Experiment Title : 4

Student Name: Rajiv Paul **UID: 20BCS1812**
Branch: CSE **Section/Group: 702 A**
Semester: 5 **Date of Performance:15/09/22**
Subject Name: Competitive Coding-I **Subject Code: 20CSP-314**

Problem Statement 1 merge-point-of-two-joined-linked-lists

Given pointers to the head nodes of 2 linked lists that merge together at some point, find the node where the two lists merge. The merge point is where both lists point to the same node, i.e. they reference the same memory location. It is guaranteed that the two head nodes will be different, and neither will be NULL. If the lists share a common node, return that node's *data* value.

Note: After the merge point, both lists will share the same node pointers.

Example

In the diagram below, the two lists converge at Node x:

```
[List #1] a--->b--->c
                   \
                    x--->y--->z--->NULL
                   /
    [List #2] p--->q
```

Function Description

Complete the findMergeNode function in the editor below.

findMergeNode has the following parameters:

- SinglyLinkedListNode pointer head1: a reference to the head of the first list
- SinglyLinkedListNode pointer head2: a reference to the head of the second list

Returns

- int: the *data* value of the node where the lists merge

Input Format

Do not read any input from stdin/console.

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

Each of the test cases is in the following format:

The first line contains an integer, $index$, the node number where the merge will occur.

The next line contains an integer, $list1_count$ that is the number of nodes in the first list.

Each of the following $list1_count$ lines contains a $data$ value for a node. The next line contains an integer, $list2_count$ that is the number of nodes in the second list.

Each of the following $list2_count$ lines contains a $data$ value for a node.

Constraints

The lists will merge.

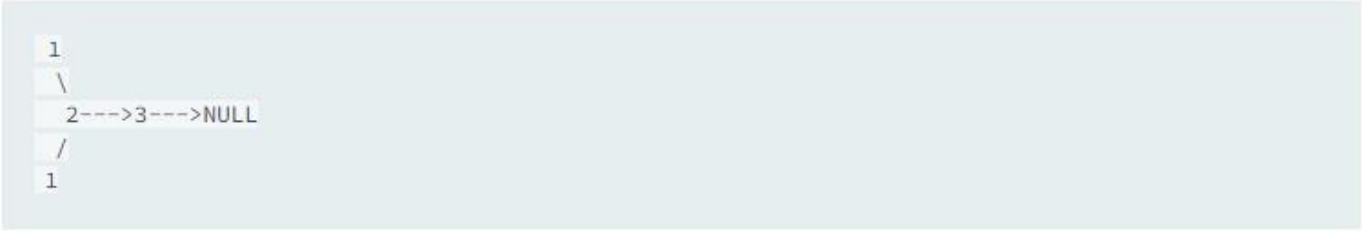
$head1, head2 \neq null$.

$head1 \neq head2$.

Sample Input

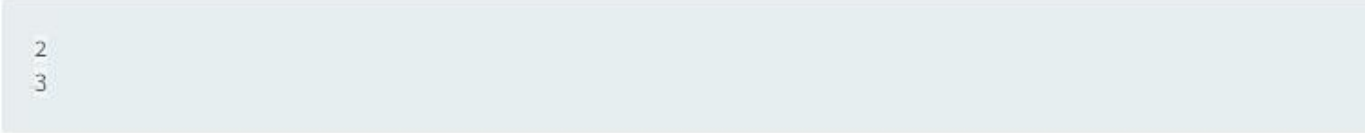
The diagrams below are graphical representations of the lists that input nodes $head1$ and $head2$ are connected to.

Test Case 0



Test Case 1

Sample Output



Solution:-

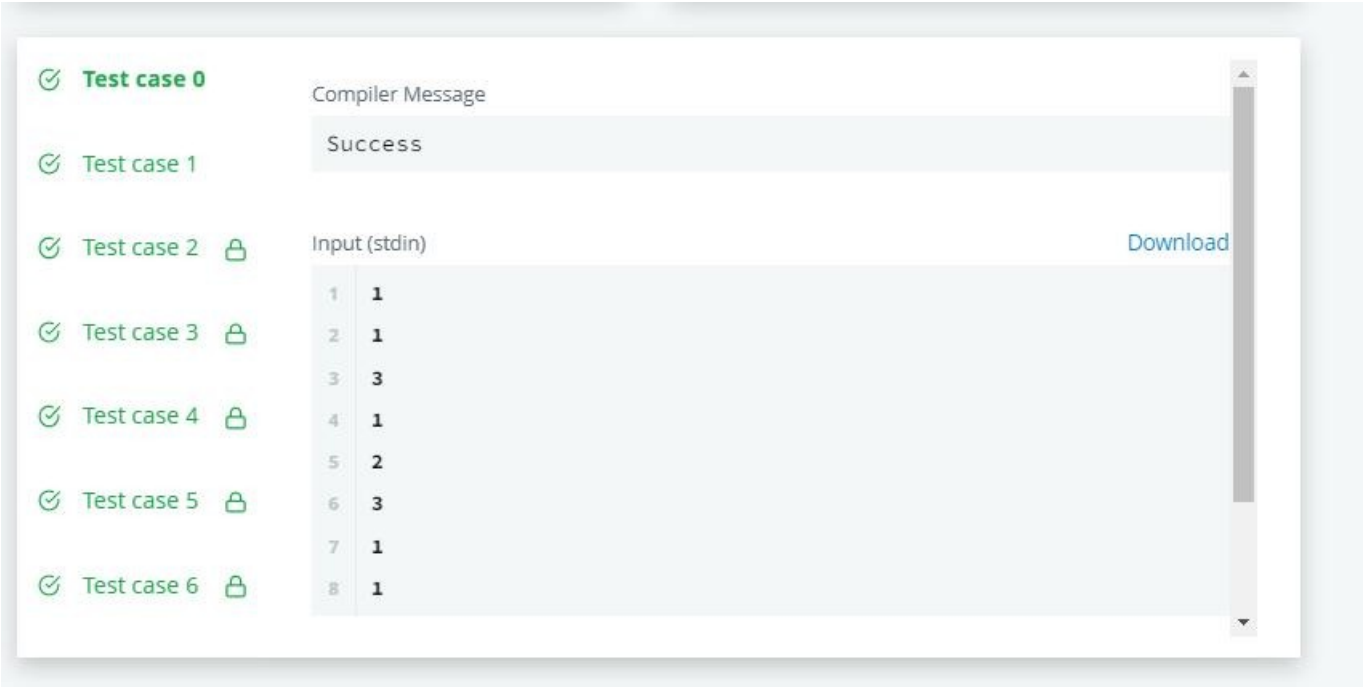
```
#include <bits/stdc+

+.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);
    }
}

int findMergeNode(SinglyLinkedListNode* headA, SinglyLinkedListNode*
    headB) {while(headA){
        SinglyLinkedListNode *tmp = headA-
        >next;headA->next = NULL;
        headA = tmp;
    }

    while(headB){
        if(headB->next ==
            NULL){return
            headB->data;
        }
        headB = headB->next;
    }
    return 0;
}
```

```
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* 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);
        }

        SinglyLinkedListNode* ptr1 = llist1-
        >head;SinglyLinkedListNode* ptr2 =
        llist2->head;

        for (int i = 0; i < llist1_count; i+
            +) {if (i < index) {
                ptr1 = ptr1->next;
            }
        }

        for (int i = 0; i < llist2_count; i+
            +) {if (i != llist2_count-1) {
                ptr2 = ptr2->next;
            }
        }

        ptr2->next = ptr1;

        int result = findMergeNode(llist1->head, llist2->head);

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

    fout.close();

    return 0;
}
```

Output:-


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
Compiler Message
Success
Input (stdin)
Download
1 1
2 1
3 3
4 1
5 2
6 3
7 1
8 1

Problem Statement 2 Cycle Detection

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.

Example

$head$ refers to the list of nodes $1 \rightarrow 2 \rightarrow 3 \rightarrow NULL$

The numbers shown are the node numbers, not their data values. There is no cycle in this list so return 0.

$head$ refers to the list of nodes $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow NULL$

There is a cycle where node 3 points back to node 1, so return 1.

Function Description

Complete the has_cycle function in the editor below.

It has the following parameter:

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

Returns

- int: 1 if there is a cycle or 0 if there is not

Note: If the list is empty, $head$ will be null.

Input Format

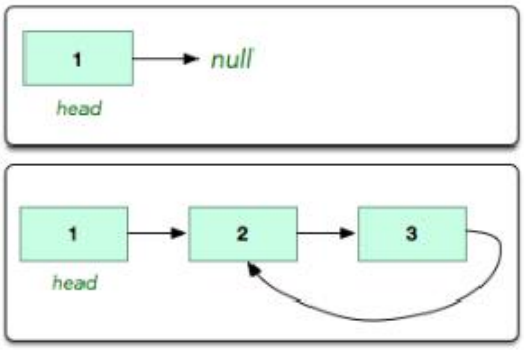
The code stub reads from stdin and passes the appropriate argument to your function. The custom test cases format will not be described for this question due to its complexity. Expand the section for the main function and review the code if you would like to figure out how to create a custom case.

Constraints

- $0 \leq$ *list size* ≤ 1000

Sample Input

References to each of the following linked lists are passed as arguments to your function:



Sample Output

```
0
1
```

Explanation

1. The first list has no cycle, so return 0.
2. The second list has a cycle, so return 1.

Solution:

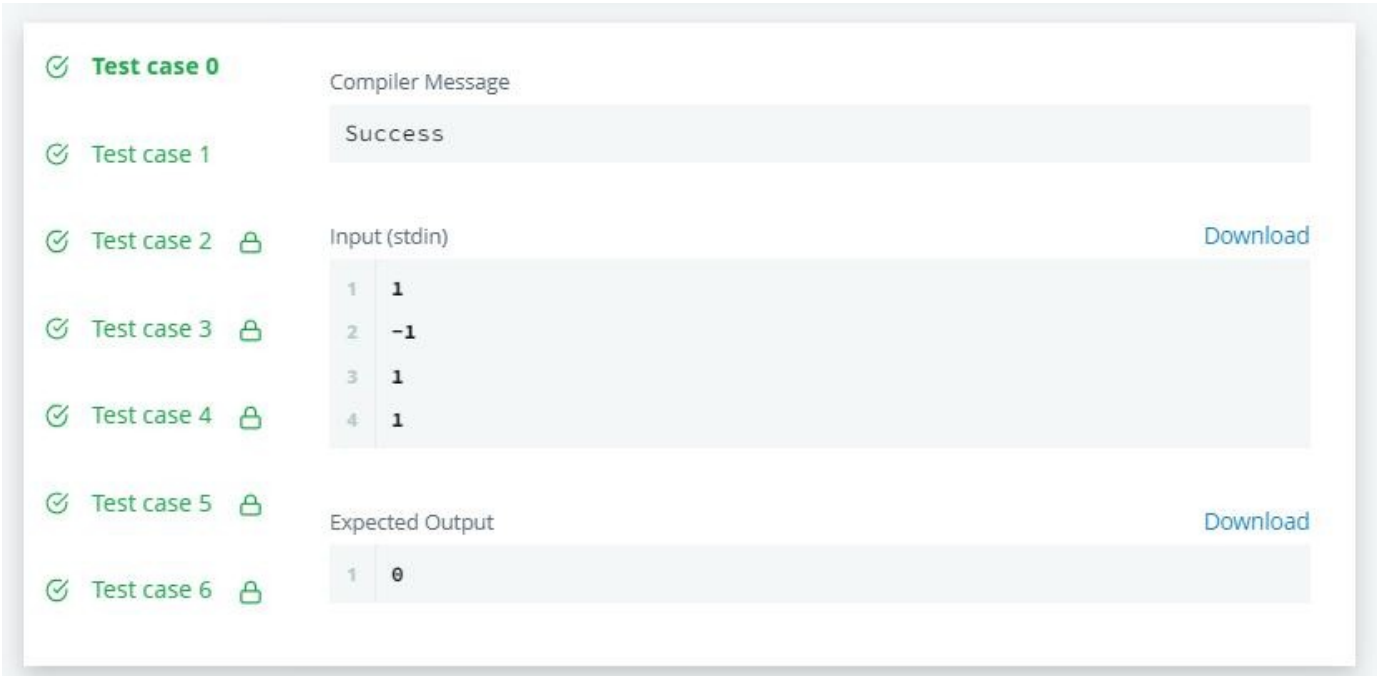
```
#include <bits/stdc+
+.h>using namespace
std;

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 has_cycle(SinglyLinkedListNode*
head) {SinglyLinkedListNode* cur1 =
```

```
head;
    SinglyLinkedListNode* cur2 =
    head;int result = 0;
    while (cur1 && cur2)
    {
        cur1    =    cur1-
        >next;   cur2    =
        cur2->next;    if
        (cur2)
        {
            cur2 = cur2->next;
        }

        if (cur1 == cur2)
        {
            result =
            1;break;
        }
    }
    return result;

}

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;
}
```

Output:


Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
Compiler Message
Success
Input (stdin)
Download
1 1
2 -1
3 1
4 1
Expected Output
Download
1 0