HEAP MANAGEMENT

Heap management involves managing memory allocation and deallocation dynamically within a heap structure. In this context, we'll explore how the allocate and free functions work, including the merging of adjacent free blocks to optimize memory usage. Here's a theoretical explanation with diagrams.

Key Concepts

Heap: A region of memory used for dynamic memory allocation.

Free Block: A portion of the heap that is available for allocation.

Allocated Block: A portion of the heap that is currently in use.

Metadata: Information stored alongside each block (both free and allocated) to manage the heap. Typically includes the size of the block and pointers to adjacent blocks.

Memory Layout

The heap is divided into blocks, each containing:

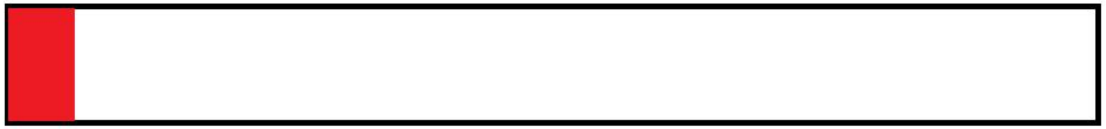
Metadata: Information about the block (e.g., size, free/allocated status).

Payload: The actual memory available for use (for allocated blocks).

Initial Heap Setup

1)Initial Free Block: When the heap is first initialized, it consists of a single large free block.

2)Metadata: This block includes metadata that tracks its size and its free/allocated status.



Heap Initially

heap size=1024 bytes

meta data size= 24bytes

Allocation Process

1)Finding a Free Block: The allocator searches for a free block that can satisfy the requested size.

2)Splitting: If the free block is larger than needed, it is split into two blocks: one allocated block and one smaller free block.

3)Updating Metadata: Metadata for the blocks is updated to reflect the allocation.

Let's simulate the allocation of memory in a heap for different data types (int, string, and double) in the specified order (100, 200, 400, "hi", 455.5) within a 1024-byte heap. We'll use metadata of 24 bytes for each block. Here's how the allocation process works:

Initial Heap State

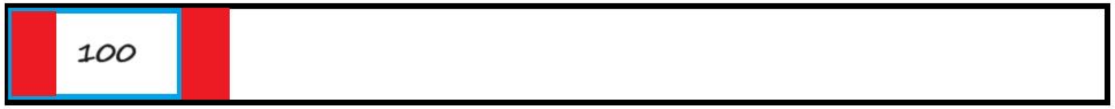
Initially, the heap is 1024 bytes with one large free block.

Allocation Process

1)Allocate 100 (int):

Size of int: 4 bytes

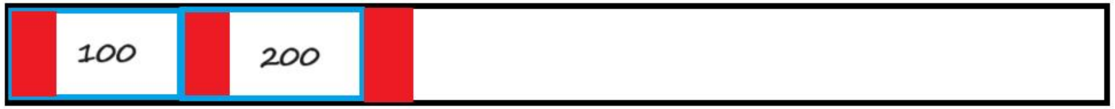
Total block size: 4 (payload) + 24 (metadata) = 28 bytes



2)Allocate 200 (int):

Size of int: 4 bytes

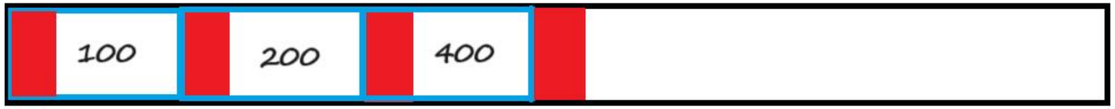
Total block size: 4 (payload) + 24 (metadata) = 28 bytes



3)Allocate 400 (int):

Size of int: 4 bytes

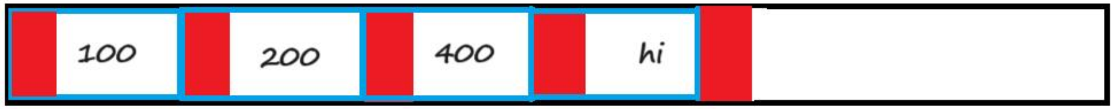
Total block size: 4 (payload) + 24 (metadata) = 28 bytes



4)Allocate "hi" (string):

Size of "hi": 3 bytes (including null terminator)

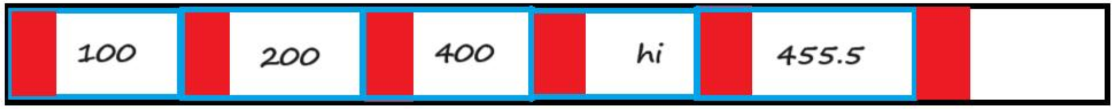
Total block size: 3 (payload) + 24 (metadata) = 27 bytes



5)Allocate 455.5 (double):

Size of double: 8 bytes

Total block size: 8 (payload) + 24 (metadata) = 32 bytes



Deallocation in Heap Management

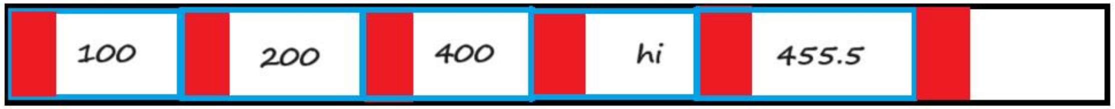
Deallocation Process

Deallocation in heap management is the process of marking previously allocated memory blocks as free, making them available for future allocations. The deallocation process involves several critical steps:

Marking the Block as Free: When a block of memory is deallocated, its status in the metadata is updated to indicate that it is no longer in use.
Merging Adjacent Free Blocks: To optimize memory usage and minimize fragmentation, adjacent free blocks are merged into a single larger block whenever possible.

Final Heap State

The heap after allocating the requested memory blocks looks like this:

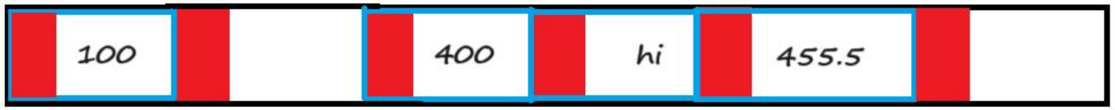


Step-by-Step Deallocation

Deallocate 200:

The metadata for the block allocated for 200 is updated to indicate that it is free.

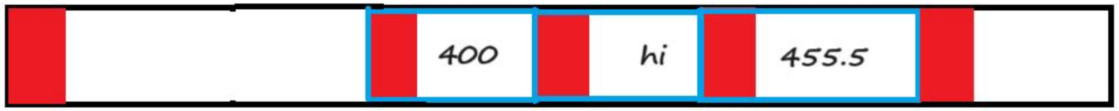
The heap now has a free block of 4 bytes where 200 was previously allocated.



Deallocate 100:

The metadata for the block allocated for 100 is updated to indicate that it is free.

The heap now has another free block of 4 bytes where 100 was previously allocated.



Merging Adjacent Free Blocks:

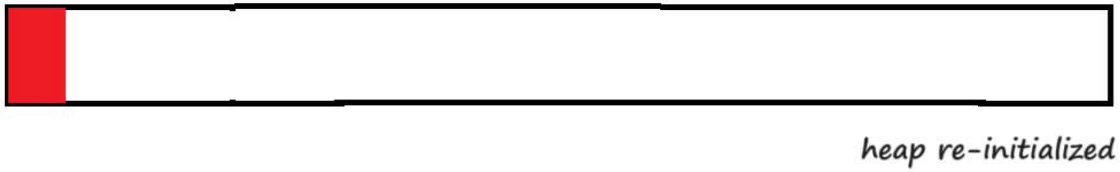
The allocator checks for adjacent free blocks and merges them to form a larger contiguous free block.

In this case, the two adjacent free blocks (4 bytes each) are merged into a single free block of 8 bytes.

Deallocation and Reinitialization in Heap Management

Reinitialization

Reinitialization involves reallocating a previously deallocated block of memory and preparing it for new data. This process can improve memory utilization by reusing free blocks instead of constantly allocating new ones.


heap re-initialized