Admin

- Today's topics
  - Finish Editor Buffer case study
  - Start Map implementation, trees
- Reading
  - Ch 9
  - Ch 13
- Café today after class

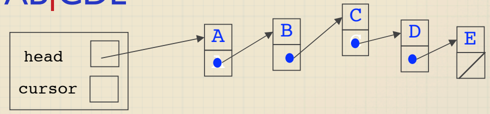
Lecture #21

Buffer: Vector vs Stack

	Vector	Stack
Buffer()	O(1)	O(1)
~Buffer()	O(1)	O(1)
moveCursorForward()	O(1)	O(1)
moveCursorBackward()	O(1)	O(1)
moveCursorToStart()	O(1)	O(N)
moveCursorToEnd()	O(1)	O(N)
insertCharacter()	O(N)	O(1)
deleteCharacter()	O(N)	O(1)
Space used	1N	2N

Buffer as linked list

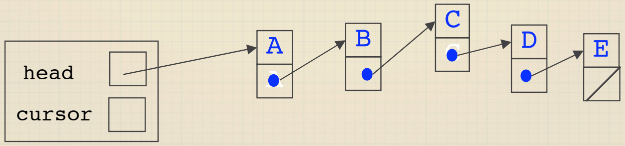
- Inspiration: contiguous memory is constraining
  - Connect chars without locality
  - Linked list to the rescue!
- Buffer contains: AB|CDE



```
// for Buffer class
private:
   struct cellT {
      char ch;
      cellT *next;
   };
   cellT *head, *cursor;
```

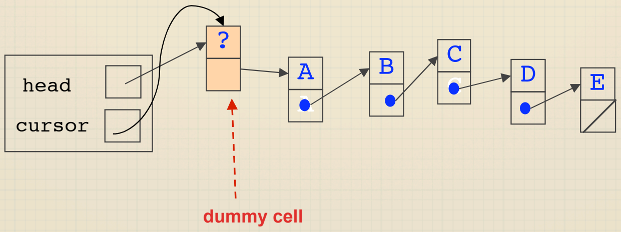
Cursor design

- Where does cursor point?
  - To cell *before* or *after*?
  - Where is cursor if between B & C?
  - After E?
  - Before A?
  - 5 letters, 6 cursor positions…



Use of dummy cell for linked list

- Add "dummy cell" to front of list
  - Simplifies logic
  - Every cell holding actual data has a predecessor
  - Cursor can point to cell before insertion point



Linked list insert/delete

```
void Buffer::insertCharacter(char ch)
{
     cellT *cp = new cellT;
     cp->ch = ch;
     cp->next = cursor->next;
     cursor->next = cp;
     cursor = cp;
}

void Buffer::deleteCharacter()
{
   if (cursor->next != NULL)  {
      cellT *old = cursor->next;
      cursor->next = old->next;
      delete old;
   }
}
```

Linked list cursor movement

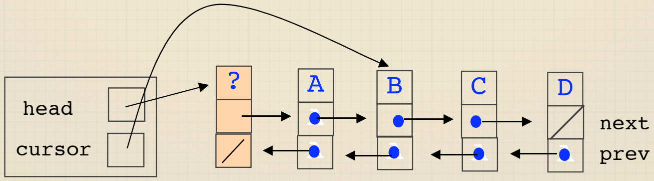
```
void Buffer::moveCursorToBegin() {
    cursor = head;
}

void Buffer::moveCursorForward() {
    if (cursor->next != NULL)
        cursor = cursor->next;
}

void Buffer::moveCursorToEnd() {
    while (cursor->next != NULL)
       moveCursorForward();
}

void Buffer::moveCursorBackward() {
   if (cursor != head) {
     cellT *cp = head;
     while (cp->next != cursor)
          cp = cp->next;
     cursor = cp;
  }
```

Compare implementations

	Vector	Stack	List
Buffer()	O(1)	O(1)	O(1)
~Buffer()	O(1)	O(1)	O(N)
moveCursorForward()	O(1)	O(1)	O(1)
moveCursorBackward()	O(1)	O(1)	O(N)
moveCursorToStart()	O(1)	O(N)	O(1)
moveCursorToEnd()	O(1)	O(N)	O(N)
insertCharacter()	O(N)	O(1)	O(1)
deleteCharacter()	O(N)	O(1)	O(1)
Space used	1N	2N	5N

Fancier linked list

- Add tail pointer to get direct access to last cell
- How to speed up moving backwards?
  - Add prev link, symmetric with next link

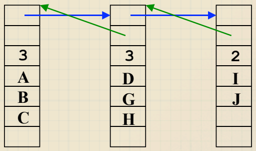


Compare implementations

	Vector	Stack	Single	Double
Buffer()	O(1)	O(1)	O(1)	O(1)
~Buffer()	O(1)	O(1)	O(N)	O(N)
moveCursorForward()	O(1)	O(1)	O(1)	O(1)
moveCursorBackward()	O(1)	O(1)	O(N)	O(1)
moveCursorToStart()	O(1)	O(N)	O(1)	O(1)
moveCursorToEnd()	O(1)	O(N)	O(N)	O(1)
insertCharacter()	O(N)	O(1)	O(1)	O(1)
deleteCharacter()	O(N)	O(1)	O(1)	O(1)
Space used	1N	2N	5N	9N

Space-time tradeoff

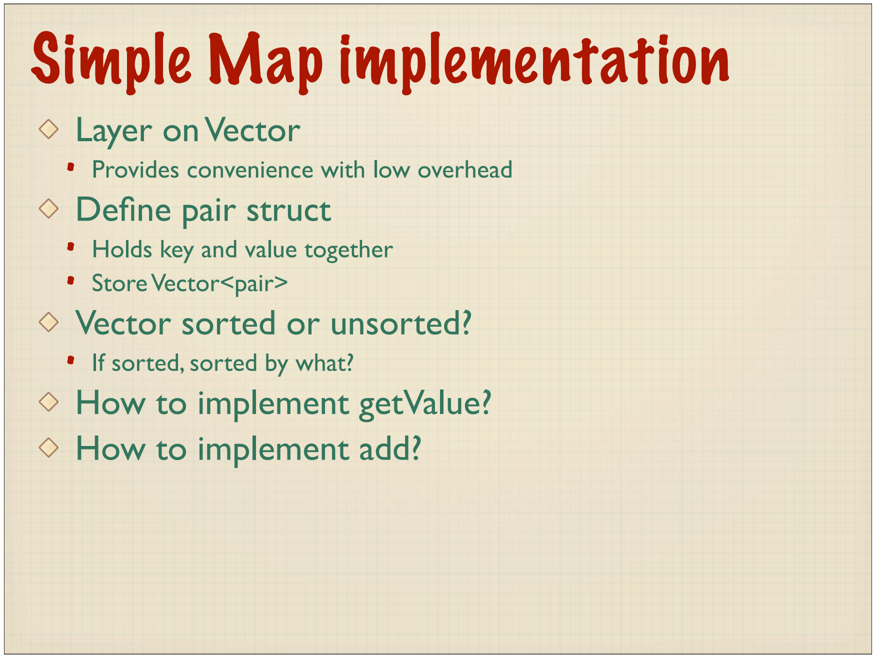
- Doubly-linked list is O(1) on all six operations
  - But, each char uses 1 byte + 8 bytes of pointers => 89% overhead!
- Compromise: chunklist
  - Array and linked list hybrid
  - Shares overhead cost among several chars
  - Chunksize can be tuned as appropriate



- Cost shows up in code complexity
  - (as you will discover on pqueue assignment)
  - Cursor must traverse both within and across chunks
  - Splitting/merging chunks on insert/deletes

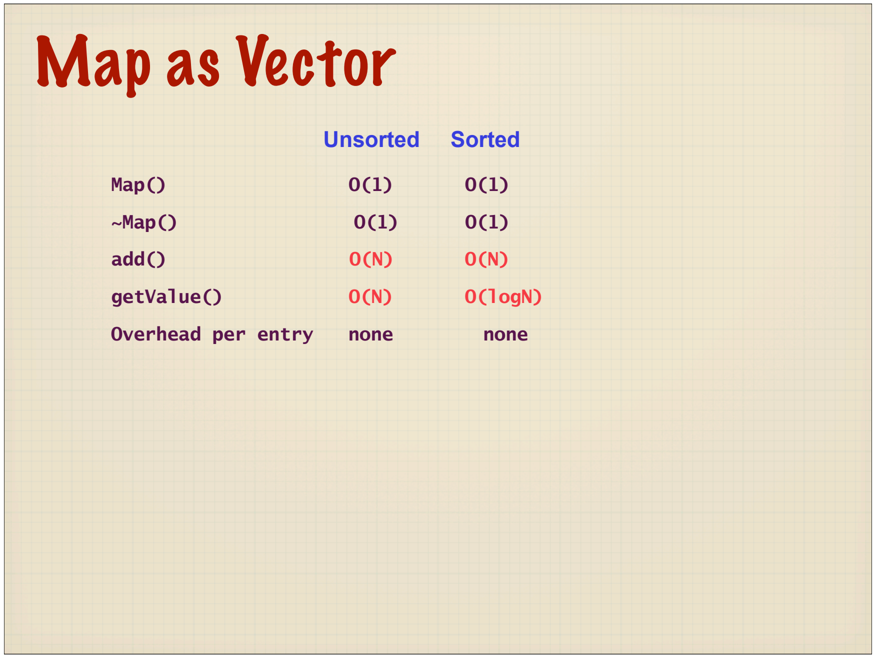
Implementing Map

- Map is super-useful
  - Any kind of dictionary, lookup table, index, database, etc.
- Stores key-value pairs
  - Fast access via key
  - Operations to optimize: add, getValue
- How to make work efficiently?



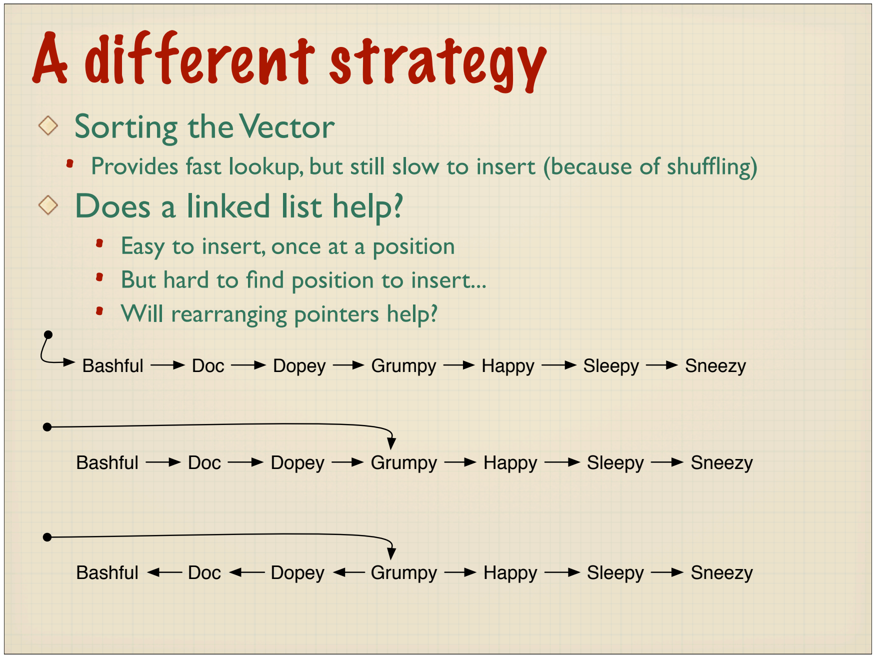
Simple Map implementation

- Layer on Vector
  - Provides convenience with low overhead
- Define pair struct
  - Holds key and value together
  - Store Vector<pair>
- Vector sorted or unsorted?
  - If sorted, sorted by what?
- How to implement getValue?
- How to implement add?



Map as Vector

	Unsorted	Sorted
`Map()`	O(1)	O(1)
`~Map()`	O(1)	O(1)
`add()`	O(N)	O(N)
`getValue()`	O(N)	O(logN)
`Overhead per entry`	none	none



A different strategy

- Sorting the Vector
  - Provides fast lookup, but still slow to insert (because of shuffling)
- Does a linked list help?
  - Easy to insert, once at a position
  - But hard to find position to insert...
  - Will rearranging pointers help?

Bashful → Doc → Dopey → Grumpy → Happy → Sleepy → Sneezy

Bashful → Doc → Dopey → Grumpy → Happy → Sleepy → Sneezy

Bashful ← Doc ← Dopey ← Grumpy → Happy → Sleepy → Sneezy