AYÇA ELİF AKTAŞ - 27802

- My implementation, allacator.cpp, implements a custom heap manager named HeapManager. This manager is designed to simulate a memory allocation system, managing the allocation and deallocation of memory blocks within a simulated heap.
- Class Structure
  - Class Name: HeapManager
  - Private Members:
    - Node: A nested structure representing a block of memory in the heap. That has attributes such as ID: Identifier for the memory block, used to track allocation, size: Size of the memory block, index: Starting index of the memory block in the heap, next: Pointer to the next Node in the list.
    - Constructor of Node: Initializes a Node with given id, size, and index.
    - head: Pointer to the first Node in the linked list, representing the start of the heap.

- Public Methods
  - Constructor (HeapManager()):Initializes the head pointer to nullptr, indicating an empty heap initially.
  - initHeap(int size):Initializes the heap with a single block of the given size. The block is marked as free (ID = -1) and starts at index 0. Returns 1 as a success indicator.
  - myMalloc(int ID, int size): Allocates a block of memory of the specified size for the given ID. Searches for a free block (ID = -1) that can accommodate the requested size. If a larger block is found, it is split into two parts: one for the allocation and the other remaining free. Updates the linked list to reflect the allocation. Returns the starting index of the allocated block or -1 if allocation is not possible.
  - myFree(int ID, int index):Frees the memory block specified by the ID and index. Marks the block as free (ID = -1) and attempts to coalesce with adjacent free blocks. Updates the linked list to reflect the deallocation. Returns 1 on successful deallocation, -1 if the specified block is not found.
  - print(): Prints the current state of the heap, displaying each block's ID, size, and index.

The implementation uses a linked list to manage memory blocks. Each Node in the list represents a block in the heap. Memory allocation (myMalloc) searches for a suitable free block and splits it if necessary. Memory deallocation (myFree) marks blocks as free and coalesces adjacent free blocks to prevent fragmentation.