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))

```
22
23      // Function to add a value
24      void add(int value, int position) {
25          Node* newNode = new Node(value);
26          if (position == 0) {
27              newNode->next = head;
28              head = newNode;
29          } else {
30              Node* current = head;
31              for (int i = 0; i < position - 1 && current != nullptr; i++) {
32                  current = current->next;
33              }
34              if (current != nullptr) {
35                  newNode->next = current->next;
36                  current->next = newNode;
37              } else {
38                  std::cerr << "Invalid position!" << std::endl;
39              }
40          }
41      }
```

2. a remove function that takes a position and removes the value stored at that position of the list and returns it
(example: myList.remove(somePosition))

```
43    // Function to remove a value at any position and return it
44    int remove(int position) {
45        if (head == nullptr) {
46            std::cerr << "List is empty!" << std::endl;
47            return -1;
48        }
49        if (position == 0) {
50            Node* temp = head;
51            int removedValue = temp->data;
52            head = head->next;
53            delete temp;
54            return removedValue;
55        } else {
56            Node* current = head;
57            for (int i = 0; i < position - 1 && current != nullptr; i++) {
58                current = current->next;
59            }
60            if (current != nullptr && current->next != nullptr) {
61                Node* temp = current->next;
62                int removedValue = temp->data;
63                current->next = temp->next;
64                delete temp;
65                return removedValue;
66            } else {
67                std::cerr << "Invalid position!" << std::endl;
68                return -1;
69            }
70        }
71    }
```

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

```
73    // Function to get the value at a given position without removing it
74    int get(int position) {
75        Node* current = head;
76        for (int i = 0; i < position && current != nullptr; i++) {
77            current = current->next;
78        }
79        if (current != nullptr) {
80            return current->data;
81        } else {
82            std::cerr << "Invalid position!" << std::endl;
83            return -1;
84        }
85    }
```

Testing:

Add function:

```
98   // Test function for adding elements to the list
99   void testAddFunction() {
100      LinkedList myList;
101
102      myList.add(1, 0);
103      myList.add(2, 1);
104      myList.add(3, 1);
105
106      // Verify the list after adding elements
107      std::cout << "List after adding elements: ";
108      myList.printList();
109  }
```

Remove function:

```
110
111  // Test function for removing elements from the list
112  void testRemoveFunction() {
113      LinkedList myList;
114
115      myList.add(1, 0);
116      myList.add(2, 1);
117      myList.add(3, 1);
118
119      int removedValue = myList.remove(1); // Remove element at position 1
120      std::cout << "Removed value: " << removedValue << std::endl;
121      std::cout << "List after removing an element: ";
122      myList.printList();
123  }
124
```

Get function:

```
125  // Test function for getting elements from the list
126  void testGetFunction() {
127      LinkedList myList;
128
129      myList.add(1, 0);
130      myList.add(2, 1);
131      myList.add(3, 1);
132
133      int valueAtIndex2 = myList.get(2);
134      std::cout << "Value at index 2: " << valueAtIndex2 << std::endl;
135  }
```