Recursion

Last updated 6/23/23

These slides introduce basic concepts of recursion

Recursion

- Recursion
  - Break a problem into smaller and smaller parts until solving it is easy
  - Typically involves a function calling itself (think nest of mirrors)

  - Requirements for a function (algorithm) to be recursive
    - Base case
      - Terminating point
      - Easy to solve
    - Must progress toward the base case each iteration
    - Function (algorithm) calls itself

Recursion

- Example – Factorial(n)
  - Base case: n = 1
  - Progress toward base:

    factorial(n) calls factorial with (n – 1)

```
int factorial(int n){
   // special case
   if(n == 0)
      return 1;

   // base case
   else if (n == 1)
      return 1;

   // movement toward base
   // decrement n --> n = 1
   else
      return n * factorial(n - 1);

}// end factorial
```

N = 5

f(5)
f(5)f(4)
f(5)f(4)f(3)
f(5)f(4)f(3)f(2)
f(5)f(4)f(3)f(2)f(1)
Returns:

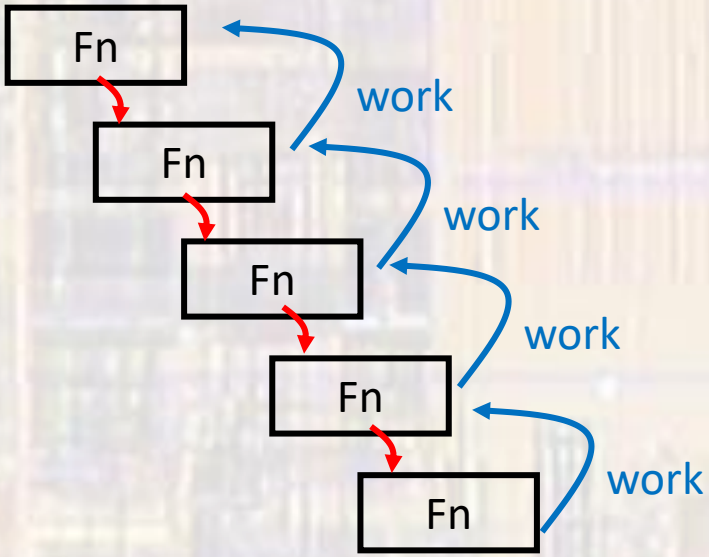
1	1
2*1	2
3*2	6
4*6	24
5*24	120

Recursion

- Types of Recursion
  - Direct
    - Function calls itself
  - Indirect
    - Function calls a second function, that calls the first function
  - Head
    - The function self-call occurs effectively at the beginning of the function
  - Tail
    - The function self-call occurs at the end of the function
  - Body
    - The function self-call occurs somewhere other than the beginning or end of the function

Recursion

- Head Recursion
  - The function self-call occurs effectively at the beginning of the function
    - The ‘work’ is done on the way back up the path
    - E.g. factorial()

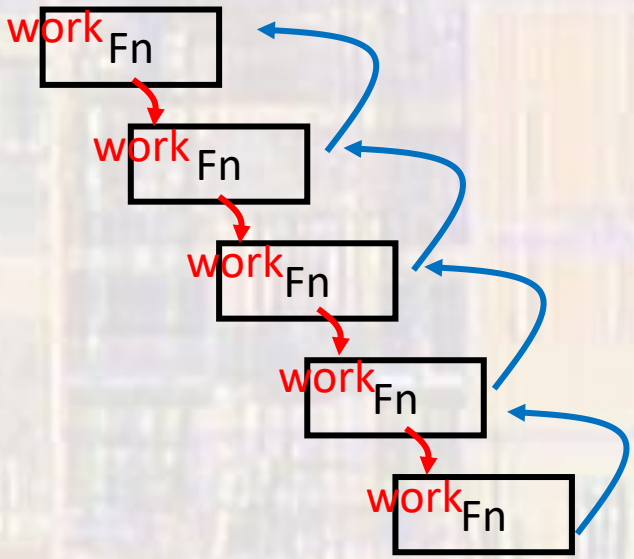



```
void count(int n){
    // base case
    if(n > 0)
        // movement toward base
        count(n - 1);

    // work done in the return path
    printf("%i ", n);

    return;
}// end count
```

1 2 3 4 5

Recursion

- Tail Recursion
  - The function self-call occurs at the end of the function
    - The ‘work’ is done on the way down the path




```
void count_down(int n){
    // base case
    if(n > 0){
        // work done in the forward path
        printf("%i ", n);
        // movement toward the base
        count_down(n - 1);
    }

    return;
}// end count_down
```

5 4 3 2 1

Recursion

- Towers of Hanoi
  - Move all discs from one tower to another
  - Only one disk can be moved at a time.
  - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
  - No disk may be placed on top of a smaller disk.

Not too bad with 3 discs, but try it with 7

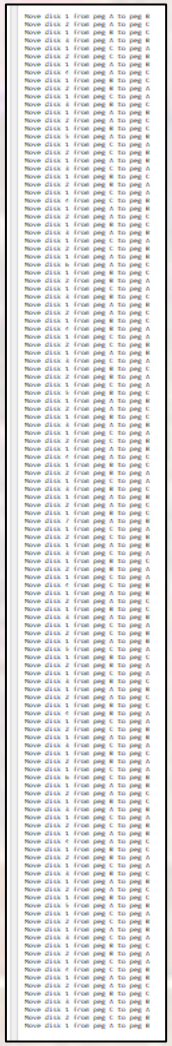
Recursion

- Towers of Hanoi

```
void toh(int num, char frompeg, char topeg, char sparepeg){
    // base case
    if (num == 1){
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }

    // moving toward base
    toh(num - 1, frompeg, sparepeg, topeg);
    printf("\n Move disk %i from peg %c to peg %c", num, frompeg, topeg);
    toh(num - 1, sparepeg, topeg, frompeg);
}// end toh
```

```
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 3 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
```

N = 3



N = 7

Recursion

- Sudoku Solver

```
-------------------------------------
|  1  7  4  |     9     |  6        |
|           |     3  8  |  1  5  7  |
|  5  3     |  7     1  |           4 |
-------------------------------------
|           7  |  3  4  9  |  8        |
|  8  4        |  5        |  3  6     |
|  3     5     |           6  |  4  7     |
-------------------------------------
|  2  8  6  |  9        |           1 |
|           |  6  2  7  |     3  8  |
|     5  3  |     8     |     9  6  |
-------------------------------------
```

Level = Medium

```
-------------------------------------
|  1  7  4  |  2  9  5  |  6  8  3  |
|  9  6  2  |  4  3  8  |  1  5  7  |
|  5  3  8  |  7  6  1  |  9  2  4  |
-------------------------------------
|  6  2  7  |  3  4  9  |  8  1  5  |
|  8  4  1  |  5  7  2  |  3  6  9  |
|  3  9  5  |  8  1  6  |  4  7  2  |
-------------------------------------
|  2  8  6  |  9  5  3  |  7  4  1  |
|  4  1  9  |  6  2  7  |  5  3  8  |
|  7  5  3  |  1  8  4  |  2  9  6  |
-------------------------------------
```

```
-------------------------------------
|     1  7  |           |     2     |
|           |        9  |           |
|  4        |  7  5     |  6        |
-------------------------------------
|        2  |  9  3     |     6     |
|           |        4  |  9        |
|  3        |        8  |           |
-------------------------------------
|        4  |           |        5  |
|  5        |  3  6     |  7        |
|           |     8     |           |
-------------------------------------
```

Level = Evil

```
-------------------------------------
|  9  1  7  |  8  4  6  |  5  2  3  |
|  6  3  5  |  2  1  9  |  8  7  4  |
|  4  2  8  |  7  5  3  |  6  9  1  |
-------------------------------------
|  8  5  2  |  9  3  1  |  4  6  7  |
|  1  7  6  |  5  2  4  |  9  3  8  |
|  3  4  9  |  6  7  8  |  1  5  2  |
-------------------------------------
|  2  6  4  |  1  9  7  |  3  8  5  |
|  5  8  1  |  3  6  2  |  7  4  9  |
|  7  9  3  |  4  8  5  |  2  1  6  |
-------------------------------------
```

Recursion

- Caveats
  - Recursion can make some problems much easier to solve but it can also introduce unnecessary complexity and cost (clk cycles and memory)
    - Function calls take clock cycles
    - Functions use stack space
    - Use a for/while loop where possible.

