CS 161
Intro to CS I

Finish Pointers/Start Recursion
In-class Exercise #3
Understanding Pointers

• Create a pointer to a double, i.e. `double *d;` and three doubles d1, d2, and d3 that get the values 7.8, 10.0, and .009.

• Now, set the pointer, d, to point to each double variable, d1, d2, and d3, printing the address and contents of each double variable along the way.
double *d has garbage, until set

```cpp
#include <iostream>
#include <cstdlib>  // gives you NULL pointer
using namespace std;

int main()
{
    double *d, d1=.78, d2=1.2;
    cout << d << endl; // what garbage is in d, if we don't set to NULL
    // how do you make a pointer point to something
    // common mistakes:
    // *d=d1;  // (core dump because you try to go to bad address NULL)
    // *d=&d1 (compiler error because trying to put double * in double)
    //  d=d1 (compiler error because trying to put double in double *)
    d=&d1;  // always fill contents before you go somewhere
    cout << &d << endl; // where d lives
    cout << &d1 << endl; // where d1 lives
    cout << d << endl; // contents of d is d1's address
    cout << *d << endl; // take me to the address in my contents, dereference
    cout << d1 << endl; // print d1's contents
    return 0;
}
```

In-class Exercise #3

Understanding Pointers

• What if you made a pointer that points to a pointer to a double, i.e. `double**dp`? Now, set `dp` to point to `d`, and use `dp` to print the address and contents of each double variable!!!
```cpp
#include <iostream>
#include <cstdlib> // gives you NULL pointer

using namespace std;

int main()
{
    double *d= NULL, d1=.78, d2=1.2;
    double **dp= NULL;

    // how do you make a pointer point to something
    // common mistakes:
    // *d=d1; //(core dump because you try to go to bad address NULL)
    // *d=&d1 (compiler error because trying to put double * in double)
    // d=d1 (compiler error because trying to put double in double *)
    d=&d1; // always fill contents before you go somewhere

    cout << &d << endl; // where d lives
    cout << &d1 << endl; // where d1 lives
    cout << d << endl; // contents of d is d1's address
    cout << *d << endl; // take me to the address in my contents, dereference
    cout << d1 << endl; // print d1's contents

    dp=&d;
    cout << "contents of d using dp: " << *dp << endl;
    cout << "contents of d1 using dp: " << **dp << endl;

    return 0;
}
```
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 << 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;

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

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

```c
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
}

BAD!!!
Computing Factorial Iteratively

factorial(4)

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

\[
factorial(4) = 4 \times 3
\]
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

factorial(4) = 4 \times 3
= 12 \times 2
= 24 \times 1
= 24
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

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

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

factorial(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(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

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(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)

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))))
  = 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))
  = 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

- 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

Executes \texttt{factorial(3)}
Recursive Factorial

factorial(4) -> 
  return 4 * factorial(3) 

  factorial(3) -> 
    return 3 * factorial(2) 

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

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(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

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)

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)

Stack

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)

return 1

returns 1

Main method

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

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 5: return 1
Step 6: return 1
Step 4: executes factorial(0)
Step 7: return 2
Step 8: return 6
Step 9: return 24

returns factorial(1)

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

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 2

Step 7: return 2

Step 8: return 6
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

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)

OSU Oregon State University