Virtual memory:

The memory allocator operates entirely within the virtual address space of a single process and knows nothing about which physical pages have been allocated to this process or the mapping from logical addresses to physical addresses; that part is handled by the operating system. So in this project you will be dealing entirely with virtual memory.

Memory Allocator:

When implementing this basic functionality in your project, we have a few guidelines. First, when requesting memory from the OS, you must use **mmap()** (which is easier to use than sbrk()). Second, although a real memory allocator requests more memory from the OS whenever it can't satisfy a request from the user, your memory allocator must call mmap() only one time (when it is first initialized).

Classic malloc() and free() are defined as follows:

- void *malloc(size_t size): malloc() allocates size bytes and returns a pointer to the allocated memory. The memory is not cleared.
- void free(void *ptr): free() frees the memory space pointed to by ptr. Otherwise, or if free(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed.

  So your functions should be implemented in similar way.

  mmap can be called as:

```
// sizeOfRegion (in bytes) needs to be evenly divisible by the page size
                void *ptr = mmap(NULL, sizeOfRegion, PROT_READ | PROT_WRITE,
    MAP_PRIVATE, fd, 0);
    if (ptr == MAP_FAILED)
                {
                 perror("mmap");
                 exit(1);
                 }
```

8-Byte Aligned Memory:

For performance reasons, Mem_Alloc() should return 8-byte aligned chunks of memory. For example if a user allocates 1 byte of memory, your Mem_Alloc() implementation should return 8 bytes of memory so that the next free block will be 8-byte aligned too. To figure out whether you return 8-byte aligned pointers, you could print the pointer this way printf("%p", ptr) . The last digit should be a multiple of 8 (i.e. 0 or 8).

Functions required:

In addition to functions specified in question you will need following functions to help implementation of required of functions:

1. To get size in pages(used for Mem_init for wrapping up size demanded in terms of page size)
2. To merge nodes(used in coalescing when you free memory)
3. To get previous of node
4. Get size in 8 bytes (used in mem_alloc)
5. Split and coalesce

Shared Library:

Beside easy linking of code there are further advantages to shared (dynamic) libraries over static libraries. When you link with a static library, the code for the entire library is merged with your object code to create your executable; if you link to many static libraries, your executable will be enormous. However, when you link to a shared library, the library's code is not merged with your program's object code; instead, a small amount of stub code is inserted into your object code and the stub code finds and invokes the library code when you execute the program. Therefore, shared libraries have two advantages: they lead to smaller executable and they enable users to use the most recent version of the library at run-time.