

Worksheet - 3

**Student Name:** ANIKET KUMAR **UID:**20BCS5306

**Branch:** BE-CSE **Semester:** 5th

**Subject Name:** Competitive Coding **Section/Group:** 20BCS\_WM-703/B

# Task:1

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 .

**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& fo ut) {

**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);

}

}

*// Complete the compare\_lists function below.*

*/\**

* *For your reference:*

*\**

* *SinglyLinkedListNode {*



* *int data;*
* *SinglyLinkedListNode\* next;*

*\* };*

*\**

*\*/*

**bool** compare\_lists(SinglyLinkedListNode\* head1, SinglyLinkedListNode\* head2) {

**while** (head1 != NULL && head2 != NULL) {

**if**(head1->data == head2->data){ head1 = head1->next;

head2 = head2->next;

}

## else {

**return** 0;

}

}

**if**(head1 == head2){

## return true;

}

## else {

**return false**;

}

}

**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');

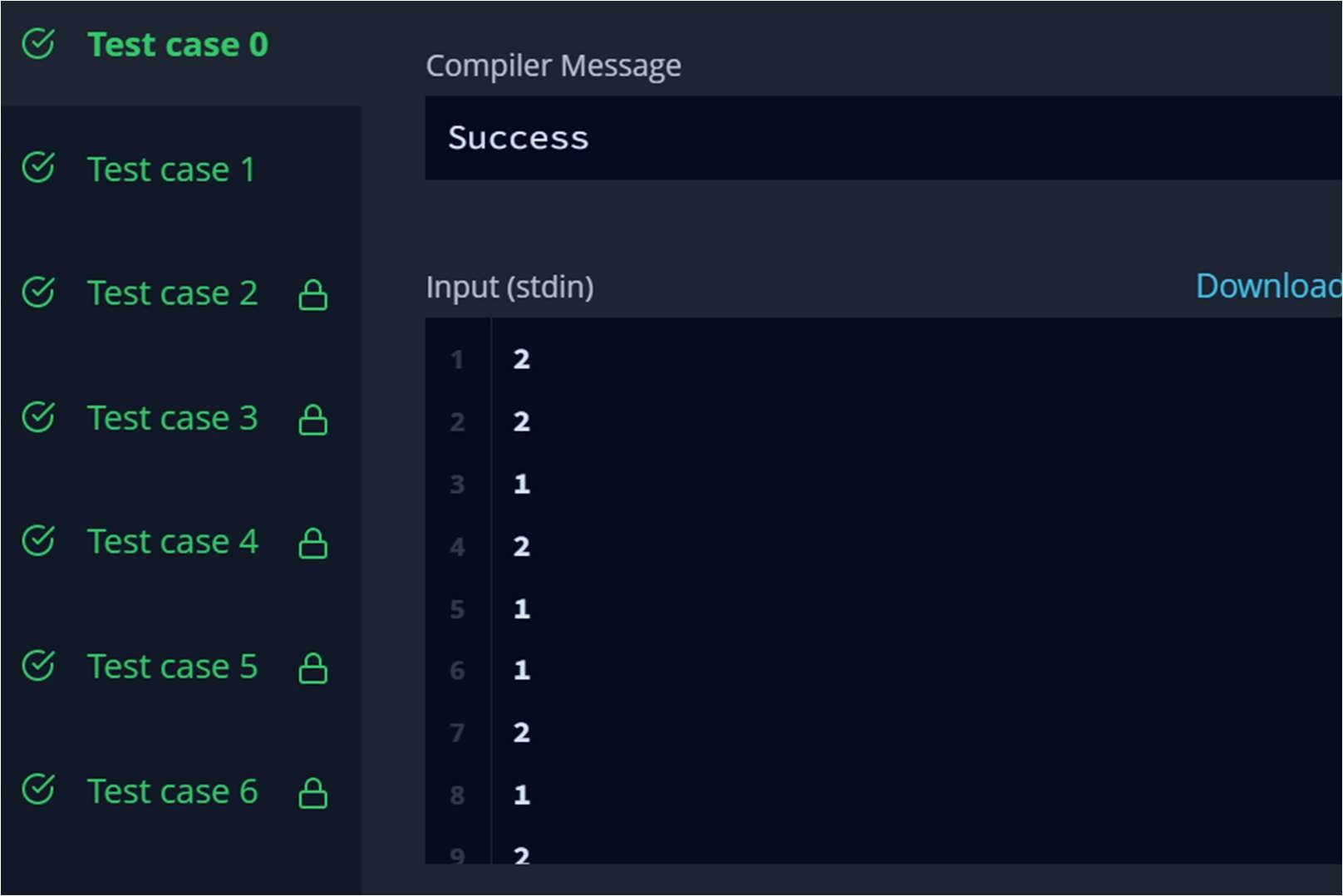
**for** (**int** i = 0; i < llist1\_count; i++) {

**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;

}

# Hacker Rank Test Case / Output:



**Task:2**

A linked list is said to contain a *cycle* if any node is visited more than once while traversing the list. Given a pointer to the head of a linked list, determine if it contains a cycle. If it does, return 1 Otherwise, return 0.

# 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& fo ut) {

**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);

}

}

*// Complete the has\_cycle function below.*

*/\**

* *For your reference:*

*\**

* *SinglyLinkedListNode {*
* *int data;*
* *SinglyLinkedListNode\* next;*

*\* };*

*\**

*\*/*

**bool** has\_cycle(SinglyLinkedListNode\* head) {

SinglyLinkedListNode\* i = head; SinglyLinkedListNode\* j = head; **while**(j !=NULL && i !=NULL){

i = i -> next;

**if**(j -> next != NULL){

j = j -> next -> next;



}**else**{

j = NULL;

}

**if**(j == i && j != NULL) **return true**;

}

## return false;

}

**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++) {

**int** index; cin >> index;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

SinglyLinkedList\* llist = **new** SinglyLinkedList();

**int** llist\_count; cin >> llist\_count;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**for** (**int** i = 0; i < llist\_count; i++) {

**int** llist\_item; cin >> llist\_item;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

llist->insert\_node(llist\_item);

}

SinglyLinkedListNode\* extra = **new** SinglyLinkedListNode(-1); SinglyLinkedListNode\* temp = llist->head;

**for** (**int** i = 0; i < llist\_count; i++) {

**if** (i == index) { extra = temp;

}



**if** (i != llist\_count-1) { temp = temp->next;

}

}

temp->next = extra;

**bool** result = has\_cycle(llist->head); fout << result << "\n";

}

fout.close();

**return** 0;

}

# Hacker Rank Test Case / Output:

