Recursion

Binary search example postponed to end of lecture

What Is Recursion?

- Recursive call
  - A method call in which the method being called is the same as the one making the call
- Direct recursion
  - Recursion in which a method directly calls itself
- Indirect recursion
  - Recursion in which a chain of two or more method calls returns to the method that originated the chain
Writing a recursive function to find $n!$

- **Factorial**($n$) = $n \times (n-1) \times (n-2) \cdots 1$
  - Factorial(4) = $4 \times 3 \times 2 \times 1 = 24$
  - Factorial(0) = 1

- **Another definition** of Factorial($n$)
  
  \[
  \begin{align*}
  &= n \times \text{Factorial}(n-1) \quad \text{if } n > 0 \\
  &= 1 \quad \text{if } n = 0
  \end{align*}
  \]

Recursive function

```c
int fact(int n) {
    if (n==0)
        return 1;
    return n * fact(n-1);
}
```

Recursion Versus Iteration

- Recursion not always 'necessary'
  - Not even allowed in some languages
- Any task accomplished with recursion can also be done without it
  - Nonrecursive: called iterative, using loops
- Recursive
  - Runs slower, uses more storage
  - Elegant solution; less coding
Some Definitions

- **Base case**
  - The case for which the solution can be stated nonrecursively

- **General (recursive) case**
  - The case for which the solution is expressed in terms of a smaller version of itself

- **Recursive algorithm**
  - A solution that is expressed in terms of (a) smaller instances of itself and (b) a base case

General format for many recursive functions

```
if (some condition for which answer is known)
  solution statement             //base case
else
  recursive function call        //general case
```

Recursive function

```
int fact(int n)
{
  if (n==0)
    return 1;
  return n*fact(n-1);
}
```

- **Call itself**
- **Fact(0)=1;**
- **Parameter to recursive call is diminished by 1**
  - Always reach base case if \( n \geq 0 \)
Box Method

- Activation record
  - Arguments
  - Local variables
  - A place holder for value returned by recursive calls
  - The return value of the function itself

Activation record for Fact(3)
Three-Question Method of verifying recursive functions

- **Base-Case Question:** Is there a nonrecursive way out of the function?
- **Smaller-Caller Question:** Does each recursive function call involve a smaller case of the original problem leading to the base case?
- **General-Case Question:** Assuming each recursive call works correctly, does the whole function work correctly?
Example Using Recursion: Fibonacci Series

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21 ...
  - Each number sum of two previous ones
- $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

```c
long fibonacci( long n )
{
    if ( n == 0 || n == 1 )
        return n;
    else
        return fibonacci( n - 1 ) + fibonacci( n - 2 );
}
```

Example Using Recursion: Fibonacci Series

$f( 3 )$
$f( 1 )$
$f( 2 )$
$f( 1 )$
$f( 0 )$
$f( 0 )$
return 1
return 1
return 0
Stacks for Recursion

• A stack
  - Specialized memory structure
  - Like stack of paper
    - place new on top
    - remove when needed from top
  - Called ‘last-in/first-out’ memory structure
• Recursion uses stacks
  - Each recursive call placed on stack
  - When one completes, last call is removed from stack

Stack Overflow

• Size of stack limited
• Long chain of recursive calls continually adds to stack
  - All are added before base case causes removals
• If stack attempts to grow beyond limit
  - Stack overflow error
• Infinite recursion always causes this

Binary Search

• Recursive function to search sorted array
  - Determines if item is in array, and if so:
    - Where in array it is
• Breaks array in half
  - Determines if item in 1st or 2nd half
  - Then searches again just that half
Checking the Recursion

- No infinite recursion:
  - Each call increases first or decreases last
  - Eventually first will be greater than last
- Stopping cases perform correct action:
  - If first > last → no elements between them
  - If key == a[mid] → correctly found!
- Recursive calls perform correct action
  - If key < a[mid] → key in 1st half - correct call
  - If key > a[mid] → key in 2nd half - correct call

Execution of Binary Search
Efficiency of Binary Search

- Extremely fast
  - Compared with sequential search
- Half of array eliminated at start!
  - Then a quarter, then 1/8, etc.
  - Essentially eliminate half with each call
- For an array of 1M elements
  - Binary search never needs more than 20 compares!
  - Logarithmic efficiency $O(\log n)$

Summary

- Reduce problem into smaller instances of same problem -> recursive solution
- Recursive algorithm has two cases:
  - Base/stopping case
  - Recursive case
- Ensure no infinite recursion
- Use criteria to determine recursion correct
  - Three essential properties