RECURSION NOTES:

- Author: Jude Thaddeau Data
- GitHub: Zero-Luminance
- Source: Neso Academy (YouTube)
  - Recursion In C
  - How To Write Recursive Functions
  - Types Of Recursion (Part 1)
  - Types Of Recursion (Part 2)
  - Advantages & Disadvantages Of Recursion

RECURSION:
- 'Recursion' is the process of defining a problem (or solution to a problem) in terms of (a simpler version of) itself
- METHOD: keep self-calling until problem is solved (base case)

Base & Recursive Cases:
- 'Base Case' is the solution to the main problem & is what each recursive call gradually works towards
- 'Recursive Case' is a sub-solution to a sub-problem of a main problem, & tells which direction the next call should go to move closer to the base case

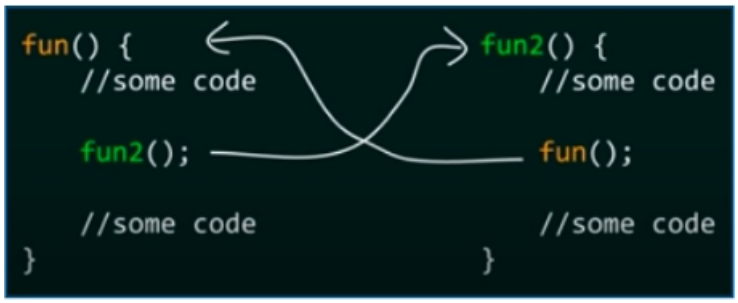
Storing The Information In Recursive Calls:
- 'Stack' is a segment of memory that plays a role in storing each recursive call: - Recursive calls ADDS a layer to the stack
- Base case is the FINAL layer of the stack
- Functions calls are executed from TOP to BOTTOM
- Example implementations: factorial, Fibonacci, sorting, etc

TYPES OF RECURSION: Direct Recursion:
- 'Direct Recursion' involves functions that ONLY calls itself
- Example: https://people.eng.unimelb.edu.au/ammoffat/ppsaa/c/binarysearch.c

```
int
binary_search(data_t A[], int lo, int hi,
                  data_t *key, int *locn) {

        int mid, outcome;

        /* if key is in A, it is between A[lo] and A[hi-1] */
        if (lo>=hi) {
                return BS_NOT_FOUND;
        }

        mid = (lo+hi)/2;

        if ((outcome = cmp(key, A+mid)) < 0) {
                return binary_search(A, lo, mid, key, locn);
        } else if (outcome > 0) {
                return binary_search(A, mid+1, hi, key, locn);
        } else {
                *locn = mid;
                return BS_FOUND;
        }
}
```

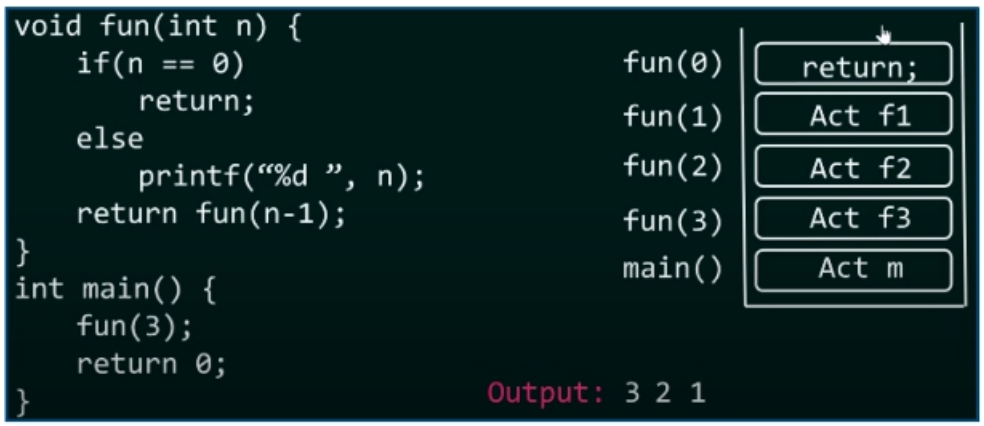
Indirect Recursion:

- 'Indirect Recursion' occurs when a function is called not by itself but by another function which may in turn call the original, or call other functions that will eventually call the original
- Example:



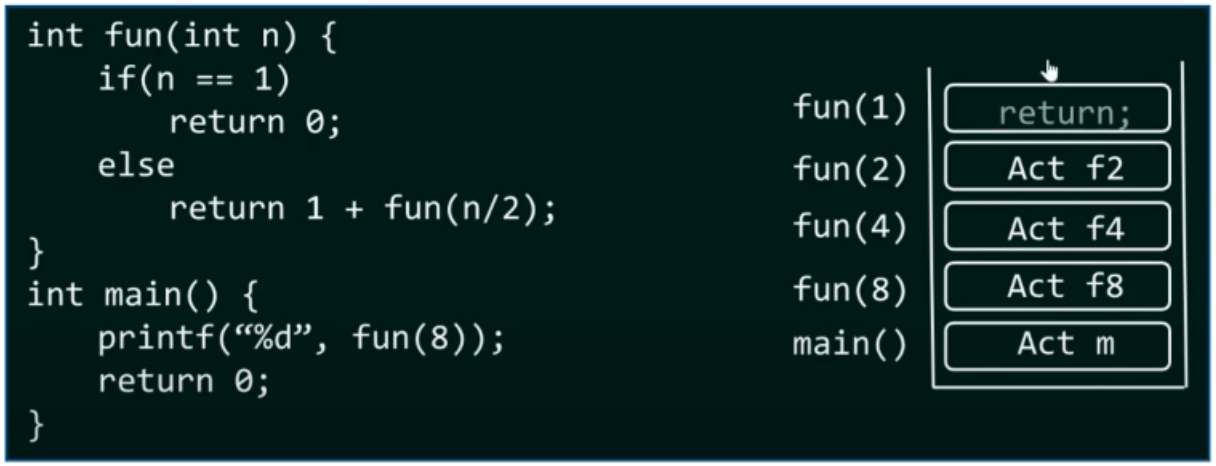
Tail Recursion:

- 'Tail Recursion' occurs when a recursive call is the LAST thing done by a function & there is NO NEED to keep record of the PREVIOUS state
- Example:



Non-Tail Recursion:

- 'Non-Tail Recursion' occurs when a recursive call is NOT the LAST thing done by a function & after returning back via the stack, there IS PREVIOUS information OR code left to evaluate
- CASE 1) Returning Information
- The return value (information) from the previous recursive call in the stack is used to EVALUATE the NEXT return value of the NEXT recursive call
- Example:



- CASE 2) Evaluating After Returns
- Input information via the recursive parameters is used to EVALUATE the REMAINING code
- All recursive calls are FIRST recorded on the stack
- When base case is reached (stack popping), local information within each recursive call is USED to EVALUATE remaining code
- Example:

```
void fun(int n) {
    if(n == 0)
        return;
    fun(n-1);
    printf("%d ", n);
}
int main() {
    fun(3);
    return 0;
}
```

fun(0) return;
fun(1) Act f1
fun(2) Act f2
fun(3) Act f3
main() Act m

Analysing Recursion:

Advantages:	Disadvantages:
- REDUCES time complexity by memorising the result of recursive calls - Fast for SMALL input values - SUPERIOR tree traversal technique	- Memory intensive on the stack - Slow for LARGE input values - Iteration is a BETTER (but less concise) alternative for some problems