# The General Dynamic Storage Allocation Problem

Author: Ilia Fatemi
SFU ID: 301415912

## Algorithm Overview

1. **Initialization (`mem_init()`):**
    - Initializes the memory system by allocating an initial memory block of size `` `MEMORY_SIZE` ``.
2. **Allocation (`my_malloc(size_t size)`):**
    - Allocates memory for a requested size.
    - Searches for free blocks in the memory to accommodate the requested size. When and if found, it will check to see if there is a free block and it will mark it as allocated. If space is available but larger than requested, split the block into an allocated block and a new free block.
    - Uses **First Fit** approach to allocating blocks.
    - Tracks allocated, free, and failed allocations.
3. **Deallocation (`my_free(void *ptr)`):**
    - Frees memory associated with the given pointer.
    - Merges adjacent free blocks if possible to avoid fragmentation.
    - Updates block statuses and sizes after deallocation.

## Data Structures Used

1. **Memory Structure:**
    - Holds metadata for each block in memory.
    - Attributes:
        i. **`size_t size`**: Total size of the memory block.
        ii. **`struct MemoryBlock* CurrentBlock`**: Pointer to the current memory block being processed.
        iii. **`struct MemoryBlock* headBlock`**: Pointer to the head of the memory block list.
2. **`MemoryBlock Structure:`**
    - This holds the metadata for each block in memory.
    - Attributes:
        i. **`size_t allocatedSize`**: Total size of the memory block.
        ii. **`size_t requestedSize`**: requested size.
        iii. **`void* startBlock`**: Pointer to the start of the block
        iv. **`void* endBlock`**: Pointer to the end of the block.
        v. **`struct MemoryBlock* prev`**: Pointer to the previous block.
        vi. **`struct MemoryBlock* next`**: Pointer to the next block.

## First-Fit Algorithm

**`my_malloc(size_t size)`** uses the First Fit approach algorithm for memory allocation. It iterates through the linked list of memory blocks, looking for a free block that can accommodate the requested size. Once a suitable free block is found:

- If the block exactly matches the requested size, it's allocated without creating a new free block.
- If the free block size is larger than the requested size, it's split into an allocated block and a new free block if the remaining free space is sufficient.

This approach allocates memory using the first available block that meets the size requirements, which aligns with the first fit strategy.

When calling **`my_free(void *ptr)`** to deallocate pointer, the function will free the memory associated with the pointer and will set the block status to FREE. Once deallocated, it will join adjacent free blocks to avoid fragmentation.

## Handling Memory Fragmentation

1. **Splitting Blocks:**
    - **Allocation**: When a requested size doesn't match an available block precisely, the code attempts to split a larger free block into two:
        i. One allocated block matching the requested size.
        ii. A new free block representing the remaining space.
    - **Deallocation**: After freeing a block, the code merges with adjacent free blocks to prevent unnecessary fragmentation.
2. **Merging Blocks:**
    - **Deallocation**: Upon freeing memory, the code checks neighboring blocks:
        i. If both previous and next blocks are free, they are merged into a single larger free block.
        ii. If only one neighboring block is free, it merges the free block with the deallocated block to prevent isolated free spaces.
3. **Optimizing Memory Usage:**
    - **Allocation**: If there's sufficient space in a free block to fulfill a requested size, it utilizes that block rather than creating new ones.
    - **Deallocation**: Merges contiguous free blocks to create larger free blocks and reduce fragmentation.

## Time Complexity

1. **`my_malloc`**: The time complexity is linear O(N) concerning the number of blocks in the memory.
2. **`my_free`**: The time complexity is constant O(1) because it directly manipulates adjacent blocks without iterating through all blocks.