Lecture 5:
Recursion
(Chapter 4)

Oct 4, 2016

Data Structures, CS102

Today's Lecture

I. Warmup, Review, Homework, etc.
II. What is Recursion?
III. How to do think Recursively
IV. How Recursion Works

Next...

I. Prelim, Review, etc

II. What is Recursion?

III. How to think Recursively

IV. How Recursion Works

V. Queues

VI. How Recursion Works

[Start] [End]

I. Prelim, Review, etc

Programmers have more fun

Knock, knock!

Programmers have more fun

Knock, knock!

Who's there?

Programmers have more fun

Knock, knock!

Who's there?

Very long ... pause....

Programmers have more fun

Knock, knock!

Who's there?

Very long ... pause....

Java.

Programmers have more fun

Reader Comments:

* Took me a while as I'm running on Java.

* Downvote. Recent versions of Java are not slow.

  You C/C++ people will forever base your opinions

  on old versions of Java and its VM.

* Upvote to counter-act your lack of a sense of humor.

* Its not a bug, its a feature.

* No, it's a bug that's been mislabeled as a feature.

Prelim, Review, Comments

Clinic for Terminal/Makefile

- this Wed (2-5)
- Send me email to confirm

Prelim, Review, Comments

Homework 3: combines ArrayStringLog + Zoombinis

Zoombini tradeoff: Coverage vs. Correctness

Prelim, Review, Comments

Breaking up command line arguments:

```
// Inside Makefile:
   ss=0
   nn=16
   mm=0
   pp="src"
   args=$(ss) $(nn) $(mm) $(pp)

   run:
        java $(p) $(args)
```

Prelim, Review, Comments

Advantage? Ability to override any part:

- `make run mm=3`
- `make run mm=1 nn=2`

Next...

I. Prelim, Review, etc

II. What is Recursion?

III. How to think Recursively

IV. How Recursion Works

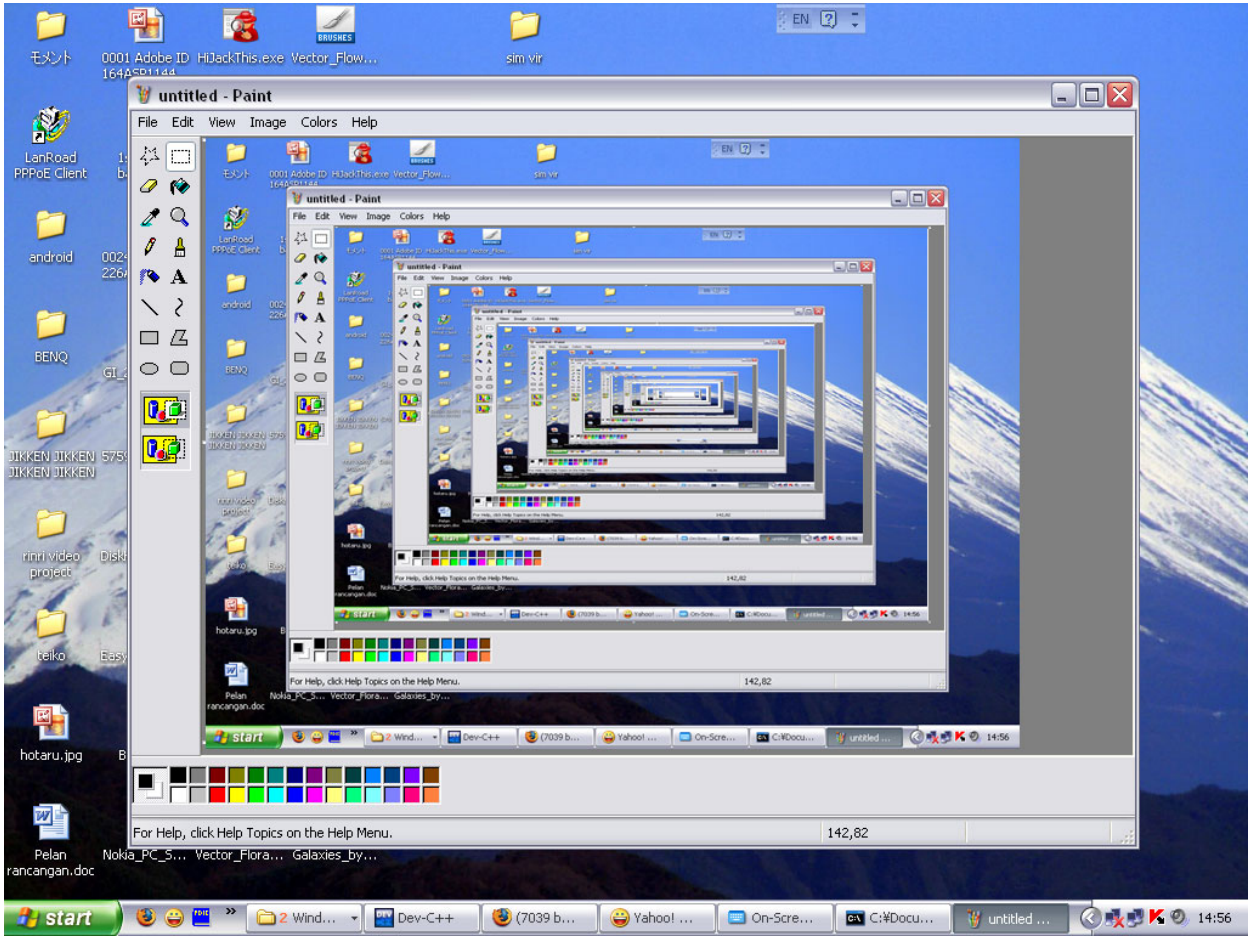
V. Queues

VI. How Recursion Works

[Start] [End]

II. What is Recursion?

Today's Topic:



Examples

What are Natural Numbers?

(Induction in High School)

Examples

What are Natural Numbers?

(Induction in High School)

A1 : (Informal)

It is a member of the set $\{0, 1, 2, \ldots\}$

Examples

What are Natural Numbers?

(Induction in High School)

A1 : (Informal)

It is a member of the set $\{0, 1, 2, \ldots\}$

A2 : (Formal)

It is either 0

or

it is 1 plus another natural number.

(and nothing else)

Examples

Recall Linked List (Lecture 3).

Examples

What is a List of Nodes?

AA :

It is either `null`

or

it is a Node `u` where `u.next` is a list.

Note: `null` is useful!

What is Recursion in Java?

Recursive classes, e.g.,

```
class Node {
    String val;
    Node next;
}
```

What is Recursion in Java?

Recursive methods, e.g.,

```
public class fac {
  public static int fac( int n) {
    if (n<=1) return 1;
    return n*fac(n-1);
  }
}
```

Math Notation: $n! = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot (n-1)! & \text{for } n \geq 1. \end{cases}$

Thus: $0! = 1, \quad 1! = 1,$

$2! = 2, \quad 3! = 6$

What is Recursion in Java?

Somewhat harder: `fibonacci numbers`

Weblink: Leonardo of Pisa or Fibonacci (ca. 1170–1240)

```
public int fib( int n) {
    if (n<2) return n;
    return fib(n-1)+fib(n-2);
}
```

Note: assume `fib(0)=0`, `fib(1)=1`.

What if...

What happens in Recursive calls?

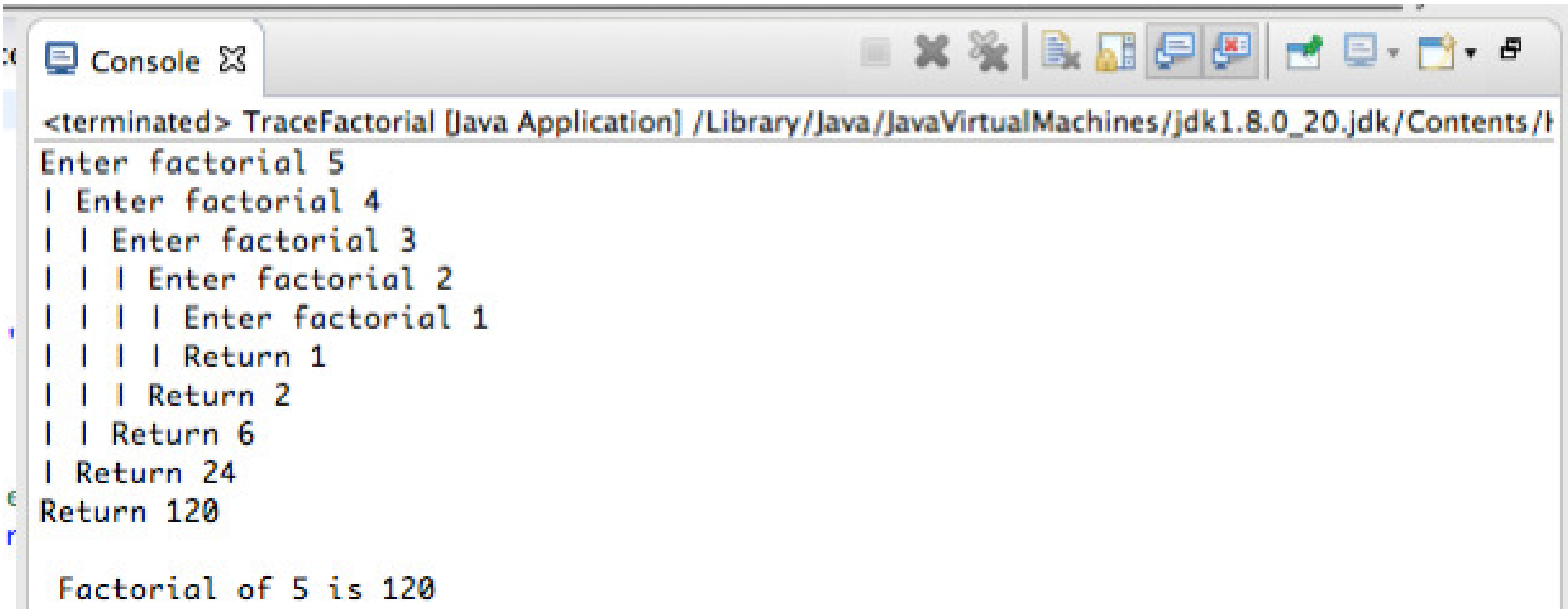
Q: What happens in calling `fac(5)`?

A: There is an implicit `recursive stack`!

What happens in Recursive calls?

Q: What happens in calling `fac(5)`?

A: There is an implicit `recursive stack`!



What happens in Recursive calls?

DEMO

* TraceFactorial

Recursive approach to Lists

Recursive Algorithms for Lists

* Create a simple linked list class called `Node`

* Recursive method to get a list of length n

* Recursive method to print a list

* Recursive method to compute the length of list

See DEMO

Recursive approach to Lists

(Develop this in class)

Construct a list of length n:

```
public static Node getList(int n) {
  if (n <= 0) return null;
  Node u = new Node(n);
  u.next = getList(n-1);
  return u;
}//getList(n)
```

Recursive approach to Lists

Computing the length of a list:

```
public static listLength(Node u) {
  if (u == null) return 0;
  return 1+ listLength(u.next);
}
```

Traversal (Printing) of a list:

```
public static print(Node u) {
  if (u == null) return;
  System.out.println(u.val);
  print(u.next);
}
```

Recursive approach to Lists

BONUS: printing in reverse is no harder!

```
public static revPrint(Node u) {
  if (u == null) return;
  revPrint(u.next);
  System.out.println(u.val);
}
```

We just exchanged the last two lines!

Tricker: reversing a list

Recursive approach to Lists

Q: What makes these recursive methods possible?

Recursive approach to Lists

Q: What makes these recursive methods possible?

A: Our recursive definition of a list!

This is an important principle

(We will see this again in binary trees)

Next...

I. Prelim, Review, etc

II. What is Recursion?

III. How to think Recursively

IV. How Recursion Works

V. Queues

VI. How Recursion Works

[Start] [End]

III. How to think Recursively

Thinking Recursively

3 Questions:

Base Case:

Make sure there is at least one (may be several)

Recursively Smaller Call:

Do you always reach Base Case? (Termination)

Why is it correct?

Inductive reasoning

Thinking Recursively

Example:

Length of list

reverse Print of list

Thinking Recursively

Does this remind you of (high school) induction?

* PROBLEM: prove that there is no largest prime number

  Let $P(n)$ be this predicate on natural numbers

  $P(n) \equiv$ "There is a prime number larger than n"

* We must prove that $P(n)$ is valid, i.e.,

  For all $n \in \mathbb{N}$, $P(n)$ is true

Thinking Recursively

Proof:

* Show basis: $P(0)$

* Show induction step: $P(n)$ implies $P(n+1)$

* CONCLUDE: P is valid

The warrant for this conclusion is called the Principle of Induction

Tower of Hanoi Problem

Web demo:
britton.disted.camosun.bc.ca/hanoi.swf

Try $n = 4$.

Tower of Hanoi Problem

Given: Tower(n) in Peg 1

Goal: Move Tower(n) to Peg 3

Tower of Hanoi Problem

Given: Tower(n) in Peg 1

Goal: Move Tower(n) to Peg 3

Recursive Solution:

```
moveTower( n, startPeg, goalPeg, auxPeg )
   moveTower( n-1, startPeg, auxPeg, goalPeg)
   movePeg( startPeg, goalPeg)
   moveTower( n-1, auxPeg, goalPeg, auxPeg)
```

Exercise: Try a non-recursive solution!

Tower of Hanoi Problem

A bit of analysis:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n-1)+1 & \text{else.} \end{cases}$$

Check: $T(2) = 3$

SUPPOSE we guess that

$$T(n) = 2^n - 1$$

Check: $T(1) = 2^1 - 1, T(2) = 2^2 - 1.$

VERIFY INDUCTIVELY:

$$T(n) = 2T(n-1)+1 = 2(2^{n-1}-1)+1 = 2^n - 1.$$

Tower of Hanoi Problem

If $n = 64$ (original Hanoi story), then
$T(64) = 18,446,744,073,709,551,615$

If each move takes one second, then time needed is 585 billion years (120 times the age of the sun).

Big Bang occurred 14.5 billion years ago,

solar system created about 4.5 billion years ago.

Moral of story?

Tail Recursion

What is it?

When there is one recursive call and it is the last statement of the program

* E.g., `fac(n)`

Tail Recursion

What is it?

When there is one recursive call and it is the last statement of the program

* E.g., `fac(n)`

Tail recursion can be replaced by a while-loop

(which is more effient)

Tail Recursion

What is it?

When there is one recursive call and it is the last statement of the program

* E.g., `fac(n)`

Tail recursion can be replaced by a while-loop

(which is more effient)

But `fib(n)` is not tail recursion

Neither is `revPrint(Node)`?

Tail Recursion

Modern compilers (including Javac) do this automatically!

Final Thoughts

Recursion is basic technique in computer science

It requires “inductive” thinking:

NEVER try to unroll the recursion

to see what happens deeper in the recursion!

QUOTE:

“`To iterate is human,`”

“`to recurse is divine !`”

Next...

I. Prelim, Review, etc

II. What is Recursion?

III. How to think Recursively

IV. How Recursion Works

V. Queues

VI. How Recursion Works

[Start] [End]

IV. How Recursion Works

Second Lecture of Week

So far: we saw what is recursion

Now, we see how recursion is implemented

Static vs. Dynamic Storage Allocation

Example:

```
class Simple {
   static int square(int n) { return n*n; }

   int cube(int n) { int a=square(n); return a*n; }

   public static void main(String[] args){
      System.out.printf("%d squared is %d\n", 5, square(5));
      System.out.printf("%d cubed is %d\n", 5, cube(5));
   }
}
```

Static vs. Dynamic Storage Allocation

The activition record for `square`:

```
class ActivationRecord {
   AddressType returnAddress; // return address
   int result; // return value
   int n; // parameter
}
```

* When we call a method, it must be loaded into memory

* When one method calls another, we need to know where it is called from (return address in memory), and how to pass back results

Static vs. Dynamic Storage Allocation

The activition record for `cube`:

```
class ActivationRecord {
   AddressType returnAddress; // return address
   int result; // return value
   int n; // parameter
   int a; // variable
}
```

* Each time we call a method, its activation record in put onto the "runtime stack"

Static vs. Dynamic Storage Allocation

Fibonacci:

```
int fib( int n ) {
   if (n<2) return n;
   return fib(n-1) + fib(n-2);
}
```

Static vs. Dynamic Storage Allocation

The runtime stack when computing `fib(3)`:

```
fib(3)
     fib(2)
          fib(1)
          return 1
          fib(0)
          return 0
     return 1
     fib(1)
     return 1
return 2
```

* DEMO with `fib(4)` to see the stack grow and shrink!

Static vs. Dynamic Storage Allocation

MAIN LESSON:

There is a runtime stack when we execute Java Programs

Thanks for Listening!

“Algebra is generous,
she often gives more than is asked of her.”

— JEAN LE ROND D’ALEMBERT (1717-83)