

## Experiment – 1.3

Name -Bhanu Pundir

Uid-20BCS1439

CLASS- 620B

Subject-Competitive Coding

### 1. Compare two linked lists

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**.

```
#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;

}

```

✓ **Test case 0**

✓ Test case 1

✓ Test case 2 

✓ Test case 3 

✓ Test case 4 

✓ Test case 5 

✓ Test case 6 

5	1
6	1
7	2
8	1
9	2
10	2
11	1
12	2

Expected Output

[Download](#)

1	0
2	1

## 2. Inserting a Node Into a Sorted Doubly Linked List

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:  $1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4 \rightarrow NULL$ .

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

using namespace std;

class
DoublyLinkedListNode {
public:
    int data;
    DoublyLinkedListNode *next;
    DoublyLinkedListNode *prev;

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

class DoublyLinkedList {
public:
    DoublyLinkedListNode *head;
    DoublyLinkedListNode *tail;

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

    void insert_node(int node_data) {
        DoublyLinkedListNode* node = new DoublyLinkedListNode(no
            de_data);

        if (!this->head) {
            this->head = node;

        } else {
```

```

        this->tail->next = node;        node->prev = this->tail;
    }

```

```

        this->tail = node;
    }
};

```

```

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

```

```

        node = node->next;

        if (node) {
            fout << 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* node = new DoublyLinkedListNode(data);

```

```

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

```

```

    if(head==NULL)
        return node;

```

```

    if(head->data > data){
        head->prev = node; node->next
        = head;

```

```

        return node;
    }
}

```

```

        DoublyLinkedListNode* next = sortedInsert(head->next, data); head->next =
next;
        next->prev = head;
return head;

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

    int t;
    cin >> t;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int t_itr = 0; t_itr < t; t_itr++) { DoublyLinkedList* llist = new
DoublyLinkedList();

        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);
        }

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

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

        print_doubly_linked_list(llist1, " ", fout); fout << "\n";

        free_doubly_linked_list(llist1);
    }

    fout.close();

    return 0;
}

```



✓ **Test case 0**

✓ Test case 1

✓ Test case 2 

✓ Test case 3 

✓ Test case 4 

✓ Test case 5 

✓ Test case 6 

Input (stdin)

Download

1 1  
2 4  
3 1  
4 3  
5 4  
6 10  
7 5

Expected Output

Download

1 1 3 4 5 10