COMP 202 – Week 13

- **Recursion is a fundamental programming technique that can provide an elegant solution certain kinds of problems**

Recursive Thinking

- **A *recursive definition* is one which uses the word or concept being defined in the definition itself**
- **When defining an English word, a recursive definition is often not helpful**
- **But in other situations, a recursive definition can be an appropriate way to express a concept**
- **Before applying recursion to programming, it is best to practice thinking recursively**

Recursive Definitions

- Consider the following list of numbers:

```
24, 88, 40, 37
```

- Such a list can be defined as

```
A LIST is a:  number
       or a:  number  comma  LIST
```

- That is, a LIST is defined to be a single number, or a number followed by a comma followed by a LIST
- The concept of a LIST is used to define itself

Recursive Definitions

- The recursive part of the LIST definition is used several times, terminating with the non-recursive part:

```
number comma LIST
  24     ,   88, 40, 37

             number comma LIST
               88     ,   40, 37

                          number comma LIST
                            40     ,   37

                                       number
                                         37
```

Infinite Recursion

- All recursive definitions have to have a non-recursive part
- If they didn't, there would be no way to terminate the recursive path
- Such a definition would cause *infinite recursion*
- This problem is similar to an infinite loop, but the non-terminating "loop" is part of the definition itself
- The non-recursive part is often called the *base case*

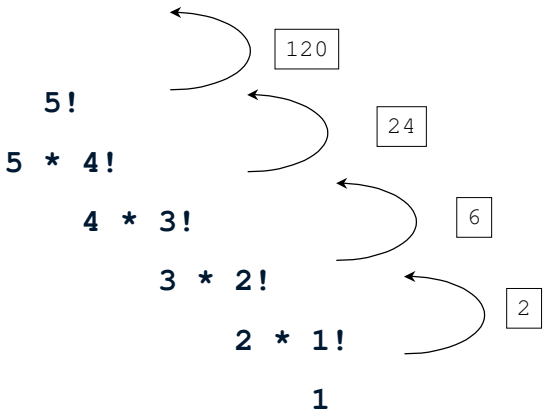
Recursive Definitions

- N!, for any positive integer N, is defined to be the product of all integers between 1 and N inclusive
- This definition can be expressed recursively as:

```
1!  =  1
N!  =  N * (N-1)!
```

- The concept of the factorial is defined in terms of another factorial
- Eventually, the base case of 1! is reached

Recursive Definitions



Recursive Programming

- **A method in Java can invoke itself; if set up that way, it is called a *recursive method***
- **The code of a recursive method must be structured to handle both the base case and the recursive case**
- **Each call to the method sets up a new execution environment, with new parameters and local variables**
- **As always, when the method completes, control returns to the method that invoked it (which may be an earlier invocation of itself)**

Recursive Programming

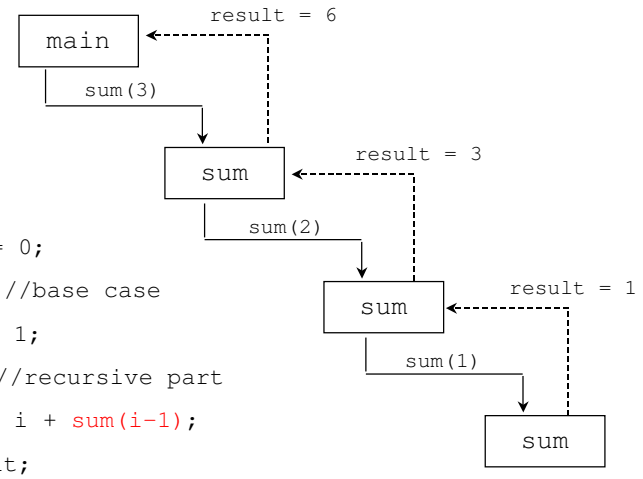
- **Consider the problem of computing the sum of all the numbers between 1 and any positive integer N**
- **This problem can be recursively defined as:**

$$\sum_{i=1}^{N} = N + \sum_{i=1}^{N-1} = N + (N-1) + \sum_{i=1}^{N-2}$$

$$= \text{etc.}$$

Recursive Programming

```
int sum(int i)
{
   int result = 0;
   if (i == 1) //base case
       result = 1;
   else        //recursive part
       result = i + sum(i-1);
   return result;
}
```



Recursive Programming

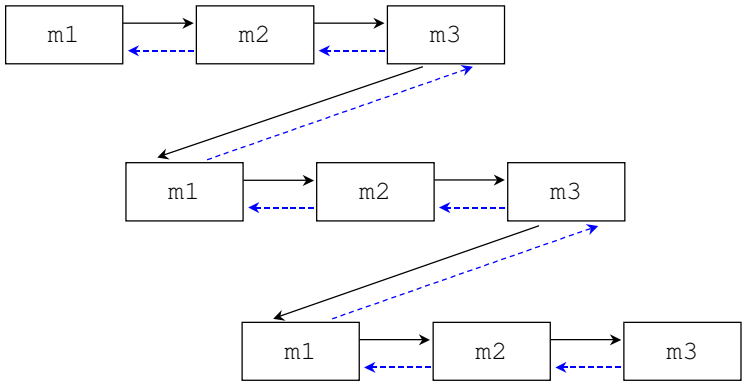
- Note that just because we can use recursion to solve a problem, doesn't mean we should (there is a lot of overhead: method calls, variable declarations, etc.)
- For instance, we usually would not use recursion to solve the sum of 1 to N problem, because the iterative version is easier to understand
- However, for some problems, recursion provides an elegant solution, often cleaner than an iterative version
- You must carefully decide whether recursion is the correct technique for any problem

- See PalindromeTesters.java

Indirect Recursion

- A method invoking itself is considered to be *direct recursion*
- A method could invoke another method, which invokes another, etc., until eventually the original method is invoked again
- For example, method `m1` could invoke `m2`, which invokes `m3`, which in turn invokes `m1` again
- This is called *indirect recursion*, and requires all the same care as direct recursion
- It is often more difficult to trace and debug

Indirect Recursion



Maze Traversal

- We can use recursion to find a path through a maze
- From each location, we can search in each direction
- Recursion keeps track of the path through the maze
- The base case is an invalid move or reaching the final destination
- See `MazeSearch.java`
- See `Maze.java`

Towers of Hanoi

- The *Towers of Hanoi* is a puzzle made up of three vertical pegs and several disks that slide on the pegs
- The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top
- The goal is to move all of the disks from one peg to another under the following rules:
  - We can move only one disk at a time
  - We cannot move a larger disk on top of a smaller one

Towers of Hanoi

- An iterative solution to the Towers of Hanoi is quite complex
- A recursive solution is much shorter and more elegant
- See `SolveTowers.java`
- See `TowersOfHanoi.java`