1. Create a linked-list that allows:
    1. an add function that takes a value and inserts it into a given position into the list (example: myList.add(someValue, somePosition))

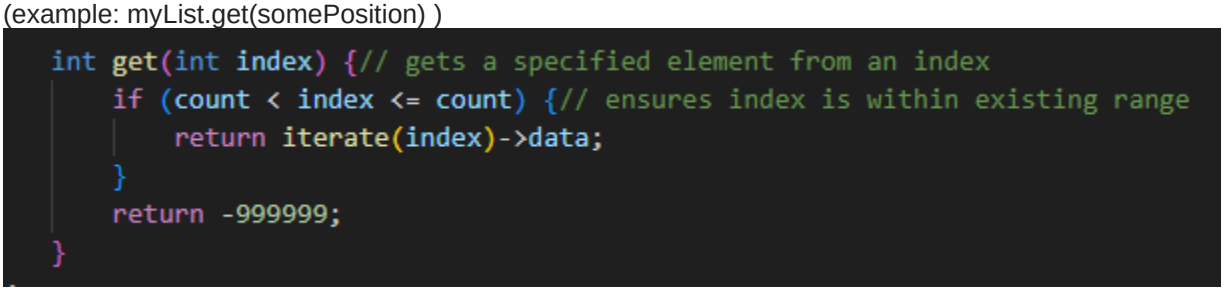
```
void add(int value, int index) {// adds an element to any point of the linked list
    if (index == 0) {// executes code to insert element at the front of the list if index is zero
        add0(value);
        return;
    }
    // sets current and previous to the proper values using iterate function
    node* curr = iterate(index-1);
    node* prev = curr;
    curr = curr->next;
    node* insert = new node();// creates the new node to be inserted
    insert->data = value;
    //resets pointers to the proper elements
    prev->next = insert;
    prev->next->next = curr;
    count += 1;
}
```

    2. a remove function that takes a position and removes the value stored at that position of the list and returns it

```
int remove(int index) { // removes an element from a specified index and returns the value
    node* curr = front;
    node* prev = front;
    if (index < count) { // ensures index is within range of list
        if (index == 0) { //executes case for if index is 0
            front = front->next;
            curr->next = nullptr;
            if (count == 0) {// removes the first element
                free(front);
                count -= 1;
            }
        }
        while (index > 0) { // iterates throuth the list until reaches desired index
            prev = curr;
            curr = curr->next;
            index -= 1;
        }
        // removes the element from the list by moveing pointers
        count -= 1;
        prev->next = curr->next;
        curr->next = nullptr;
        return curr->data;
    }
    return -999999;
}
```

    (example: myList.remove(somePosition))

3. a get function that takes a position and returns that value without removing it (example: myList.get(somePosition))



```
int get(int index) {// gets a specified element from an index
    if (count < index <= count) {// ensures index is within existing range
        return iterate(index)->data;
    }
    return -999999;
}
```

2. Be sure to include at least one test function for each piece of functionality that should verify that your code is working! This should be at least one test per behavior, likely more. You can make these tests in a source file with a main where your tests are either directly in the main or inside their own standalone functions (please do not neglect the importance of testing!)

```
// cout << "remove(0): " << myList.remove(0) << endl;
cout << "remove(1): " << myList.remove(1) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(2): " << myList.remove(2) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(3): " << myList.remove(3) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(0): " << myList.remove(0) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(3): " << myList.remove(3) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(0): " << myList.remove(0) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "remove(0): " << myList.remove(0) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;
```

```
cout << "remove(0): " << myList.remove(0) << endl;
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "iterate function: " << myList.get(3) << endl;

myList.add0(42);
myList.add0(41);
myList.add0(40);
myList.add0(39);
myList.print();
cout << "list length: " << myList.list_len() << endl;

cout << "iterate function: " << myList.get(3) << endl;

myList.add(99, 2);
myList.add(20,0);
myList.print();
cout << "list length: " << myList.list_len() << endl;


cout << "end program" << endl;
return 0;
```