CS 161
Intro to CS I

Finish Pointers/Start Recursion
In-class Exercise #3
Understanding Pointers

• Create a pointer to a double, i.e. `double *d;` and three doubles `d1`, `d2`, and `d3` that get the values `7.8`, `10.0`, and `.009`.

• Now, set the pointer, `d`, to point to each double variable, `d1`, `d2`, and `d3`, printing the address and contents of each double variable along the way.
In-class Exercise #3

Understanding Pointers

• What if you made a pointer that points to a pointer to a double, i.e. `double **dp`? Now, set `dp` to point to `d`, and use `dp` to print the address and contents of each double variable!!!
Demo...
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;
    }
}
```
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 << endl;
  }
}
```
What is different when we call after?

- Recursive Solution

```cpp
void draw_rect(int i) {
    if(i>0){
        //Base case
        cout << "******" << endl;
        cout << "*   *

        cout << "******"
        cout << "******" << endl << endl;
        draw_rect(--i); //Recursive call
    }
}
```
Example: Factorial

• Definition

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

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

factorial(4) = 4 * 3

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

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

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

\[
\text{factorial}(4) = 4 \times 3 \\
= 12 \times 2 \\
= 24 \times 1
\]

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

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

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

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

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

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

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

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

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

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

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

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

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

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
Recursive Factorial

factorial(4) → factorial(3) → factorial(2) → factorial(1) → factorial(0) → return 24

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

Executes factorial(4)
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Executes factorial(3)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 4: executes factorial(0)

Stack

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)

Stack

Executes factorial(2)

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

Step 9: return 24

Executes factorial(1)

Stack

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

OSU Oregon State University
Main method

Space Required for factorial(4)
Space Required for factorial(3)
Space Required for factorial(2)
Space Required for factorial(1)
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 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6
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)

returns 1

Stack

<table>
<thead>
<tr>
<th>Space Required</th>
<th>for factorial(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Required</td>
<td>for factorial(1)</td>
</tr>
<tr>
<td>Space Required</td>
<td>for factorial(2)</td>
</tr>
<tr>
<td>Space Required</td>
<td>for factorial(3)</td>
</tr>
<tr>
<td>Space Required</td>
<td>for factorial(4)</td>
</tr>
</tbody>
</table>

Main method
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 \times \text{factorial}(3)

Step 1: executes factorial(3)

return 3 \times \text{factorial}(2)

Step 2: executes factorial(2)

return 2 \times \text{factorial}(1)

Step 3: executes factorial(1)

return 1 \times \text{factorial}(0)

Step 5: return 1

Step 4: executes factorial(0)

returns factorial(0)

Main method

<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Required for factorial(1)</td>
</tr>
<tr>
<td>Space Required for factorial(2)</td>
</tr>
<tr>
<td>Space Required for factorial(3)</td>
</tr>
<tr>
<td>Space Required for factorial(4)</td>
</tr>
<tr>
<td>Main method</td>
</tr>
</tbody>
</table>
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

Space Required for factorial(1)

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

Step 5: return 1

Step 6: return 1

Step 7: return 2

returns factorial(2)
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(3)

Step 5: return 1

Step 6: return 2

Step 7: return 3

Step 8: return 6

Main method

Step 0: executes factorial(4)

Space Required for factorial(4)

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

Step 9: return 24

returns factorial(4)