CS 161
Intro to CS I
Finish Recursion
What is different when we call after?

- Recursive Solution

```cpp
void draw_rect(int i) {
    if(i>0){ //Base case
        cout << "******" << endl;
        cout << "*         *" << endl;
        cout << "******" << endl << endl;
        draw_rect(--i); //Recursive call
    }
}
```
Example: Factorial

• Definition

\[ 0! = 1; \]
\[ n! = n \times (n-1) \times \ldots \times (n-(n-1)) \times 1 = n \times (n-1)!; \quad n > 0 \]
Iterative Factorial

factorial(0) = 1;
factorial(n) = n*n-1*n-2*...*n-(n-1)*1;

long factorial(int n) {
    long fact;
    if(n==0 or n==1)
        fact=1;
    else
        for(fact=n; n > 1; n--)
            fact=fact*(n-1);
    return fact;
}
Recursive Factorial

factorial(0) = 1;

factorial(n) = n * factorial(n-1);

long factorial(int n) {
    if (n == 0) // Base case
        return 1;
    else
        return n * factorial(n-1); // Recursive call
}
Computing Factorial Iteratively

factorial(4)

\[ \text{factorial}(0) = 1; \]
\[ \text{factorial}(n) = n \times (n-1) \times \ldots \times 2 \times 1; \]
Computing Factorial Iteratively

factorial(4) = 4 * 3

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;
Computing Factorial Iteratively

factorial(4) = 4 * 3
= 12 * 2

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;
Computing Factorial Iteratively

factorial(4) = 4 * 3
    = 12 * 2
    = 24 * 1

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;
Computing Factorial Iteratively

factorial(4) = 4 * 3
    = 12 * 2
    = 24 * 1
    = 24

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;
Computing Factorial Recursively

factorial(4)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);
Computing Factorial Recursively

\[ \text{factorial}(4) = 4 \times \text{factorial}(3) = 4 \times (3 \times \text{factorial}(2)) \]

factorial(o) = 1;
factorial(n) = n*factorial(n-1);
Computing Factorial Recursively

\[
\begin{align*}
\text{factorial}(4) &= 4 \times \text{factorial}(3) \\
&= 4 \times (3 \times \text{factorial}(2)) \\
&= 4 \times (3 \times (2 \times \text{factorial}(1)))
\end{align*}
\]

factorial(0) = 1;  
factorial(n) = n \times \text{factorial}(n-1);
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)
  = 4 * ( 3 * factorial(2))
  = 4 * ( 3 * (2 * factorial(1)))
  = 4 * ( 3 * ( 2 * (1 * factorial(0))))
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * ( 3 * (2 * factorial(1)))
= 4 * ( 3 * ( 2 * (1 * factorial(0))))
= 4 * ( 3 * ( 2 * (1 *1)))

factorial(0) = 1;
factorial(n) = n*factorial(n-1);
Computing Factorial Recursively

\[
\text{factorial}(4) = 4 \times \text{factorial}(3) \\
= 4 \times (3 \times \text{factorial}(2)) \\
= 4 \times (3 \times (2 \times \text{factorial}(1))) \\
= 4 \times (3 \times (2 \times (1 \times \text{factorial}(0)))) \\
= 4 \times (3 \times (2 \times (1 \times 1))) \\
= 4 \times (3 \times (2 \times 1))
\]
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)
    = 4 * (3 * factorial(2))
    = 4 * ( 3 * (2 * factorial(1)))
    = 4 * ( 3 * ( 2 * (1 * factorial(0))))
    = 4 * ( 3 * ( 2 * 1))
    = 4 * (3 * 2)
Computing Factorial Recursively

\[
\text{factorial}(4) = 4 \times \text{factorial}(3) \\
= 4 \times (3 \times \text{factorial}(2)) \\
= 4 \times (3 \times (2 \times \text{factorial}(1))) \\
= 4 \times (3 \times (2 \times (1 \times \text{factorial}(0)))) \\
= 4 \times (3 \times (2 \times (1 \times 1))) \\
= 4 \times (3 \times (2 \times 1)) \\
= 4 \times (3 \times 2) \\
= 4 \times 6
\]
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)
    = 4 * (3 * factorial(2))
    = 4 * (3 * (2 * factorial(1)))
    = 4 * (3 * (2 * (1 * factorial(0))))
    = 4 * (3 * (2 * (1 * 1)))
    = 4 * (3 * (2 * 1))
    = 4 * (3 * 2)
    = 4 * 6
    = 24

factorial(0) = 1;
factorial(n) = n * factorial(n-1);
Differences

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory

needs to be shallow
don’t keep calling over & over w/ in
Recursive Factorial

factorial(4)

Executes factorial(4)

Step 0: executes factorial(4)
Step 1: executes factorial(3)
Step 2: executes factorial(2)
Step 3: executes factorial(1)
Step 4: executes factorial(0)
Step 5: return 1
Step 6: return 1
Step 7: return 2
Step 8: return 6
Step 9: return 24
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Executes factorial(3)

Step 1: executes factorial(3)

Step 2: executes factorial(2)

Step 3: executes factorial(1)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 4: executes factorial(0)
Recursive Factorial

factorial(4)

return 4 * factorial(3)

Step 0: executes factorial(4)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 2: executes factorial(2)

Step 3: executes factorial(1)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Executes factorial(2)

Stack

Space Required for factorial(3)
Space Required for factorial(4)
Main method
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 2: executes factorial(2)

return 2 * factorial(1)

Step 3: executes factorial(1)

Step 4: executes factorial(0)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Executes factorial(1)

Stack

Space Required for factorial(2)
Space Required for factorial(3)
Space Required for factorial(4)
Main method
Recursive Factorial

factorial(4)
  return 4 * factorial(3)
    return 3 * factorial(2)
      return 2 * factorial(1)
        return 1 * factorial(0)
          Executes factorial(o)

Step 0: executes factorial(4)
Step 1: executes factorial(3)
Step 2: executes factorial(2)
Step 3: executes factorial(1)
Step 4: executes factorial(0)
Step 5: return 1
Step 6: return 1
Step 7: return 2
Step 8: return 6

Stack
  Space Required for factorial(4)
  Space Required for factorial(3)
  Space Required for factorial(2)
  Space Required for factorial(1)
  Main method

OSU Oregon State University
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 2: executes factorial(2)

return 2 * factorial(1)

Step 3: executes factorial(1)

return 1 * factorial(0)

Step 4: executes factorial(0)

return 1

returns 1

Main method

Space Required for factorial(4)

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(1)

Space Required for factorial(0)

Main method
Recursive Factorial

Step 0: executes factorial(4)

return 4 * factorial(3)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 2: executes factorial(2)

return 2 * factorial(1)

Step 3: executes factorial(1)

return 1 * factorial(0)

Step 4: executes factorial(0)

return 1

Step 5: return 1

returns factorial(0)

Main method

Space Required for factorial(4)

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(1)

Space Required for factorial(0)

Stack

Main method
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 2: executes factorial(2)

return 2 * factorial(1)

Step 3: executes factorial(1)

return 1 * factorial(0)

Step 4: executes factorial(0)

returns factorial(1)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 9: return 24

Space Required for factorial(4)

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(1)

Main method

Stack
Recursive Factorial

Step 0: executes factorial(4)
Step 1: executes factorial(3)
Step 2: executes factorial(2)
Step 3: executes factorial(1)
Step 5: return 1
Step 6: return 1
Step 7: return 2

return 2 * factorial(1)
return 3 * factorial(2)
return 4 * factorial(3)

returns factorial(2)
Recursive Factorial

main method

Stack

Step 0: executes factorial(4)

Step 1: executes factorial(3)

Step 2: executes factorial(2)

Step 3: executes factorial(1)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

returns factorial(3)

factorial(4)

return 4 * factorial(3)

return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)

Space Required for factorial(4)
Recursive Factorial

Step 0: executes factorial(4)
Step 1: executes factorial(3)
Step 2: executes factorial(2)
Step 3: executes factorial(1)
Step 4: executes factorial(0)
Step 5: return 1
Step 6: return 1
Step 7: return 2
Step 8: return 6
Step 9: return 24
Helping Visualize Recursion

• [http://pythontutor.com/cpp.html](http://pythontutor.com/cpp.html)
  – C++ (experimental)
In-class Exercise

• Get into groups of 4 – 5.

• Write your own recursive `int pwr()` function that takes two integers as arguments and returns the integer result.
  – What does the function prototype look like?
  – Now, write the function definition...

```c
int pwr(int b, int e)
{
    int result = 1;
    for (int i = 0; i < e; i++)
    {
        result *= b;
    }
    return result;
}
```
```cpp
#include <iostream>

using namespace std;

long factorial(int n) {
    //cout << n << endl;
    if (n == 0)      // Base case
        return 1;
    else
        return n * factorial(n - 1);  // Recursive call
}

long factorial_iter(int n) {
    long f = 1;
    for (int i = 1; i <= n; i++)
        f *= i;
    return f;
}

int pwr(int b, int e) {
    if (e == 0)
        return 1;
    else
        return b * pwr(b, e - 1);
}

int main() {
    cout << pwr(2, 8) << endl;
    cout << factorial_iter(10000000) << endl; // gives 0 because of overflow
    cout << factorial(10000000) << endl; // seg faults
    return 0;
}
```