Ferris Hussein

Assignment 1: Malloc

Program Description:
In this project I implemented a static char array as our main memory to be allocated and freed as needed. I tried to be as space efficient as possible and by doing so I created the metadata to store the size requested as a short. This data was stored in a struct called “miniblock” which had a size of 2 bytes. I used negative and positive values to distinguish betIen whether a particular block has been used or not. Positive meaning it has been used or allocated and negative meaning it has not been used.

Implementation of mymalloc():
Because I only have a short as part of our metadata, the maximum amount of space a user can request is 4094 bytes. In the function mymalloc, I first start by doing an error check. I check whether the requested size (which is stored as an int) is properly requested. Meaning, the user cannot request space less than 1 byte or greater than 4094 byted. If so, I print a Malloc Error. If this condition is passed, I create a pointer “ptr” of type miniblock and initialize the main memory by declaring the size as -4094 where the negative indicates that the memory has not been used. I then loop through using “ptr” to look for the first available space that can be allocated. If there is no available space, I print a Malloc Error. If there is available space, I check whether I can split it or not. If the block cannot be split I allocate and if it can be split it then I allocate accordingly.

Implementation of myfree():
The function myfree, takes in a void pointer to a specific place in memory that needs to be freed. I start traversing through the char array and locating the block to be freed. If I have fully traversed through the entire char array and I are not able to find the block to be freed, I print a Free Error. I also check if the pointer is already not being used by checking its sign. If the pointers size is already negative, I do not need to free the pointer and I print a Free Error. If the pointer block has bypassed all the error checks I then negate the size of the pointer indicated that the block has been freed. After freeing this block, I then check whether the previous block and next block is also free. If the previous block is freed and the next block is free I merge the previous block, the pointer block, and the next block together. If the previous block is free and the pointer block is free but the next bock is not free, I merge just the previous block and the pointer block. If the next block is free and the pointer block is free, but the previous block is free I merge the pointer and the next block. Though, if both the next block and previous block are both not free, no merging occurs.

Implementation of print():
The print function starts off with an error check to see whether the main memory is initialized. If not I print Print Error and return the function. I then traverse through the entire main memory and I print each block.

Workload Data and Findings

In the file memgrind.c, I tested several workloads as well as workloads of my own. Testcases A-D were required by the assignment, testcases E-F are two workloads of my own making, and I included an extra testcase G that counted the number of times I can malloc 1 byte. For testcases A-F, I iterated through each 100 times and recorded and printed the amount of time each took in seconds. I also recorded and printed the time it took per each iteration of each testcase as well.