

ED1 Memory Manager

# Overview | Objective

This lab is designed to show a simple implementation of a Memory Management system. It will test your familiarity with pointers, cyclic doubly linked lists, and problem solving skills.

All functions will be implemented inside the FS Student Game Engine, this mimics the real world scenario of working within an existing code base. Code will be written in MMHeap\_TODO.cpp and will consist of the following functions. Reading the class declaration in MMHeap.h is also advised.

Comment out the calls to our solution version and write your own implementation, when you do this you will also need to comment in the thread protection. If you change the flow of execution be sure to lock / unlock the heap appropriately.

To view memory usage metrics pull down the output console with “o” then use “u/i” to reach the memory page. To dump this information (and more), use the following command in the input console **MetricDump.**

To execute a core dump, open up the input console (~) and use the command **CoreDump**.

# Code Section

You will need to complete the following functions within the Heap class.

void Heap::Init(unsigned int poolSizeInBytes)

This function should allocate our memory pool using malloc and imbed the memory descriptors.

This function should also setup the internal Heap members.

Header\* Heap::FindBlock(unsigned int allocSize)

This function should find the first available memory pool that is larger enough to store allocSize. If it doesn’t find one large enough it should return null.

char\* Heap::Allocate(unsigned int allocSize)

This function should use the FindBlock() function to find a free memory segment.

If there is an available pool, you should detach it from the free list, mark it’s header and footer as used, and return a pointer to where the actual memory was allocated. Make sure that you update your free list appropriately, especially if you need to split the memory block into a two smaller blocks

void Heap::DeAllocate(void\* data)

This function takes in a pointer to the actual memory, so it needs to mark the header and footer for this block as free, and check to see if you can merge in either (or both) directions. Make sure to set up the cyclic linked list correctly.

# Rubric | Grading Breakdown

Commenting should be appropriate and used where necessary.

Your completed lab should be warning free.

Your completed lab should be error/crash free, a submission that crashes will result in a 0%.

Your project should be free of memory leaks.

Failure to follow directions will result in the loss of additional points.

|  |  |
| --- | --- |
| Init | 30% |
| FindBlock | 20% |
| Allocate | 50% |

The deallocate function is the lab assignment for your lab after the first test.

|  |  |
| --- | --- |
| Deallocate – Right | 40% |
| Deallocate – Left | 40% |
| Deallocate - Middle | 20% |

# Submissions

Lab is due at **the end** of the lab period. Have a lab instructor grade your assignment; you still need to turn in the assignment on VFILER, grades will be posted on LMS. If you are off campus, late assignments can be turned in through LMS.

This lab must be turned in using the .zip file format using as *LastName.FirstName.lab\_name.zip*.

To create the zip file, run the “make submission folder.bat” file. It will create a folder one level higher called “zip\_this\_and\_turn\_in,” leave it named that. Zip it, rename the zip file and turn it in on VFILER.