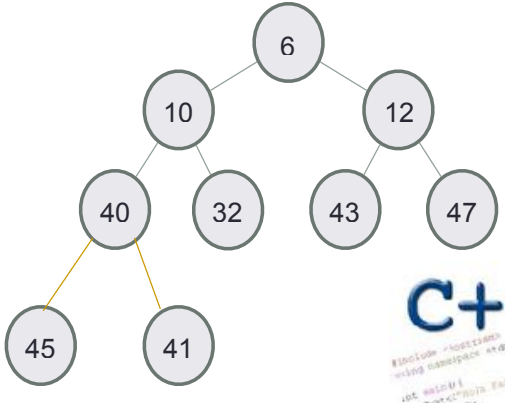
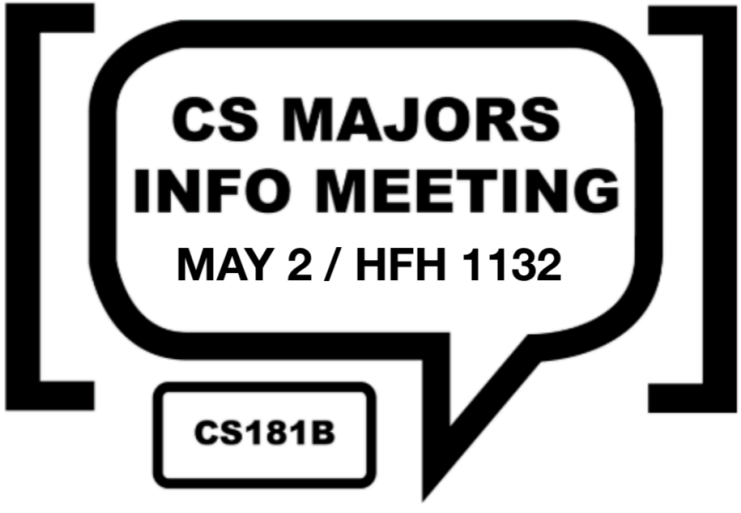
RECURSION

Problem Solving with Computers-II





?

CS185

CS165A/B

CS181B

CS178

3:30-5:00 PM
FREE FOOD

UNSURE ABOUT MAJOR-FIELD ELECTIVES?
COME LEARN ABOUT ELECTIVE CHOICES
THAT INTEREST YOU, AND FULFILL
YOUR GRADUATION ADVISING REQUIREMENT!

Let recursion draw you in….

- Many problems in Computer Science have a recursive structure…
- Identify the “recursive structure” in these pictures by describing them

Recursion as a tool for solving problems

Describe the problem in terms of a smaller version of itself!

[Recursive algorithms]

To wash the dishes in the sink:

Wash the dish on top of the stack] → Work done to reduce the problem size

If there are no more dishes] → Stopping condition

you are done!

Else:

Wash the *remaining* dishes in the sink] ← Recursive step

Compute the factorial of a number

n! is the number of different arrangements/permutations of n things

$n! = n * (n-1) * (n-2) * \ldots * 1, \quad \text{if } n >= 1$

$= 1, \quad \text{if } n = 0$

$\rightarrow (n-1)!$

```
int fact(int n) {
line1// if (n == 0 || n == 1) // Base case (smallest version of the problem)
line2//     return 1;
line3//  int result = fact(n-1);
line4//  return n * result;
   }
```

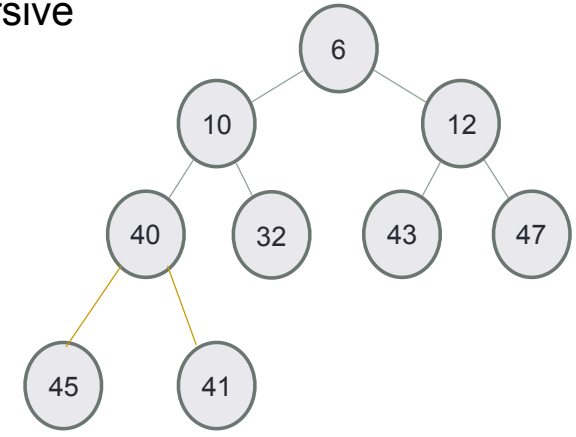
→ combine to a single statement

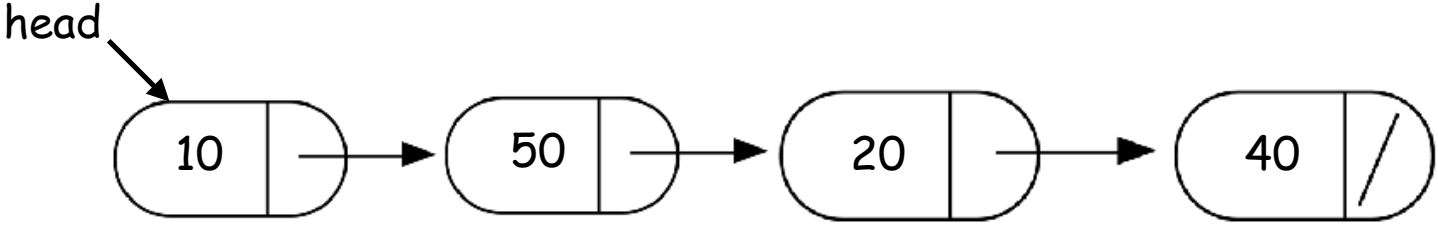
```
return n * fact(n-1);
```

Examples in this course

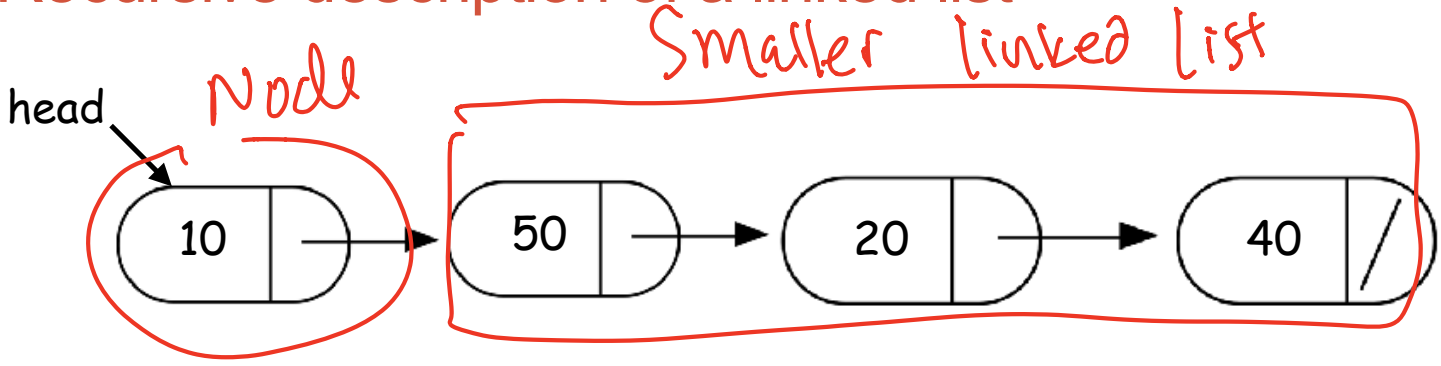
Ask questions about data structures that have a recursive structure like linked lists and trees:

- Find the sum of all the elements in this tree
- Print all the elements in the tree
- Count the number of elements in this tree





Recursive description of a linked list



- A non-recursive description of the linked list:
  A linked list is a chain of nodes

- A recursive description of a linked-list:
  A linked list is a node, followed by a smaller linked list

Sum the elements in a linked list

head → 10 → 50 → 20 → 40 → /

first node

rest of the list

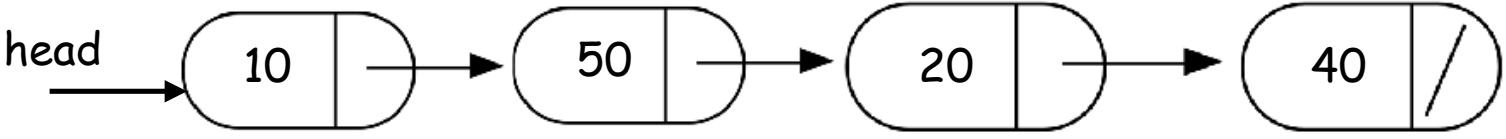
head 0

```
Sum of the elements in the linked list:
    If the linked list is empty,
      return 0
   else
       return Value of the first node +
              Sum the elements in the rest of the list
```

work to make the input smaller

Recursive call: use the same algorithm to calculate the sum of the elements of the rest of the list

Search for an element in a linked list



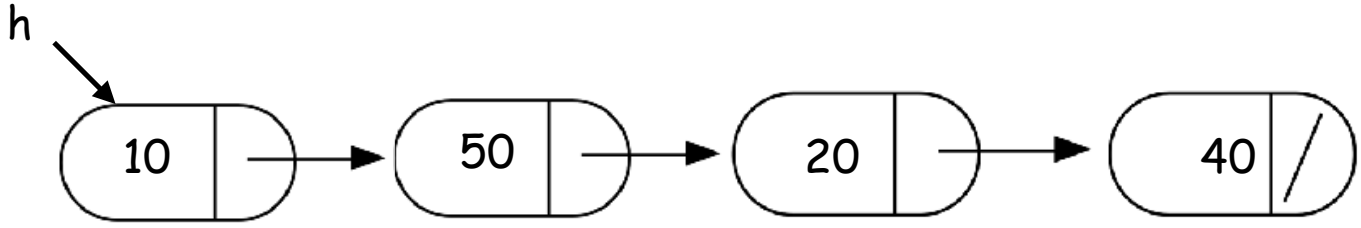
Search for an input value in the linked list:

```
        If the value of the first node == input value
                    return true
        else
                    Search in the rest of the list
```

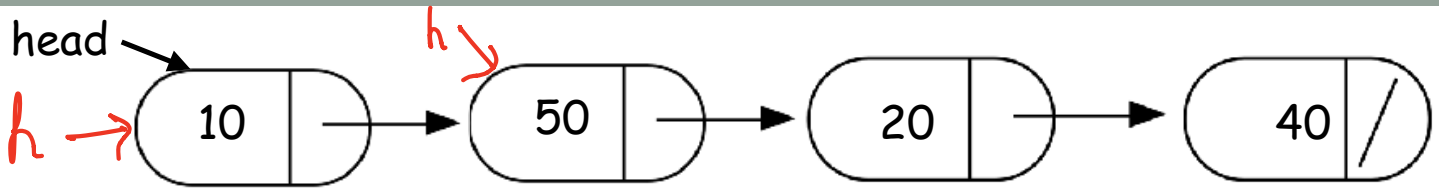
The base case

```
int IntList::search(Node* h, int value){
    // Solve the smallest version of the problem
    // BASE CASE!!
   if(!h) return false; // If the list is empty return false
   if ( h->info == value)
                return true;

   return search(h->next, value);

}
```

pointer to the first node in the linked list (This changes with every recursive call)




```
int IntList::search(Node* h, int value){

    // BASE CASE!!
    if(!h) return false;
    if (h->value == value)
       return true;
    // RECURSIVE CASE:
    return search(h->next, value);
  }
```




```
int IntList::search(Node* h, int value){

    // BASE CASE!!
    if(!h) return false;
    if (h->value == value)
        return true;

    // RECURSIVE CASE:
    search(h->next, value);
}
```

// Found 50
// this will be executed

This call evaluates to true when h->next is a pointer to 50
However, it returns "nothing"

What is the output of
cout<<search(head, 50);

A. Segmentation fault
B. Program runs forever
C. Prints true or 1 to screen
D. Prints nothing to screen // junk value
E. None of the above

Helper functions

- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion

For example

```
bool IntList::search(int value){

   return search(head, value);
    //helper function that performs the recursion.

}
```

public method of the class hard to recurse on because we cannot "pass in" a linked list

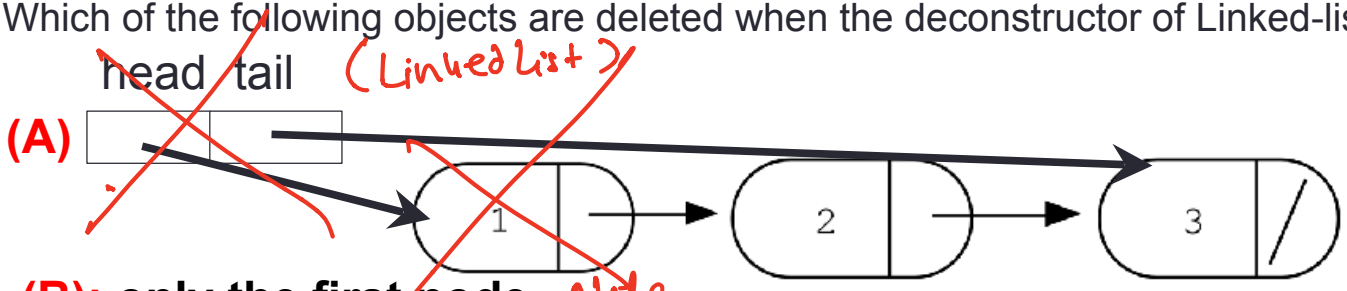
Define a helper function

Recursive deconstructors

```
LinkedList::~LinkedList(){
   delete head; //This is really a call
}                  to Node's de-constructor
```

```
class Node {
    public:
        int info;
        Node *next;
};
```

Which of the following objects are deleted when the deconstructor of Linked-list is called?



(B): only the first node

(C): A and B

(D): All the nodes of the linked list

(E): A and D

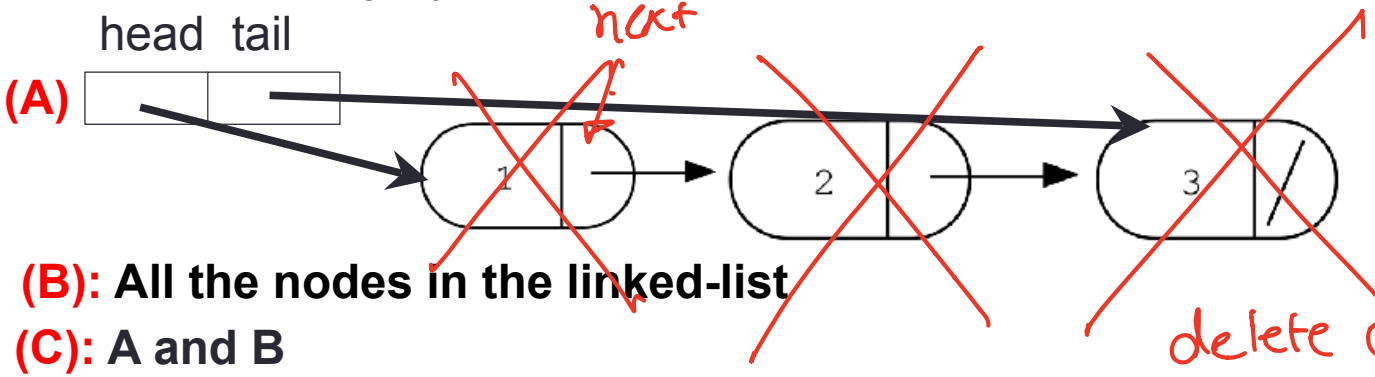
class LinkedList {
Node *head;
Node *tail;
}

Recursive deconstructors

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```

Which of the following objects are deleted when the deconstructor of Linked-list is called?



(B): All the nodes in the linked-list

(C): A and B

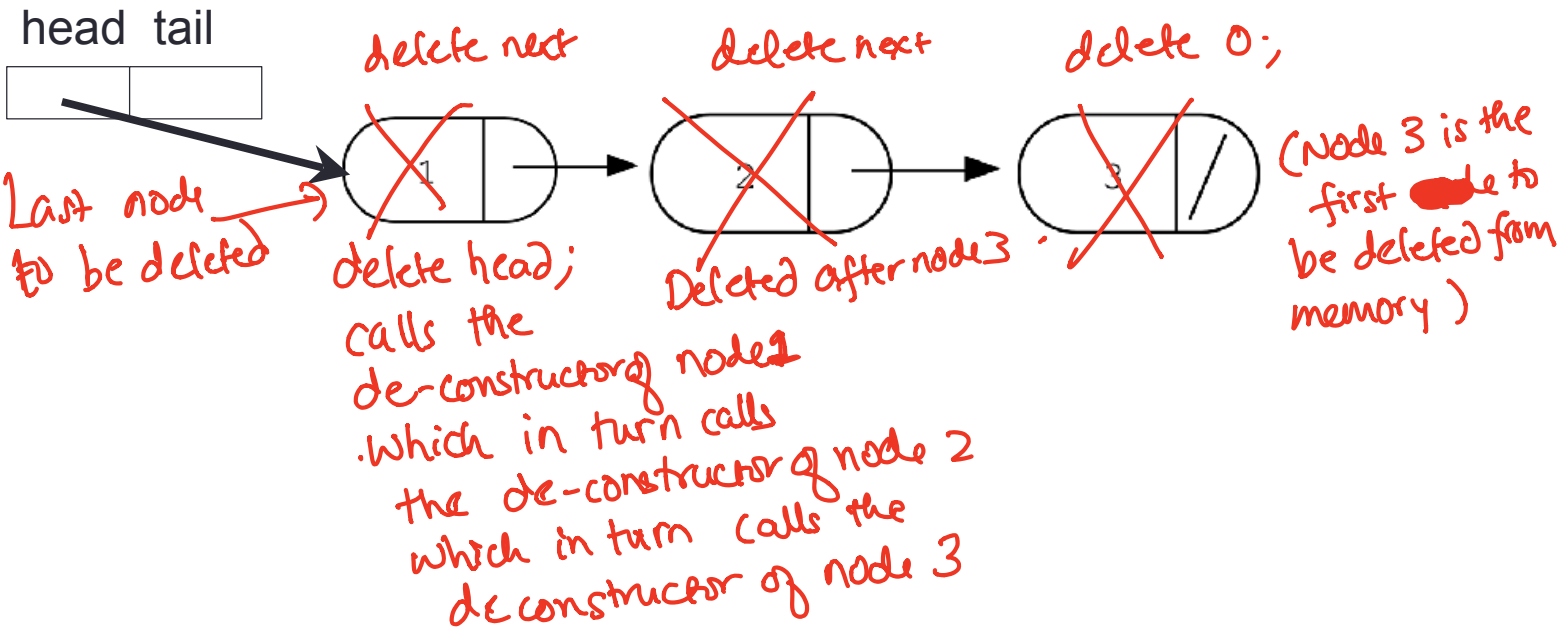
(D): Program crashes with a segmentation fault

(E): None of the above

delete 0;
does nothing.

```
LinkedList::~LinkedList(){
   delete head;//This starts a chain of Node deletions
}
```

```
Node::~Node(){
→ delete next;
}
```



How is PA02 going?

Checkpoint - Thurs. May 3

Get your design checked off before implementing your classes

A. **Done**
B. **Completed designing my classes but haven't implemented them yet**
C. **I understand how to approach the PA, haven't designed by classes yet**
D. **I don't quite understand how to approach the assignment**
E. **Haven't read it yet**

Design your classes and have an outline of how you would use your classes to implement the game.

PA02

(store as a linkedlist)

Alice's hand

Bob's hand

Play the game on these sample cards to help design your classes & the algorithm for the game.

Next time

- More on Running time analysis