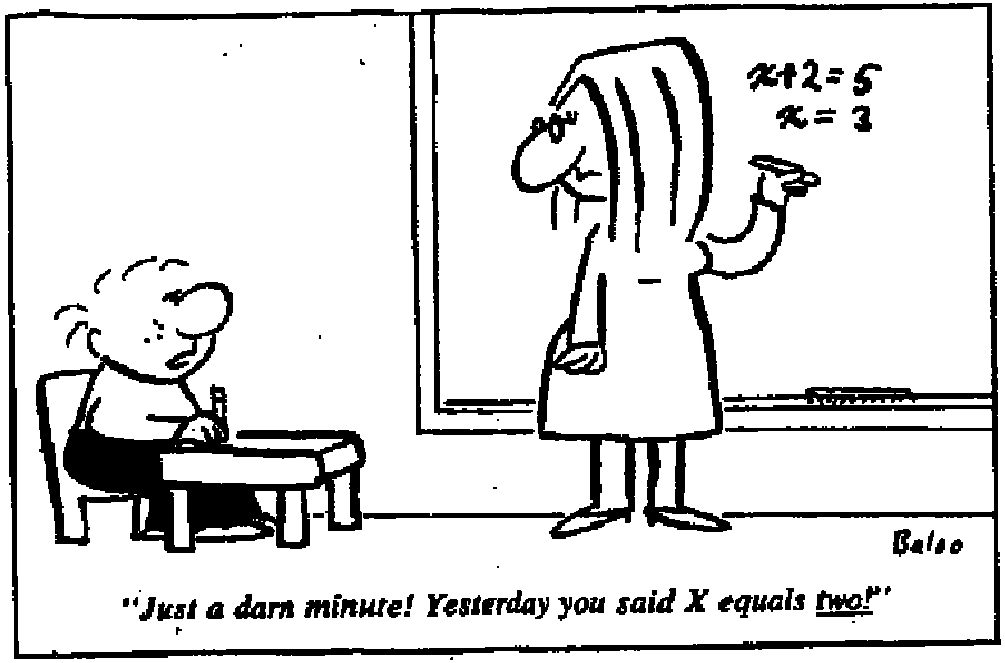
CIV 201

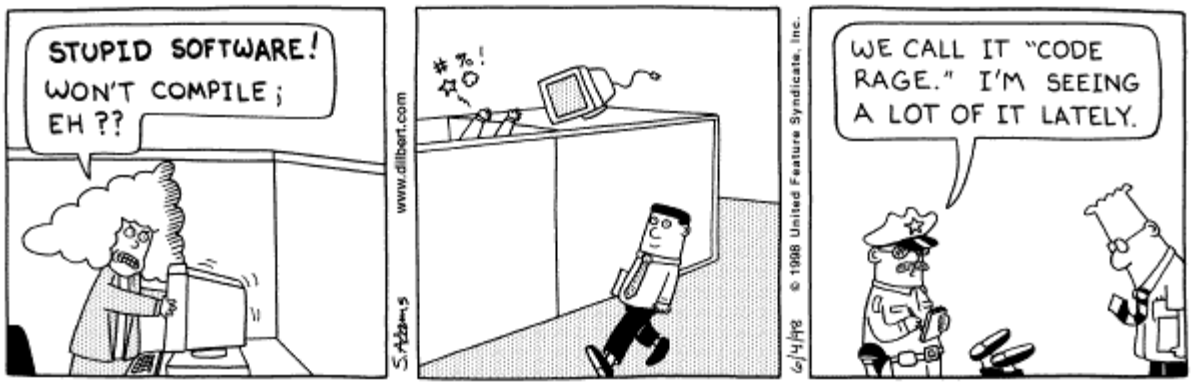
COMPUTER METHODS FOR PROBLEM SOLVING

Lecture 9
Call-by-Value and Recursion



Hints on Programming

- Don’t wait until the last day to start.
- Compile on your local machine. Use elmo, zoe, etc. only if you must -- they are busy machines.
- Develop incrementally. Start with a working program. Make small changes. Check that the program still works.



Examples of incremental development

- Come prepared, having read the assignment, having read the textbook, having listened in lecture, having prepared the outline below.
- Copy working version of **`Integra.java`** for **`civ201a`**.
- Add a method **`func()`** to evaluate a function. For starters, let the function be $f(x) = x^2$.
- Call **`func()`** from **`integrate()`** with a few choice values and print out the values computed to make sure **`func()`** works properly.
- Write code in **`paint()`** to draw the function defined by **`func()`**.
- Write a method **`rect()`** to compute the area using the rectangle rule. Put a call to this method in **`integrate()`**. Use **`System.out.println`** to print out the answer. Run with **`lowx = 0`**, **`high x = 1`**, **`n = 1`**. Compare this answer against a **hand calculation**. Try some other simple cases.
- Make a new method called **`trap()`**. Copy the code in **`rect()`** to **`trap()`**. Modify the code just copied so that it implements the trapezoidal rule. Test as above.
- Add code to **`paint()`** to illustrate the approximating rectangles. Test with small values of **`n`**.
- Add code to **`paint()`** to illustrate the approximating trapezoids. Test with small values of **`n`**.
- Change function implemented in **`func()`** to the complicated exponential function given in the assignment.

Call-by-Value

Can a method change the data in its argument list?

```
public class CallByValue
{
      public static void main(String[] args)
      {
         int n;
         n = 37;

         System.out.println("before: n = " + n);
         SetToSix(n);
         System.out.println("after:  n = " + n);
      }

      static void SetToSix(int n)
      {
         System.out.println("inside: n = " + n);
      }
}
```

The **n** in **main** is *local* to **main**.

The value of **n** in **SetToSix** is *initially* set to the value of the corresponding variable on the argument list in the method from which the call originated. In this case 37.

The **n** in **SetToSix** is *local* to **SetToSix**.

Can you guess the last line of output?

```
before:  n = 37
inside:  n = 37
after:   n = __
```

What about Global Variables?

Can you guess the output?

```
public class CallByValue
{
    static int N;        // a GLOBAL variable

    public static void main(String[]  args)
    {
        N = 314;

        System.out.println("N = " + N);
        SetToSix();
        System.out.println("N = " + N);
    }

    static void SetToSix()
    {
        N = 6;
    }
}
```

```
before:  N = 314
after:   N = ___
```

The argument list is empty. Would the answer be the same if **N** had been passed as an argument?

What about Arrays?

```
public class CallByValue
{
    public static void main(String[] args)
    {
        double[] x = new double[12];
        for (int j=0; j<12; j++) { x[j] = j; }
        System.out.print   (“before: x[7] = ” + x[7]);
        ComputeSquares(x);
        System.out.println(“after: x[7] = ” + x[7]);
        System.out.print   (“before: x[4] = ” + x[4]);
        ComputeSqrt(x[4]);
        System.out.println(“after: x[4] = ” + x[4]);
    }
    static void ComputeSquares(double[] ex)
    {
        for (int j=0; j<12; j++) {
            ex[j] = ex[j]*ex[j];
        }
    }
    static void ComputeSqrt(double y)
    {
        y = Math.sqrt(y);
    }
}
```

What do you think is the output this time?

```
before: x[7] = ___   after: x[7] = ___
before: x[4] = ___   after: x[4] = ___
```

Moral: The *value* that is copied to the called method is a pointer, not the entire pile of stuff to which the pointer points.

What about Class Variables?

```
class Zip
{
    int zip; double lat; double lon; double x; double y;
}
public class CallByValue
{
    public static void main(String[] args)
    {
        Zip z;
        z = new Zip();
        z.zip = 90210;
        z.lat = 34.09;
        z.lon = 118.41;
        System.out.print   (“before: zip  = ” + z.zip);
        SetToPrinceton(z);
        System.out.println(“after: zip = ” + z.zip);
    }
    static void SetToPrinceton(Zip zed)
    {
        zed.zip = 8544;
    }
}
```

z is a class variable.
But **z.zip** is an **int**.

And the output is

```
before: zip = 90210
after:  zip = _____
```

Moral: Class variables are pointers just as arrays are. Therefore, argument passing works the same as it did with arrays.

Recursion

Consider the *factorial* function: $n! = n \times (n-1) \times ... \times 3 \times 2 \times 1$

The *natural* way to program it uses the following recursive definition:

$n! = n \times (n-1) \qquad 0! = 1$

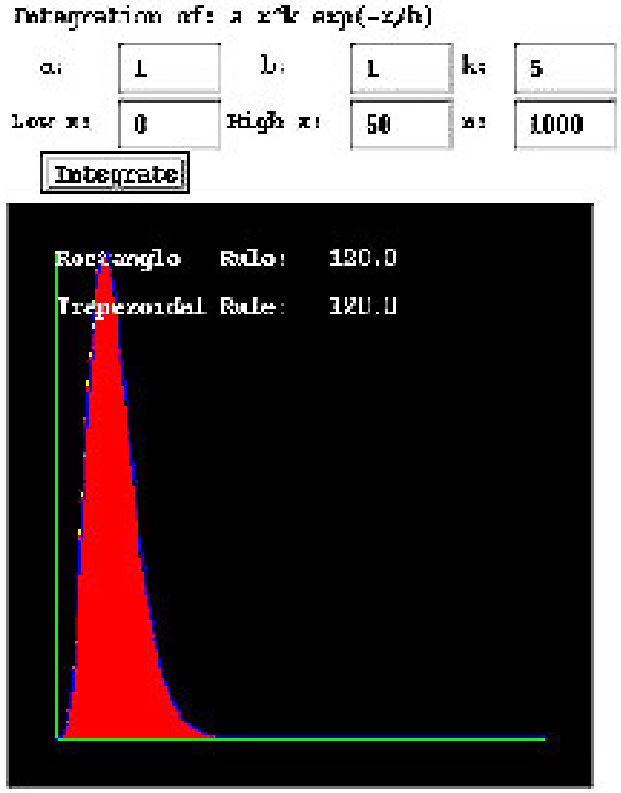
Here's the Java program:

```
import ccj.*;
public class Factorial
{
    public static void main(String[] args)
    {
        int n, n_fact;
        while (true) {
            n = Console.in.readInt("Enter an integer: ");
            if (n<0) break;
            n_fact = factorial(n);
            System.out.println("n factorial = " + n_fact);
        }
    }
    static int factorial(int k)
    {
        if (k==0) {return 1;}
        else      {return k*factorial(k-1);}
    }
}
```

Digression -- Factorials and Integration

Amazing but true: $n! = \int_0^\infty x^n e^{-x} dx$

Can use a slightly modifed **`Integra.java`** to compute these integrals:



Works even with positive real numbers and even some negative numbers:

$$(-1/2)! = \sqrt{p}$$