CS 161
Intro to CS I

Recursion
Odds and Ends

- Assignment 4 due Sunday
- Questions?

```c
#include <cstdlib>

int x = atoi(s.c_str());

double y = atof(s.c_str());
```
Exercise Pointers vs. References

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

```c
int *p2, *p, x = 50;
```

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

```c
p = &x;
p2 = &p;
cout << p << 150;
cout << p2 << 150;
```
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;
    }
}
```

`draw_rect(4);`
Example: Drawing Rectangles

• Recursive Solution

```c
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
  
  \[ 0! = 1; \]
  
  \[ n! = n \times (n-1) \times \ldots \times (n-(n-1)) \times 1 = n \times (n-1)! \quad ; \quad n > 0 \]
Iterative Factorial

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

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

factorial(4) = 4 * 3

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

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

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

factorial(4) = 4 * factorial(3)
factorial(0) = 1;  
factorial(n) = n*factorial(n-1);

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

factorial(0) = 1;
factorial(n) = n*factorial(n-1);
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)))
= 4 \times (3 \times (2 \times 1))
= 4 \times (3 \times 2)

factorial(0) = 1;
factorial(n) = n \times 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)))
= 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)
= 4 * 6
= 24

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

• Pros
  – Readability

• Cons
  – Efficiency
  – Memory
Recursive Factorial

Executes factorial(4)

factorial(4)

Stack

Main method
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
Recursive Factorial

factorial(4)

Step 0: executes factorial(4)

return 4 * factorial(3)

Step 1: executes factorial(3)

return 3 * factorial(2)

Step 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Executes factorial(2)

Stack

Space Required for factorial(3)

Space Required for factorial(4)

Main method
Recursive Factorial

factorial(4)
  return 4 * factorial(3)
    Step 1: executes factorial(2)
    return 3 * factorial(2)
      Step 2: executes factorial(1)
      return 2 * factorial(1)
      return 1 * factorial(0)
      return 1

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

Executes factorial(1)

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

Executes factorial(0)

Stack

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)
return 1

returns 1

Stack

Stack Required for factorial(0)
Stack Required for factorial(1)
Stack Required for factorial(2)
Stack Required for factorial(3)
Stack Required for factorial(4)
Main method
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 5: return 1
Step 4: executes factorial(0)
returns 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)

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

returns factorial(2)
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

Main method
Space Required for factorial(4)
Stack
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)