1. How would modify your code so that pages are saved to disk when memory capacity is exhausted?

Consider setting up a paging mechanism, customizing a page size, and implementing a swap strategy using the associated I/O functions. Use the previous pm_heap to represent each paging. Consider modifying the pm_malloc() and pm_free() functions to update the data in the paging system to do synchronized swapping and releasing.

2. How would you load previously saved pages back in?

Set up a swap function that is executed when the main function is called to a page that does not exist in memory to place the page in memory. As for the exchange strategy, LRU is used.

3. Does your design work for a virtual heap?

Yes, it works for a virtual heap.

4. Do you need to change your design?

No, I don't need to change my design.

5. What kind of file access would you use? Which C functions would you need?

sprint(), fprintf(), fwrite(), fread() and so on.

6. How would you detect when the heap is exhausted?

I consider tracking the lifecycle of each block of memory by intercepting calls to memory allocation and release functions. For example, each time a block of memory is successfully allocated, the memory allocation information (such as the pointer to it, file name, function name, line number, and number of bytes requested) is added to a global chain table; whenever a block of memory is freed, the corresponding memory information is then removed from the chain table. In this way, when the program ends, the remaining memory information nodes in the chain table correspond to those memory that have not been freed.

7. How would you know that you needed to reload a page?

A flag bit can be set to indicate presence on disk or presence in memory, and when the flag bit indicates on disk the load function is called to load it from disk back into memory.

8. What kind of page replacement strategy would you implement and how?

I might be going to use the LRU. If a piece of data has not been accessed in the recent past, it is very unlikely that it will be accessed in the future, i.e., the data that has not been accessed for the longest time should be eliminated when the bounded space is full of data. I'm considering a doubly linked list for the implementation.