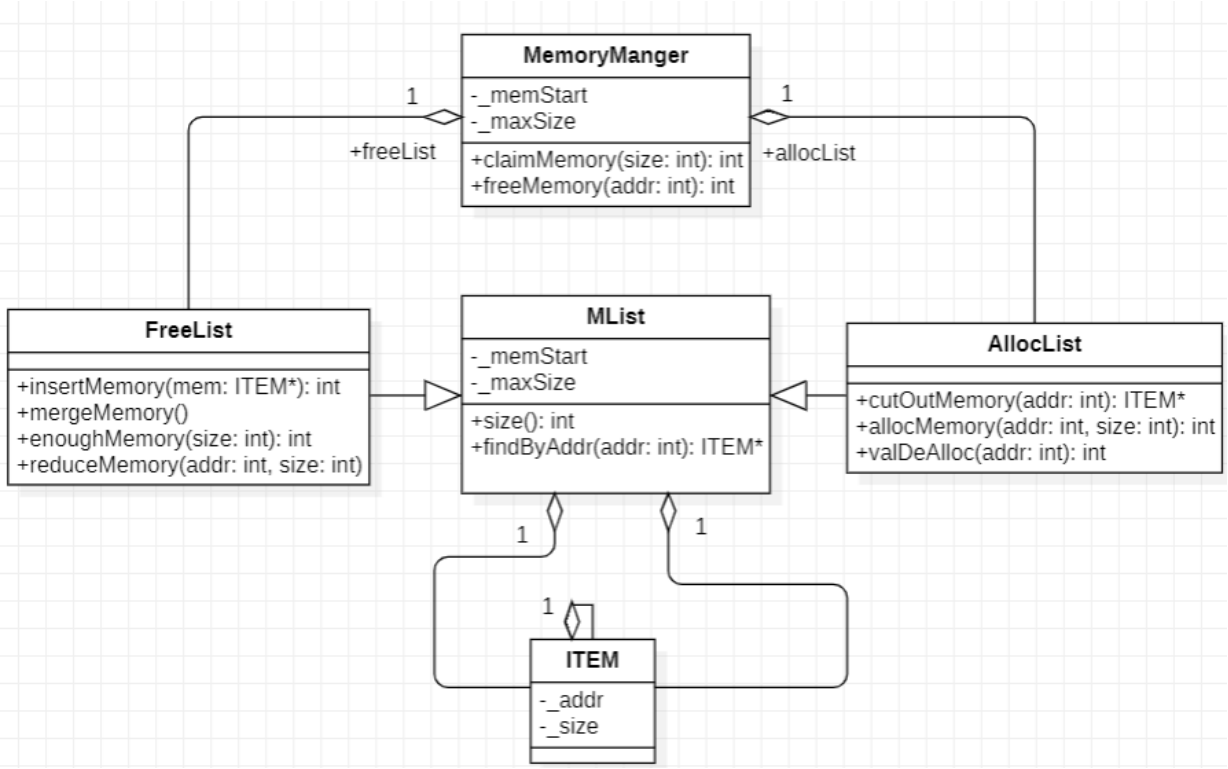
Memory manager assignment

Task description:

In this assignment we are going to implement a Memory Manager that is used by an Operating System to keep track of dynamic memory allocation (allocation from heap). In our case the equivalent to "malloc/free" will be called *claimMemory* and *freeMemory.* My task was to implement these functions and make it so that every requirement is met during this process.

Class diagram:



Memory Manager	
Main class, manages memory by using 2 lists. One FreeList and one AllocList. The system is able to use and handle memory operations through its simple interface.	
claimMemory(in size:int): int	Claims memory.
freeMemory(in addr:int): int	Frees memory.

FreeList	
LinkedList storing available memory in chunks. Its responsibility lies in managing all the free memory related operations.	
insertMemory(in mem:ITEM*): int	*Insert the cut out memory from the cutOutMemory function to the earliest possible place after the first element which has a larger address than the cut-out memory chunk.*

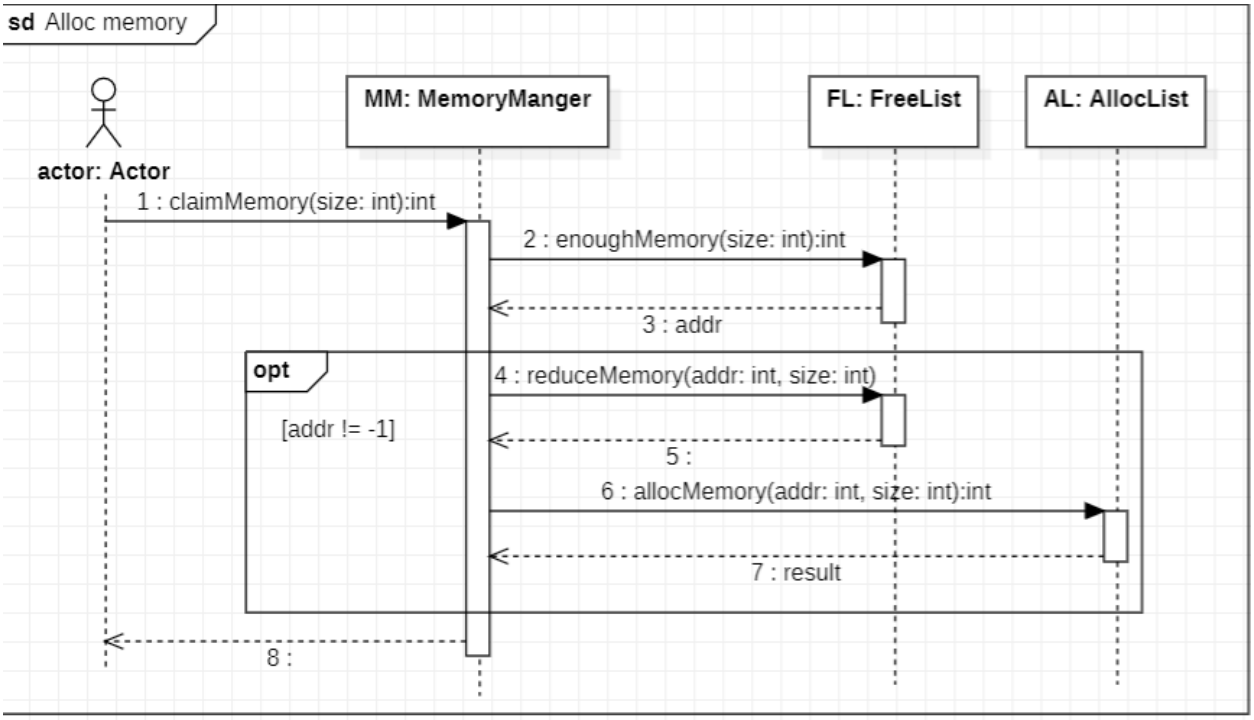
mergeMemory()	*With a while loop merges the free memory blocks until no more merging Is possible. If list size is more than 1 than this function runs. It checks the addr + size of the left list element than the addr of the right list element, if they are equal than merging can happen. This is done until all the elements in the array are checked.*
enoughMemory(in size:int): int	Checks that the given size in the parameter could be allocated. Iterates through the freeList and check that is there any element which has a max size >= to the given size of the parameret. If there is one than the address of that segment gets returned from the function, if there isn't any than -1 gets returned.
reduceMemory(in addr:int, in size:int)	From the return value of enoughMemory function the addr from where the available free space is accessible, it removes from that address a given size of memory and restructures the array in a new way. If space was left in the freeList element (allocSize is smaller than that elements maxSitze) then just make that element smaller. If size = maxsize than remove that element completely and rearrange.

AllocList	
LinkedList storing allocated memory in chunks. Its responsibility lies in managing all the allocated memory related operations.	
cutOutMemory(in addr:int): ITEM*	*After finding the address, it removes it from the allocList, returns it back as a return parameter from the function then connects the remaining linked list ITEMS so the list stays continuous.*
allocMemory(in addr:int, in size:int)	
valDeAlloc(in addr:int): int	Function which evaluates that the given addr could be freed. It returns 0 if it is a valid, otherwise -1.

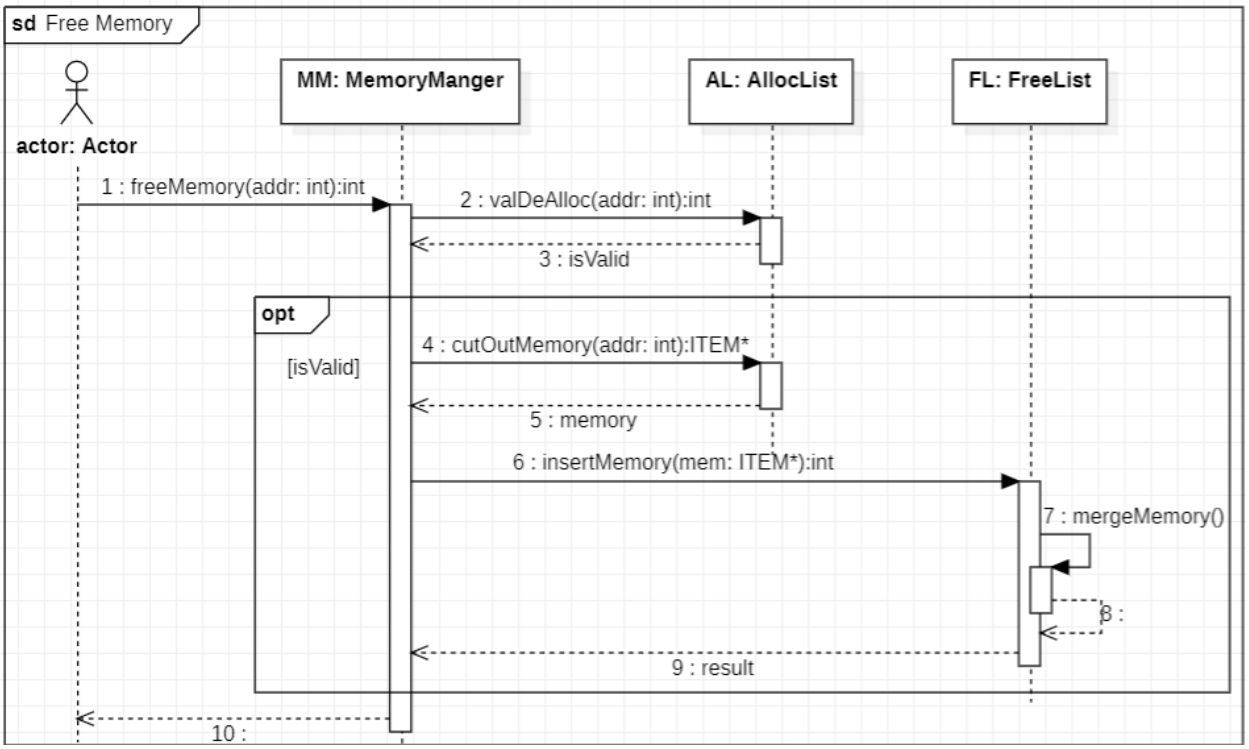
MList	
Common base class of the 2 before mentioned LinkedLists, has all basic list related functionalities.	
size(): int	Returns the size of the list
findByAddr(in addr:int): ITEM*	Finds an item by its address

ITEM
An elment representing one piece of a list.

Sequence diagrams:

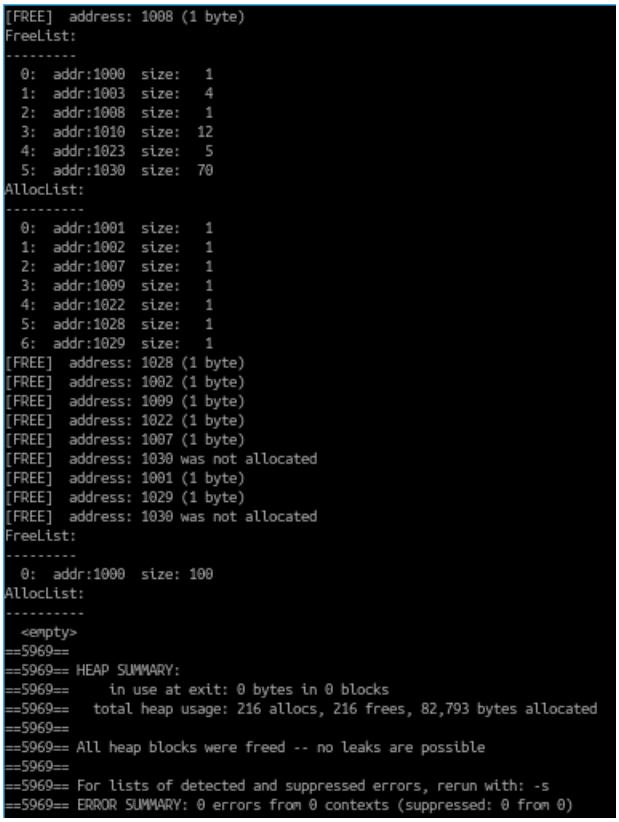


Sequence diagram visualizes the memory claiming process.



Sequence diagram visualizes the memory freeing process.

Valgrind stack:



```
[FREE]  address: 1008 (1 byte)
FreeList:
---------
  0:  addr:1000  size:   1
  1:  addr:1003  size:   4
  2:  addr:1008  size:   1
  3:  addr:1010  size:  12
  4:  addr:1023  size:   5
  5:  addr:1030  size:  70
AllocList:
----------
  0:  addr:1001  size:   1
  1:  addr:1002  size:   1
  2:  addr:1007  size:   1
  3:  addr:1009  size:   1
  4:  addr:1022  size:   1
  5:  addr:1028  size:   1
  6:  addr:1029  size:   1
[FREE]  address: 1028 (1 byte)
[FREE]  address: 1002 (1 byte)
[FREE]  address: 1009 (1 byte)
[FREE]  address: 1022 (1 byte)
[FREE]  address: 1007 (1 byte)
[FREE]  address: 1030 was not allocated
[FREE]  address: 1001 (1 byte)
[FREE]  address: 1029 (1 byte)
[FREE]  address: 1030 was not allocated
FreeList:
---------
  0:  addr:1000  size: 100
AllocList:
----------
  <empty>
==5969==
==5969== HEAP SUMMARY:
==5969==     in use at exit: 0 bytes in 0 blocks
==5969==   total heap usage: 216 allocs, 216 frees, 82,793 bytes allocated
==5969==
==5969== All heap blocks were freed -- no leaks are possible
==5969==
==5969== For lists of detected and suppressed errors, rerun with: -s
==5969== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Implementation:

I decided to choose a doubly linked list as my base link implementation, since it is faster, when well implemented easier to use and it was simply more interesting and challenging for me to do it that way. Later on I regretted my decision because 80% of my work on this assignment was debugging and hair pulling, writing to the console and stuff like that.

I first went ahead and started with the documentation, throughout planning, then class diagram, and sequence diagrams. By writing down what and how my functions should work beforehand I did not have to make any design braking architectural decisions when implementing the code.