Recursion (and execution trees) Carol Zander

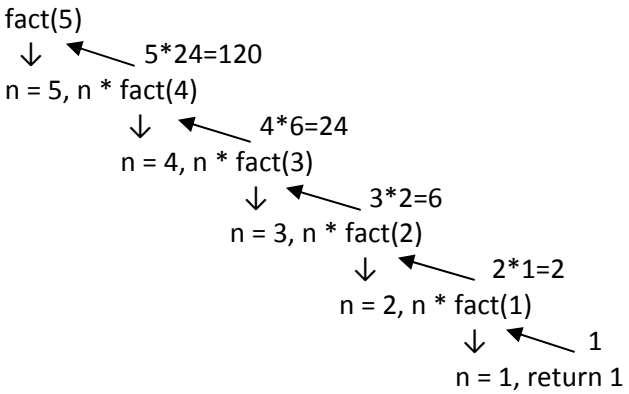
Example – Factorial

Recursive definition for Factorial of n, fact(n) = n! (e.g., 4! = 4 · 3 · 2 · 1) are defined by 0! = 1,
and fact(n) =n · fact(n-1) , n = 0, 1, 2, … .

```
// does not take into consideration overflow
int fact(int n) {
   if (n < 0) return -1;
   if (n <= 1) return 1;
   return n * fact(n-1);
}

int main() {
   cout ≪ fact(5) ≪ endl;
   return 0;
}
```

Draw an execution tree (showing the calls and returns) of the execution of fact(5):



Example – Towers of Hanoi

The Towers of Hanoi is a mathematical game or puzzle. It consists of three pegs, and a number of disks of different sizes which can be put onto a peg. The puzzle starts with the disks in a stack in order from smallest to largest on one peg, smallest at the top. The objective is to move the entire stack from one peg to another peg, obeying the following rules:

- Only one disk may be moved at a time
- One move consists of taking the top disk from one of the pegs and putting it onto another peg, on top of the other disks on that peg
- No disk may be placed on top of a smaller sized disk

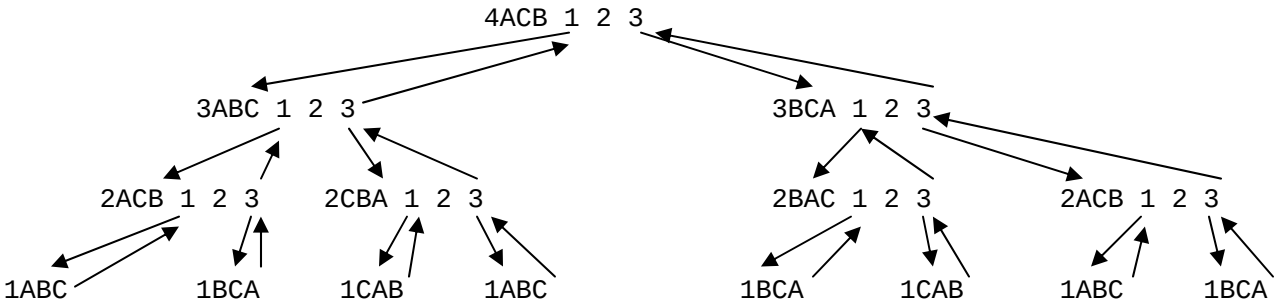
The puzzle was invented by the French mathematician Édouard Lucas under the name N. Lucas de Siam in 1883. There is a legend about a temple in Benares with a dome which marked the center of the world. Within the dome, there is a large room with three posts in it surrounded by 64 golden disks. The priests of Hanoi, move these disks, in accordance with the rules of the puzzle. According to the legend, when the last move of the puzzle is completed, the universe will come to an end. (There are many variations on this legend.)
If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them $2^{64}-1$ seconds or roughly 585 billion years to finish.

```
void moveDisks (int n, char fromPeg, char toPeg, char auxPeg) {
   if (n == 1) {                                          // base case
      cout << setw(6) << n
           << "            " << fromPeg
           << "            " << toPeg << endl;
   }
   else {                                                 // recursive step
      moveDisks(n-1, fromPeg, auxPeg, toPeg);             // call this 1
      cout << setw(6) << n                                // call this 2
           << "            "  << fromPeg
           << "            " << toPeg << endl;
      moveDisks(n-1, auxPeg, toPeg, fromPeg);             // call this 3
   }
}

int main() {
   cout << "Move Disk #      From Peg     To Peg" << endl;
   moveDisks(4, 'A', 'C', 'B');

   return 0;
}
```

Draw an execution tree (showing the calls and returns) of the execution of movedisks(4, 'A', 'C', 'B') and display output:



Output:

Move Disk #	From Peg	To Peg
1	A	B
2	A	C
1	B	C
3	A	B
1	C	A
2	C	B
1	A	B
4	A	C
1	B	C
2	B	A
1	C	A
3	B	C
1	A	B
2	A	C
1	B	C

Example – find depth in a general binary tree (not necessarily binary search tree)

Find the depth of a node in the tree. Return 0 for the root, -1 for an object not found. One solution:

```
int depth(BinaryTreeNode *current, const Object &target) const {
   if (current == NULL) return = -1;
   if (*current->item == target) return 0;

   int tryBranch = depth(current->left, target);
   if (tryBranch == -1) tryBranch = depth(current->right, target);
   if (tryBranch == -1) return -1;
   else return 1 + tryBranch;
}
```

Consider the following tree and find the depth of the target "rr":