UCLA Samueli Computer Science

CS32: Introduction to Computer Science II

Discussion Week 3

Junheng Hao, Arabelle Siahaan

Apr. 19, 2019

Announcements

UCLA Samueli Computer Science

- Project 2 is due on Tuesday, April 23.
- Midterm 1 is scheduled on Thursday, April 25.

Outline Today

UCLA Samueli Computer Science

- Data structures
- Linked List
- Stack
- Queue
- Project 2: Guide

Data Structure

UCLA Samueli Computer Science

Before we talk about doing things arrays, let's talk about data structure first!

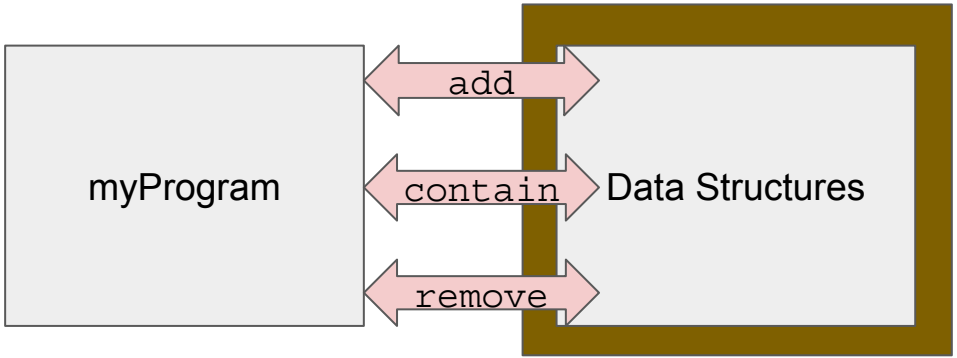
CS32 is not just to write code and run. It is more about organizing data. We call an organization scheme a data structure. For every structure, we define:

- How to store/organize the data items?
- Method to add new data / remove data
- Apply functions on data: most importantly, how to search data?

We also need to know pros and cons of each data structure as well as its efficiency or complexity of algorithms with these data structures.

Arrays (as an example)

- Most basic data structure and most important!



For example, the **Sequence** *class you have done in homework 1!*

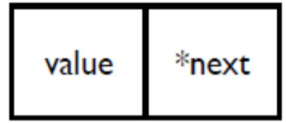
- Add / Contain / Remove → Complexity?
- More implementations: getIndexOf, getFrequencyOf, …
- More questions: Using array to implement Linked List? How to sort a array or maintain a sorted array?

Linked List: Review

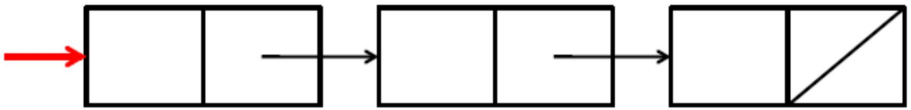
Basis

UCLA Samueli Computer Science

- Minimum Requirement
  - Key component as unit: Node (with `value` and `pointer` to next node)
  - Head pointer → points to the first term
  - Loop-free (except in some special case: circular listed list)
- Regular operations
  - Insertion
  - Search
  - Removal
- Pros and cons
  - Efficient insertion, flexible memory allocation, simple implementation
  - High complexity of search




```
typedef int ItemType;
Struct Node
{
      ItemType value;
      Node *next;
};
```

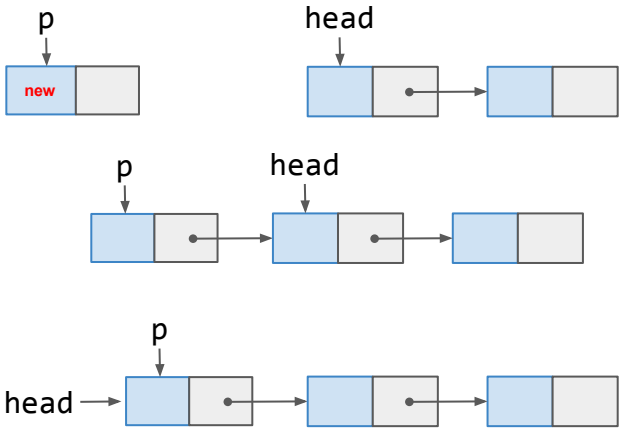


Linked List

Insertion: Add a new node to a list

UCLA Samueli Computer Science

- Example: Insert as head in a list
- Steps
  a) Create a new node and call the pointer p
  b) Make its `next` pointer point to the first item
  c) Make the head pointer to the new node



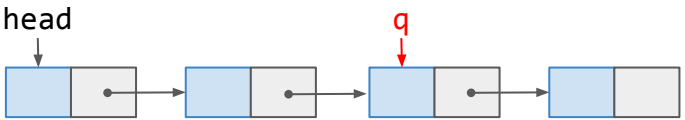

```
//Skeleton: Linked list insertion
//====================================

//insert as head
p->next = head;
Head = p;

//insert after end: End node: q
q->next = p;
p->next = nullptr;

//insert in the middle: node q
p->next = q->next;
q->next = p;
```

Linked List

Search

UCLA Samueli Computer Science

- Steps
  a) Find matched node and return
  b) If no match, return NULL




```
// Skeleton Code: Linked list search
// ==================================


Node* Search(int key, Node* head){
  Node *q = head;
  while(q != NULL)
  {
    if(q -> value != key) q = q -> next;
    else return q;
  }
  return NULL;
}
```

Linked List

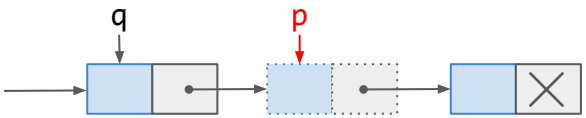
Removal

UCLA Samueli Computer Science

- Remember to set the previous node q's `next` pointer to point the next node of p

```
q->next = p->next;
delete p
```

- What if `p == head`? What if `p` prints to the last node in the linked list?




```
// Skeleton Code: Linked list removal
// ===================================

void remove(int valToRemove, Node* head) {
      Node *p = head, *q = NULL;
      while (p != NULL) {
            if (p->value == valToRemove)
                  break;
            q = p;
            p = p->next;}
      if (p == NULL) return;
      if (p == head) //special case
            head = p->next;
      else
            q->next = p->next;
      delete p;
}
```

Linked List

Conclusion

UCLA Samueli Computer Science

- Pros:
  - Efficient insertion (add new data items)
  - Flexible memory allocation
- Cons:
  - Slow search (search is more important than insertion and removal in real situations)

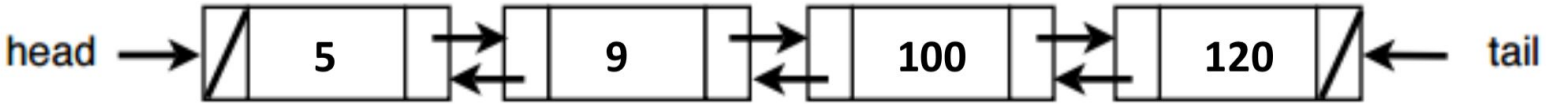
- Many variations
  - Doubly linked lists
  - Sorted linked lists
  - Circularly linked lists

Sorted Linked List

Motivation and properties

UCLA Samueli Computer Science

- Do we need to search the entire linked list?
- What if we store all values in an ascending sorted (or descending order)?

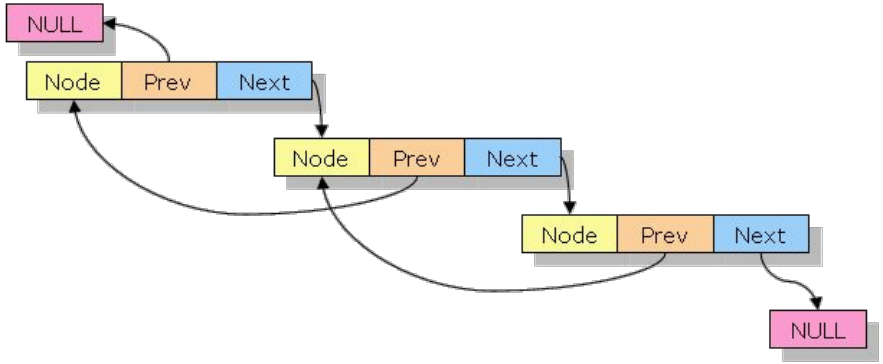


- How do you change insertion function? → Worksheet Q6
  - Find the node **q** whose value is the greatest lower bound to the new node **p**.
- How do you change search function?
  - Early stop when we see a node which stores a value that is larger than key for ascending sorted linked list.
- How do you update removal functions?

Double Linked List

Data structures and properties

UCLA Samueli Computer Science



- A linked list where each node has two pointers:
  - Next – pointing to the next node
  - Prev – pointing to the previous node
- Features
  - head, tail pointers
  - `head->prev = NULL; tail->next = NLL;`
  - `head == tail == NULL` when doubly linked list is empty

```
typedef int ItemType;
Struct Node
{
        ItemType value;
        Node *next;
        Node *prev;
};
```

Double Linked List

Insertion: How many cases to consider?

UCLA Samueli Computer Science

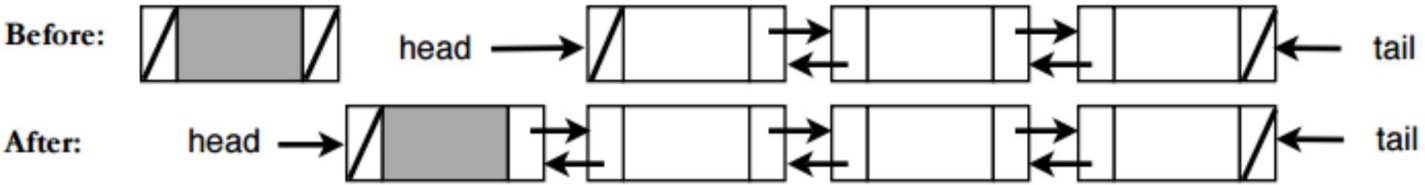
- Four cases:
  - Insert before the head
  - Insert after the tail
  - Insert somewhere in the middle
  - When list is empty ⬅

Double Linked List

Insertion: Before head / After tail

UCLA Samueli Computer Science

- Steps for insertion before head:
  - Set the `prev` of head to the new node `p`
  - Set the `next` of `p` to `head`
  - `p` becomes the new `head`
  - `head->prev = NULL;`
- Steps for insertion after tail:
  - Similar to insertion before head (try it yourself!)



Double Linked List

Insertion: In the middle of the list

UCLA Samueli Computer Science

- Steps for insertion in the middle (after node q):
  - Fix the next node of q first: `Node *r = q->next;`
  - Point both next of q and prev of r to p: `q->next = r->prev = p;`
  - Point both sides of p to q and r respectively: `p->prev = q; p->next = r;`

- You can do that without the help of pointer `r`

```
p->prev = q;
p->next = q->next;
q->next = q->next->prev = p;
```

Double Linked List

Insertion to empty list / Search

UCLA Samueli Computer Science

- Insertion to an empty list

```
head = tail = p;
p->next = p->prev = NULL;
```

- Search in doubly linked list
  - Similar to standard linked list
  - Can be done either from head or tail

Double Linked List

Removal

UCLA Samueli Computer Science

- Removal is more complex!
- Consider the following cases:
  - Check if the node p is the head (`p == head`). Let this boolean be A.
  - Check if the node is the tail (`p == tail`). Let this boolean be B.
- Different cases:
  - Case 1 (A, but not B): P is the head of the list and there is more than one node.
  - Case 2 (B, but not A): P is the tail of the list, and there is more than one node.
  - Case 3 (A and B): P is the only node.
  - Case 4 (not A and not B): P is in the middle of the list.

Double Linked List

Removal

UCLA Samueli Computer Science

```
void removeNodeInDLL(Node *p, Node& *head, Node& *tail)
{
      if (p == head && p == tail) //case 3
            head = tail = NULL;
      else if (p == head) {
            //case 1
            head = head -> next;
            head -> prev = NULL; }
      else if (p == tail) {
            //case 2
            tail = tail -> prev;
            tail -> next = NULL; }
      else {
            //case 4
            p -> prev -> next = p -> next;
            p -> next -> prev = p -> prev; }
      delete p;
}
```

Double Linked List

Copy a doubly linked list (and more)

UCLA Samueli Computer Science

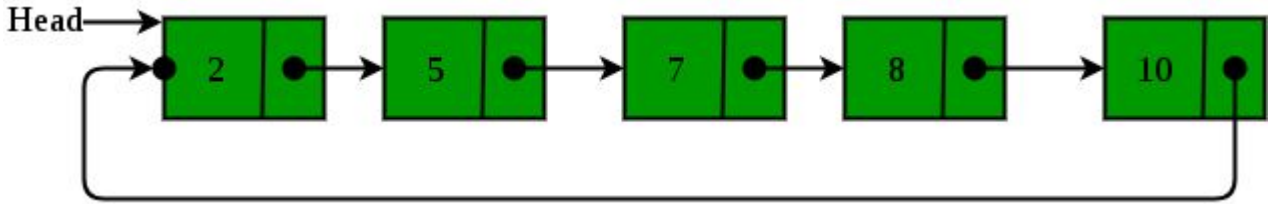
- Steps
  - Create head and tail for the new list
  - Iterate through the old list. For each node, copy its value to a new node.
  - Insert the new node to the tail of the new list.
  - Repeat until we have iterated the entire old list.
  - Set `NULL` before head and next of tail.

- Tips for linked list problems
  - To draw diagrams of nodes and pointers will be extremely helpful.
  - When copying a linked list, only copy stored values to new nodes. Do not copy pointers.
  - You need to check **edge cases**! (We'll talk about it later!)

Circular Linked List

Motivation and properties

UCLA Samueli Computer Science



- Linked list where all nodes are connected to form a circle.
  - There is no NULL at the end.
  - Can be a singly circular linked list or doubly circular linked list.
- Pros:
  - Any points can be head (starting point).
  - Implementation for queue or Fibonacci Heap.
  - Fit to repeatedly go around the list.
- It is also very tricky though.

Problem: Reverse Linked List

Leetcode questions #206

UCLA Samueli Computer Science

Question: How to reverse a (single) linked list?

Example:

```
Input: 1->2->3->4->5->NULL
Output: 5->4->3->2->1->NULL
```

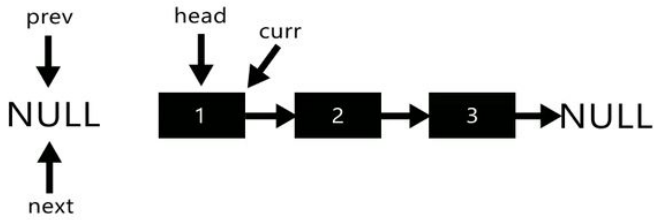
```
// One possible solution
Node* reverseList(struct ListNode* head)
{
     Node *prev=NULL,*current=head,*next;
     while(current) {
          next = current->next;
          current->next = prev;
          prev = current;
          current = next;
     }
     return prev;
}
```

Problem: Reverse Linked List

Leetcode questions #206

UCLA Samueli Computer Science

Let's see what happens in these lines of codes! [Link]

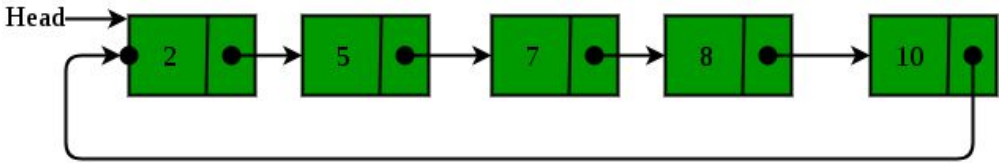


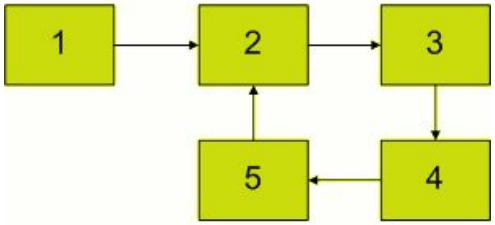

```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Circular vs Linked List with Loop

UCLA Samueli Computer Science

- Two different tasks:
  - Tell whether the linked list is circular
  - Tell whether there is a loop in the linked list (this is much harder!)





Suggestions on Linked List

- Drawing pictures and carefully tracing through your code, updating the picture with each statement, can help you find bugs in your code.
- Check any list operations for these:
  - In a typical, middle-of-the-list case?
  - At the beginning of the list?
  - At the end of the list?
  - For the empty list?
  - For a one-element list?
- Another validation technique is for every expression of the form **`p->something`**, prove that you can be sure **`p`** has a well-defined, non-null value at that point.

Hints for Project 2

UCLA Samueli Computer Science

Task: to implement a couple of algorithms that operate on sequences by a doubly-linked list rather than an array.

```
class Sequence{
  public:
    Sequence();
    bool empty() const;
    int size() const;
    int insert(int pos, const ItemType& value);
    int insert(const ItemType& value);
    bool erase(int pos);
    int remove(const ItemType& value);
    bool get(int pos, ItemType& value) const;
    bool set(int pos, const ItemType& value);
    int find(const ItemType& value) const;
    void swap(Sequence& other);
};
int subsequence(const Sequence& seq1, const Sequence&
seq2);
void interleave(const Sequence& seq1, const Sequence&
seq2, Sequence& result);
```

Note & Reminders:

1. Still use type alias for multiple data type.
2. Destructor, copy constructor and assignment operator are still needed.
3. Sequence algorithms (`subsequence`, `interleave`) are defined out of the `Sequence` class.
4. Do not make changes that are not allowed. Check the spec!
5. Report (doubly-linked list implementation, pseudocode for non-trivial algorithms, test cases)

Project 2: Design of Doubly-Linked Lists

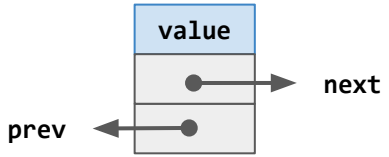
UCLA Samueli Computer Science

- How would you design your doubly-linked list?
- As required in the spec (report), show a typical `Sequence` and an empty `Sequence`. Is the list circular? Does it have a dummy node? What's in your list nodes (key/value and pointers)?
- What does a doubly-linked list with dummy node look like? And why do we design it? → To easily solve the “edge” case (or annoying border case)!
- Check the supplementary linked list note!

Project 2: Doubly-Linked List as Start

UCLA Samueli Computer Science

```
typedef int ItemType;
Struct Node
{
      ItemType value;
      Node *next;
      Node *prev;
};
```

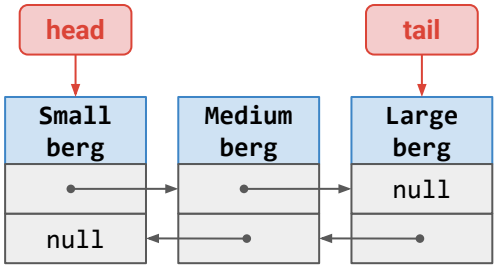


- How to remove the first item from a list?
- Wrong! Error when list is empty!
- Still flawed! Error when list has only one element!

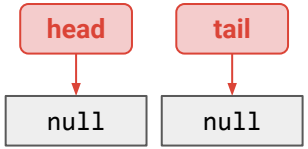
```
Node* oldHead = head;
head = head->m_next;
head->m_prev = nullptr;
delete oldHead;
```

```
if (head != nullptr) {
  Node* oldHead = head;
  head = head->m_next;
  head->m_prev = nullptr;
  delete oldHead;
}
```

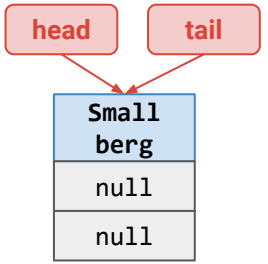
Version 1.0 DLL (3 elements)



Version 1.0 DLL (empty)



Version 1.0 DLL (1 element)

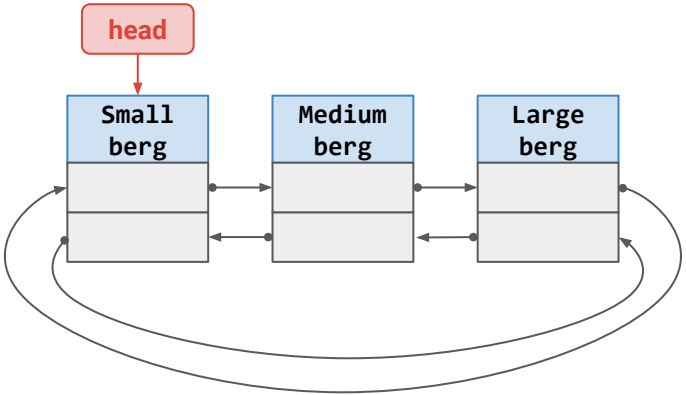


Project 2: Circular Doubly-Linked List

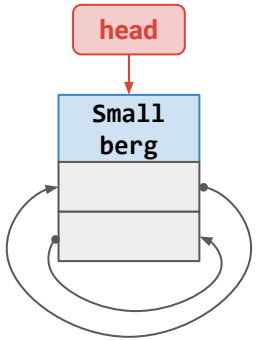
UCLA Samueli Computer Science

- The structure of linked lists you choose leads to many special cases and it is somewhat hard to deal with!
- Let's make a circular doubly-linked list! → Get rid of `nulls`! `Tail` is no longer needed!

Version 2.0 CDLL (3 elements)



Version 2.0 CDLL (1 element)

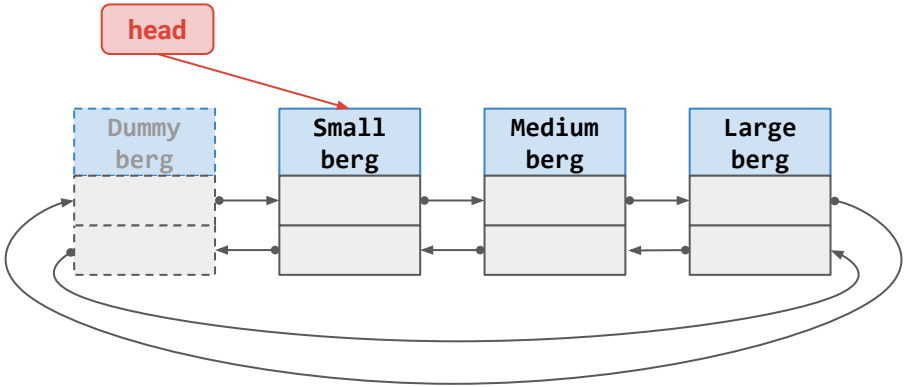


Project 2: Circular Doubly-Linked List

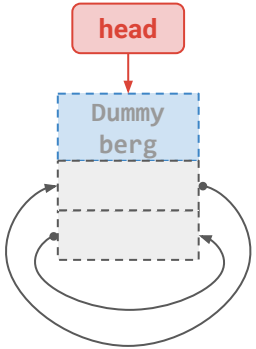
UCLA Samueli Computer Science

- You may still have problems with zero or one elements!
- Make a dummy node! → Just to make your linked list is always non-empty!
- Now it's time to think about all the functions you need to implement in `Sequence` class!

Version 3.0 CDLL (3 elements)



Version 3.0 CDLL (empty list)

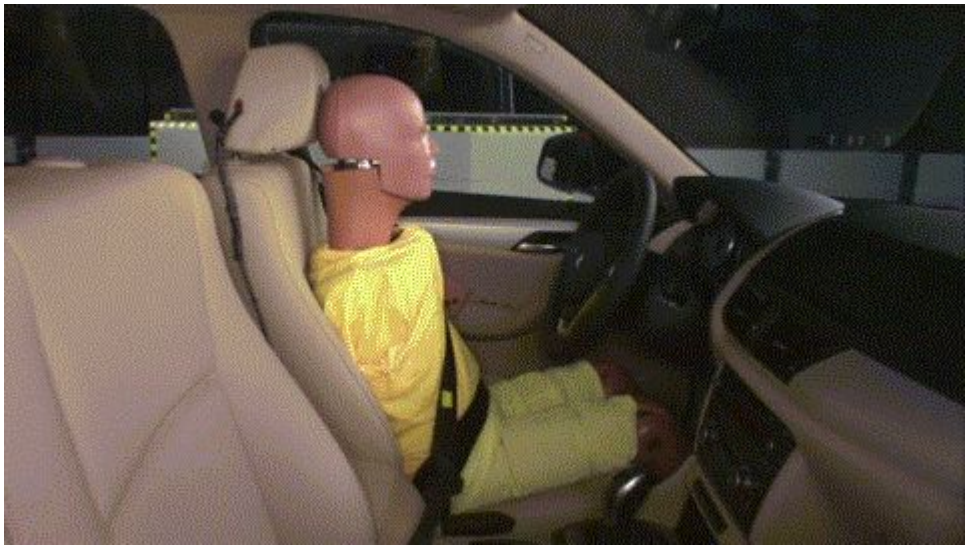


Project 2: Design test cases?

UCLA Samueli Computer Science

One final note when you are working on Homework 2: Do some USEFUL testing!

```
// Example: to test remove function

Sequence s;
s.insert(0, "a");
s.insert(1, "b");
s.insert(2, "c");
s.insert(3, "b");
s.insert(4, "e");
assert(s.remove("b") == 2);
assert(s.size() == 3);
string x;
assert(s.get(0, x)  &&  x == "a");
assert(s.get(1, x)  &&  x == "c");
assert(s.get(2, x)  &&  x == "e");
```



Make sure that you follow the requirements of "Turning it in"!

UCLA Samueli Computer Science

Break Time! (5 minutes)

Q & A

Group Exercises: Worksheet 1

UCLA Samueli Computer Science

- Exercise problems from **Worksheet 2** (see “LA worksheet” tab in CS32 website). Answers will be posted after all discussions.
- Questions for today: [*No. of questions*]

UCLA Samueli Computer Science

Thank you!

Q & A