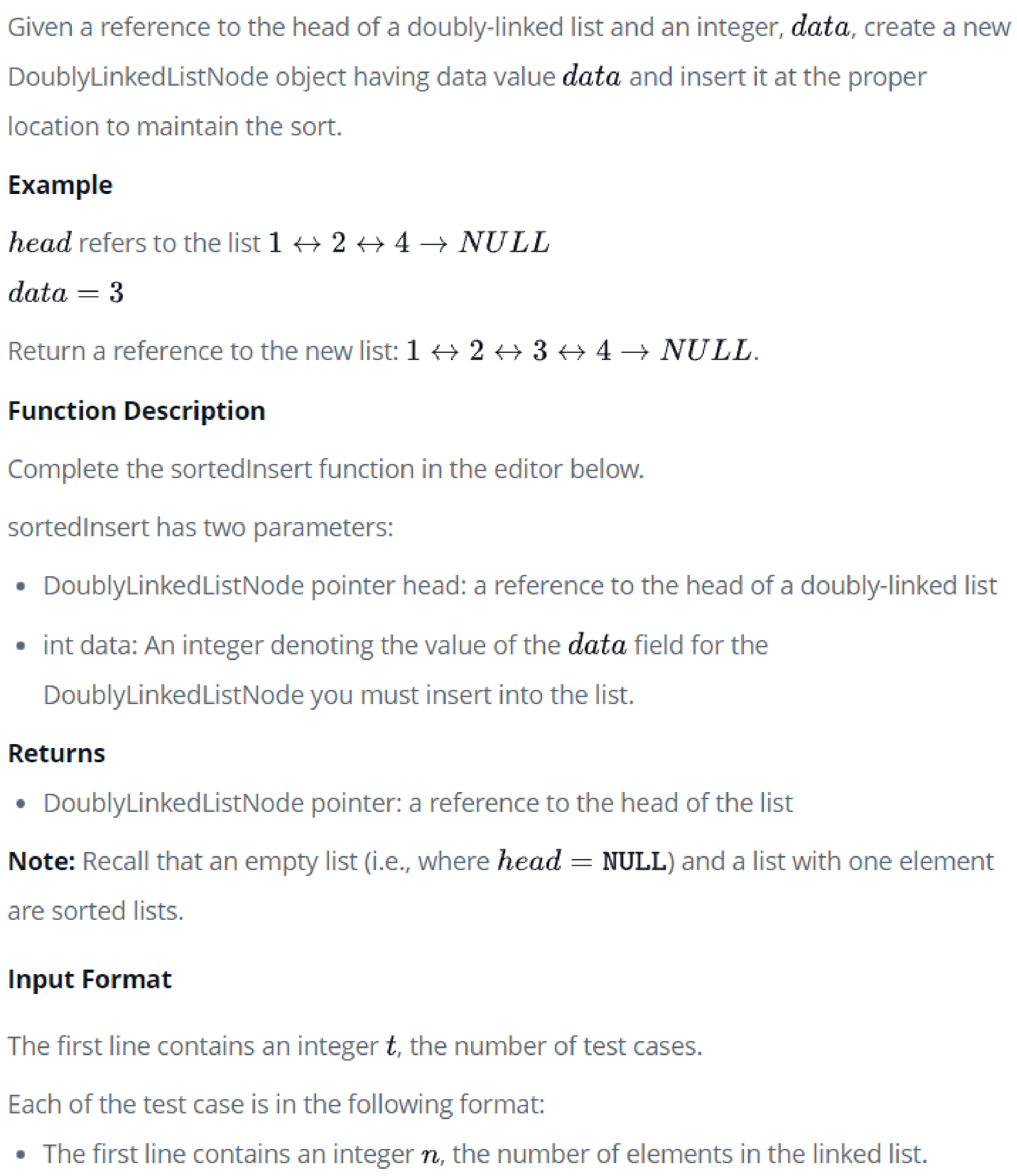
Congratulations

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

```
Success
```

```
2
2
1
2
1
1
2
1
2
```

Q2



Given a reference to the head of a doubly-linked list and an integer, $data$, create a new DoublyLinkedListNode object having data value $data$ and insert it at the proper location to maintain the sort.

Example

$head$ refers to the list $1 \leftrightarrow 2 \leftrightarrow 4 \rightarrow NULL$

$data = 3$

Return a reference to the new list: $1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4 \rightarrow NULL$.

Function Description

Complete the sortedInsert function in the editor below.

sortedInsert has two parameters:

- DoublyLinkedListNode pointer head: a reference to the head of a doubly-linked list
- int data: An integer denoting the value of the $data$ field for the DoublyLinkedListNode you must insert into the list.

Returns

- DoublyLinkedListNode pointer: a reference to the head of the list

Note: Recall that an empty list (i.e., where $head = \texttt{NULL}$) and a list with one element are sorted lists.

Input Format

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

Each of the test case is in the following format:

- The first line contains an integer n, the number of elements in the linked list.

- Each of the next n lines contains an integer, the data for each node of the linked list.
- The last line contains an integer, $data$, which needs to be inserted into the sorted doubly-linked list.

Constraints

- $1 \leq t \leq 10$
- $1 \leq n \leq 1000$
- $1 \leq DoublyLinkedListNode.data \leq 1000$

Sample Input

```
STDIN   Function
-----   --------
1       t = 1
4       n = 4
1       node data values = 1, 3, 4, 10
3
4
10
5       data = 5
```

Sample Output

```
1 3 4 5 10
```

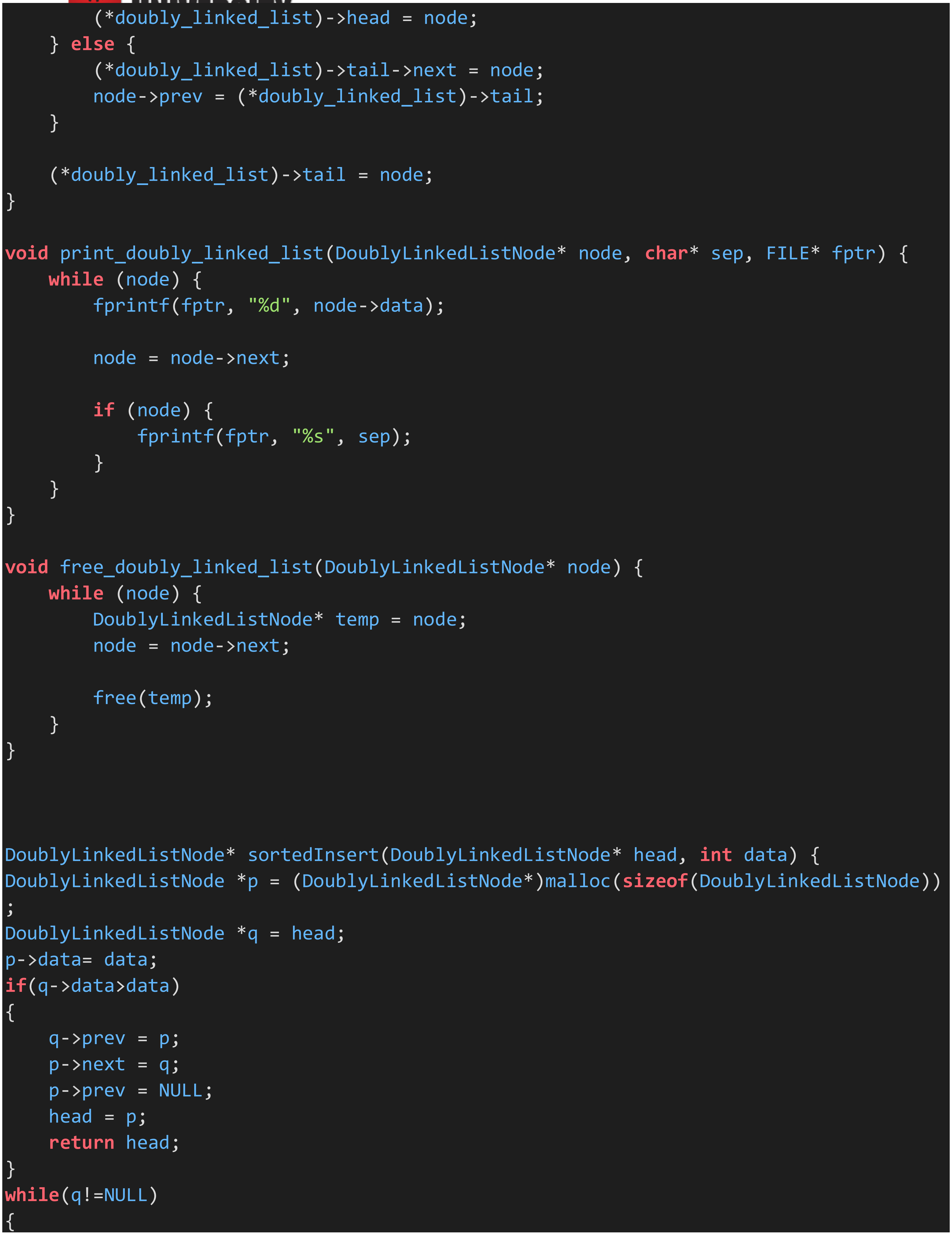
Explanation

The initial doubly linked list is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 10 \rightarrow NULL$.

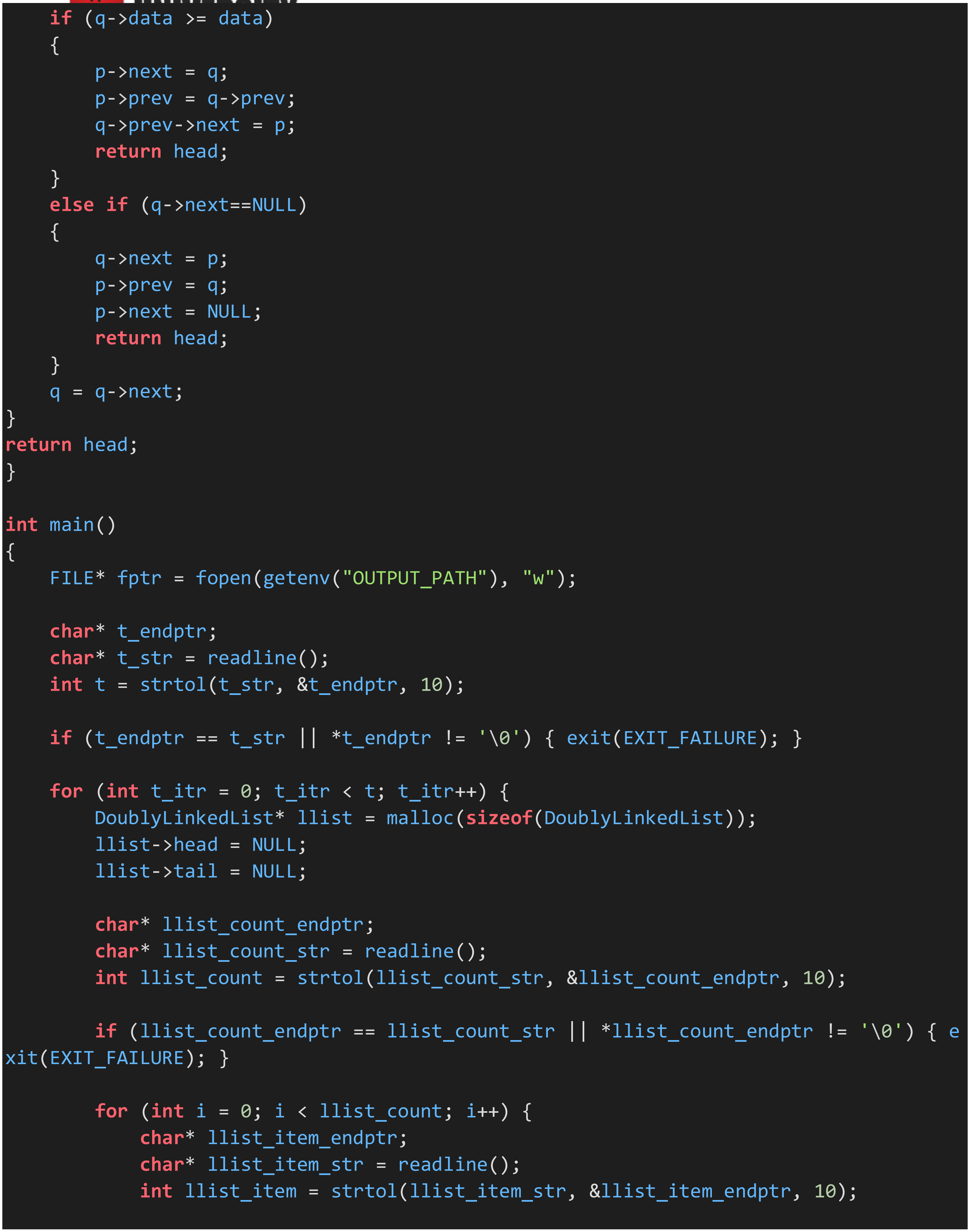
The doubly linked list after insertion is: $1 \leftrightarrow 3 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 10 \rightarrow NULL$

Code:

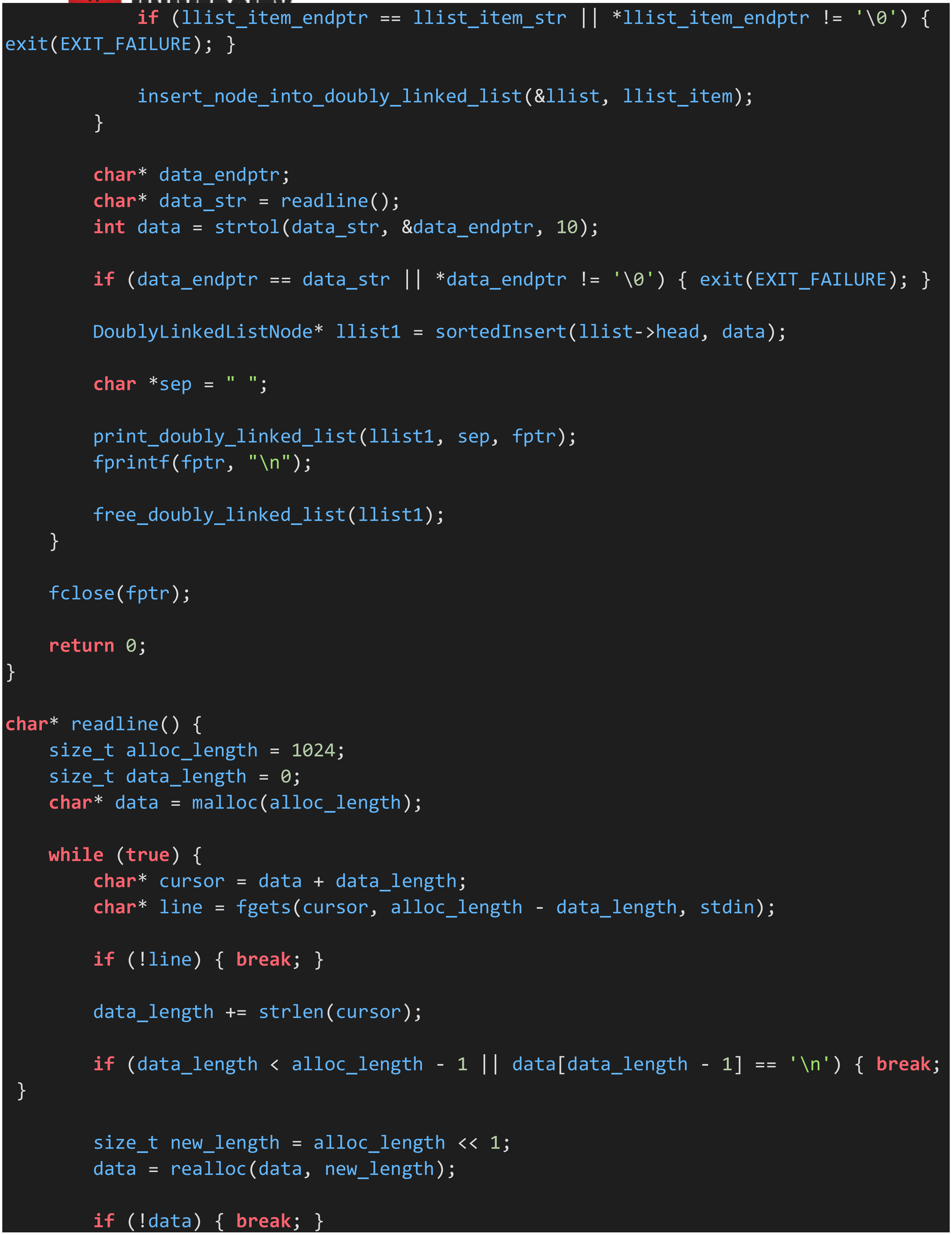
```
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

typedef struct DoublyLinkedListNode DoublyLinkedListNode;
typedef struct DoublyLinkedList DoublyLinkedList;

struct DoublyLinkedListNode {
    int data;
    DoublyLinkedListNode* next;
    DoublyLinkedListNode* prev;
};

struct DoublyLinkedList {
    DoublyLinkedListNode* head;
    DoublyLinkedListNode* tail;
};

DoublyLinkedListNode* create_doubly_linked_list_node(int node_data) {
    DoublyLinkedListNode* node = malloc(sizeof(DoublyLinkedListNode));

    node->data = node_data;
    node->next = NULL;
    node->prev = NULL;

    return node;
}

void insert_node_into_doubly_linked_list(DoublyLinkedList** doubly_linked_list, int n
ode_data) {
    DoublyLinkedListNode* node = create_doubly_linked_list_node(node_data);

    if (!(*doubly_linked_list)->head) {
```



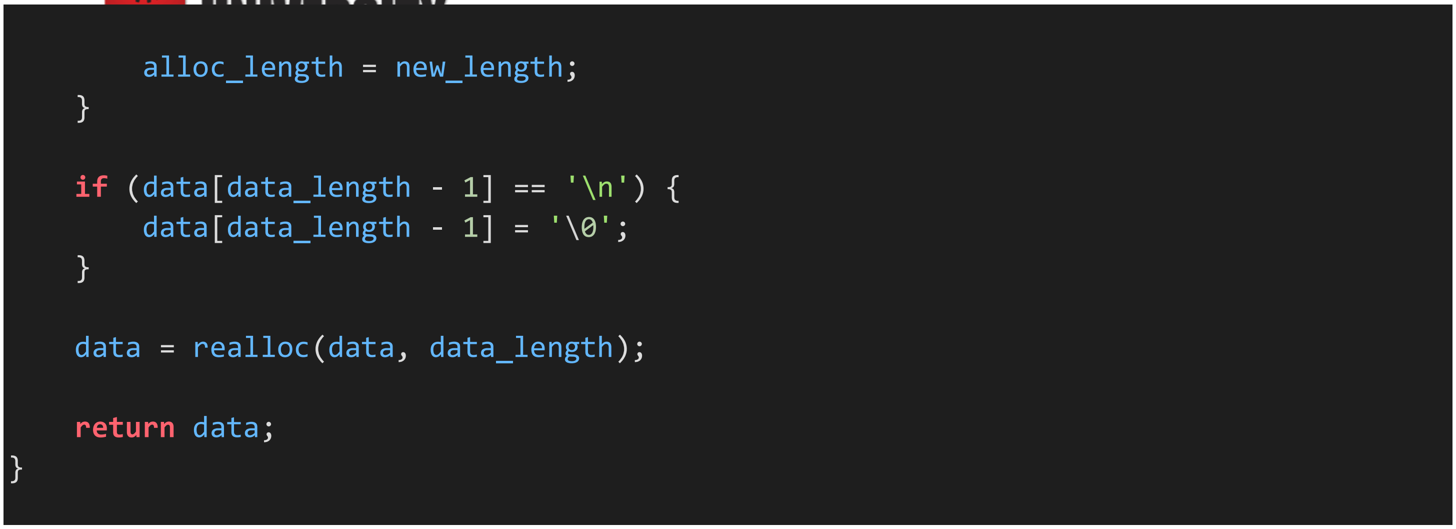
```
        (*doubly_linked_list)->head = node;
    } else {
        (*doubly_linked_list)->tail->next = node;
        node->prev = (*doubly_linked_list)->tail;
    }

    (*doubly_linked_list)->tail = node;
}

void print_doubly_linked_list(DoublyLinkedListNode* node, char* sep, FILE* fptr) {
    while (node) {
        fprintf(fptr, "%d", node->data);

        node = node->next;

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

void free_doubly_linked_list(DoublyLinkedListNode* node) {
    while (node) {
        DoublyLinkedListNode* temp = node;
        node = node->next;

        free(temp);
    }
}



DoublyLinkedListNode* sortedInsert(DoublyLinkedListNode* head, int data) {
DoublyLinkedListNode *p = (DoublyLinkedListNode*)malloc(sizeof(DoublyLinkedListNode))
;
DoublyLinkedListNode *q = head;
p->data= data;
if(q->data>data)
{
    q->prev = p;
    p->next = q;
    p->prev = NULL;
    head = p;
    return head;
}
while(q!=NULL)
{
```



```
    if (q->data >= data)
    {
        p->next = q;
        p->prev = q->prev;
        q->prev->next = p;
        return head;
    }
    else if (q->next==NULL)
    {
        q->next = p;
        p->prev = q;
        p->next = NULL;
        return head;
    }
    q = q->next;
}
return head;
}

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++) {
        DoublyLinkedList* llist = malloc(sizeof(DoublyLinkedList));
        llist->head = NULL;
        llist->tail = NULL;

        char* llist_count_endptr;
        char* llist_count_str = readline();
        int llist_count = strtol(llist_count_str, &llist_count_endptr, 10);

        if (llist_count_endptr == llist_count_str || *llist_count_endptr != '\0') { e
xit(EXIT_FAILURE); }

        for (int i = 0; i < llist_count; i++) {
            char* llist_item_endptr;
            char* llist_item_str = readline();
            int llist_item = strtol(llist_item_str, &llist_item_endptr, 10);
```

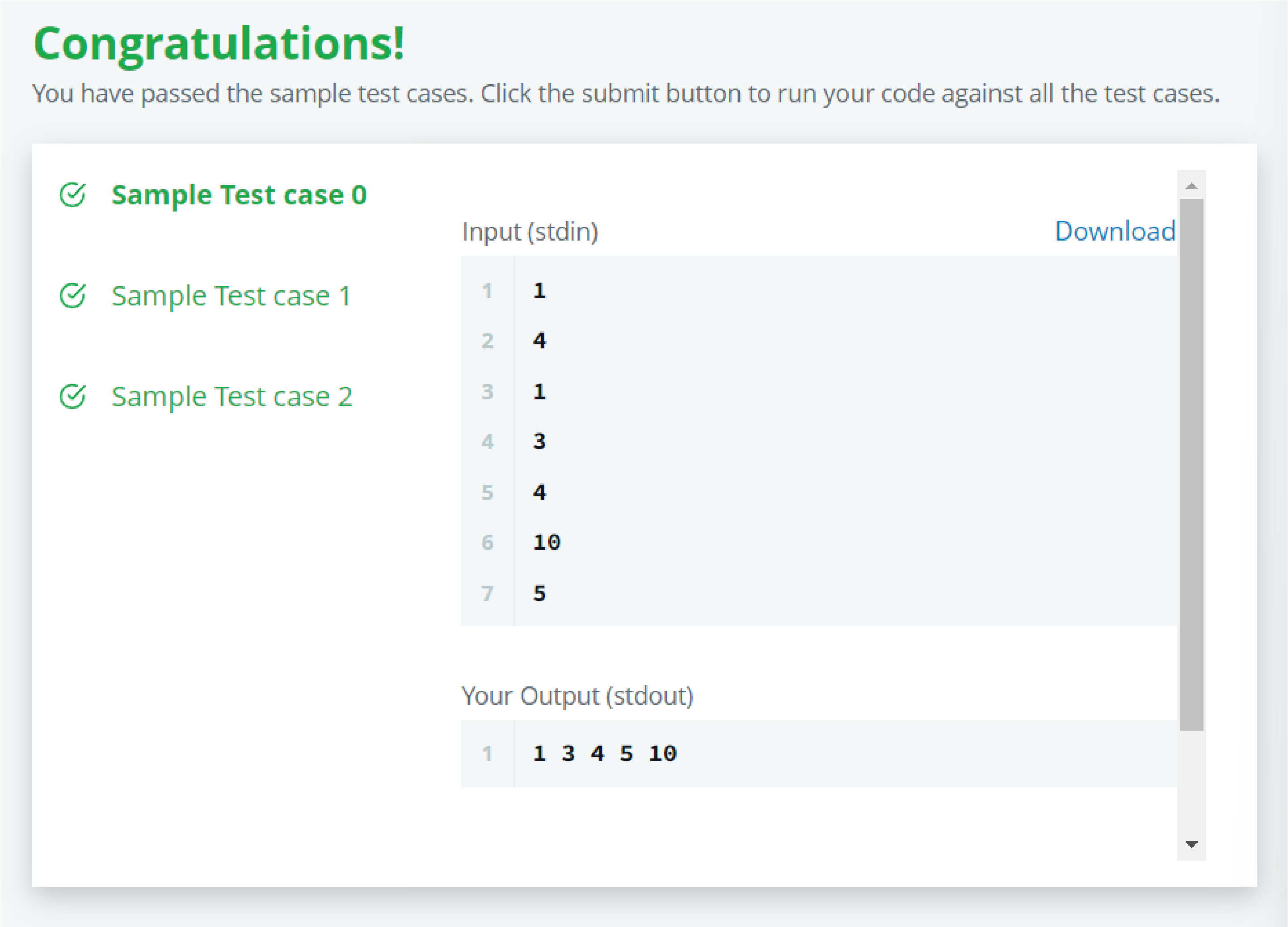


```
            if (llist_item_endptr == llist_item_str || *llist_item_endptr != '\0') {
exit(EXIT_FAILURE); }

            insert_node_into_doubly_linked_list(&llist, llist_item);
        }

        char* data_endptr;
        char* data_str = readline();
        int data = strtol(data_str, &data_endptr, 10);

        if (data_endptr == data_str || *data_endptr != '\0') { exit(EXIT_FAILURE); }

        DoublyLinkedListNode* llist1 = sortedInsert(llist->head, data);

        char *sep = " ";

        print_doubly_linked_list(llist1, sep, fptr);
        fprintf(fptr, "\n");

        free_doubly_linked_list(llist1);
    }

    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;
}
```

OUTPUT:



Congratulations!

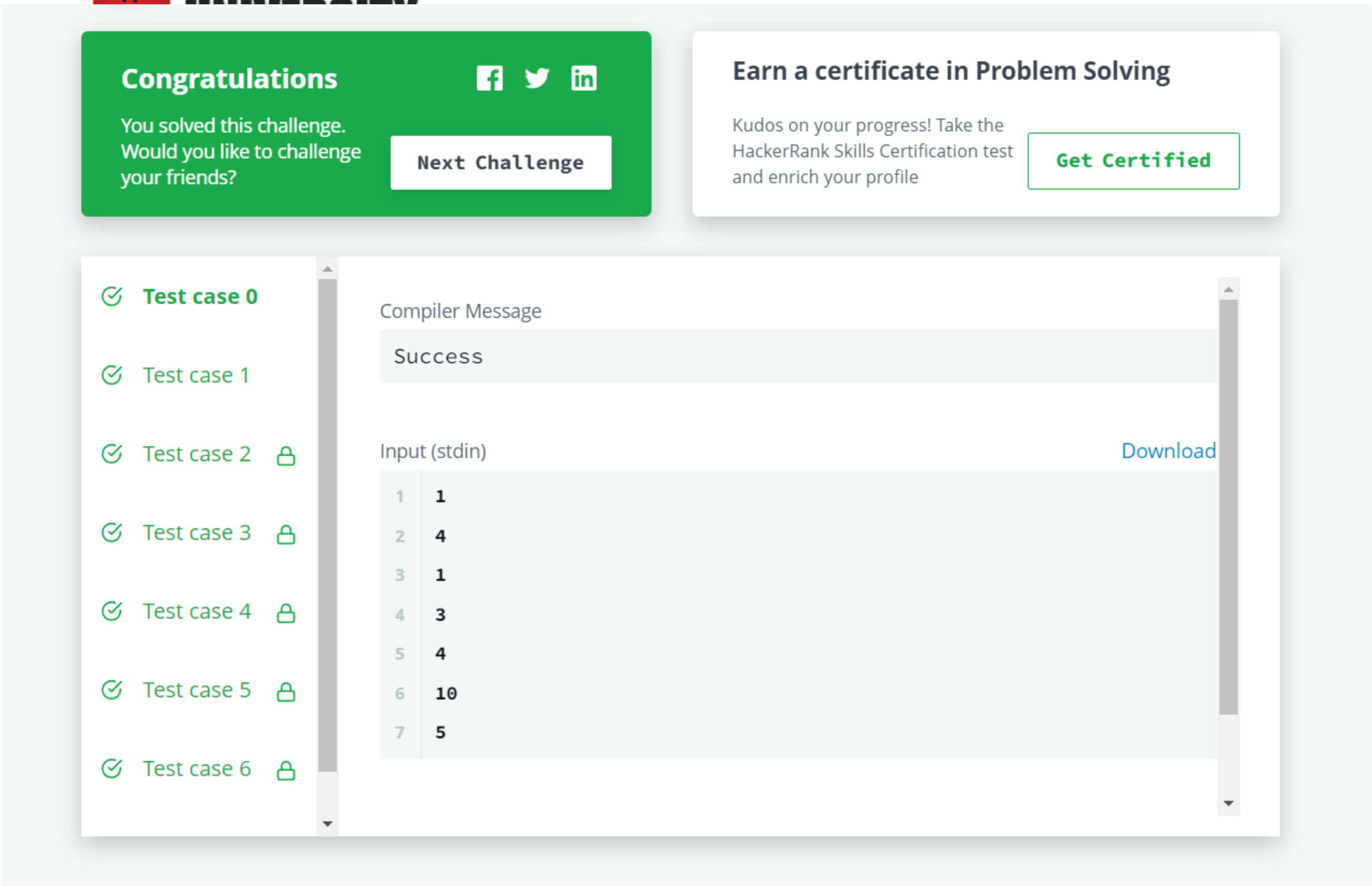
You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
1
4
1
3
4
10
5
```

Your Output (stdout)

```
1 3 4 5 10
```



Congratulations

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

Earn a certificate in Problem Solving

Kudos on your progress! Take the HackerRank Skills Certification test and enrich your profile

Compiler Message

```
Success
```

Input (stdin)

```
1
4
1
3
4
10
5
```