CS 225 Spring 2019 :: TA Lecture Notes
2/6 Stack & Queue

By Wenjie

- **Different Implementations of List**

  Arrays are sequential blocks of memory where one element follows immediately after the other.

c	s	2	2	5
index: [0]	[1]	[2]	[3]	[4]

  As we for the linked list, we have to implement and analyze ADT functionality like insert, remove, get, ect.

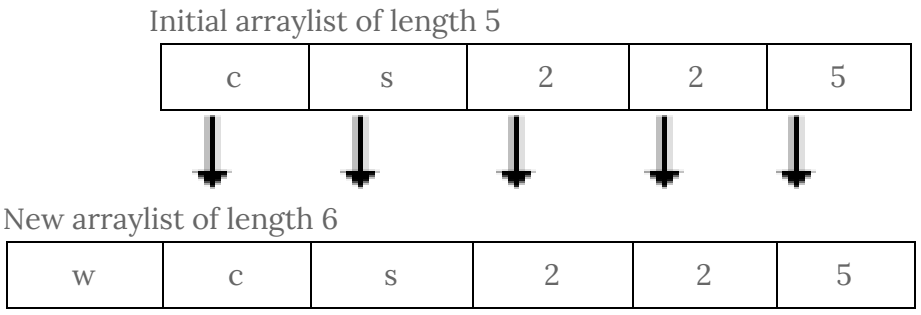
  - **ArrayList**
    - Same abstract datatype
    - Store everything in an array

```
1   #pragma once
2   template <typename T>
3   class List {
4     public:
...     /* ... */
14    private:
15      T * arr_;       // the content array
16
17      int capacity;   // the maximum size possible;
18                         the allocated array size
19
20      int count;      // the size in use; the number
21                         of current elements
22  };
```

CS 225 Spring 2019 :: TA Lecture Notes 2/6 Stack & Queue

By Wenjie

- How would we insert an element to the back of the list? Place the element at the index right after the last element and increment the index. However we may reach the full capacity and in that case we need to expand our array. We will explore two strategies for expanding arrays.
    - Strategy 1: every time we run out of space, create a new array of length (old_lenght + 1) and copy over elements plus the new element. Every time we add an element, we have to repeat the above process. This means that we have to copy n elements each time we insert which takes O(n) time per insert. If we try increasing size to (old_lenght + k), where k is some arbitrary integer, we would still be taking O(n) time for each insert. This is a very inefficient strategy.



    - Strategy 2: every time we run out of space, create a new array that is twice as big as the initial array and copy over elements plus the new element. In this case we will not be creating a new array and copying over every time we insert, in fact we will have running time of O(n) for n insertions which means that the running time per insertion is O(1)! With this strategy, most of the insertions will take O(1) and only occasionally we will have an insertions that takes O(n). This is efficient! The reason we double is the time vs. space trade off. When we are doubling, our array is always half full and we are not wasting much space.
- What happens if we want to add in front? Then We need to move every element one spot ahead and then insert the new element to the front. In other words, we need to copy n element. Therefore, the running time is O(n) again.

CS 225 Spring 2019 :: TA Lecture Notes
2/6 Stack & Queue

By Wenjie

- Resizing strategy: a trade-off
  - An experiment: initial capacity = 2, insert n elements
  - small increase (+2) each time:
    - Total time complexity: 2+4+6+...+(2n) = O(n^2)
    - Each insertion: **O(n)** amortized
  - double the size each time:
    - Total time complexity: 2+4+8+...+log n = O(n)
    - Each insertion: **O(1)** amortized
    - Not wasting too much space and a better runtime

- **Linked List vs Array List**

	Singly Linked List	**ArrayList**
Insert/Remove at Front	O(1)	O(1) (amortized)
Insert at a given element	O(1)	O(n) needs to copy everything after the element
Remove a given element	O(1)	O(n)
Insert an arbitrary element	O(n)	O(n)
Remove at arbitrary location	O(n)	O(n)

- **std::vector**
  - The c++ implementation of a List encapsulating an array
  - So insert will take O(n) worst case
  - “push_back” takes O(1) since it adds to the end of the array

CS 225 Spring 2019 :: TA Lecture Notes
2/6 Stack & Queue

By Wenjie

- **Stack ADT**
  - We can think of stack as a pile of papers on the desk or a stack of plates. We always remove from the top of the stack (imagine trying to remove the bottom plate from the stack of plates, we are very likely to break them all). Stack is said to be a “last in - first out” (LIFO) data structure.
  - Operations
    - Push data
    - Pop data
    - Create a new stack
    - Check empty

- **Queue ADT**
  - We can think of queue as a line in a store, the first person in line gets served first (if we serve the last person in line before the others, we will get a lot of angry customers). Queue is said to be a “first in -first out” (FIFO) data structure.
  - Operations
    - Enqueue
    - Dequeue
    - Create new
    - Check empty