

Class 10: Recursion

Introduction to Computation and Problem Solving

Prof. Steven R. Lerman
and
Dr. V. Judson Harward

Methods and Variables, 1

```
public class Variables
{
    public static void main(String[] args)
    {
        int i = 42;
        i = incr( i );
        System.out.println( "i = " + i );
        i = decr( i );
        System.out.println( "i = " + i );
    }

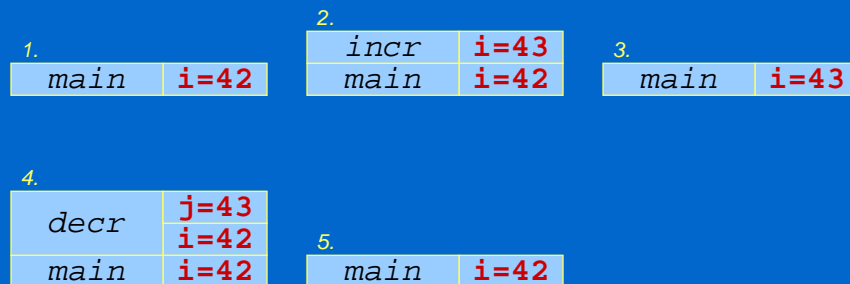
    public static int incr( int i )
    { return ++i; }

    public static int decr(int j)
    {
        int i = j-1; return i;
    }
}
```

2

Methods and Variables, the Stack

Are the different `i` variables going to interfere with each other?



3

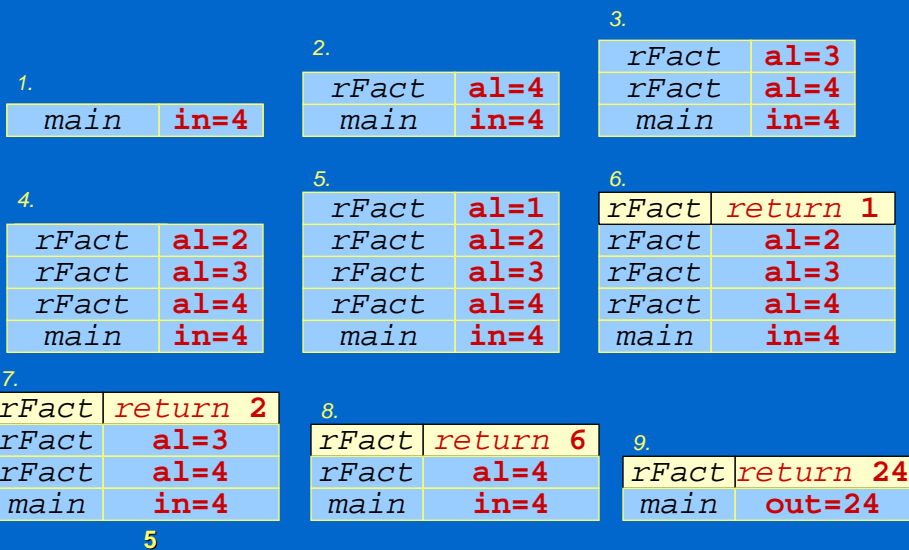
Recursion: Can a Method Call Itself?

```
import javax.swing.*;

public class RecursiveFactorial {
    public static void main(String[] args) {
        String iString = JOptionPane.showInputDialog(
            "Factorial of?");
        int in = Integer.parseInt( iString );
        long out = rFact( in );
        System.out.println( "The factorial is " + out);
        System.exit( 0 );
    }
    public static long rFact ( long al ) {
        if ( al < 0 )
            throw new IllegalArgumentException();
        else if ( al == 1 || al == 0 )
            return 1;
        else return al*rFact( al - 1 );
    }
}
```

4

Recursive Method on the Stack



Recursion

- A recursive algorithm takes a problem and tries to express the answer in terms of a smaller version or versions of the same type of problem.
 - $\text{factorial}(n) = n * \text{factorial}(n - 1)$;
- A recursive algorithm must always have a **termination condition** or **base case** for which it can compute the answer directly.
 - $\text{factorial}(1) = 1$
- The computer then unwinds the previous calls on the stack combining results using the recursive algorithm.
 - $\text{factorial}(n) = n * \text{factorial}(n - 1)$;
- 3 steps: check for (1) subdivide problem, (2) recognize base case / termination condition, and (3) combine results

Recursion, 2

The reason to use recursion is not that it is particularly efficient, but that it is often clearer.

- A factorial can be computed just as easily using iteration.

```
public static long iFact( long l ) {  
    if ( l < 0 )  
        throw new IllegalArgumentException();  
    long f = 1;  
    while ( l > 1 )  
        f = f * l--;  
    return f;  
}
```

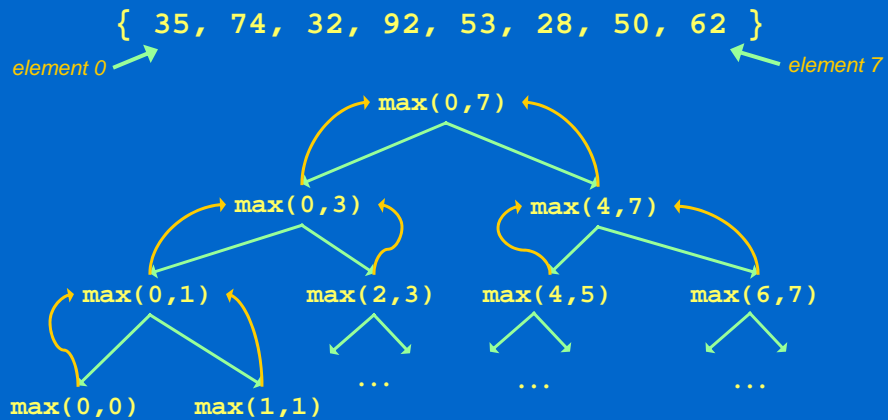
7

Recursion, 2nd Example Finding the Maximum of an Integer Array

- Can we find the maximum of an integer array iteratively?
- Can we do it recursively?
 - How do we subdivide the problem?
 - What's the base case / termination condition?
 - How do we combine results?

8

Recursion, 2nd Example Finding the Maximum of an Integer Array, 1



9

Recursion, 2nd Example Finding the Maximum of an Integer Array, 2

```

1. public class ArrayMax
2. {
3.     public static void main(String[] args) {
4.         int[] arr = { 35, 74, 32, 92, 53, 28, 50, 62 };
5.         int max = arrayMax( arr );
6.         System.out.println( "The array maximum is " + max );
7.     }
8.     public static int arrayMax( int[] a )
9.     { return arrayMax( a, 0, a.length - 1 ); }
10.    public static int arrayMax( int[] a, int start, int end ) {
11.        if ( start == end )
12.            return a[ start ];
13.        int mid = ( start + end ) / 2;
14.        int max1 = arrayMax( a, start, mid );
15.        int max2 = arrayMax( a, mid+1, end );
16.        if ( max1 > max2 ) return max1;
17.        else return max2;
18.    }
19.}
  
```

10

Finding the Maximum of an Integer Array, 3

- Notice there are 2 versions of `maxArray()`, the introductory version that accepts the whole array as an argument and the recursive version.
- How does this method subdivide the problem?
 - Line 13 is crucial. Convince yourself it works.
- What line implements the base case / termination condition?
- What line combines results?

11

Exponentiation

Exponentiation, done 'simply', is inefficient

- Raising x to y power can take $y-1$ multiplications:
 - E.g., $x^7 = x * x * x * x * x * x * x$
- Successive squaring is much more efficient, but requires some care in its implementation
- For example: $x^{48} = (((x * x * x)^2)^2)^2$ uses 6 multiplications instead of 47.
- What is an estimate of how many multiplications it takes to raise x to the y power using successive squaring?
 - Hint: to find $x^{1,000,000,000}$, squaring takes 42 operations while the simple method takes 999,999,999!

12

Exponentiation cont.

- Odd exponents take a little more effort:
 - $x^7 = x * (x*x*x)^2$ uses 4 operations instead of 7
 - $x^9 = x * ((x*x)^2)^2$ uses 4 operations instead of 9
- We can generalize these observations and design an algorithm that uses squaring to exponentiate quickly.
- Writing this with iteration and keeping track of odd and even exponents is possible but tricky.
- It is most naturally written as a recursive algorithm
 - We write a series of 3 identities and then implement them as a Java function!

13

Exponentiation, cont.

Three identities:

- $x^1 = x$ (small enough)
- $x^{2n} = x^n * x^n$ (reduces problem)
- $x^{2n+1} = x * x^{2n}$ (reduces problem)

14

Recursive Exponentiation

```
public class IntPower
{
    public static void main( String [] args ) {...}
    public static BigInteger rIntPower(BigInteger b, int e)
    {
        if ( e == 0 )
            return BigInteger.ONE;
        else if ( e == 1 )
            return b;
        else if ( e % 2 == 1 )
            return rIntPower(b, e - 1).multiply(b);
        else {
            BigInteger b1 = rIntPower(b, e / 2);
            return b1.multiply(b1);
        }
    }
}
```

15

Recursive Exponentiation, 2

- Why do we use `BigInteger` instead of standard `long` arithmetic.
- What is the base case / termination condition?
- How many ways do we subdivide the problem?
- How are results combined?

16

Recursive Exponentiation, main()

```
public static void main(String[] args) {
    String bStr =
        JOptionPane.showInputDialog("Base?");
    BigInteger base = new BigInteger(bStr);
    String eStr =
        JOptionPane.showInputDialog("Exponent?");
    int exp = Integer.parseInt(eStr);
    BigInteger res = rIntPower( base, exp );
    System.out.println( "Result is " + res );
    System.exit( 0 );
}
```

17

Recursion and Iteration

- It's a tricky exercise to write the exponentiation iteratively
 - Try it if you have time and are interested!
- It's often easier to see a correct recursive implementation
 - Recursion is often closer to the underlying mathematics
- There is a mechanical means to convert recursion to iteration, used by compilers and algorithm designers. It's complex, and is used to improve efficiency.
 - Overhead of method calls is noticeable, and converting recursion to iteration within a method speeds up execution
 - Small or infrequently used methods can be left as recursive

18

Exercise 1

- Download `JavaFiles.zip` for Lecture 10 and unpack it into a new folder. Create a new project using the files.
- Open the file `ReverseArray.java`. It provides the framework for a recursive reverse array method as shown on the next slide.

19

ReverseArray

```
public class ReverseArray
{
    public static void main(String[] args)
    {
        int[] arr = { 35, 74, 32, 92, 53, 28, 50, 62 };
        reverseArray( arr );
        for ( int i = 0; i < arr.length; i++ )
            System.out.print( " " + arr[i] );
        System.out.println();
    }
    public static void reverseArray( int[] a )
    { reverseArray( a, 0, a.length - 1 ); }

    public static void reverseArray( int[] a,
        int left, int right )
    {
        // Write your code here
    }
}
```

20

Exercise 1, cont.

- Like `ArrayMax`, there are two versions of the `reverseArray()` method: one to introduce the problem, and then the recursive version that you will need to complete.
- Why don't the methods return anything?
- Go ahead and write the recursive `reverseArray()` method.
 - What will the termination condition be?
 - How will you reduce the size of the recursive problem?
 - How will you combine results?

21

Exercise 2

- An example sequence is defined as:
 - $q_0 = 0$
 - $q_n = (1 + q_{n-1})^{1/3}$
- Write a recursive method to compute q_n
- Open `Sequence.java` (or type it from next page)
 - `main()` is written for you
 - The recursive method 'signature' is written also
 - Complete the recursive method.
- Save/compile and run or debug it
 - Try $n = 10$, $n = 20$

22

Sequence.java

```
import javax.swing.*;

public class Sequence {
    public static void main(String[] args) {
        String input= JOptionPane.showInputDialog("Enter n");
        int n= Integer.parseInt(input);
        double lastTerm= q(n);
        System.out.println("Last term: "+ lastTerm);
    }

    public static double q(int n) {
        // Write your code here
    }
}

// Sample output:
n: 0 answer: 0.0
n: 1 answer: 1.0
n: 2 answer: 1.2599210498948732
n: 3 answer: 1.3122938366832888
```

23