Shina Kolisetti
Richard Gerdes
C214 Systems Programming

Assignment 6 : Error detecting malloc() and free()

The program we implemented uses malloc() and free() library calls for dynamic memory allocation. We used the static char array that description provided MEMBLOCK_SIZE (5000).

For the assignment we were supposed to implement memory allocation and memory freeing. We have two methods:-
memMalloc which takes in a unsigned int and and returns a pointer to a block of the requested size.
memFree which takes in a void pointer that is returned from memMalloc and then frees the malloced space.

For extra credit

Structure
malloced memory is prefaced by a header. these headers are connected by a circular linked list.

Implementation
malloc() - looks for the first empty piece of memory starting at the end of the section of memory to be used. This is easy to achieve since the implementation used a circular linked list structure and the last block of memory is linked to the first. When a block of large enough size if found memory is allocated from the end of so that the list does not need to be modified as much. By going through the list in reverse, we will reuse large memory blocks that have been freed.

free()- when freeing a pointer, multiple checks are performed to confirm that the address passed in is valid. the first check is whether the pointer passed in falls in the range of address that are used by the program. if not, no memory is freed and the function returns. The next check that is done if for a valid block, and to protect from double frees. This check confirms that the pointer is in fact preceded by a valid header and that it is part of the link list by checking that pointers in the header are valid and point to other valid headers before and after the block. It also confirms that the processing and successor headers point back the that block. If one of these checks fails, the block of memory is either invalid, already freed, or had been corrupted by overwriting the end of another block. Once it had been confirmed that the pointer passed in is that start of a valid inuse memory chunk and that it has a valid header, the block is marked as free and removed from the linked list.

calloc() - the calloc function is simple. It mallocs a new chunk of memory and then memsets its value to all 0's.

realloc() - confirms that block passed in is valid. It then checks to see if the memory block is followed by a free block that is large enough to be included in the block or shifted to allow the block being resized to be expanded to include that range. If it can be, the memory block is extended and the original pointer is returned. If the block can not be extended, then the function mallocs a new block of the required size and copies the contents of the old block over. The old block of memory is then freed and the new block is returned.

printStats() - print stats counts the memory blocks and sums the amount of memory that has been stored by the program. It also outputs the given block count for both free and in use blocks.

printList() - the print list function prints out a list of all valid memory blocks and given data.

intiMemoryBlock() - on the first call, this function initiates the memory block and sets up the headers for the initial free chunk.

Features:
leak checking:

the function *printStats()* finds the amount of bytes that are both free and in use. it displays the count of bytes for each as well as the count of how many blocks that data is in. The functions also recognizes blocks of data that have been lost due to corruption

the *printStats()* function has been registered with atexit so that the stats are displayed once the program exits.

run protection

the program has been designed to not be dependant on which function gets called first. on the first run of each function, they call a initialize function (*intiMemoryBlock()*) that configures the header of the memory block so that it can be used or traversed by the other functions.