CS 161
Introduction to CS I
Lecture 21

• Recursion
Assignment 5 – Treasure Chest

• Define your own struct (type) with at least 4 attributes
• Create a program to store items of that type in a treasure chest and keep track of the total collection value
• User can:
  • Add item
  • Remove item
  • Display item
  • Add an item with random properties
  • <Your choice here>

No live demo
(README.txt instead)
Midterm 2

• Midterm 2: content through **week 7** (but no structs)
• Review questions (and solutions) are on course website
• Bring your questions to class on Wednesday
  • Stuck on pointers? Functions? 2D arrays?
• Review session: Thursday 2/27, 6-7 p.m., **LINC 228**
• Midterm: Friday 2/28, 2-2:50 p.m., **LINC 100**
• Format: true/false, multiple choice, one page short answer
  • Scantron sheet: fill in bubbles with #2 pencil
• Bring to midterm: **student ID and #2 pencil(s)**
Let's calculate factorials

- Mathematical definition
  
  \[ 
  0! := 1; 
  \]

  \[ 
  n! := n \times (n-1) \times \ldots \times 1 
  \]

  \[ 
  := n \times (n-1)! \quad \text{if } n > 0 
  \]
Iterative factorial

Iterative definition:

\[
\text{factorial}(0) := 1; \\
\text{factorial}(n) := n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1;
\]

1. int factorial(int n) {
2.     int fact;
3.     if (n==0)
4.         fact = 1;
5.     else
6.         for (fact=n; n > 1; n--)
7.             fact = fact * (n-1);
8.     return fact;
9. }

See lec21-factorial-iterative.cpp
Computing Factorial Iteratively

factorial(4) = 4 * 3
   = 12 * 2
   = 24 * 1
   = 24

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;
Recursion

• A recursive definition includes a mention of itself
  • "My descendants are my children + all of my children's descendants."
  • "My keys are located in this room or in some other room."
• A recursive function includes [at least one] call to itself
  • Base case: when to stop (simplest case)
  • Recursive step: a general statement that reduces the task (eventually) to a base case
Recursive Factorial

Recursive definition:
Base case: \( \text{factorial}(0) = 1; \)
Recursive step: \( \text{factorial}(n) = n \times \text{factorial}(n-1); \)

```c
int factorial(int n) {
    if (n == 0) /* Base case */
        return 1;
    else /* recursive call */
        return n * factorial(n - 1);
}
```

See lec21-factorial-recursive.cpp
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

factorial(4)

Executes factorial(4)

Stack

Space for factorial(4)

Space for main()
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

Stack

Space for factorial(4)
Space for factorial(3)
Space for main()
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)

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)

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 5: return 1

Step 6: return 1

Step 7: return 2

Step 8: return 6

Step 9: return 24

Executes factorial(1)

Space for factorial(1)

Space for factorial(2)

Space for factorial(3)

Space for factorial(4)

Space for main()
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 4: executes factorial(0)

Recursive Factorial

Executes factorial(0)
Stack
Space for factorial(0)
Space for factorial(1)
Space for factorial(2)
Space for factorial(3)
Space for factorial(4)
Space for main()
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)

return 1

returns 1

2/24/2020
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(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 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)

Stack

Space for factorial(2)
Space for factorial(3)
Space for factorial(4)
Space for main()
Recursive Factorial

return 1
factorial(4)
return 4 * factorial(3)
return 3 * factorial(2)
return 2 * factorial(1)
return 1 * factorial(0)
Step 5: return 1
Step 6: return 1
Step 7: return 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)
returns factorial(2)

Stack

Space for factorial(3)
Space for factorial(4)
Space for main()
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

returns factorial(3)

factorial(4)
return 4 \times \text{factorial}(3)
return 3 \times \text{factorial}(2)
return 2 \times \text{factorial}(1)
return 1 \times \text{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

returns factorial(4)
When is recursion useful?

- Problems that have a "nested" or recursive structure and would be hard to write in an iterative fashion
  - Water in a river is the sum of the water from each tributary
- Recursion breaks the problem into one small step + "the rest of the solution"
Your turn: descendants

• "My number of descendants is my number of children + the sum of my children's descendants."
  • What is the base case?
  • What is the recursive step?
Your turn: descendants

• "My number of descendants is my number of children + the sum of my children's descendants."
  
  • What is the base case?
    • No children: \#descendants = 0
  
  • What is the recursive step?
    • \#descendants = \#children + \#descendants(child1) + \#descendants(child2) + ...
  
  • This would be quite difficult to do in an iterative way!
Your turn: exponents

• Compute $\text{base}^{\text{exp}}$ in a recursive function
  • What is the function prototype?
  • What is the base case?
  • What is the recursive step?
Your turn: exponents

• Compute \(\text{base}^\text{exp}\) in a recursive function called \(\text{pwr}()\)
  • What is the function prototype?
    • \(\text{int pwr(int base, int exp);}\)
  • What is the base case?

• What is the recursive step?
Your turn: exponents

• Compute \texttt{base}^{\texttt{exp}} in a recursive function called \texttt{pwr()}
  • What is the function prototype?
    • \texttt{int pwr(int base, int exp);} 
  • What is the base case?
    • \texttt{exp = 0: return 1} 
  • What is the recursive step?
    • \texttt{exp > 0: return base * pwr(base, exp-1)}
Exponent implementation

1. int pwr(int base, int exp) {
2.   if (exp == 0) /* base case */
3.     return 1;
4.   else
5.     /* recursive call */
6.     return base * pwr(base, exp - 1);
7. }

See lec21-power-recursive.cpp
Gotchas

- Failure to specify base case => stack overflow
- Failure to reach base case => stack overflow
  - Problem doesn't get smaller

```c
int myfun(int n) {
    if (n == 0)
        return 0;
    else
        return myfun(n);
}
```
What vocabulary did we learn today?

- Recursion
- Base case
- Recursive step
What ideas and skills did we learn today?

• How to design solutions with recursive definitions
• How to translate a recursive definition into a recursive function
• Merits of iteration versus recursion
Week 8 begins!

- Attend lab (laptop required)
- Read Rao lesson 7 (pp. 158-161)
  Read Miller lecture 8: http://www.doc.ic.ac.uk/~wjk/C++Intro/RobMillerL8.html
- Start on design for Assignment 5 (due Sunday, March 1)

See you Wednesday (midterm review)!

- Bring your questions about material from weeks 1-7