Today’s topics

Java
  Recursion

Upcoming
  Graphics

Reading
  Great Ideas, Chapter 4
  (begin Chapter 5 for graphics)
Recursion

- Actually have been using *it* in everyday life
- **Dictionary Example**
  - Need *dictionary* to use a *dictionary*
    - Look up word
    - Definition may use words we don’t understand
    - Look up those words
    - Etc.
- Can be confusing
- **Use Clone Model** to sort this out
  - Like using *multiple dictionaries*
- Recursion implies a *self-referential* process
- In computing we say a function *invokes itself*
Recursion

- **Recursion is a *Divide and Conquer* strategy**
  - Decomposing problem
    1. Base Case (*halting case*)
    2. Recursive case (which must get us *closer to solution*)
  - If we don’t have base case, *infinite recursion*
    - Very much like an infinite loop
    - (very bad!)

- **Factorial Example**
  - Definition of N Factorial (domain is non-negative integers)
    - N! = N * (N-1)!
    - 0! = 1
  - Note we defined factorial (or !) in terms of !
  - Which part corresponds to the base case?

- **In class demo…**
Factorial Program

```java
public class RecFact extends java.applet.Applet implements ActionListener {

    TextField mInstruct, mResults;
    IntField gN;
    Button bFact;

    public void init() {
        mInstruct = new TextField(70);
        mInstruct.setText(
            "Enter N, then press button for factorial");
        gN = new IntField(10);
        gN.setLabel( "N");
        bFact = new Button("Factorial");
        mResults = new TextField(70);
        mResults.addActionListener(this);
        add(mInstruct); add(gN); add(bFact); add(mResults);
    }
}
```
public int fact(int n) {
    if (n == 0) {
        return 1;
    }
    return n * fact(n - 1);
}

public void actionPerformed(ActionEvent event) {
    int k;
    k = gN.getInt();
    mResults.setText(k + " factorial = " + fact(k));
}
Using the Clone Model for \( f = \text{fact}(5) \)

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]

\[
\text{public int fact(int n)\{}
\quad \text{if (n==0) return 1;}
\quad \text{return } n \times \text{fact(n-1);} \}
\]
Recursive vs Iterative

- Notice that recursive solution required *No Loop*
  - Repetition is implicit in the process
- Could have used iterative (looping) approach:
  ```java
  public int fact(int n)
  {
    int prod = 1;
    while(n > 0)
    {
      prod = prod * n;
      n = n - 1;
    }
    return prod;
  }
  ```
  - Is actually simpler for this problem
    - For some problems, recursion is much easier (when comfortable with it)
- Watch the SIZE OF THE NUMBERS !!!!
Exponentiation

- Want to calculate $x$ to the $n^{th}$ power
  - Also written as $x^n$
- Brute force approach
  - $x^n = x \times x \times x \times \ldots \times x$
  - How many multiplications?
  - Can we do better?
  - How would you calculate $7^{64}$ with simple 4-function calculator?
- Might calculate $49=7 \times 7$. Then can use $49^{32}$
  - How many multiplications now?
  - Carry on with this idea: $2401 = 49 \times 49$.
    - Leaves us with $2401^{16}$
Exponentiation Recursively

- Want to calculate $x$ to the $n^{th}$ power recursively
- Base case: $N = 0$
  \[ x^0 = 1.0 \]
- Recursive case: $N$ is an even number
  \[ x^N = x^{N/2} \times x^{N/2} \]
- Recursive case: $N$ is an odd number
  \[ x^N = x \times x^{N/2} \times x^{N/2} \]
- Ready to put this into code
public class Recurse extends java.applet.Applet implements ActionListener {

    IntField gN;
    DoubleField gX;
    Label lN, lX;
    Button bFact, bExp;
    TextField mResults;
    int k, n;
    double x;

    public void init() {
        lN = new Label("N");
        lX = new Label("X");
        mInstruct = new TextField(60);
        mInstruct.setText("Enter N and X, then press button for function");
Recursive Expon.2

gN = new IntField(10);
gX = new DoubleField(10);
bFact = new Button("Factorial");
bExp = new Button("Exponential");
mResults = new TextField(60);
bFact.addActionListener(this);
bExp.addActionListener(this);
add(mInstruct); add(lN); add(gN); add(lX); add(gX);
add(bFact); add(bExp); add(mResults);
}

public void actionPerformed(ActionEvent event) {
    Object cause = event.getSource();
    if (cause == bFact) {
        n = gN.getInt();
        x = gX.getDouble();
        mResults.setText(n + " factorial = " + fact(n));
    }
}
Recursive Expon.3

```java
if (cause == bExp) {
    n = gN.getInt();
    x = gX.getDouble();
    mResults.setText(
        x + " to the " + n + " power = " + expon(x, n));
}
}

int fact(int n) {
    if (n<=1) {
        return 1;
    }
    return n * fact(n-1);
}
```
Recursive Expon.4

double expon(double x, int n) {
    double xnot;
    if (n == 0) {
        return 1.0;
    }
    xnot = expon(x, n/2);
    if (n == 2*(n/2)) { // or if (n%2 == 0) i.e., is it even?
        return xnot * xnot;
    } else {
        return x * xnot * xnot;
    }
}
}
Other uses of Recursion

- **Recursion sometime associated with self-similar structure**
  - Fractals are a graphic instantiation of similar ideas
  - Will look at this later

- **Processing folders (directories)**
  - Each folder may contain files or other folders
  - Folder containing folders is self-referential

- **Processing tree-like data structures**
  - Important in computer science
  - (think of your family tree)

- **Many other applications**

- **Recursion can be expensive**
  - Each invocation of a method (function) incurs overhead
  - Use iteration when this is obvious solution (e.g. N!)

- **For many complicated problems, recursive solution is easier!**
Church-Markov-Turing Thesis

Any non-trivial computer language that one can invent is apparently capable of computing no more and no fewer functions than all other nontrivial programming languages.

This part of Java lets you solve all kinds of problems and implement all computer algorithms.

See how far we have gotten!