CS 225 Spring 2019 :: TA Lecture Notes 2/6 LIST Impl

By Wenjie

- **Insert at front**
  - First of all, create a new node with the input data. Then we set new node's next node to the current head. Finally set head to new node

```
14  template <typename T>
15  void List::insertAtFront(T & t, unsigned index) {
16       ListNode * node = new ListNode(t);
17       node->next = head_;
18       head_ = node;
19  }
```

- **Find an element**
  - This will return a reference to a ListNode pointer, so then nothing new created - below is the version of recursive Solution:

List.hpp

```
1   // Recursive Solution:
2   template <typename T>
3   typename List<T>::ListNode *& List<T>::_index(unsigned index) {
4   //return a reference to a ListNode pointer
5   //function name is "_index" (helper function)
6     return _index_helper(index, _head);
7   }
8
9   ListNode *& _index_helper(unsigned index, ListNode *& node) {
10    if (index == 0) {
11      return node;
12    } else {
13      return _index(index - 1, node -> next);
14  }
15  // shift one in each recursive call
```

CS 225 Spring 2019 :: TA Lecture Notes 2/6 LIST Impl

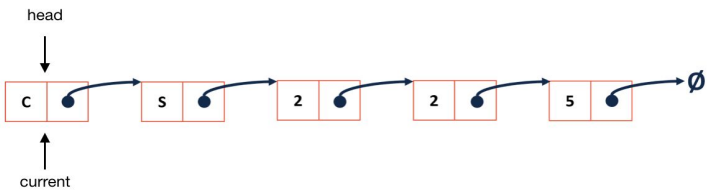
By Wenjie

- **Iterative Solution**

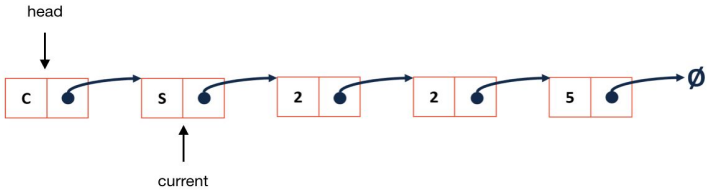
List.hpp

```
1   // Iterative Solution:
2   template <typename T>
3   typename List<T>::ListNode *& List<T>::_index(unsigned index) {
4   //return a reference to a ListNode pointer
5   //function name is "_index" (helper function)
6     if (index == 0) {
7       return head;
8     } else {
9       ListNode *thru = head;
10      for (unsigned i = 0; i < index - 1; i++) {
11        thru = thru->next;
12      }
13      return thru->next;
14    }
15  }
```

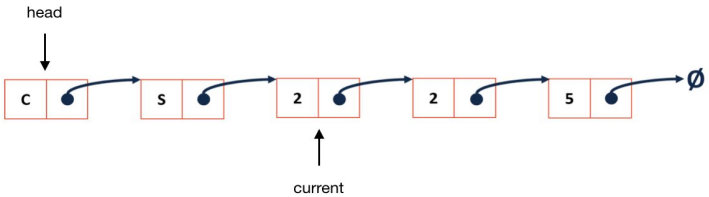
1. Initialization



2. First iteration, moved the current by one



3. Another iteration, moved current by another one



CS 225 Spring 2019 :: TA Lecture Notes 2/6 LIST Impl

By Wenjie

- Write a boundary check so that we do not evaluate NULL pointers at the end of the list in most of the cases

- **Operator [ ]**
  - Implement a “array-like” access by index operator. For example, if we have List<int> list; then we can do list[3]

List.hpp

```
1  template <typename T>
2  T & List<T>::operator[](unsigned index) {
3    ListNode *& d = _index(index);
4    return d -> data;
5  }
```

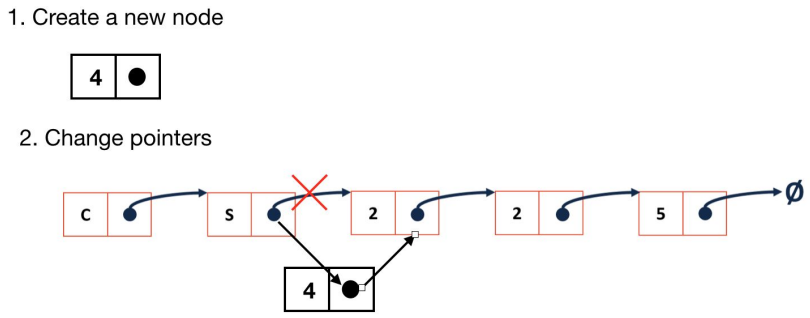
- **Insert**
  - We should make sure to insert before an indexed node. Since if we use pass by reference, we can treat the node as head, and then do the exact same as in we do in inserting at front.

List.hpp

```
1  template <typename T>
2  void List<T>::insert(const T & t, unsigned index) {
3    ListNode *& node = _index(index);
4    ListNode * newNode = new ListNode(t);
5    newNode -> next = node;
6    node = newNode;
7  }
```

CS 225 Spring 2019 :: TA Lecture Notes
2/6 LIST Impl

By Wenjie



- **We need to use the reference of "node"**, since we need to change its pointing location to new node. Otherwise we will create a new copy of a node pointer, which does not change the original list structure

- **Remove**
  - Let node point to the node -> next
    - Something like: `node = node -> next;`
    - Also, be careful about leaking memory! The most common reason of causing that is it lost access to the original node without deleting properly.
  - Therefore we take care of memory problem by:
    - Use a temp to copy the original node address
    - Delete that original node

List.hpp

```
1  template <typename T>
2  T & List<T>::remove(unsigned index) {
3    ListNode *& node = _index(index);
4    ListNode * temp  = node;
5    T & data = node -> data;
6    node = node -> next;
7    delete temp;
8    return data;
   }
```