CS 161
Intro to CS I

Recursion
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

recursive
Iterative Factorial

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

factorial(4)

def factorial(o) = 1;
def factorial(n) = n*(n-1)*...*2*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

\[
\text{factorial}(4) \\
\text{factorial}(0) = 1; \\
\text{factorial}(n) = n \times \text{factorial}(n-1);
\]
Computing Factorial Recursively

factorial(4) = 4 * factorial(3)

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

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

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

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

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

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

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

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

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

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

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
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
Differences

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory

\[\text{not a reason to avoid recursion}\]
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)

Main method

Stack

Space Required for factorial(4)

Main method
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 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)

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)

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

Stack
Recursive Factorial

factorial(4)
  \[ \text{Step 0: executes factorial(4)} \]
  \[ \text{return 4 * factorial(3)} \]
  \[ \text{Step 1: executes factorial(3)} \]
  \[ \text{return 3 * factorial(2)} \]
  \[ \text{Step 2: executes factorial(2)} \]
  \[ \text{return 2 * factorial(1)} \]
  \[ \text{Step 3: executes factorial(1)} \]
  \[ \text{return 1 * factorial(0)} \]

Executes factorial(0)

Stack

<table>
<thead>
<tr>
<th>Space Required for factorial(0)</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>Main method</td>
</tr>
</tbody>
</table>

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

returns 1

---

Space Required for factorial(0)
Space Required for factorial(1)
Space Required for factorial(2)
Space Required for factorial(3)
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)

Step 5: return 1

returns factorial(0)

Space Required for factorial(4)

Space Required for factorial(3)

Space Required for factorial(2)

Space Required for factorial(1)

Stack

Main method

OSU Oregon State University
Recursive Factorial

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

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

returns factorial(2)
Recursive Factorial

Main method

Space Required for factorial(4)

Stack

5
4
3
2
1

factorial(4)

return 4 * factorial(3)

Step 0: executes factorial(4)

return 3 * factorial(2)

Step 1: executes factorial(3)

return 2 * factorial(1)

Step 2: executes factorial(2)

return 1 * factorial(0)

Step 3: executes factorial(1)

Step 4: executes factorial(0)

return 1

Step 5: return 1

Step 6: return 2

Step 7: return 6

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)
Step 5: return 1
Step 6: return 1
Step 7: return 2
Step 8: return 6
Step 9: return 24

returns factorial(4)
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...
```cpp
#include <iostream>

using namespace std;

int power(int base, int exponent) {
    int p = 1;
    for(int i = 0; i < exponent; i++)
        p *= base;
    return p;
}

int pwr(int base, int exponent) {
    // base
    if(exponent == 0)
        return 1;
    else
        return base * pwr(base, exponent-1);
}

int main() {
    int base, exp;
    cout << "Enter base and exponent, ex. 2 4: " << endl;
    cin >> base;
    cin >> exp;
    cout << power(base, exp) << endl;
    cout << pwr(base, exp) << endl;
    return 0;
}
```
In-class Exercise

Pointers vs. References

• What if you made a pointer (p2) that points to a pointer (p) to an int (x)?
  – What would the picture look like?
  – Write the code for this picture.

• Can you make this same picture for references?
  – What if you had two references, r and r2?

---

OSU Oregon State University