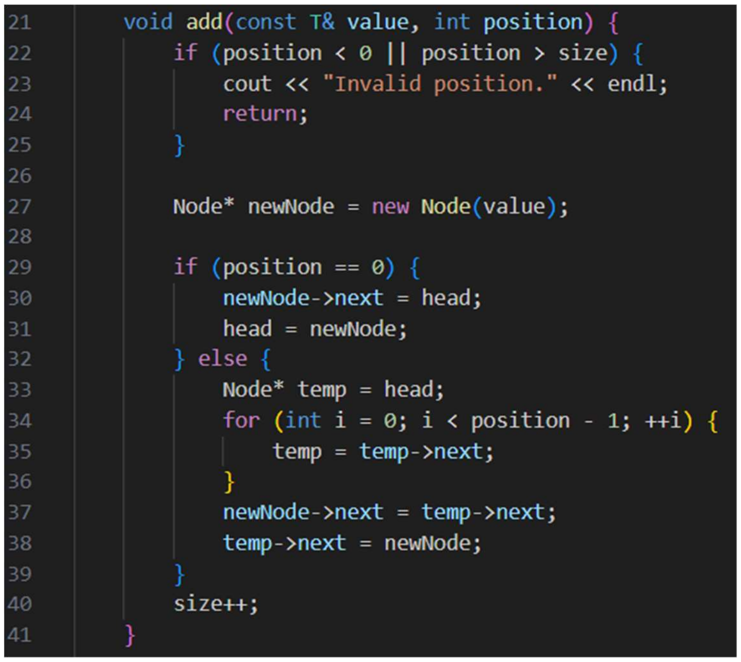
Leland Wendel

CS-260

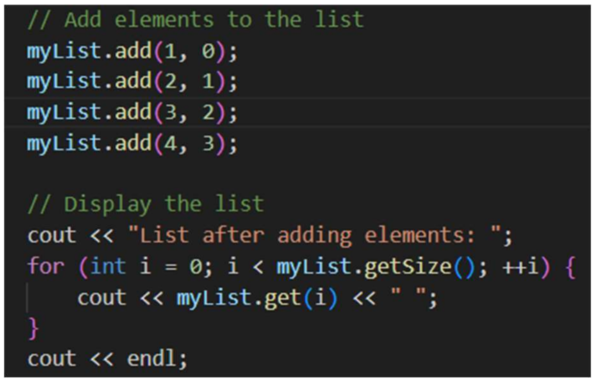
Assignment 4

1. **An add function that takes a value and inserts it into a given position into the list.**



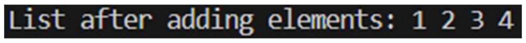
```
21      void add(const T& value, int position) {
22          if (position < 0 || position > size) {
23              cout << "Invalid position." << endl;
24              return;
25          }
26
27          Node* newNode = new Node(value);
28
29          if (position == 0) {
30              newNode->next = head;
31              head = newNode;
32          } else {
33              Node* temp = head;
34              for (int i = 0; i < position - 1; ++i) {
35                  temp = temp->next;
36              }
37              newNode->next = temp->next;
38              temp->next = newNode;
39          }
40          size++;
41      }
```

The add function takes two arguments - the value to be added and the position (int) in the list where that value will be added. The block of code from lines 22-25 checks if the inputted position exists, tossing the value out and displaying an error message if it doesn't. The block of code from lines 29-39 adds the new node to the input position in the list. If the new position is zero, the node becomes the head of the list. Otherwise, the list is iterated over to find the node just before the input position and the pointers are updated to insert the new node. The code at line 40 increments the list size after adding the new node.



```
// Add elements to the list
myList.add(1, 0);
myList.add(2, 1);
myList.add(3, 2);
myList.add(4, 3);

// Display the list
cout << "List after adding elements: ";
for (int i = 0; i < myList.getSize(); ++i) {
    cout << myList.get(i) << " ";
}
cout << endl;
```

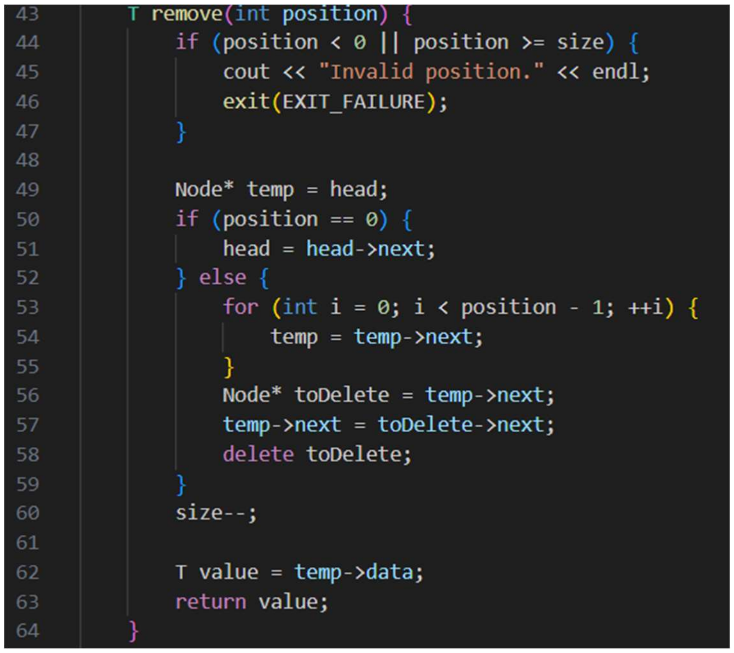
- Main test



```
List after adding elements: 1 2 3 4
```

- Output

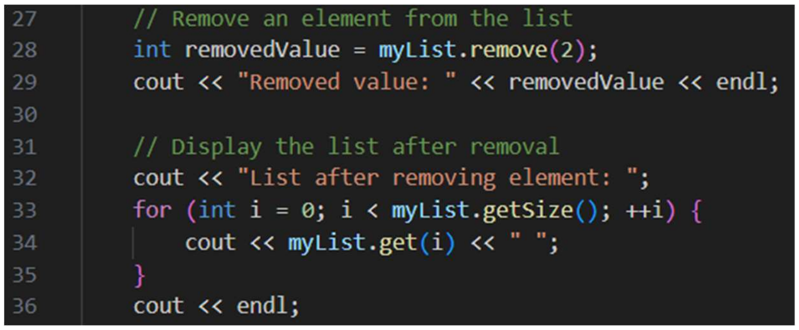
2. **A remove function that takes a position and removes the value stored in that position of the list and returns it.**



```
43      T remove(int position) {
44          if (position < 0 || position >= size) {
45              cout << "Invalid position." << endl;
46              exit(EXIT_FAILURE);
47          }
48  
49          Node* temp = head;
50          if (position == 0) {
51              head = head->next;
52          } else {
53              for (int i = 0; i < position - 1; ++i) {
54                  temp = temp->next;
55              }
56              Node* toDelete = temp->next;
57              temp->next = toDelete->next;
58              delete toDelete;
59          }
60          size--;
61  
62          T value = temp->data;
63          return value;
64      }
```

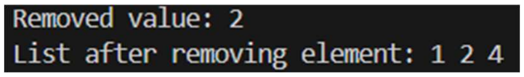
The remove function takes an integer position and returns a value type T, which is the value to be removed. The block of code from lines 44-47 checks if the input position is valid, displaying an error message and exiting the program if not. At line 49, a temporary pointer 'temp' is initialized at the head of the list – this pointer is used to locate the node just before the input position. The code at lines 50 & 51 updates the head pointer if the head is selected for removal. The block of code from lines 53-59 iterates over the list until 'temp' finds the node "position -1", and then updates the "next" pointer to "position – 1". This removes the input node from the list, stores the node to be deleted, and then deletes the node removed from the list. Line 60 subtracts the deletion and resizes the list and lines 62 & 63 return the value stored in the removed node if needed.

Main test:



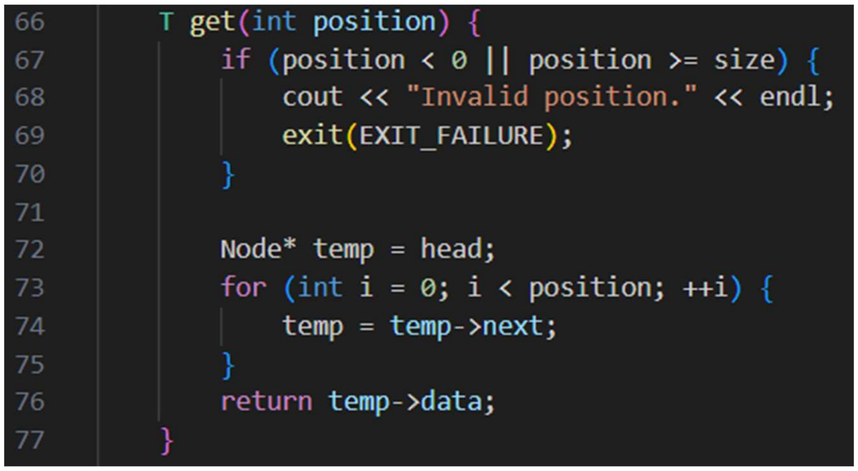
```
    // Remove an element from the list
    int removedValue = myList.remove(2);
    cout << "Removed value: " << removedValue << endl;

    // Display the list after removal
    cout << "List after removing element: ";
    for (int i = 0; i < myList.getSize(); ++i) {
        cout << myList.get(i) << " ";
    }
    cout << endl;
```

Output:



```
Removed value: 2
List after removing element: 1 2 4
```

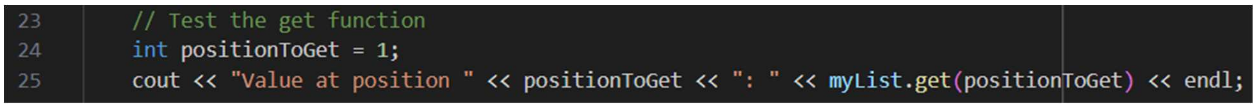
3. **A get function that takes a position and returns that value without removing it.**



```
66      T get(int position) {
67          if (position < 0 || position >= size) {
68              cout << "Invalid position." << endl;
69              exit(EXIT_FAILURE);
70          }
71  
72          Node* temp = head;
73          for (int i = 0; i < position; ++i) {
74              temp = temp->next;
75          }
76          return temp->data;
77      }
```

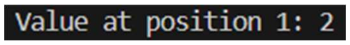
The get function takes an integer position as an argument and returns a value of type T, which is the value stored in that position. The block of code from lines 67-70 checks if the input position is valid and displays an error message and exits the program if not. At line 72, a temporary pointer 'temp' is initialized to the head of the list. The 'temp' pointer will iterate over the list to locate the node at the input position. The block of code from lines 73-76 loops over the list until 'temp' locates the node at the input position, moving "int position" times to move 'temp' to the proper node. Then, it returns the value stored in the node that 'temp' is pointing to.

Main test:



```
    // Test the get function
    int positionToGet = 1;
    cout << "Value at position " << positionToGet << ": " << myList.get(positionToGet) << endl;
```

Output:



```
Value at position 1: 2
```