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

• Definition

0! = 1;

n! = n * (n-1) * … * (n-(n-1)) * 1 = n * (n-1)! ; n > 0
Iterative Factorial

factorial(0) = 1;
factorial(n) = n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot n-(n-1)\cdot1;

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

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

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

Computing Factorial Iteratively

factorial(4)

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

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

\[
\begin{align*}
\text{factorial}(0) &= 1; \\
\text{factorial}(n) &= n \times (n-1) \times \ldots \times 2 \times 1;
\end{align*}
\]
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

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

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))))

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))))

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

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

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

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

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

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

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

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
Recursive Factorial

Factorial(4) → Factorial(3) → Factorial(2) → Factorial(1) → Factorial(0)

Steps:
- 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

```
return factorial(4)
```

```
return 4 * factorial(3)
```

```
return 3 * factorial(2)
```

```
return 2 * factorial(1)
```

```
return 1 * factorial(0)
```

Step 0: executes \texttt{factorial}(4)

Step 1: executes \texttt{factorial}(3)

Step 2: executes \texttt{factorial}(2)

Step 3: executes \texttt{factorial}(1)

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)

Step 1: executes factorial(3)

return 3 * factorial(2)

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(2)
Recursive Factorial

factorial(4)

return 4 * factorial(3)

Step 0: executes factorial(4)

Executes factorial(1)

return 3 * factorial(2)

Step 1: executes factorial(3)

return 2 * factorial(1)

Step 2: executes factorial(2)

Stack

Space Required for 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)

return 1 * 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

Executes factorial(0)
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 \times factorial(3)

Step 1: executes factorial(3)

return 3 \times factorial(2)

Step 2: executes factorial(2)

return 2 \times factorial(1)

Step 3: executes factorial(1)

return 1 \times factorial(0)

Step 4: executes factorial(0)

return 1

Space Required
for factorial(0)

Space Required
for factorial(1)

Space Required
for 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)

return 1 * factorial(0)

Step 5: return 1

Step 4: executes factorial(0)

returns factorial(0)

Stack

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

OSU Oregon State University
Recursive Factorial

(factorial(4) →
return 4 * factorial(3)

Step 0: executes factorial(4)

(factorial(3) →
return 3 * factorial(2)

Step 1: executes factorial(3)

(factorial(2) →
return 2 * factorial(1)

Step 2: executes factorial(2)

(factorial(1) →
return 1 * factorial(0)

Step 3: executes factorial(1)

(factorial(0) →
return 1

Step 4: executes factorial(0)

Step 5: return 1

Step 6: return 1

Step 7: return 2

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

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

returns factorial(3)
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 factorial(0)

returns factorial(1)

returns factorial(2)

returns factorial(3)

returns factorial(4)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 9: return 24
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...