Chapter 13
Recursion

FORDHAM

CISC 5004 - Programming C++
Sam Kamens
Fall, 2022

Recursion: Introduction

- Algorithm
  - Sequence of steps required to solve a problem
- Recursive Algorithm
  - Breaks the problem into smaller subproblems, and applies the same algorithm to the smaller problems
  - Requires a base case

Recursive Functions

- A function that calls itself is a **recursive function**
  - Could be indirect (e.g. `f() -> g() -> f()`)
- Example: CountDown

Recursive Binary Search

- The algorithm
- Guessing Game
- Searching a sorted list

Debugging recursive functions

- Add output statements
  - Indent the statements to show the depth of recursion
  - Example: Find String Match
  - Hands-on example

Creating a Recursive Function

1. Write the base case
    - There can be > 1 base case
2. Write the recursive case

- Example: Factorial
- Common Errors
    - Not covering all base cases
    - Not reaching a base case
    - Results: Infinite recursion
- Common pattern: function plus helper

Is recursion an appropriate choice?

- Is there a natural recursive solution?
- Is the recursive solution actually better?

Coding Example

- Prime Number Checker

Recursive Math Functions

- Recursive Fibonacci
- Greatest Common Divisor

Exploration of all possibilities

- Word Scramble viewed as a tree
- Recursive Implementation
- Shopping Spree
- Traveling Salesman

Stack Overflow

- Each function call adds a new **stack frame** on the stack
- If the recursion is too deep (i.e. too many function calls), the stack may grow too large for its allocated memory
  - This is known as **stack overflow** and will likely cause the program to crash
  - This is often an indication of unintended infinite recursion
- Stack size can be increased in some cases
- Otherwise, algorithm must be changed to avoid the overflow
- One possible solution: Tail Recursion
  - Recursive call is the last call in the function
  - May be optimized by the compiler

C++ Examples/Labs

- Recursively Output Permutations
- Number Pattern