Group:

David Ha	:	dth69
Michael Martinez	:	mbm193

Test Plan:

a) For our library to be correct the mymalloc function must be able to take a given size and if possible allocate a memory block of that size to the heap memory. It is important that it checks the memory range as well as whether or not certain memory is already allocated or if it is free.

   Our myfree function must be able to take a pointer and use it to search through the allocated memory to check if the given address has been allocated or not, and if so to deallocate that memory. When free deallocates a block, it must also coalesce and merge the memory blocks that were on its head or tail in order to allow us to free that space and use it later.

   The header file must be able to take malloc and free and reference it to mymalloc and myfree respectively. We must also declare all of the methods that it will call from our mymalloc program.

b) To test whether memory blocks are being allocated we will need to check whether the current node has a specified block size given to it from one of our arguments. To check whether that block is allocated our current node must also have a Boolean type check to see if that chunk has been freed or if it still allocated.

   In our myfree function we will check those nodes for information on whether to proceed with deallocation or to return the specific feedback on why it is not possible. In the event of a null pointer, we cannot return anything. The myfree function will also handle coalescing to ensure that memory blocks are removed from our struct by merging the ends of the blocks it is connected to after being freed.

c) A printList() function will allow us to create a full list of all the chunks allocated in the heap memory, including the block that is free and the size available. This will print out a message with the chunk number and its size, whether it has been freed or not, and the address of that block. This function is only used when debugging mymalloc through its own arguments and is not necessarily called via our header file.

The printList() function iterates through each node from our head until the current node is NULL which indicates the tail has been reached. As it iterates through each node in the structure it prints the following code:

```
("Chunk %d: %d Free %d Address %p\n", count, curr->BlockSz, curr->free, curr->start_address)
```

- The count is the current position of the block given as an integer starting from 1.
- The curr->BlockSz node information gives us the total size that memory block has allocated.
- The curr->free node information is a Boolean check to see whether the block is allocated or if it is free. In this check we want to make sure that it is allocated, otherwise there has been an issue while coalescing our blocks.
- The curr->start_address node will give us the address of that node.

Messages are also printed out for any checks that our arguments may have failed to pass in mymalloc or myfree. These checks in mymalloc include whether there is enough memory available within the memory range when given a size, if that size is a valid size, which in the case of size_t is any integer not 0. In myfree the checks include whether there has already been memory allocated, if so it searches through each chunk to see if the pointer address exists and whether or not it has already been freed, and finally if the pointer does not exist after going through each memory block node.

d) For our first custom stress tests we chose to do a combination of the previous tests where we include a random check to decide whether to immediately deallocated a memory block or not. And whatever memory has not been deallocated immediately will be deallocated after reaching 120 total mallocs.

For our second custom stress test we did a simple 2-d array allocation and deallocation which would take twice as long as the 2nd given stress test where we allocate 120 blocks and then deallocate them after.