Recursion

- Often phrased as “a function calls itself”
  - function actually calls a clone, not itself
  - each clone has its own variables/parameters/state
  - when a clone is finished, execution continues with the function that called the clone (may be a …)

- Recursion is useful in solving lots of problems, but should be avoided on occasion
  - lots of function calls, clones made
  - total # of clones called vs # in existence at one time
  - see Fibonacci (bad) and logarithmic power (good)
Rules of recursion

- There must be a base case, no recursive calls
- Each call must make progress towards the base case
  - call is similar, but simpler

```c
int CountNodes(Node * list)
{
    if (list == 0) return 0;
    else return 1 + CountNodes(list->next);
}
```

- Make the recursive call, use the result to determine what to do/what to return
  - something has happened as result of recursion
Reading words (yet again)

Node * ReadWords(istream & input)
//pre: input readable
//post: returns linked list of words in input, in order
{
    string word;
    if (input >> word)
    {
        return new Node(word, ReadWords(input));
    }
    return 0;    // make sure list is terminated!
}

● **Node has a constructor, what does it look like?**
  ➤ **how many parameters?**
  ➤ **default values?**
Recursion and Reading

- What to return when no words left?
- Each call returns a complete list

```cpp
struct Node {
    string info;
    Node * next;
    Node(const string & s, 
         Node * link = 0) 
        : info(s), 
          next(link) 
    {  
    }
};
```