# Lab 4: LinkedList - Stack - Queue

## 1 Singly Linkedlist

Following is a representation of a singly linked list

```
struct NODE{
    int key;
    NODE* p_next;
};
```

```
struct List{
    NODE* p_head;
    NODE* p_tail;
};
```

Complete the following functions to fulfill the given requirements (linkedlist without `p_tail`):

1. Initialize a `NODE` from a given integer:
   - `NODE* createNode(int data)`
2. Insert an integer to the head of a given linkedlist:
   - `void addHead(Node* &pHead, int data)`
3. Insert an integer to the tail of a given linkedlist:
   - `void addTail(Node* &pHead, int data)`
4. Remove the first `NODE` of a given linkedlist:
   - `void removeHead(Node* &pHead)`
5. Remove the last `NODE` of a given linkedlist:
   - `void removeTail(Node* &pHead)`
6. Remove all `NODE` from a given linkedlist:
   - `void removeAll(Node* &pHead)`
7. Remove an integer before a value of a given linkedlist:
   - `void removeBefore(Node* &pHead, int val)`
8. Remove an integer after a value of a given linkedlist:
   - `void romveAfter(Node* &pHead, int val)`
9. Insert an integer at a position of a given linkedlist:
   - `bool addPos(Node* &pHead, int data, int pos)`
10. Remove an integer at a position of a given linkedlist:
    - `void RemovePos(Node* &pHead, int pos)`
11. Insert an integer before a value of a given linkedlist:
    - `void addBefore(Node* &pHead, int data, int val)`
12. Insert an integer after a value of a given linkedlist:
    - `void addAfter(Node* &pHead, int data, int val)`
13. Print all elements of a given linkedlist:
    - `void printList(Node* &pHead)`
14. Count the number of elements linkedlist:
    - `int countElements(Node* &pHead)`
15. Count the number of appearances of a value in a given linkedlist:
    - `int countAppearance(Node* &pHead, int value)`
16. Create a new `List` by reverse a given linkedlist:
    - `Node* reverseList(Node* &pHead)`
17. Remove all duplicates from a given linkedlist:
    - `void removeDuplicate(Node* &pHead)`
18. Remove all `key` value from a given linkedlist:
    - `bool removeElement(Node* &pHead, int key)`

Complete the following functions to fulfill the given requirements (linkedlist with given `p_tail`):

1. Initialize a `List` from a give `NODE`:
   - `List* createList(NODE* p_node)`
2. Insert an integer to the head of a given `List`:
   - `bool addHead(List* &L, int data)`
3. Insert an integer to the tail of a given `List`:
   - `bool addTail(List* &L, int data)`
4. Remove the first `NODE` of a given `List`:
   - `void removeHead(List* &L)`
5. Remove the last `NODE` of a given `List`:
   - `void removeTail(List* &L)`
6. Remove all `NODE` from a given `List`:
   - `void removeAll(List* &L)`
7. Remove an integer before a value of a given `List`:
   - `void removeBefore(List* &L, int val)`
8. Remove an integer after a value of a given `List`:
   - `void romveAfter(List* &L, int val)`
9. Insert an integer at a position of a given `List`:
   - `bool addPos(List* &L, int data, int pos)`
10. Remove an integer at a position of a given `List`:
    - `void RemovePos(List* &L, int pos)`
11. Insert an integer before a value of a given `List`:
    - `bool addBefore(List* &L, int data, int val)`
12. Insert an integer after a value of a given `List`:
    - `bool addAfter(List* &L, int data, int val)`
13. Print all elements of a given `List`:
    - `void printList(List* L)`
14. Count the number of elements `List`:
    - `int countElements(List* L)`
15. Count the number of appearance of a value in a given linkedlist:
    - `int countAppearance(List* L, int value)`
16. Create a new `List` by reverse a given `List`:
    - `List* reverseList(List* L)`
17. Remove all duplicates from a given `List`:
    - `void removeDuplicate(List* &L)`
18. Remove all `key` value from a given `List`:
    - `bool removeElement(List* &L, int key)`

## 2 Doubly Linkedlist

Following is representation of a doubly linked list:

```
struct d_NODE{
    int key;
    d_NODE* pNext;
    d_NODE* pPrev;
};
```

```
struct d_List{
    d_NODE* pHead;
    d_NODE* pTail;

};
```

Implement functions to execute the operations from a singly linkedlist section.

## 3 Stack - Queue

Following is the representation of a Singly linked list node:

```
struct NODE{
    int key;
    NODE* pNext;
};
```

Utilize the Linked list above, define the data structure of Stack and Queue, then implement functions to execute the following operations:

1. Stack
    - **Initialize** a stack from a given key.
    - **Push** a key into a given stack.
    - **Pop** an element out of a given stack, return the key's value.
    - **Count** the number of elements of a given stack.
    - Determine if a given stack **is empty**.
2. Queue
    - **Initialize** a queue from a given key.
    - **Enqueue** a key into a given queue.
    - **Dequeue** an element out of a given queue, return the key's value.
    - **Count** the number of elements of a given queue.
    - Determine if a given queue **is empty**.