EECS 280 Discussion #3

Week of January 21

Outline

* Administrivia
* Tail Recursion
* Function Pointers

Announcements

* Assignment #2
  * Due January 31st @ 11:59 PM
* Discussion Slides
  * My slides will be available online before discussion, official notes to follow on CTools
  * http://www-personal.umich.edu/~nlderbin/eecs280

Outline

* Administrivia
* Tail Recursion
* Function Pointers

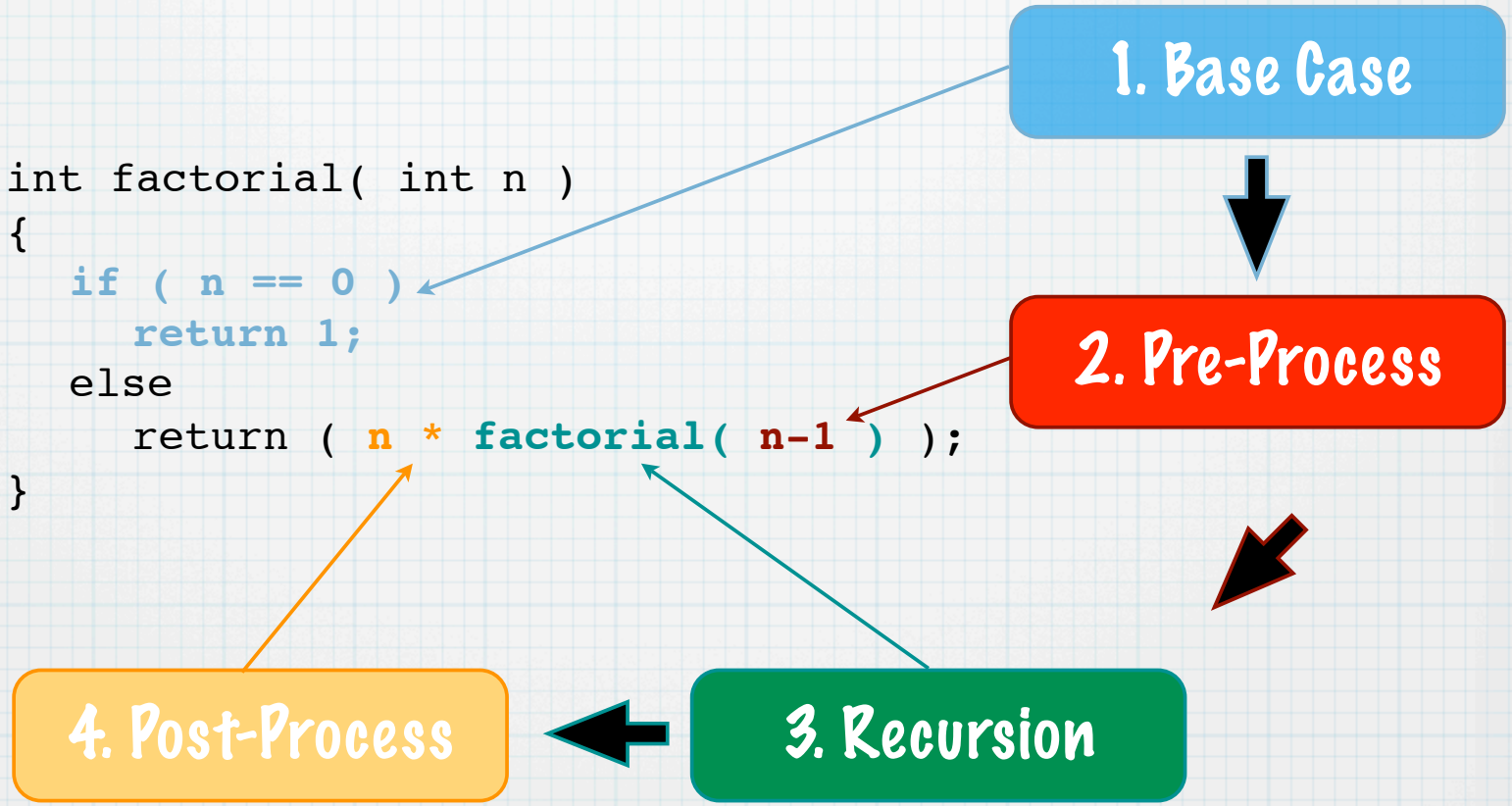
Recursion Review

- Style of programming whereby a function calls itself
- Usually consists of components
  - Base Case(s): small case solution
  - Pre-processing: large case -> small case
  - Recursive Call(s): process small case
  - Post-processing: process small case solution

Recursion Breakdown

```
int factorial( int n )
{
   if ( n == 0 )
      return 1;
   else
      return ( n * factorial( n-1 ) );
}
```

Recursion Breakdown



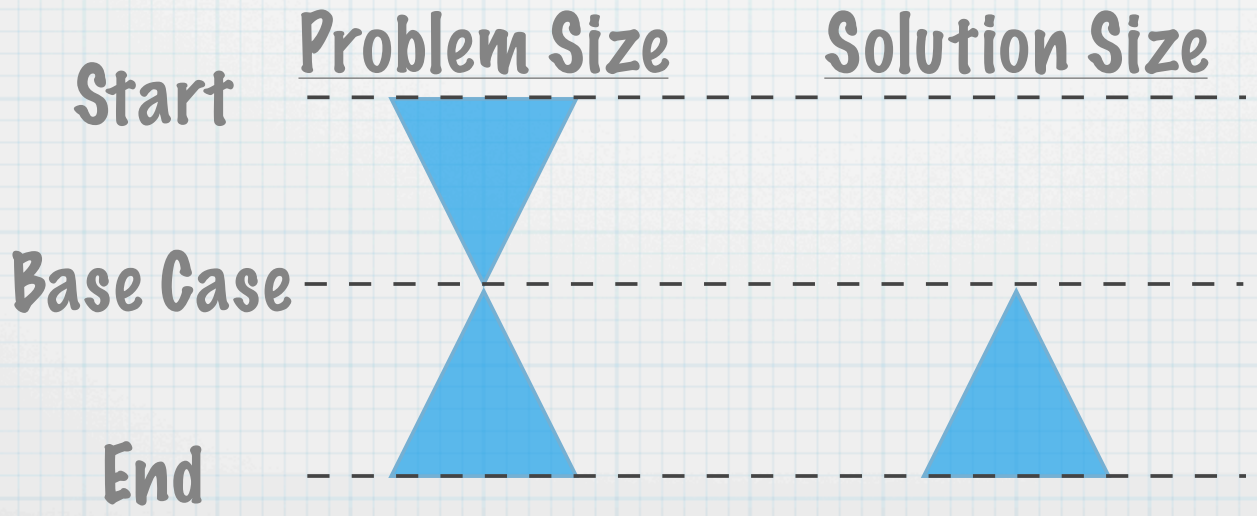
factorial(3)

factorial(3) = 3 * factorial(2) = 3 * 2 = 6

factorial(2) = 2 * factorial(1) = 2 * 1 = 2

factorial(1) = 1 * factorial(0) = 1 * 1 = 1

factorial(0) = 1



Tail Recursion

- Style of recursive programming whereby the return value of the recursive call is not manipulated
  - Tail Recursion has no post-processing

Recursion Breakdown

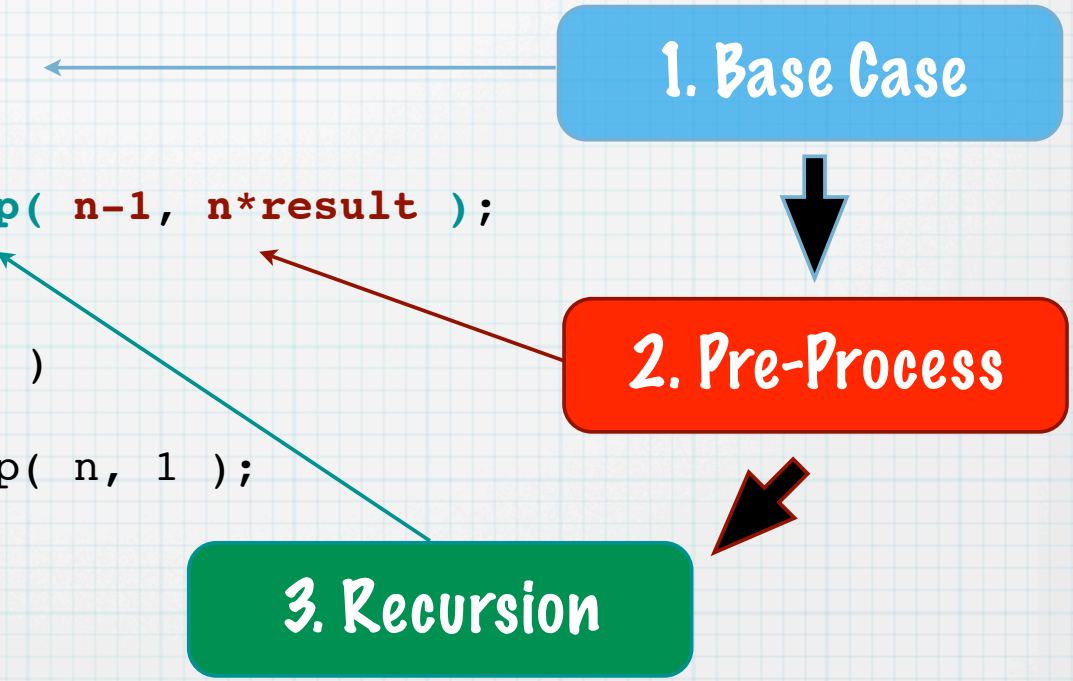
```
static int fact_help( int n, int result )
{
   if ( n == 0 )
      return result;
   else
      return fact_help( n-1, n*result );
}

int fact_tail( int n )
{
      return fact_help( n, 1 );
}
```

Recursion Breakdown

```
static int fact_help( int n, int result )
{
   if ( n == 0 )
      return result;
   else
      return fact_help( n-1, n*result );
}

int fact_tail( int n )
{
      return fact_help( n, 1 );
}
```



fact_tail(3)

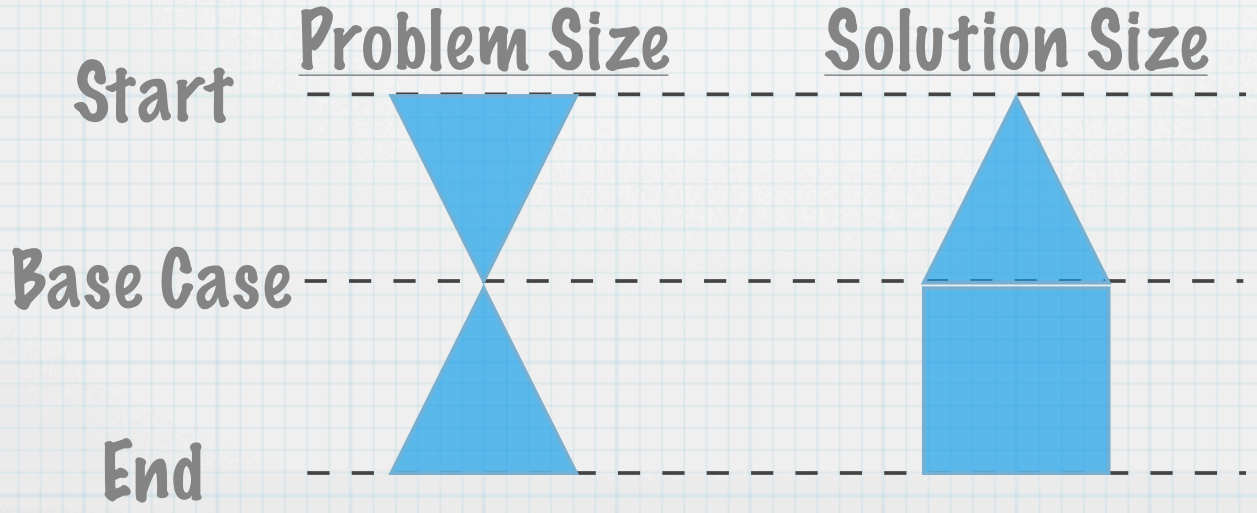
fact_tail(3) = fact_help(3, 1) = 6

fact_help(3, 1) = fact_help(2, 3) = 6

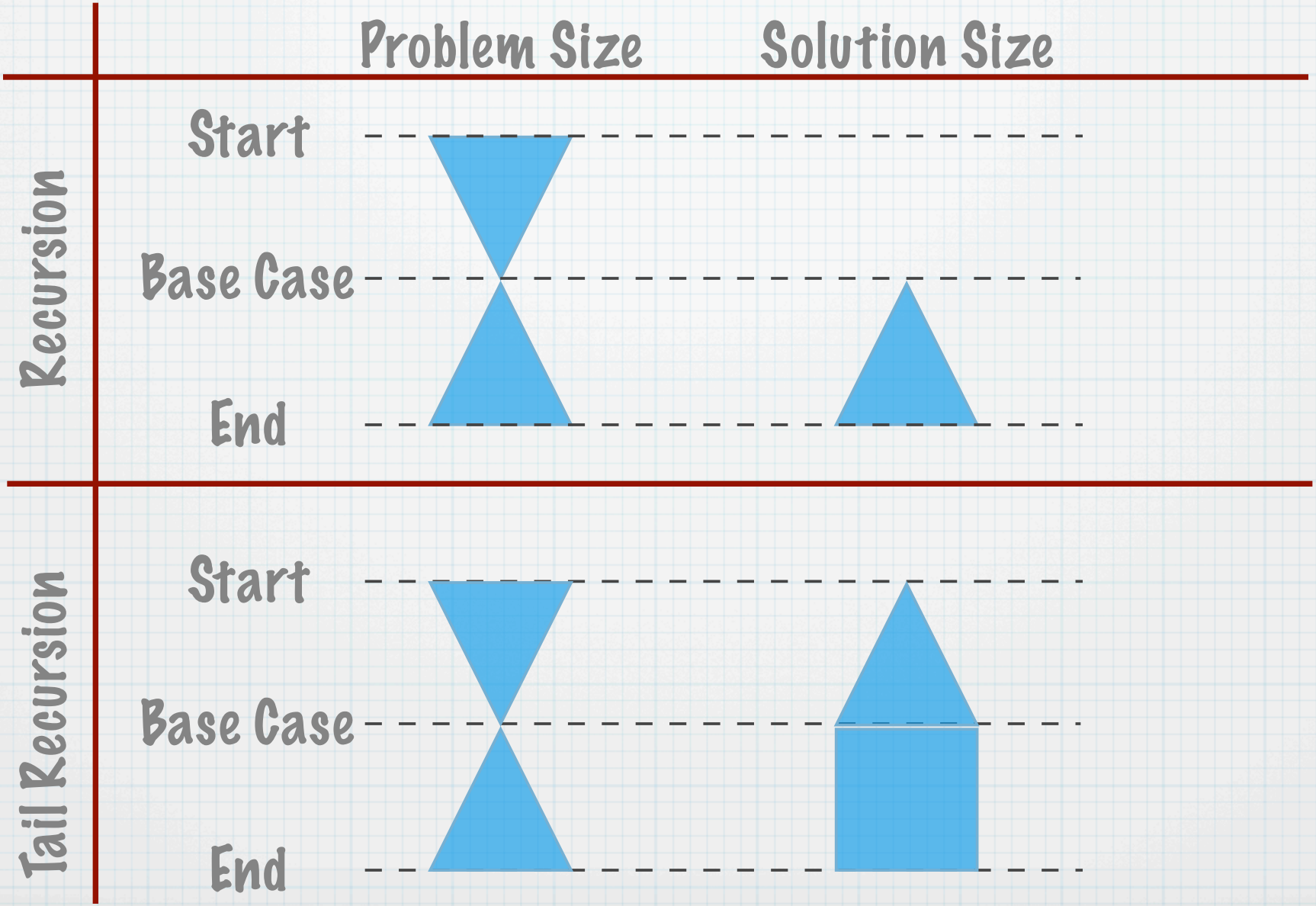
fact_help(2, 3) = fact_help(1, 6) = 6

fact_help(1, 6) = fact_help(0, 6) = 6

fact_help(0, 6) = 6



Recursion Comparison



Pin the Tail!

Consider recursive function foo, which of these statements are indicative of a tail recursive function?

✓
```
return foo(x-1);
```

```
return foo( x + foo(y) );
```

✓
```
return foo( bar(x-1, y) + bar(x, y-1) );
```

```
return foo(x-1) + bar(y);
```

```
return bar(y) + foo(x-1);
```

List Example

```
list_t sum_piecewise( list_t x, list_t y )
   // REQUIRES: x and y are the same length
   // EFFECTS: returns the sum of lists x and y
{
   if ( list_isEmpty(x) )
      return list_make();

   return list_make( list_first(x) + list_first(y),
                     sum_piecewise( list_rest(x), list_rest(y) );
}
```

List Example - Tail

```
static list_t sum_piecewise_help( list_t x, list_t y, list_t result )
   // REQUIRES: x and y are the same length
   // EFFECTS: returns the sum of lists x and y
{
   if ( list_isEmpty(x) )
      return reverse( result );

   return sum_piecewise_help( list_rest(x), list_rest(y),
            list_make( list_first(x) + list_first(y), result ));
}

list_t sum_piecewise_tail( list_t x, list_t y )
{
   return sum_piecewise_help( x, y, list_make() );
}
```

Outline

* Administrivia
* Tail Recursion
* Function Pointers

Code Reuse Discussion

* Reasons NOT to copy-paste code:
  * large code chunks can usually be extracted to functions
  * variable conflicts in destination
  * re-factoring difficulties

Motivating Example

Consider the following function:

```
int count_greater_than( list_t list, int n );
// EFFECTS: returns the number of elements in list
//          greater than n
```

What if we also wanted

- count_less_than
- count_prime
- count_ ...

Function Pointers

Consider the following solution:

```
int count_predicate( list_t list, bool (*fn)(int) )
// EFFECTS: returns the number of elements in list
//          for which fn() returns true
{
   int counter = 0;

   while ( !list_isEmpty( list ) )
   {
      if ( fn( list_first( list ) ) )
         counter++;

      list = list_rest( list );
   };

   return counter;
}
```

Potential Predicates

```
bool is80s( int n )
{
   return ( ( n >= 80 ) && ( n <= 89 ) );
}

bool isEven( int n )
{
   return ( ( n % 2 ) == 0 );
}

bool lessThan100( int n )
{
   return ( n < 100 );
}
```

Final Thoughts

- Assignment #2 due next Thursday
  - Work out functions by hand first
  - Give yourself time to think and sleep
- Additional function pointer discussion will be available in discussion notes via CTools later this week