AKFA UNIVERSITY

Algorithms and Data Structures
BS-III Fall 2022
LAB NO: 05

Name	
Student ID	
Date	

Task 1. Finding the middle node of the linked list using iteration

This task is taken from leetcode.com https://leetcode.com/problems/middle-of-the-linked-list/

Given the head of a singly linked list, return the middle node of the linked list.

If there are two middle nodes, return the second middle node.

Test cases:

Test case 1:

List obj;

obj.insertAtLast(3);

obj.insertAtLast(1);

obj.insertAtLast(2);

obj.insertAtLast(5);

obj.insertAtLast(15);

There are five nodes in the list. The output should be the following

Middle node is: 2

Test case 2:

List obj;

obj.insertAtLast(7);

obj.insertAtLast(1);

obj.insertAtLast(9);

obj.insertAtLast(3);

obj.insertAtLast(5);

obj.insertAtLast(15);

There are six nodes in the list. There are two middle nodes, however, it should print the second middle node. Therefore, the output should be the following

Middle node is: 3

Task 2. Copy function

Create a function called CopyList that takes an object of the List class and copy it into the object for whom the function is called.

Test cases:

Test case 1:

List obj;

obj.insertAtLast(3);

obj.insertAtLast(1);

obj.insertAtLast(2);

obj.insertAtLast(5);

obj.insertAtLast(15);

List obj2;

obj2.CopyList(obj);

obj2.printAll();

Output:

3 1 2 5 15

Test case 2:

List obj;

obj.insertAtLast(3);

obj.insertAtLast(1);

obj.insertAtLast(2);

obj.insertAtLast(5);

obj.insertAtLast(15);

List obj2;

obj2.CopyList(obj);

obj.insertAtLast(18);

obj.insertAtLast(13);

obj2.printAll();

Output:

3 1 2 5 15

Note: You can use printAll function that was provided in Lab 4_1

Task 3: Union of unordered linked list

Create a function named unionList that takes the union of two lists, stores it into another list, and return the list object that contain the union

```
#include<iostream>
#include<conio.h>
using namespace std;
struct Node
{
	int data;
	Node* next;
};
class List
{
	Node* head, *last;
public:
	List() { head =last= NULL; }
	void insertAtLast(int data)
	{
		Node* nN = new Node;
		nN->data = data;
		nN->next = NULL;
		if (!head)
		{
			head = last = nN;
		}
		else
		{
```

```
                last->next = nN;
                last = nN;
        }
}
void printAll()
{
        Node* curr = head;
        while (curr)
        {
                cout << curr->data << "   ";
                curr = curr->next;
        }
}
bool isInList(Node* ptr, int data)
{
        while (ptr)
        {
                if (ptr->data == data)
                        return true;
                ptr = ptr->next;

        }
        return false;
}
List unionList(List obj)
{
        Node* curr1 = head, *curr2 = obj.head;
        List newList;
```

Your logic here

```
        return newList;
    }
};
```

Test case 1:

```
void main()
{
        List obj1, obj2;
        obj1.insertAtLast(5);
        obj1.insertAtLast(15);
        obj1.insertAtLast(25);
        obj1.insertAtLast(35);
        obj1.insertAtLast(45);
        obj1.insertAtLast(5);
        obj1.insertAtLast(5);

        obj2.insertAtLast(15);
```

```
        obj2.insertAtLast(25);
        obj2.insertAtLast(4);
        obj2.insertAtLast(5);
        obj2.insertAtLast(51);
        List obj3 = obj1.unionList(obj2);
        obj3.printAll();
        _getch();
}
```

Output:

5 15 25 35 45 4 51

Note: You should use isInList function for the implementation of unionList function. This function takes the data and head pointer of a list, and return true if the item is already in the linked list. Also, the order of the data in obj3 doesn't matter.

Task 3: Intersection of unordered linked list

Create a function named intersectList that takes the union of two lists, stores it into another list, and return the list object that contain the union

```
#include<iostream>
#include<conio.h>
using namespace std;
struct Node
{
        int data;
        Node* next;
};
class List
{
        Node* head, *last;
public:
        List() { head =last= NULL; }
        void insertAtLast(int data)
        {
                Node* nN = new Node;
                nN->data = data;
                nN->next = NULL;
                if (!head)
                {
                        head = last = nN;
                }
                else
                {
                        last->next = nN;
                        last = nN;
```

```
		}
	}
	void printAll()
	{
		Node* curr = head;
		while (curr)
		{
			cout << curr->data << "   ";
			curr = curr->next;
		}
	}
	bool isInList(Node* ptr, int data)
	{
		while (ptr)
		{
			if (ptr->data == data)
				return true;
			ptr = ptr->next;

		}
		return false;
	}
	List intersectList(List obj)
	{
		Node* curr1 = head, *curr2 = obj.head;
		List newList;
```

Your logic here

```
		return newList;
	}
};
```

Test case 1:

```
void main()
{
	List obj1, obj2;
	obj1.insertAtLast(5);
	obj1.insertAtLast(15);
	obj1.insertAtLast(25);
	obj1.insertAtLast(35);
	obj1.insertAtLast(45);
	obj1.insertAtLast(5);
	obj1.insertAtLast(5);

	obj2.insertAtLast(15);
	obj2.insertAtLast(25);
	obj2.insertAtLast(4);
```

```
        obj2.insertAtLast(5);
        obj2.insertAtLast(51);
        List obj3 = obj1.intersectList(obj2);
        obj3.printAll();
        _getch();
}
```

Output:

15 25 5

Note: You should use isInList function for the implementation of intersectList function. This function takes the data and head pointer of a list, and return true if the item is already in the linked list. Also, the order of the data in obj3 doesn't matter.