Week 7 - Monday

COMP 2000

Last time

- What did we talk about last time?
- Swing menus
- Started recursion

Questions?

Project 2

Recursion Examples

Exponentiation

- Similarly, exponentiation is repeated multiplication
- Thus, $x^y = x \cdot x \cdot x \ldots \cdot x$ (y times)
- Base case (***y*** = 0):
  - $x^0 = 1$
- Recursive case (***y*** > 0):
  - $x^y = x \cdot x^{y-1}$

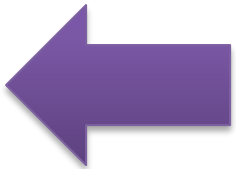
- There is a more efficient way to do this, but you'll have to take COMP 2100 to talk about it

Code for exponentiation

```
public static double power( double x, int y ){

  if( y == 0 )
     return 1.0;
  else
     return x * power( x, y - 1 );

}
```



Base Case

Recursive Case

Summing the first *n* numbers

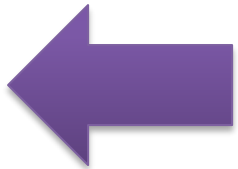
- What if we want to sum the values from 1 up to ***n***?
- $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + (n-1) + n$
- Base case (***n*** = 1):
  - $\sum_{i=1}^{1} i = 1$
- Recursive case (***n*** > 1):
  - $\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$
- True, this sum is $\frac{n(n+1)}{2}$, but don't worry about that

Code for summing up to *n*

```
public static int sumUpTo( int n ){

  if( n == 1 )
     return 1;
  else
     return n + sumUpTo( n - 1 );

}
```



Base Case

Recursive Case

It doesn't have to be mathematical

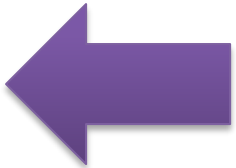
- We could play with strings, too
- What if I want to count the number of uppercase or lowercase E's in a string ***s***?
- Base case (length(***s***) = 0):
  - eCount(***s***) = 0
- Recursive cases (length(***s***) > 0):
  - If ***s*** starts with 'e' or 'E', eCount(***s***) = 1 + eCount(rest of ***s***)
  - Otherwise, eCount(***s***) = eCount(rest of ***s***)

Code for counting E's

```
public static int eCount( String s ){

  if( s.length() == 0 )
     return 0;
  else if(s.charAt(0) == 'e' || s.charAt(0) == 'E')
     return 1 + eCount(s.substring(1));
  else
     return eCount(s.substring(1));


}
```



Base Case

Recursive Cases

Recursive hints

- Always look at the return types
- Are you returning the right thing in all cases?
- Do you have at least one base case to stop the recursion?
- Do you have at least one recursive case to move forward?
- Try not to assign variables
- Don't use loops (unless explicitly told to)
- Don't use member variables or global variables
- Don't try to do everything at once!
  - Just unwrap one layer…

Recursion Tricks

Comparison to loops

- Loops often use indexes to keep track of how far you are in the process
- Sometimes that index is used only to determine when a loop is going to terminate
- At other times, the index value is needed for work done in the loop
- Consider this loop to reverse an array:

```
for(int i = 0; i < array.length/2; ++i) {
      int temp = array[i];
      array[i] = array[array.length - i - 1];
      array[array.length - i - 1] = temp;
}
```

Extra information

- Recursion sometimes requires similar information that can be passed along to each recursive call
- This information could be an index into a **`String`** or an array
- In graph or tree algorithms, it might be the parent node you visited previously
- There are recursive methods with 10 or more parameters
- There's nothing wrong with that, provided that you actually *need* them all

Summing an array

- What if we want to sum the values in an array called ***array***?
- We need some extra information: current index
- Base case (***index*** = ***length***):
  - Sum(from ***index*** onward):
    - 0 (Nothing left to sum)
- Recursive case (***index*** < ***length***):
  - Sum(from ***index*** onward):

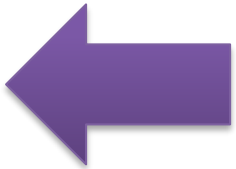
    array[***index***] + Sum(from ***index*** + 1 onward)

Code for summing an array

```
public static double sum(double array[], int index) {

  if( index == array.length )
     return 0.0;
  else
     return array[index] + sum(array, index + 1);

}
```



Base Case

Recursive Case

Reversing a `String`

- What if we want to reverse the contents of a string called ***s***?
- We need some extra information: current index
- Base case (***index*** = ***length***):
  - Reverse(from ***index*** onward):

    "" (Nothing left to reverse)
- Recursive case (***index*** < ***length***):
  - Reverse(from ***index*** onward):

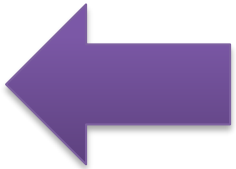
    s[***length*** – ***index*** - 1] + Reverse(from ***index*** + 1 onward)

Code for reversing a `String`

```
public static String reverse(String s, int index) {

  if( index == s.length() )
     return "";
  else
     return s.charAt(s.length() - index - 1) +
          reverse(s, index + 1);

}
```



Base Case

Recursive Case

Waiting for the recursion to come back

- All of the recursion we have shown so far doesn't do much after its recursive call returns
  - In actual fact, we have often waited for the return to add, multiply, or concatenate a value
  - If we simply returned the result of the previous method, it would be **tail recursion**
- Some recursive methods do significant work *before* making a recursive call
- Some recursive methods do significant work *after* making a recursive call
- Some do both!

Using the stack to go in reverse

- All stacks (including the call stack) are first-in last-out (FILO) structures
- In situations where we want to deal with things in backwards order, we can use this natural reversing tendency
- For example, if we want to print out a **`String`** in reverse, we can recurse through each character and print them as the recursion returns
- Doesn't make sense yet?

Printing a `String` in reverse

- What if we want to print the contents of a string called ***s*** *in reverse*?
- We need some extra information: current index
- Base case (***index*** = ***length***):
  - ReversePrint(from ***index*** onward):

    Print nothing
- Recursive case (***index*** < ***length***):
  - ReversePrint(from ***index*** onward):

    ReversePrint(from ***index*** + 1 onward)

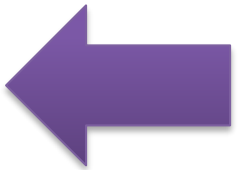
    Then print ***s*** [***index***]

Code for printing a `String` in reverse

```
public static void reversePrint(String s, int index)
{


  if( index < s.length() ) {
          reversePrint(s, index + 1);
          System.out.print(s.charAt(index));
  }


}
```



(Empty)
Base Case

Recursive
Case

Reversing a `String` (the remix)

- We can even use this approach to reverse a string in a different manner than we did before
- Base case (***index*** = ***length***):
  - Backwards(from ***index*** onward):

    "" (Nothing left to reverse)
- Recursive case (***index*** < ***length***):
  - Backwards(from ***index*** onward):

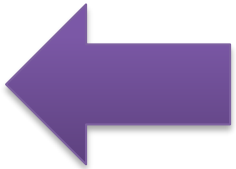
    Backwards(from ***index*** + 1 onward) + ***s***[***index***]

Remixed code for reversing a `String`

```
public static String backwards(String s, int index) {

  if( index == s.length() )
     return "";
  else
     return backwards(s, index + 1) + s.charAt(index);

}
```



Base Case

Recursive Case

Mid-Semester Evaluations

Upcoming

Next time…

- More recursion

Reminders

- Keep reading Chapter 19
- Keep working on Project 2
- **InSocial Risk Advisors are looking for a consultant**
  - They need help linking together some services with Zapier
  - Should be a small amount of work, but it might open up other opportunities
  - If interested, send a resume to Jim Waterwash
  - Get his contact information from me