PARAS PATANGE: u7291471

The following is in relation to the Dynamic memory allocator which allocates and deallocates memory in explicit free lists, with the use of headers and footers to keep track of allocated and unallocated free blocks using the first fit placement method. My implementation also contains simple time coalescing.

Data Structures:

- Structs:
    - Header : A data type for the header, and contains size,metadata,allocation status and neighbors.
    - Footer: A data type for the footer, and contains size, and allocation.
    - Full_block : A data type which contains information about the block specifically the if its been allocated and its size.
    - Boundary_tag: A data type to improve coalescing contains size, and allocation.

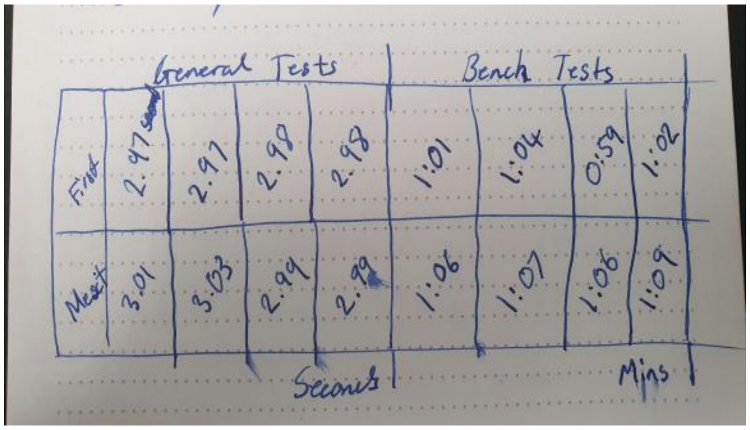
When the user makes a request to my_malloc, the memory allocator searches through the free list and finds the first block which is free (based on requested size), according to the assignment specifications we are asked to round to the nearest multiple of 8(and include enough space for the header and footer of the block), If a block is found which is free the a block pointer to the content is returned. Once the block is found then the memory is assigned and then using the block to data function the address is stored. If no free space is found then, it will return Null, an extension which I thought would be better was to just extend the heap. More accurate details of what each function does is found on the code itself as comments above the respective functions.

I have implemented linear coalescing, which basically uses the simple logic that if in memory, the current block that you are at + the size of the header of the block is equal to the next block's address, that block is then determined to be a neighbor of the current block. After Which it is coalesced/linked which minimally decreases the external fragmentation. The extension to that is constant time coalescing which I have tried to implement however may not function properly as the footer struct may not be containing all relevant information. The general idea of it was to create a header and footer to each chunk of data, which would be stored. When constant coalescing was to be performed, the constant coalescing function would first remove the blocks from their list, then they are joined together based on the 4 cases (Free block to the left and right, Free block to the right, Free block to the left and no free blocks on either side). I have also commented out boundary tags as they were not functional (Attempted code can be seen at bottom of assignment). I also attempted to use boundary tags in my constant time coalescing function. However due to poor time management I wasn't able to properly debug and implement.

Optimisation Summary:
I have not included, however I tried to implement some of the following:

- Multiple Free Lists:
  Attempting to implement multiple free explicit lists, my idea in doing such was to create multiple segregated linked lists, in which specific lists would store certain chunk sized data, for example there could exist a maximum amount of lists which could all have their callable chunk limits.
- Boundary Tags:
  I tried to implement a better form of coalesce via boundary tags, and I have used footers.
- Constant Time Coalescing:
  I attempted to implement this, using boundary tags and the logic mentioned above/in the code.

Placement Policies:
I experimented with First-Fit and Next-Fit, both indicated fast behavior, however experimentally the faster one was First fit. I chose not to include Best-Fit in the test as its nature was to search through all of the list and find the best slot for memory, which is far slower in practice than Next and First Fit. Below are some real world timings and tests that I performed to measure time taken and which to implement in my Dynamic memory Allocator. (Note: Bench tests weren't working, as such I measured time to fail the test). Another observation I made apart from tests, was the existence of an infinite loop, this causes super long bench times and some segmentation errors.



	General Tests				Bench Tests			
First	2.97 secs	2.97	2.98	2.98	1:01	1:04	0:59	1:02
Next	3.01	3.03	2.99	2.99	1:06	1:07	1:06	1:09
	Seconds				Mins			

Challenges Encountered:
The challenges I faced along the way were related to optimisation, due to some improper time management I was unable to properly attempt some of the tasks even with some ideas. A challenging aspect for me was the coalescing in constant time.I feel like I had the right Idea however I didn't have the time to implement it. I also spent a significant portion of my time researching ideas and how to implement them, specifically Free lists and Constant TIme Coalescing.