RECURSION

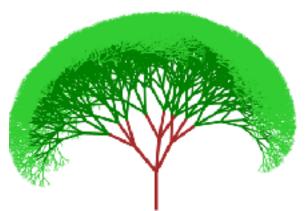


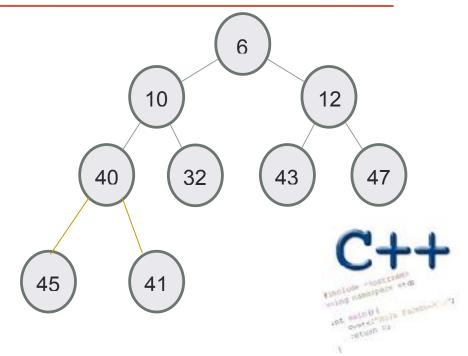




Problem Solving with Computers-II









CS178

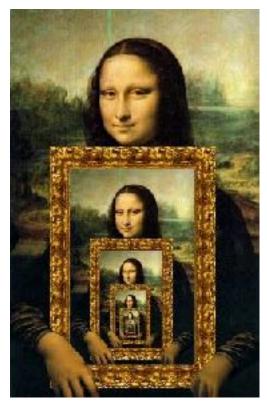
UNSURE ABOUT MAJOR-FIELD ELECTIVES?

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!

To wash the dishes in the sink:

Wash the dish on top of the stack

If there are no more dishes

you are done!

Else:

Wash the *remaining* dishes in the sink

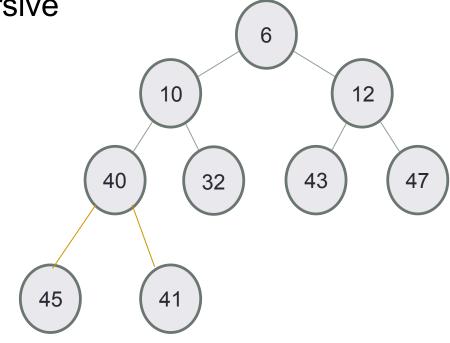
Compute the factorial of a number

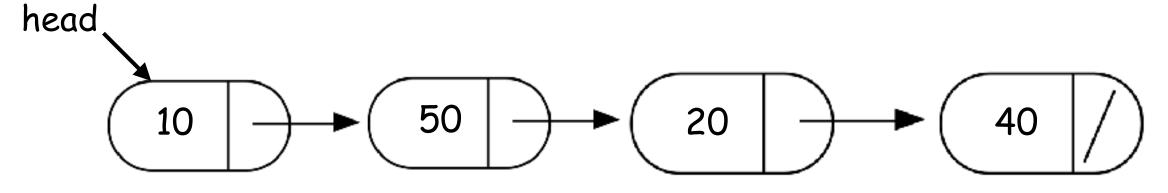
```
n! = n^* (n-1)^* (n-2)^* ....*1, if n>=1
= 1 , if n=0
```

Examples in this course

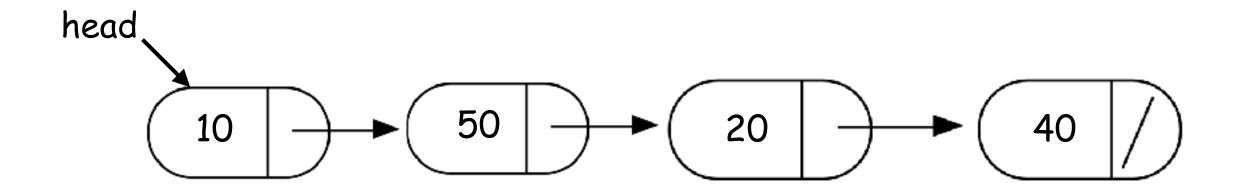
Ask questions about data structures that have a recursive structure like linked lists and trees:

- 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



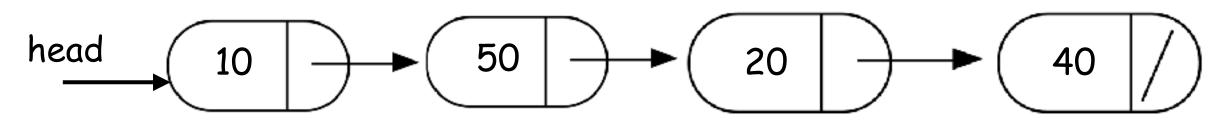


Recursive description of a linked list



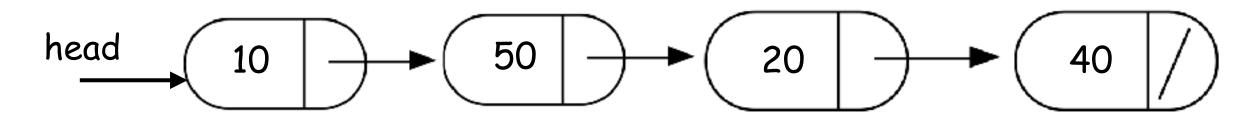
- A non-recursive description of the linked list:
 - A linked list is a chain of nodes
- A recursive description of a linked-list:
 A linked list is a node, followed by a smaller linked list
 - A linked list is a node, followed by a smaller linked list

Sum the elements in a linked list



```
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
```

Search for an element in a linked list



Search for an input value in the linked list:

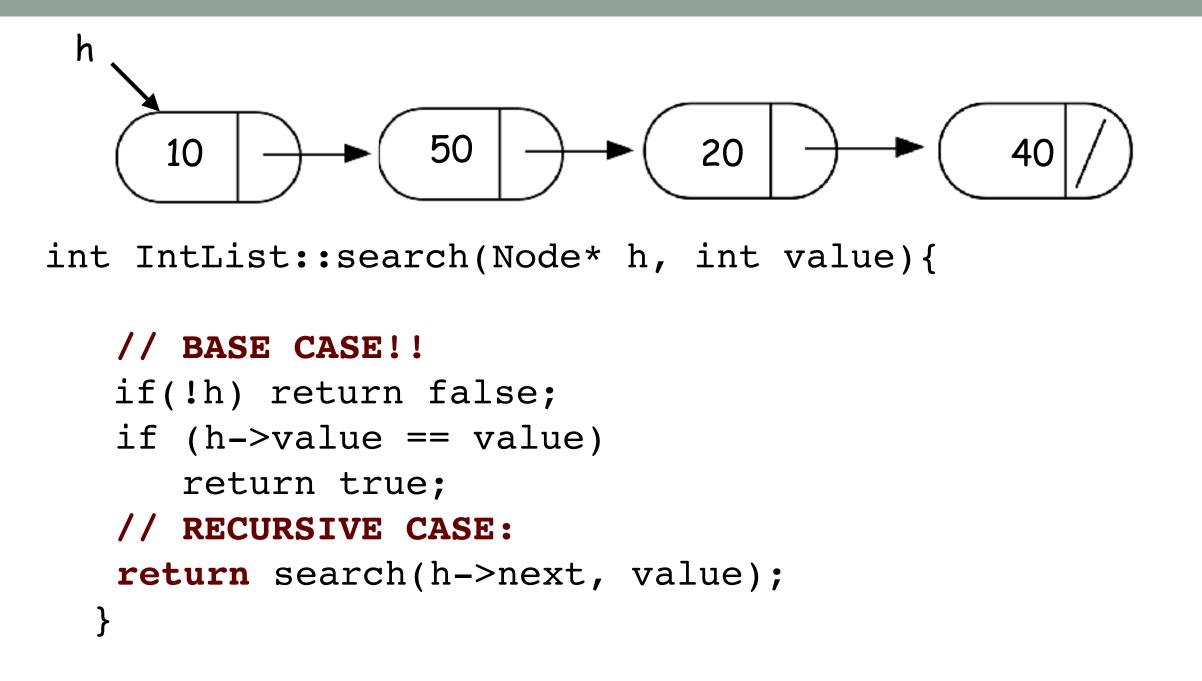
If the value of the first node == input value return true

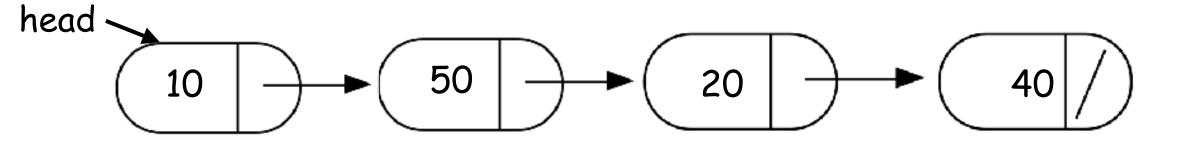
else

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;
```





```
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);
```

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
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.
```

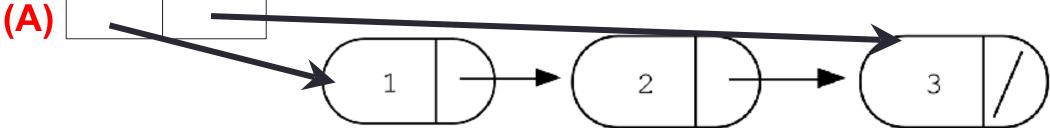
Recursive deconstructors

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

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

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

head tail



(B): only the first node

(C): A and B

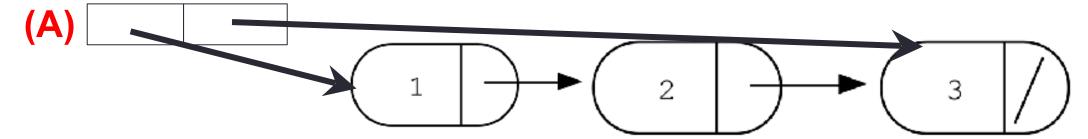
(D): All the nodes of the linked list

(E): A and D

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? head tail



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

(C): A and B

(D): Program crashes with a segmentation fault

(E): None of the above

```
LinkedList::~LinkedList(){
   delete head;
}
head tail
Node::~Node(){
   delete next;
}
```

How is PA02 going?

- 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



PA02



















Performance questions

- How efficient is a particular algorithm?
 - CPU time usage (Running time complexity)
 - Memory usage
 - Disk usage
 - Network usage
- Why does this matter?
 - Computers are getting faster, so is this really important?
 - Data sets are getting larger does this impact running times?

How can we measure time efficiency of algorithms?

One way is to measure the absolute running time

• Pros? Cons?

```
clock_t t;
t = clock();

//Code being timed
t = clock() - t;
```

Which implementation is significantly faster?

```
function F(n) {
    if(n == 1) return 1
    if(n == 2) return 1
return F(n-1) + F(n-2)
}
```

```
function F(n) {
  Create an array fib[1..n]
  fib[1] = 1
  fib[2] = 1
  for i = 3 to n:
     fib[i] = fib[i-1] + fib[i-2]
  return fib[n]
}
```

A. Recursive algorithm

B. Iterative algorithm

C. Both are almost equally fast

A better question: How does the running time scale as a function of input size

```
function F(n) {
    if (n == 1) return 1
    if (n == 2) return 1
    return F(n-1) + F(n-2)
}
function F(n) {
    Create an array fib[1..n]
    fib[1] = 1
    fib[2] = 1
    for i = 3 to n:
        fib[i] = fib[i-1] + fib[i-2]
        return fib[n]
}
```

The "right" question is: How does the running time scale? E.g. How long does it take to compute F(200)?let's say on....

NEC Earth Simulator



Can perform up to 40 trillion operations per second.

The running time of the recursive implementation

The Earth simulator needs 2^{95} seconds for F_{200} .

Time in seconds

210

220

230

240

270

Interpretation

17 minutes

12 days

32 years

cave paintings

The big bang!

```
function F(n) {
    if (n == 1) return 1
    if (n == 2) return 1
return F(n-1) + F(n-2)
}
```

Let's try calculating F₂₀₀ using the iterative algorithm on my laptop.....

Next time

More on Running time analysis