Enum
The enum NodeType is used to define two constants, HOLE and PROCESS, which represent the two types of memory segments: holes (unused memory segments) and processes (allocated memory segments).

Struct Nodeofsmolchain
size: The size of the memory block.
type: The type of the memory block, which can be either HOLE or PROCESS.
next and prev: Pointers to the next and previous nodes in the linked list.

Struct Nodeofbigchain
size: The size of the memory block.
remainder: The remaining size of the memory block after allocations.
ptr: The pointer to the actual memory block.
virtual_address: The virtual address associated with the memory block.
subchain: A pointer to the linked list of sub-guccha nodes.
next and prev: Pointers to the next and previous nodes in the main memory chain.

The `get_pages` function takes malloc input and splits into pages and remaining hole size

The `combine_holes` function combines adjacent holes in the sub-guccha list to create larger available memory blocks.

The `creategucche` function creates a sub-guccha list for a given main branch.
Create gucche for a given main branch (main branch has gucche which may have type hole or process and must add up to allotted page size)
Use size attr of main branch and make linked list of gucche accordingly then link head of linked list to main node -> subchain

The `mems_init` function initializes the memory space for the allocator.
Within this function, an initial memory space is allocated using the mmap system call, enabling subsequent memory management operations. The allocated memory is set to a predefined page size, typically 4096 bytes. The function then sets up the initial node of the main memory chain, referred to as BIGhead, which stores essential information about the memory block. Fields such as the size, remainder, actual pointer (ptr), and virtual address (virtual_address) are appropriately initialized.

The `createBIGnode` function creates a new node for the main memory chain.
Takes input as needed size and inserts a new node into doubly linked list of big nodes also finds remainder to check for holes during later allocs
Error handling done for mmap error

The `mems_malloc` function is used to allocate memory.
It iterates through the main memory chain and the sub-guccha list to find available memory for allocation.
This function is designed to allocate memory blocks based on the specified size requirements. It traverses through the nodes of the main memory chain, assessing the availability of memory space by examining the sub-guccha linked lists. By iterating through the sub-guccha nodes, the function identifies suitable locations for memory allocation, prioritizing segments marked as 'HOLE' (unused memory segments) and ensuring that the available space meets the requested size. Upon finding a suitable memory block, the function adjusts the memory allocation status, converts the identified 'HOLE' into a 'PROCESS' segment, and returns the virtual address of the allocated memory block. If no suitable memory block is found, the function triggers the creation of a new node in the main memory chain to accommodate the requested memory size, subsequently returning the virtual address of the newly allocated memory.

The `mems_print_stats` function prints the current status of the memory allocator, including the main memory chain and its sub-guccha lists.
This function traverses the main memory chain, printing detailed information about each memory block, including its starting and ending virtual addresses, as well as the types and sizes of the sub-guccha linked lists associated with each block. By displaying the memory allocation status in a structured manner, this function allows users to visualize the allocation patterns and the distribution of 'HOLE' and 'PROCESS' segments, facilitating a clear understanding of the memory management strategy employed by the custom allocator. Additionally, the function calculates and presents various statistics, such as the total number of pages used, the amount of unused space, the length of the main chain, and an array containing the lengths of the sub-chains. Overall, mems_print_stats serves as a valuable tool for monitoring and analyzing the memory allocation behavior and efficiency of the custom memory allocator.

The `mems_free` function is responsible for freeing memory.
It searches for the allocated memory block and changes its type to a hole, allowing it to be reused.
When invoked, this function locates the memory block corresponding to the provided virtual address within the main memory chain. It then examines the sub-guccha linked list associated with the identified memory block to find the specific sub-guccha node representing the allocated memory segment. Upon locating the designated node, the function changes its type from 'PROCESS' to 'HOLE', indicating that the memory segment is now available for potential future allocations. This process effectively releases the memory block for reuse, subsequently updating the total unused space and the remainder of the corresponding memory block within the main memory chain. By facilitating the seamless deallocation of memory segments, the mems_free function contributes to the efficient management and utilization of memory resources within the custom memory allocator.

The `mems_get` function returns a pointer to the memory location specified by the virtual pointer `v_ptr`.

When called, this function navigates through the main memory chain, searching for the memory block that encompasses the given virtual pointer. Once the corresponding memory block is identified, the function calculates the relative offset between the virtual pointer and the starting virtual address of the memory block, providing the actual memory location within the allocated memory space. By returning the adjusted memory location, mems_get enables users to access the precise memory content stored at the specified virtual address, supporting seamless data retrieval and manipulation within the custom memory allocator. This function enhances the accessibility and usability of the memory management system, allowing for efficient and accurate data handling within the allocated memory space.

The `mems_finish` function is used to deallocate all the memory used by the allocator, including the main memory chain and its sub-guccha lists.
This function systematically traverses the main memory chain, initiating the deallocation process for each memory block and the associated sub-guccha linked lists. By utilizing the munmap system call, the function deallocates the memory blocks, releasing the allocated memory resources back to the operating system. Furthermore, it ensures the comprehensive deallocation of the memory utilized by the allocator, including the main memory chain and its corresponding sub-guccha linked lists.