CS 161
Intro to CS I
Recursion
Recursion

• What is it?
  – Function that calls itself 1 or more times (directly or indirectly)
  – Has 1 or more base case for stopping
  – Inductive reasoning: general case must eventually be reduced to a base case
Example: Drawing Rectangles

• Iterative Solution:

```cpp
void draw_rect(int i) {
    for( ; i > 0; i--){
        cout << "*****" << endl;
        cout << "*   *" << endl;
        cout << "*****" << endl;
        cout << "*****" << endl;
    }
}
```
Example: Drawing Rectangles

- Recursive Solution

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

• Definition

\[0! = 1;\]

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

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

\[
\text{factorial}(0) = 1;
\]
Computing Factorial Iteratively

factorial(4) = 4 * 3

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

\[
\text{factorial}(4) = 4 \times 3 \\
= 12 \times 2
\]

\[
\text{factorial}(n) = n \times (n-1) \times ... \times 2 \times 1;
\]
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

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

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

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

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

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

\[ \text{factorial}(0) = 1; \]
\[ \text{factorial}(n) = n \times \text{factorial}(n-1); \]

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

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

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

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

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

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

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

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

\[
\text{factorial}(n) = \begin{cases} 
1 & \text{if } n = 0 \\
 n \times \text{factorial}(n-1) & \text{if } n > 0 
\end{cases}
\]

\[
\text{factorial}(4) = 4 \times \text{factorial}(3) \\
= 4 \times (3 \times \text{factorial}(2)) \\
= 4 \times (3 \times (2 \times \text{factorial}(1))) \\
= 4 \times (3 \times (2 \times (1 \times \text{factorial}(0)))) \\
= 4 \times (3 \times (2 \times (1 \times 1))) \\
= 4 \times (3 \times (2 \times 1)) \\
= 4 \times (3 \times 2) \\
= 4 \times 6 \\
= 24
\]
Differences

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
Recursive Factorial

Executes factorial(4)

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

Main method

Space Required for factorial(4)
Space Required for factorial(3)
Space Required for factorial(2)
Space Required for factorial(1)
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

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Space Required for factorial(4)

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(1)

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)  
return 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)

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)

Executes factorial(0)

Stack

Space Required for factorial(1)
Space Required for factorial(2)
Space Required for factorial(3)
Space Required for factorial(4)
Main method
factorial(4) 
return 4 \times \text{factorial}(3) 
return 3 \times \text{factorial}(2) 
return 2 \times \text{factorial}(1) 
return 1 \times \text{factorial}(0) 

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 4: executes factorial(0) 

Stack

\begin{array}{|c|}
\hline
\text{Space Required for factorial(0)} \\
\text{Space Required for factorial(1)} \\
\text{Space Required for factorial(2)} \\
\text{Space Required for factorial(3)} \\
\text{Main method} \\
\hline
\end{array}
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

Step 5: return 1

returns factorial(0)

Main method

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

return 4 * factorial(3)

return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)

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

returns factorial(1)
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)

Step 5: return 1
Step 6: return 1
Step 7: return 2
Step 8: return 6

Main method

3
Space Required for factorial(3)
Space Required for factorial(2)
Space Required for factorial(4)
Space Required for factorial(3)
Space Required for factorial(2)
Space Required for factorial(1)

Stack

OSU Oregon State University
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

return 1 * factorial(0)
return 2 * factorial(1)
return 3 * factorial(2)
return 4 * factorial(3)
returns factorial(3)
Recursive Factorial

Step 9: return 24

Step 8: return 6

Step 7: return 2

Step 6: return 1

Step 5: return 1

Step 4: executes factorial(0)

return 1

factorial(4)

return 4 * factorial(3)

return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)

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)

returns factorial(4)

Main method

3

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(4)

5

Stack

OSU Oregon State University
In-class Exercise #4

• Get into groups of 4 – 5.

• Write your own recursive \textit{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...