Memory Manager Assignment

Terence Munro

Contents

1. Problem Statement

The goal of Question 1 was to create a dynamic memory manager class that can allocate/deallocate blocks of memory to the user when requested. For the purposes of this assignment we simulate our memory by using a character array on the heap.

The class needed to be able to:

- Allocate new blocks of memory returning NULL if there was not enough memory left
- Reallocate an old block of memory smaller or larger given a new size also returning NULL if the resize was not possible due to limited space.
- Free a block of memory for use again later.
- Return the amount of free space remaining.
- Defragment the memory array if it had become fragmented from use (function called Compact)
- Dump the memory contents to the screen for debugging/assessment purposes.
- Output the memory content to the output stream using the << operator

2. User Requirements

The following outlines the user requirements for the program:

The user should be able to:

- instanciate the class with as large initial memory block as they want to simulate with within reason (e.g. cannot go past the amount of memory they have or greater than the max size of a signed integer).
- retrieve the amount of available memory.
- allocate memory and use it for any type of data they want to put in it.
- reallocate an existing memory block to a smaller or larger size.
- retrieve a print out of all the memory in hexidecimal format.
- defragment the memory.
- perform a deep copy on the memory manager and its memory block.

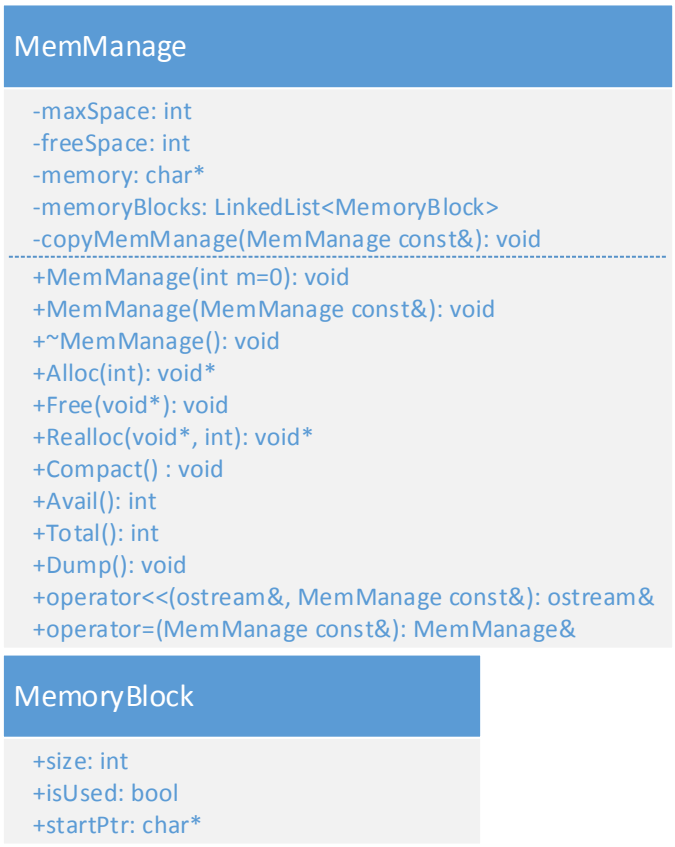
3. Software Requirements

The following outlines the software requirements for the program:

- The memory that is allocated must be private
- A linked list must be used to keep track of the memory blocks
- Alloc must return NULL if not enough space is available
- Realloc must return NULL if not enough space is available or if not possible due to fragmentation
- Dump must print to standard out the memory content formatted in hexidecimal.
- The << operator must be overloaded to allow users to get the memory content in a outputstream.

4. Software Design

UML Diagram



Function List

- copyMemManage(*mm*)
    - Private function: Performs a deep copy of inputted MemManage object
    - Special note: Deep copying the linked lists would result in the pointers stored in the information of each node to be a shallow copy thus still pointing to *rhs*'s internal memory space. The pointers are reconstructed to point to the correct memory space after copying the linked list.
- MemManage(*s*)
    - Constructor: initalizes the memory, the linked list and the variables.
    - Input parameter *s* refers to the maximum amount of memory that this class will manage defaulting to 0 if the user does not specify.
- MemManage(*mm*)
    - Copy Constructor: performs a deep copy of inputted parameter.
    - Inputted parameter must be of type MemManage
    - Calls copyMemManage(*mm*)
- ~MemManage()
    - Destructor: cleans up any memory allocated during the use of the object

- Alloc(*s*)
    - Public function: Creates a block of memory of *s* size.
    - Inputted parameter is an integer.
    - Returned value of the function is either a pointer to the memory location in private variable *memory* or NULL if allocation was not possible
- Realloc(*ptr*, *s*)
    - Public function: Resizes a block of memory or creates a new block and freeing the old block.
    - Inputted parameters are a managed pointer and the new size.
    - Returned value of the function is:
        - A pointer to the memory location in private variable *memory*
        - NULL if the allocation was not possible
        - NULL if the *ptr* was not allocated by this memory manager
- Free(*ptr*)
    - Public function: Frees a managed block of memory for reuse.
    - Inputted parameter is a managed pointer
    - If inputted parameter is not a managed pointer function has no effect
- Compact()
    - Public function: Defragments memory blocks
    - Function removes all unused blocks and moves the remaining blocks into the spaces so that all free space is at the end of the private *memory* variable.
- Avail()
    - Public function: Returns the amount of free space left in the memory
- Total()
    - Public function: Returns the amount of space the memory manager is managing
- Dump()
    - Public function: Prints memory content to standard out formatted as hexidecimal values.
    - Calls the << operator
- Operator<<
    - Public function overload: Exports the memory content as hexidecimal values to the output stream.
- Operator =
    - Public function overload: Deep copies the right hand object

- Calls the copyMemManage function

Data structures in the software

LinkedList

- **Name:** memoryBlocks
- **Type:** List
- **Purpose:** To track all the memory blocks that have been allocated and freed
- **Used in functions:** Alloc, Realloc, Free, Compact, Copy function, Constructers and Destructors

Struct

- **Name:** MemoryBlock
- **Type**: struct, data store
- **Purpose**: Store information about individual memory block. Stored in each node of the linked list.
- **Used in functions:** Same as linked list

Array

- **Name:** memory
- **Type:** character array
- **Purpose:** simulate internal memory of a computer and store all the data user puts in their allocated blocks

Detailed Design – Pseudocode for all non-standard and non-trivial algorithms that operate on datastructures

Alloc

```
If size > free space
        Return with NULL
Else if memoryBlocks are empty
        Create memory block pointed to the start of the internal memory
Else
        Search for an unused memory block that is big enough
        If not found and exists enough space
                Create new memory block pointed to the element after the last element in the list
        If found and larger than required
                Create new memory block set as unused that represents the remainder space and add it to the list
        If found and exactly the right size
                Set to used and return the pointer
```

If not possible
    Return with NULL

Realloc

Find pointer in memoryBlocks list
If not found or new size exceeds free space
    Return with NULL
If new size is the same as old size
    Return pointer
If new size is smaller than old size
    Create new memory block that is the difference between the old size and the new size set to unused
    Change the size of the current memory block
    Return pointer
If new size is larger than old size
    And it's the last element added
        Extend its size and return pointer
    Or if the following block is unused and is large enough to fit the difference
        Change the following blocks size to the difference between the needed extention and its current size and move its pointer
        Extend the current blocks size and return the pointer
    Or there is not enough space where it is but there is enough space elsewhere
        Alloc new space
        Move elements from old space to new space
        Free old space
If all else fails
    Return with NULL

Compact

Iterate through all the blocks of memory
    If the space is unused
        Add unused size to offset variable
        Delete unused block
    Else if offset > 0 *(and space is used)*
        Move starting pointer and its data back by the offset
        Null the old space

5. Requirement Acceptance Tests

Software Requirement No	Test	Implemented (Full/Partial/None)	Test Results (Pass/Fail)	Comments (for partial implementation or failed test results)
1	Retrieve the amount of available memory	Full	Pass	
2	Allocate memory and use it for any type of data they want to put in it	Full	Pass	
3	Reallocate an existing memory block to a smaller or larger size	Full	Pass	
4	Retrieve a print out of all the memory in hexidecimal format	Full	Pass	
5	Defragment the memory	Full	Pass	
6	Perform a deep copy of the memory manager and its memory	Full	Pass	
7	The memory that is allocated must be private	Full	Pass	
8	A linked list must be used to keep track of the memory blocks	Full	Pass	
9	Alloc must return NULL if not enough space is available	Full	Pass	
10	Realloc must return NULL if not enough space is available or if not possible due to fragmentation	Full	Pass	

Software Requirement No	Test	Implemented (Full/Partial/None)	Test Results (Pass/Fail)	Comments (for partial implementation or failed test results)
11	Dump must print to standard out the memory content formatted in hexidecimal	Full	Pass	
12	User can outstream the memory manager to get the memory content	Full	Pass	

6. Detailed Software Testing

Unit tests are only available in the 2013 project as the 2008 edition of Visual Studio uses a different framework and it would have been to much extra time to convert the syntax

No	Test	Expected Results	Actual Results
1.0	**Unit tests**		**Screenshot provided in appendix**
1.1	MemManage_AllocatesMemory Start MEM_SIZE = 16 - Alloc (sizeof int) - Alloc (sizeof float) - Set int to 10 - Set float to 1.2	Ptr != NULL Avail() = MEM_SIZE – (sizeof int + sizeof float) Outstream << m = 0A 00 00 00 9A 99 99 3F 00 00 00 00 00 00 00 00\n	All as expected
1.2	MemManage_FreesMemory - Alloc pointer - Set value to pointer - Check size of mem manager is as expected	Size = starting size – allocated size Printout is hexidecimal equivalent of set value with 00 for unused and null pointers Size = starting size again after free Printout is all 00s	All as expected

No	Test	Expected Results	Actual Results
	- Check printout is as expected - Free pointer - Check new size is as expected - Check printout is as expected - Check ptr is NULL	Ptr == NULL	
1.3	MemManage_ReallocatesMemory - Allocate pointer - Realloc pointer	Pointer != NULL Check printouts	As expected
1.4	MemManage_Compact - Fill up memory manager - Free some holes - Run compact	Check printout is as expected Compare printout after compact	As expected
1.5	MemManage_DeepCopy - Full example given below	Full example given below	As expected

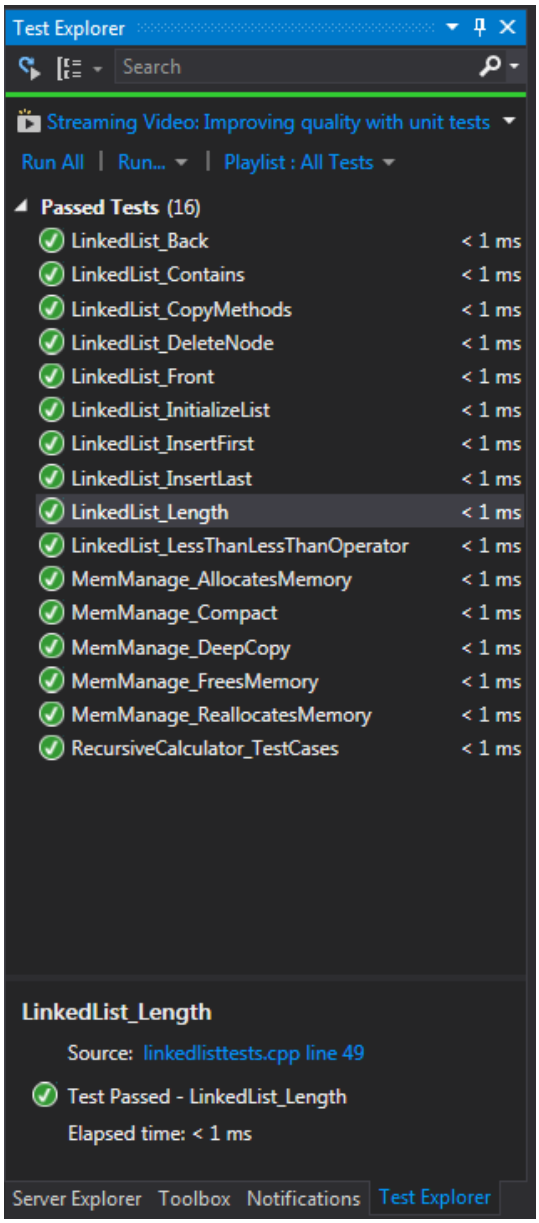
Example unit test:

```
TEST_METHOD(MemManage_DeepCopy)
{
	MemManage m(16), cpy(m);
	stringstream s1, s2;

	Assert::AreEqual<int>(16, m.Avail());
	Assert::AreEqual<int>(16, cpy.Avail());

	char* str = (char*)m.Alloc(6);
	Assert::AreEqual<int>(10, m.Avail());
	Assert::AreEqual<int>(16, cpy.Avail());
	sprintf_s(str, 6, "hello");

	cpy = m;
	Assert::AreEqual<int>(10, m.Avail());
	Assert::AreEqual<int>(10, cpy.Avail());

	s1 << m;
	s2 << cpy;
	Assert::AreEqual<basic_string<char>>(s1.str(), s2.str());
}
```

7. User Instructions

- User can load up the project using the Visual Studio 2008 solution file in the same directory as this Documentation.
- Statically compiled executable is available in the Release folder
- If the user wanted run user tests themselves they need to use Visual Studio 2013 and load up the VS2013 solution in a folder external to the question 1 and 2 folders labelled VS2013

8. Appendix



```
Free Space = 46
7A 65 72 6F 00 6F 6E 65 00 74 77 6F 00 74 68 72
65 65 00 66 6F 75 72 00 66 69 76 65 00 73 69 78
00 73 65 76 65 6E 00 65 69 67 68 74 00 6E 69 6E
65 00 74 65 6E 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00

Free Space = 39
7A 65 72 6F 00 6F 6E 65 00 74 77 6F 00 74 68 72
65 65 00 66 6F 75 72 00 66 69 76 65 00 00 00 00
00 73 65 76 65 6E 00 00 00 00 00 00 00 6E 69 6E
65 00 74 65 6E 00 73 69 78 74 65 65 6E 00 65 69
67 68 74 65 65 6E 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
Free Space = 65
7A 65 72 6F 00 00 00 00 00 74 77 6F 00 00 00 00
00 00 00 66 6F 75 72 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 74 65 6E 00 73 69 78 74 65 65 6E 00 65 69
67 68 74 65 65 6E 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
Free Space = 40
7A 65 72 6F 00 6E 69 6C 00 74 77 6F 00 00 00 00
00 00 00 66 6F 75 72 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 74 65 6E 00 73 69 78 74 65 65 6E 00 65 69
67 68 74 65 65 6E 00 74 77 65 6E 74 79 00 73 65
76 65 6E 74 79 20 74 68 72 65 65 00 00 00 00 00
00 00 00 00
Free Space = 37
7A 65 72 6F 00 6E 69 6C 00 74 68 72 65 65 00 00
00 00 00 73 65 76 65 6E 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 74 65 6E 00 73 69 78 74 65 65 6E 00 65 69
67 68 74 65 65 6E 00 74 77 65 6E 74 79 00 73 65
76 65 6E 74 79 20 74 68 72 65 65 00 00 00 00 00
00 00 00 00
Alloc(50) returned 00000000

Free Space = 37
Mem:
7A 65 72 6F 00 6E 69 6C 00 74 68 72 65 65 00 00
00 00 00 73 65 76 65 6E 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 74 65 6E 00 73 69 78 74 65 65 6E 00 65 69
67 68 74 65 65 6E 00 74 77 65 6E 74 79 00 73 65
76 65 6E 74 79 20 74 68 72 65 65 00 00 00 00 00
00 00 00 00
Copy:
7A 65 72 6F 00 6E 69 6C 00 74 68 72 65 65 00 73
65 76 65 6E 00 74 65 6E 00 73 69 78 74 65 65 6E
00 65 69 67 68 74 65 65 6E 00 74 77 65 6E 74 79
00 73 65 76 65 6E 74 79 20 74 68 72 65 65 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
Press any key to continue . . .
```