CS 161
Intro to CS I

Finish Recursion/Begin Memory Model
Odds and Ends

• Assignment 5 posted
• Assignment 4 demo this week
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

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)

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

factorial(4) = 4 * 3
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

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

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

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

factorial(4)
Computing Factorial Recursively

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

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

factorial(4) = 4 \times factorial(3)
= 4 \times (3 \times factorial(2))
Computing Factorial Recursively

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

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

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

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(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))
Computing Factorial Recursively

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

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

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

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
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)

Step 1: executes factorial(3)

return 3 * 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)

Space Required for factorial(4)

Space Required for factorial(3)

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)

Step 4: executes factorial(0)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Executes 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)

Executes factorial(0)

Main method

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

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

Stack

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 4: executes factorial(0)

Step 5: return 1

returns factorial(0)
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

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)
returns factorial(2)
Step 5: return 1
Step 6: return 1
Step 7: return 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)

return 1 * factorial(0)

Step 4: executes factorial(0)

returns factorial(3)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 9: return 24
Recursive Factorial

Main method

Stack

returns 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

factorial(4)

return 4 * factorial(3)

return 3 * factorial(2)

return 2 * factorial(1)

return 1 * factorial(0)
In-class Exercise #4

• 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...
Stack vs. Heap

• Static vs. Dynamic
Static vs. Dynamic

• Static Semantics
  – Assign address of variable
    int *i, j=2;
    i=&j;

• Dynamic Semantics
  – Create memory
  – Assign memory to pointer
    int *i=NULL;
    i=new int;
    *i=2;
What About Memory Leaks?

• What happens here...

... 
int main () {
    int *i=NULL;  //created in main function
    while(1) {
        i = new int;
    }
}
Fixing Memory Leaks...

• What happens here...

...  
int main () {  
    int *i=NULL; //created in main function  
    while(1) {  
        i = new int;  
        delete i; //free memory that i points to, preventing mem leaks  
    }  
}
Dynamic Memory Demo...