CS 131 SI - Week 5 - Linked Lists and Doubly Linked Lists

1. **Illustrate** (not code) how the following operations would look with a SinglyLinkedList:
    a. Create an empty linked list

    b. Add ‘z’ to the front of the list

    c. Add ‘c’ to the front of the list

    d. Remove from the front of the list

    e. Add ‘g’ to the front of the list

    f. Add ‘o’ to the front of the list

    g. Add ‘d’ to the front of the list

    h. Clear the list

2. What are the essential components an iterator must have?

3. Checkmark the ADTs you believe would be the fastest at the following operations:
   **HINT:** Think about edge cases
   < ELT and LIST are template types. ELT is an element and LIST is another list>

| Operations | FixedVector | SinglyLinkedList | DoublyLinkedList |
|---|---|---|---|
| insert_front(ELT e) | | | |
| remove_front(ELT e) | | | |
| insert_back(ELT e) | | | |
| remove_back(ELT e) | | | |
| get(int index) | | | |
| clear() | | | |
| concatenate (LIST other) | | | |
| print() | | | |

4. Create the following methods in the below class (assume all other code is similar to the SinglyLinkedList class in the class GitHub page). **Perform any necessary assertions.**

```
#include <iostream>
#include <cassert>
using namespace std;

template <typename ELT>
class SinglyLinkedList {
    private:
        SNode* _head;  // same as SinglyLinkedNode on the github page
        int length;
    public:
        // … other code here …
        void print_reverse() {




        }
```

```
        void concatenate(SinglyLinkedList<ELT>& other_list) {




        }
        ELT get(int i) {




        }
    // … other code …
};
```

5. If the above code was written for a DoublyLinkedList class, would the above class need to be changed in any way? Explain.

6. Without referring class GitHub code, implement the following DoublyLinkedNode and DoublyLinkedList class:

```
template<typename ELT>
class DoublyLinkedNode {
    DoublyLinkedNode(DoublyLinkedNode<ELT>* prev,
                     ELT element, DoublyLinkedNode<ELT>* next) {



    }
    DoublyLinkedNode<ELT>* prev() { return _prev; }
    ELT element() { return _element; }
    DoublyLinkedNode<ELT>* next() { return _next; }
    void set_prev(DoublyLinkedNode<ELT>* prev) { _prev = prev; }
    void set_element(ELT element) { _element = element; }
    void set_next(DoublyLinkedNode<ELT>* next) { _next = next; }
};
```

```
// … other code …
template <typename ELT>
class DoublyLinkedList {
        private:
                DoublyLinkedNode<ELT> *_header, *_trailer;
                int _length;
        public:
                DoublyLinkedList() {


                }
                ~DoublyLinkedList() {


                }
                int length() { return _length; }
                bool is_empty() {


                 }
                void add_back(ELT e) {




                 }
                void remove_front() {




                 }
                void remove_back() {


                }
                void clear() {


                }
};
```