**"Heap", "Stack" Memory & Overhead**

In C programming, "heap" and "stack" refer to two different areas of memory used for different purposes. "Overhead" generally refers to the additional resources or computational cost associated with certain operations.

**1. Heap:**

- The heap is a region of memory used for dynamic memory allocation. It's where you can allocate memory during runtime using functions like `malloc`, `calloc`, and `realloc`.

- Memory allocated on the heap persists until it's explicitly deallocated using `free`.

- Since memory on the heap can be allocated and deallocated in a more flexible manner compared to the stack, it's commonly used for data structures like linked lists, trees, and dynamic arrays.

**2. Stack:**

- The stack is a region of memory used for function call management and local variable storage.

- When a function is called, its local variables and function call information (such as return address) are stored on the stack.

- The memory used for the stack is managed in a last-in-first-out (LIFO) manner, meaning that the most recently added data is the first to be removed.

- Since stack memory is managed in a strict order and local variables have a well-defined scope, the stack is usually faster for memory access compared to the heap.

**3. Overhead:**

- Overhead refers to the extra resources (like time or memory) consumed by a process that are not directly contributing to the primary task.

- In C, overhead can manifest in various ways. For example, function call overhead involves the cost of setting up function call frames on the stack, passing parameters, and returning values.

- In memory management, overhead might refer to the extra memory consumed by data structures used to manage memory allocation, like block headers in dynamically allocated memory chunks.

**Here's a simple example illustrating these concepts:**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main() {**

**// Stack-allocated integer**

**int stackValue = 10;**

**// Heap-allocated integer**

**int \*heapValue = (int \*)malloc(sizeof(int));**

**\*heapValue = 20;**

**printf("Stack Value: %d\n", stackValue);**

**printf("Heap Value: %d\n", \*heapValue);**

**// Deallocate heap memory**

**free(heapValue);**

**return 0;**

**}**

In this example, `stackValue` is stored on the stack, and `heapValue` is allocated on the heap using `malloc`. After printing the values, the heap memory is deallocated with `free`.