CS 161
Intro to CS I

Finish Pointers/Start Recursion
Odds and Ends

• Last week to demo Assignment 3!!
• Assignment 4 questions
#include <iostream>
#include <string>
using namespace std;

void fun(string *s, string *t) {
    cin >> *s;
    cin >> *t;
}

void fun(string &s, string &t) {
    cin >> s;
    cin >> t;
}

int main() {
    string str, str1;

    fun(&str, &str1);
    cout << str << " " << str1;

    fun(str, str1);
    cout << str << " " << str1;

    return 0;
}
More Understanding Pointers

• What if you made a pointer that points to a pointer to an int, i.e. `int **p2`? Now, set p2 to point to p, and use p2 to print the address and contents of the int s 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;
        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;
    }
}
```
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 ... \times (n-(n-1)) \times 1 = n \times (n-1)! ; n > 0
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;
}

OSU Oregon State University
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 \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

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

factorial(4)
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

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

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

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

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

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

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

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

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

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

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

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

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
Recursive Factorial

factorial(4)

Executes 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

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

return 1

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 9: return 24
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)`
  - Returns: `4 * factorial(3)`
- **Step 1:** executes `factorial(3)`
  - Returns: `3 * factorial(2)`
- **Step 2:** executes `factorial(2)`
  - Returns: `2 * factorial(1)`
- **Step 3:** executes `factorial(1)`
  - Returns: `1 * factorial(0)`
- **Step 4:** executes `factorial(0)`
  - Returns: 1

**Stack**

- Main method
- Space Required for `factorial(4)`: 4
- Space Required for `factorial(3)`: 5
- Space Required for `factorial(2)`: 3
- Space Required for `factorial(1)`: 1

**Recursive Factorial**

```
Stack
```

**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)

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

Executes factorial(o)

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)

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)
Main method

Main method

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

return 1

Step 5: return 1

returns factorial(0)
Recursive Factorial

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

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)  
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  
Step 7: return 2

returns factorial(2)

Main method

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

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

return 2 * factorial(1)

return 3 * factorial(2)

Step 2: executes factorial(2)

Step 1: executes factorial(3)

return 4 * factorial(3)

Step 0: executes factorial(4)

return 24
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

return 1 * factorial(0)

return 2 * factorial(1)

return 3 * factorial(2)

return 4 * factorial(3)

Step 0: executes factorial(4)

returns factorial(4)