



COS30008

Data Structures and Patterns

Recursion

Recursion, Linked Lists, and ADTs

Overview

- Recursion
- Singly-Linked Lists
- Abstract Data Types

References

- Bruno R. Preiss: Data Structures and Algorithms with Object-Oriented Design Patterns in C++. John Wiley & Sons, Inc. (1999)
- Richard F. Gilberg and Behrouz A. Forouzan: Data Structures - A Pseudocode Approach with C. 2nd Edition. Thomson (2005)
- Russ Miller and Laurence Boxer: Algorithms Sequential & Parallel. 2nd Edition. Charles River Media Inc. (2005)
- Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo: C++ Primer. 5th Edition. Addison-Wesley (2013)

Recursion

- If a procedure contains within its body calls to itself, then this procedure is said to be **recursively defined**.
- This approach of program specification is called recursion and is found not only in programming.
- If we then define a procedure recursively, then there must exist **at least one sub-problem that can be solved directly**, that is without calling the procedure again.
- **A recursively defined procedure must always contain a directly solvable sub-problem. Otherwise, this procedure does not terminate.**

Problem-Solving with Recursion

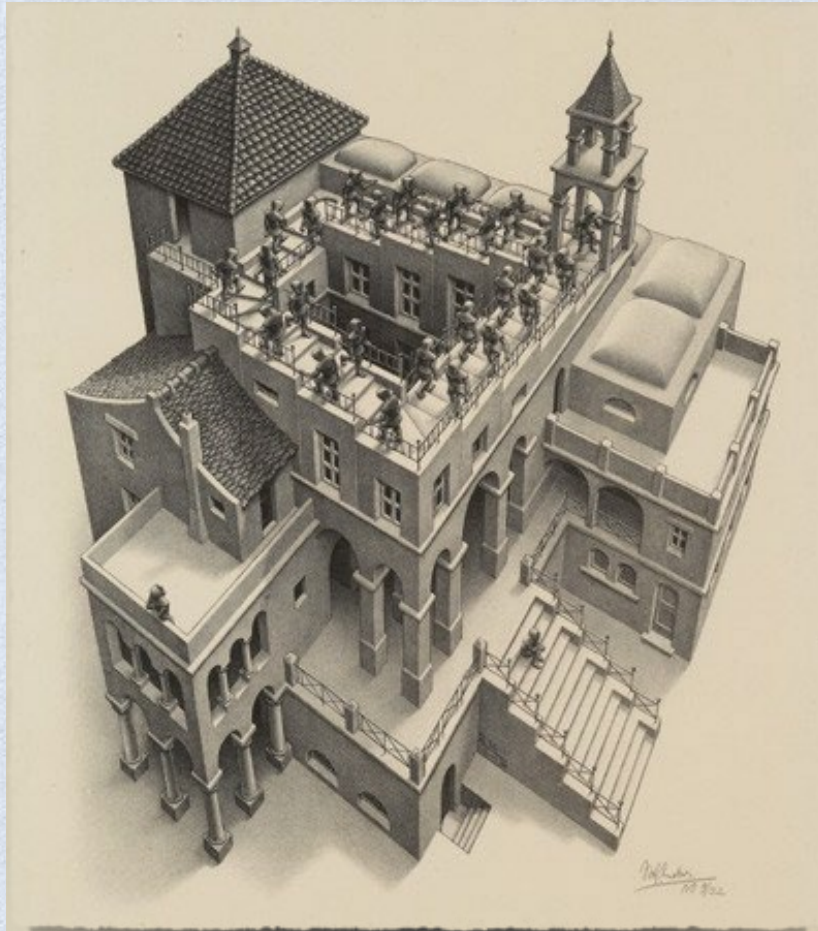
- Recursion is an important problem-solving technique in which a given problem is reduced to smaller instances of the same problem.
- The general structure of a recursive definition is

$$X = \dots X \dots$$

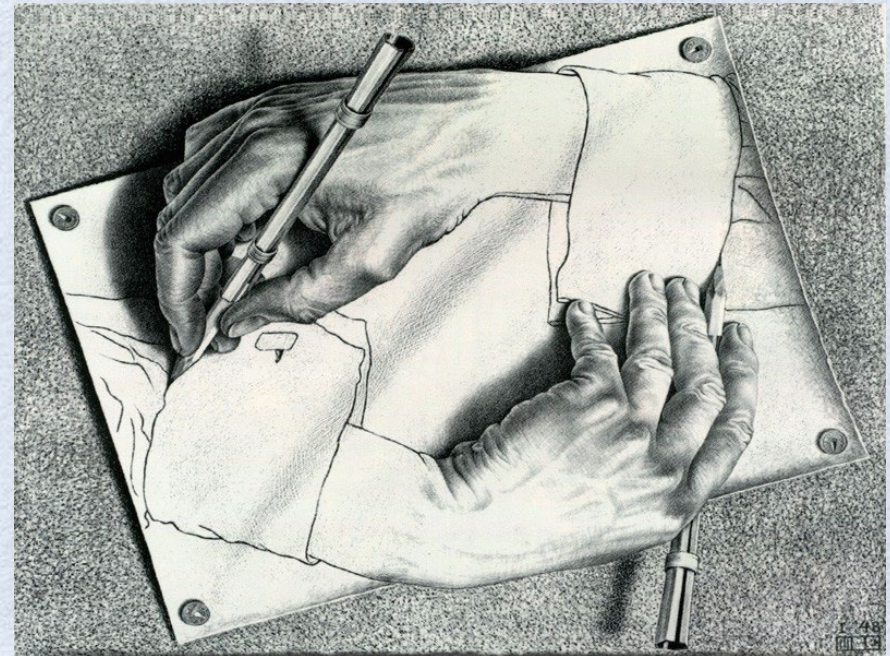
Left-hand side

Right-hand side

Impossible Recursive Structures



<http://www.mcescher.com/>

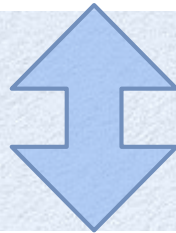


Fair use of material subject to copyright: Low resolution images for educational presentation on recursion.

Impossible Structures in C++:

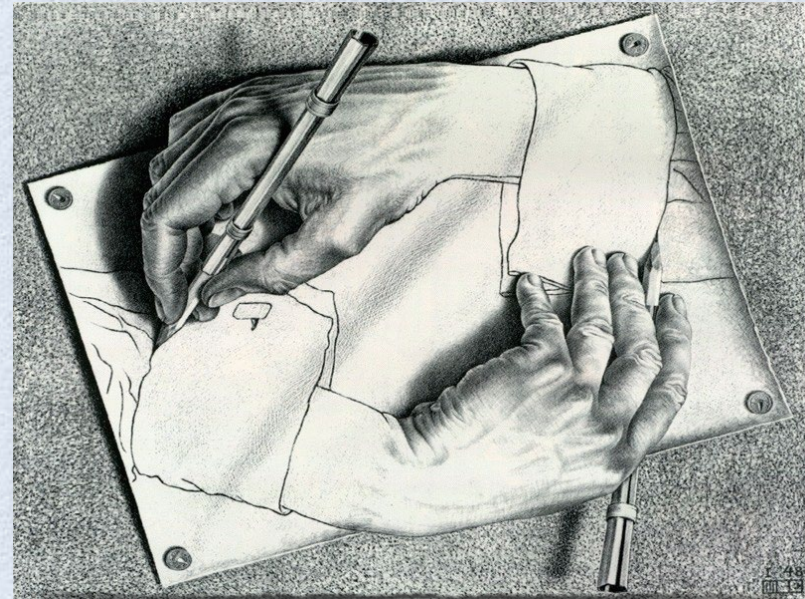
```
#include "ClassB.h"
```

```
class ClassA  
{  
    use ClassB  
};
```



```
#include "ClassA.h"
```

```
class ClassB  
{  
    use ClassA  
};
```



Forward Declaration

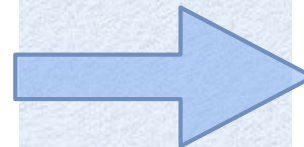
- Unlike in Java or C#, we cannot define mutual recursive classes in C++.
- We must resolve the dependency manually and use forward declarations in specifications.
- In the implementations, we must include all specifications, so that the compiler can resolve the dependencies.

```
class ClassB;
```

```
class ClassA  
{  
    use ClassB  
};
```

```
class ClassA;
```

```
class ClassB  
{  
    use ClassA  
};
```



```
#include "ClassA.h"  
#include "ClassB.h"
```

```
implement ClassA  
implement ClassB
```


Basic Recursive Problems

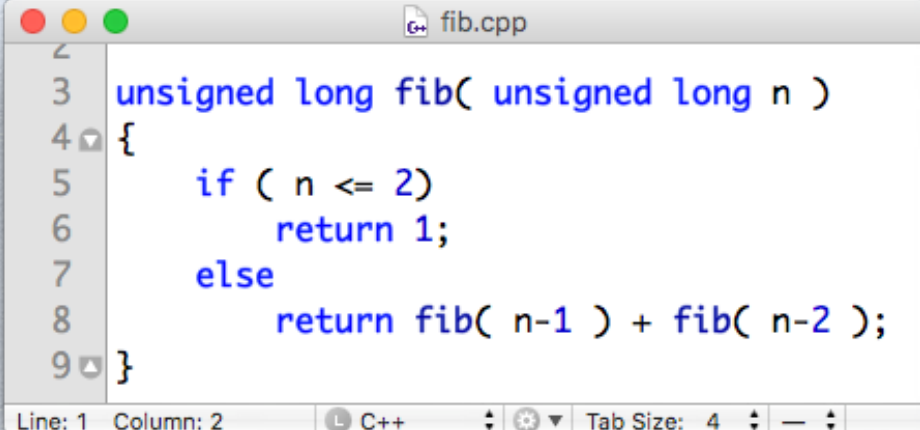
Fibonacci

- The Fibonacci numbers are the following sequence of numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- In mathematical terms, the sequence $F(n)$ of Fibonacci numbers is defined recursively as follows:

$$F(1) = 1$$

$$F(2) = 1$$

$$F(n) = F(n-1) + F(n-2)$$



```
1
2
3 unsigned long fib( unsigned long n )
4 {
5     if ( n <= 2)
6         return 1;
7     else
8         return fib( n-1 ) + fib( n-2 );
9 }
```

Line: 1 Column: 2 C++ Tab Size: 4

Recursive Problem-Solving: Factorials

- The factorial for positive integers is

$$n! = n * (n - 1) * \dots * 1$$

- The recursive definition:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

Calculating Factorials

- The recursive definition tells us exactly how to calculate a factorial:

$$\begin{aligned}4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \\ &= 24\end{aligned}$$

Recursive step: $n=4$

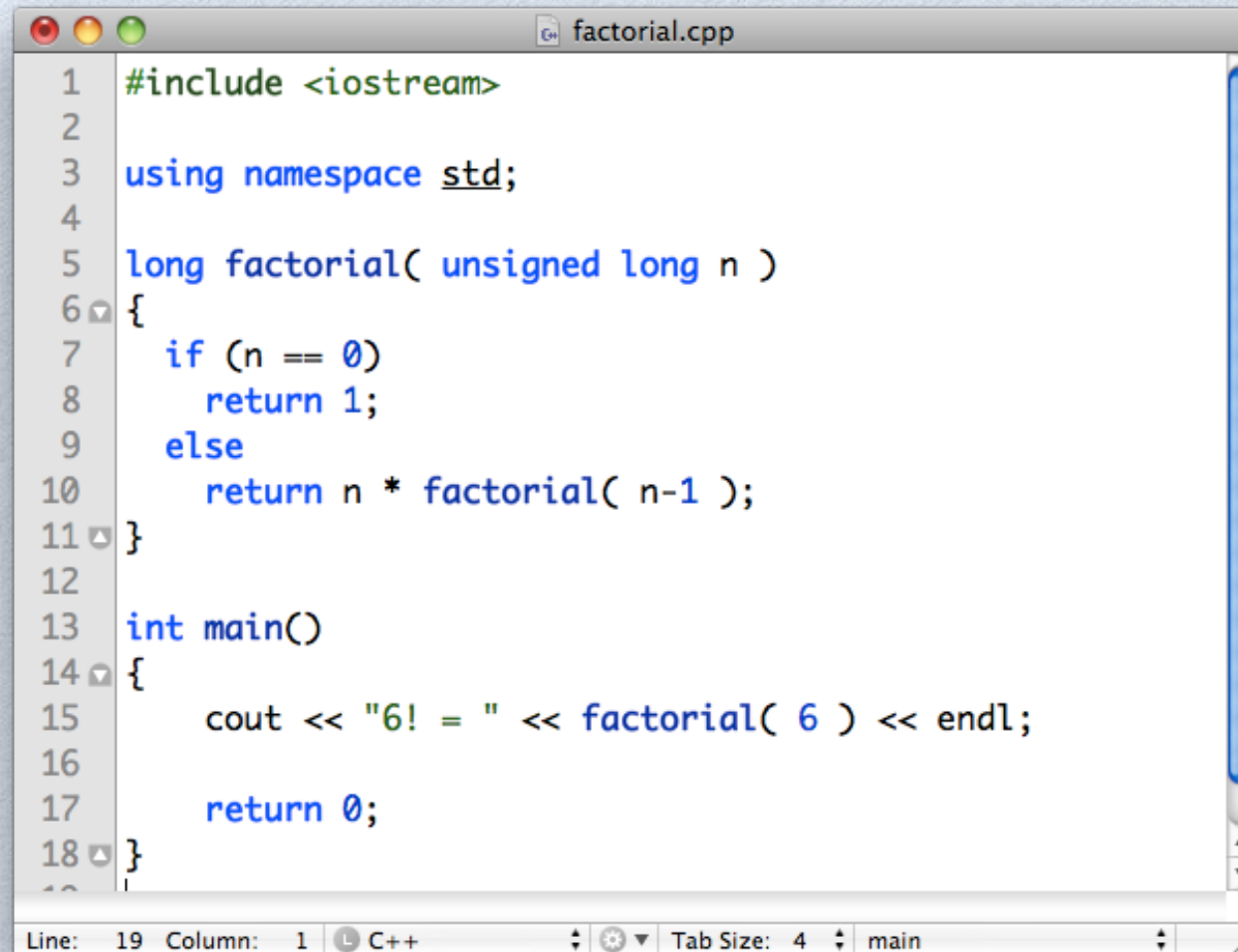
Recursive step: $n=3$

Recursive step: $n=2$

Recursive step: $n=1$

Stop condition: $n=0$

Recursive Factorial

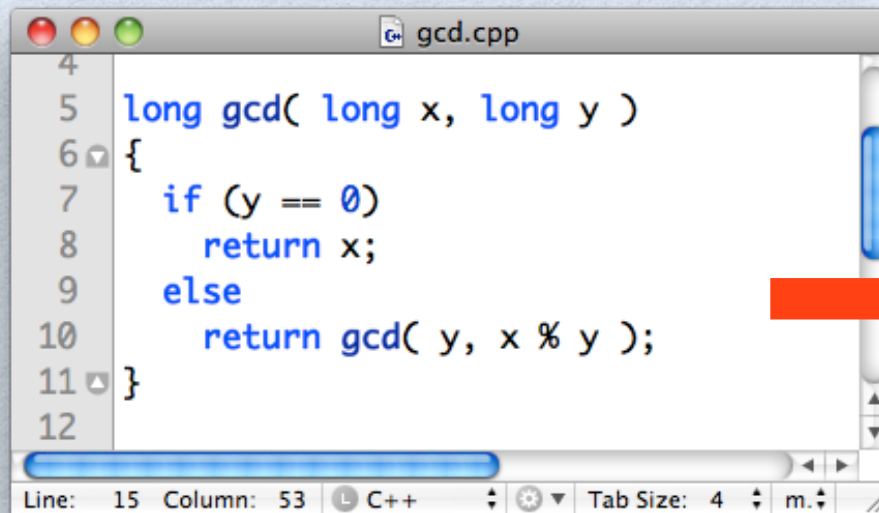


```
1  #include <iostream>
2
3  using namespace std;
4
5  long factorial( unsigned long n )
6  {
7      if (n == 0)
8          return 1;
9      else
10         return n * factorial( n-1 );
11 }
12
13 int main()
14 {
15     cout << "6! = " << factorial( 6 ) << endl;
16
17     return 0;
18 }
```

The screenshot shows a code editor window with a title bar containing three colored buttons (red, yellow, green) and a file icon followed by the text 'factorial.cpp'. The editor area displays C++ code for a recursive factorial function. The code is line-numbered from 1 to 18. Line 1: '#include <iostream>'. Line 2: empty. Line 3: 'using namespace std;'. Line 4: empty. Line 5: 'long factorial(unsigned long n)'. Line 6: '{'. Line 7: ' if (n == 0)'. Line 8: ' return 1;'. Line 9: ' else'. Line 10: ' return n * factorial(n-1);'. Line 11: '}'. Line 12: empty. Line 13: 'int main()'. Line 14: '{'. Line 15: ' cout << "6! = " << factorial(6) << endl;'. Line 16: empty. Line 17: ' return 0;'. Line 18: '}'. The status bar at the bottom shows 'Line: 19 Column: 1', a C++ icon, 'C++', a settings icon, 'Tab Size: 4', and 'main'.

Tail-Recursion

- A function is called **tail-recursive** if it ends in a recursive call that does not build-up any deferred operations.



```
gcd.cpp
4
5 long gcd( long x, long y )
6 {
7     if (y == 0)
8         return x;
9     else
10        return gcd( y, x % y );
11 }
12
```

Line: 15 Column: 53 C++ Tab Size: 4 m.



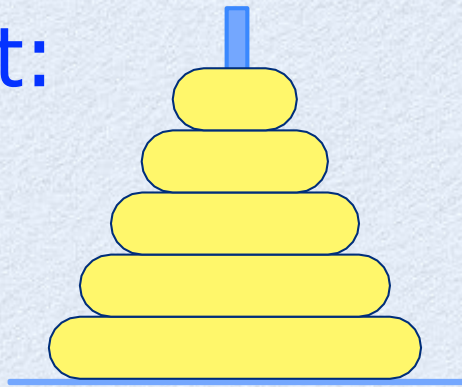
gcd(1246, 234):
↳ gcd(234, 76)
↳ gcd(76, 6)
↳ gcd(6, 4)
↳ gcd(4, 2)
↳ gcd(2, 0)
↳ 2

Towers of Hanoi

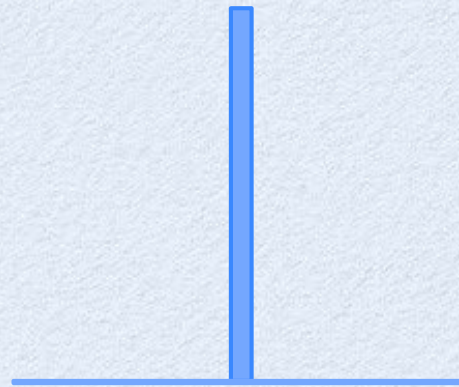
- Problem:
 - Move disks from a start peg to a target peg using a middle peg.
- Challenge:
 - All disks have a unique size and at no time must a bigger disk be placed on top of a smaller one.

Towers of Hanoi: Configuration

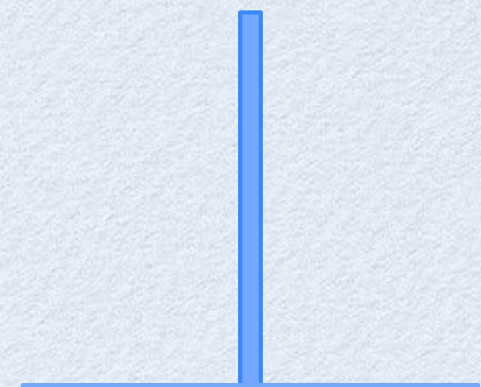
Start:



Start

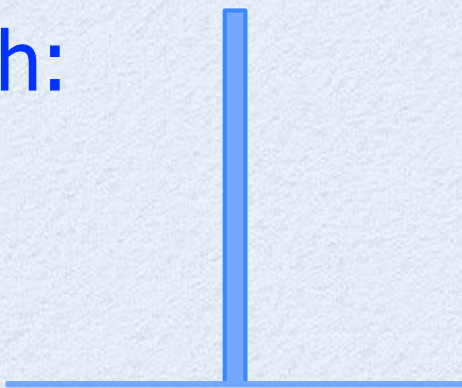


Middle

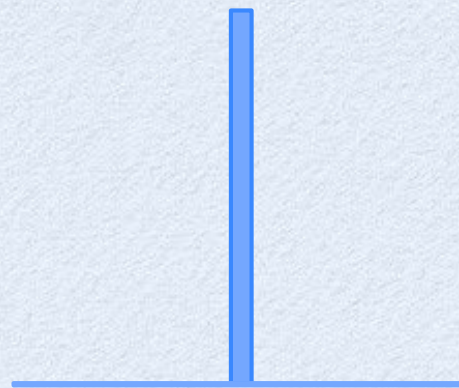


Target

Finish:



Start



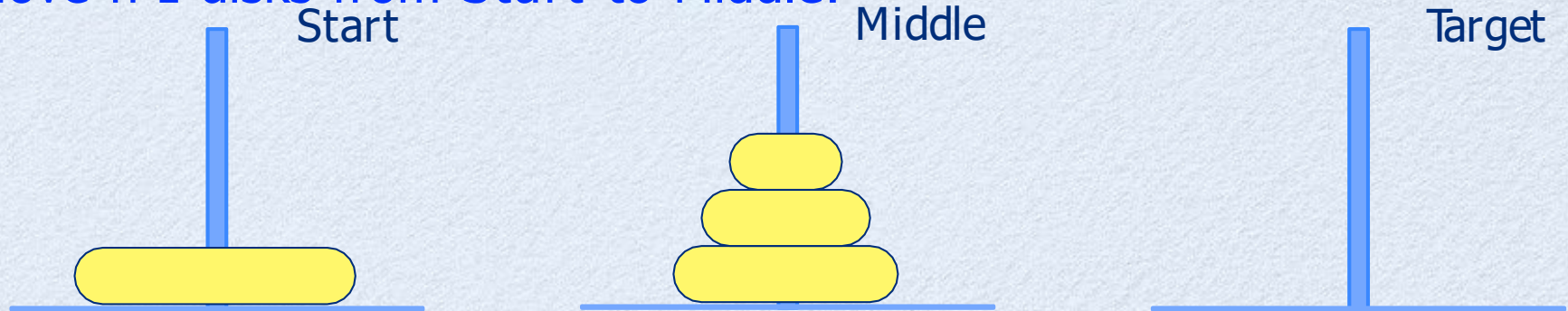
Middle



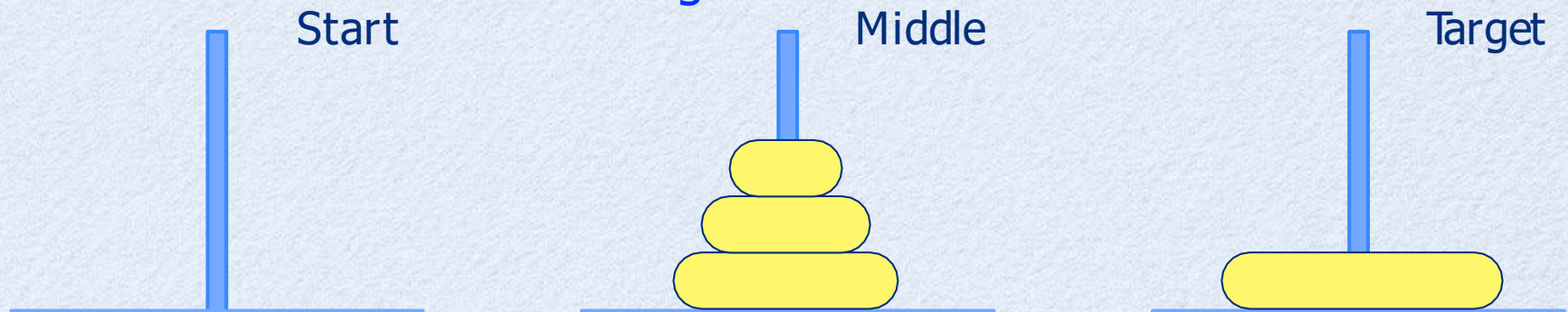
Target

A Recursive Solution

1. Move $n-1$ disks from Start to Middle:



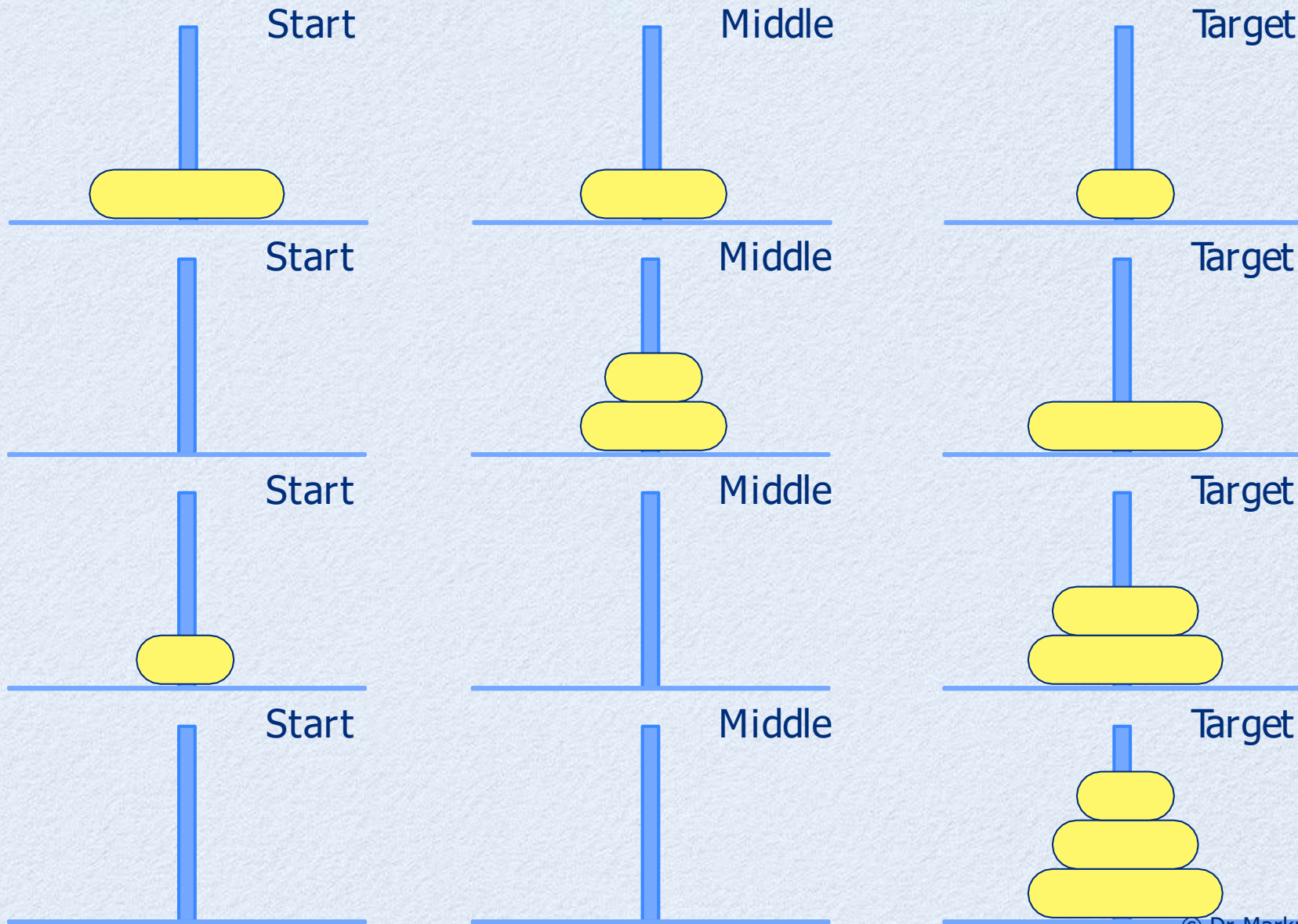
2. Move 1 disk from Start to Target:



3. Move $n-1$ disks from Middle to Target:



A Recursive Solution: Intermediate



The Recursive Procedure

```
hanoi.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  void move( int n, string start, string target, string middle )
6  {
7      if ( n > 0 )
8      {
9          move( n-1, start, middle, target );
10         cout << "Move disk " << n << " from " << start
11             << " to " << target << "." << endl;
12         move( n-1, middle, target, start );
13     }
14 }
15
16 int main()
17 {
18     move( 3, "Start", "Target", "Middle" );
19
20     return 0;
21 }
```

Line: 22 Column: 1 C++ Tab Size: 4 main

```
COS30008
Kamala: COS30008 Markus$ ./hanoi
Move disk 1 from Start to Target.
Move disk 2 from Start to Middle.
Move disk 1 from Target to Middle.
Move disk 3 from Start to Target.
Move disk 1 from Middle to Start.
Move disk 2 from Middle to Target.
Move disk 1 from Start to Target.
Kamala: COS30008 Markus$ _
```


**Recursion is a prerequisite
for linked lists!**