.TH P2 "Memory Management"

.SH NAME
.P
P2 - Memory Management

.SH SYNOPSIS
.P
The main purpose of this project was to learn about implementing a memory manager in c++ whose features include initializing, tracking, allocating, and deallocating sections of memory.

.SH DESCRIPTION
.P

From MemoryManager.h :

.RS
I first created a header file which included three separate sections. The first section was a listNode struct which was used to characterize sections of memory as nodes with a value representing its size, a value for its position in memory, and a variable stating if the node was a hole or used. The next section included the MemoryManager class declarations, where the MemoryManager class is responsible for allocating/deallocating memory. Finally, there were the allocator algorithm declarations which were separate from the MemoryManager class.
.RE

From MemoryManager.cpp :

.RS
Within the MemoryManager.cpp file, initially the allocator algorithms for bestFit and worstFit were defined. The rest of the file was dedicated to defininig the methods of the MemoryManager class. The consturctor initializes the algorithm type used and sets the length of word used. The destructor calls the shutdown() method which releases (empties vectors) all memory associated with the initialized memory block. The initialize function checks that the requested size is valid and if so creates a memoryBlock and stores the starting point of the block for tracking memory. As mentioned above, shutdown() releases all memory initialized for tracking. The allocate method loops through the vector of nodes to find a suitable hole for allocation. It then checks to see if there is extra unused space and if so it adjusts the memory for later allocations using splitNode(). The free method finds the specified node based on the memory address and deallocates it for later use. This function also checks for adjacent holes and merges them as necessary. The setAllocator method simply sets the algorithm type used as bestFit or worstFit. The dumpMemoryMap method uses POSIX calls to write the list of holes to a new txt file. The flags O_CREAT and O_WRONLY were used for this. The getList method returns a vector that contains the holes in memory. The getBitMap method returns a stream of bits in a vector showing if the words are used(1) or free(0). The other get functions are simple functions with return statements to get information about the wordSize, start, or limit of the memory block created.

.RE

.SH TESTING
.P
After implementing all these methods I tested that the code compiled using a makefile which included the following commands and built the library functions:

```
MemoryManager: MemoryManager.cpp MemoryManager.h
      g++ -c MemoryManager.cpp -o MemoryManager.o
      ar cr libMemoryManager.a MemoryManager.o
```

After testing that this worked I linked the CommandLineTest.cpp file to the library using the following command:

```
c++ -std=c++17 -o program_name CommandLineTest.cpp -L ./MemoryManager -
lMemoryManager
```

Following this command, I ran the test file using ./program_name and found that I passed all test cases. To test for memory leaks however, I needed to use the valgrind software. After installing valgrind, I ran the following command to test for memory leaks:

```
valgrind --leak-check=yes ./program_name
```

Following this, I found that all allocations had accompanying frees and there were no memory leaks possible.

.SH LINK
.P
https://youtu.be/d7odRYRuxys

.SH REFERENCES/CITATIONS
.P
https://ufl.instructure.com/files/74718896/download?download_frd=1
.P
https://www.cs.rit.edu/~ark/lectures/gc/03_00_00.html

.SH AUTHOR
.P
Michael Gerber