Project 1 Malloc Library

Design:

As part of this project we need to implement two major system functions (i.e. malloc() and free()). To this I have stored all the free heap spaces as a doubly linked list with the head being **free_head**. This list also holds required metadata (like the len of the memory block).

Malloc():

Both **ff_malloc()**, and **bf_malloc()** have similar implementations. The only difference is the way we chose the free block in the list. Therefore I am going to explain the malloc function together in the report. There are two ways to allocate the requested size.
1) existing free space
2) allocating extra free space through the system function sbrk()

We are going to choose option 1 wherever possible to avoid leaving a huge memory footprint by our function. Therefore we are going to iterate through the linked list to find free space which is greater than the size requested. (Note here is the only place where there is a difference between **ff_malloc()** and **bf_malloc()**. For ff_malloc we stop at the first free space where we find extra space. Whereas for bf we iterate throughout the list to find the best fit free space (i.e. the extra space after allocation should be as minimal as possible)).

If we do find a free space where it satisfies the size condition, we will reallocate the extra space if any back to the free list (i.e **split_extra_space()**). We do this by creating a new metadata for the extra space and inserting this new metadata into the free list in the appropriate position. After doing this we remove the current free space from the free list and allocate it to the caller function (i.e **allocateFromFreeSpace()**).

If we are not able to find any free space in the list which meets our criteria, we have no choice other than to expand the heap and allocate the new free space (i.e **expandHeap()**). To do this we use a system function called sbrk() and we request for size_requested_by_user + metadata_size. We fill in this metadata with information like the len (or the space that is being allocated in this block).

Free():

Both ff_free() and bf_free() have the exact same logic. Therefore going to treat it as one. Free function is going to check the metadata of the memory block and is going to assign this memory block to the free list. After allocating the free list, we are going to see if the newly allocated memory block can be coalesced into either the previous free block or the next free block or both.

Performance Report:

Best Fit:

Small Range:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
Execution Time = 0.003733 seconds
Fragmentation  = 0.077215
```

Equal Size:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 22.199386 seconds
Fragmentation  = 0.450000
```

Large Size:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 56.497395 seconds
Fragmentation  = 0.041318
```

First Fit:

Small Range:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
Execution Time = 0.004131 seconds
Fragmentation  = 0.099130
```

Equal Size:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 22.039864 seconds
Fragmentation  = 0.450000
```

Large Size:

```
[ka266@vcm-30704:~/ece-650/ece-650-project1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 42.600430 seconds
Fragmentation  = 0.093421
```

Analysis of Performance Report:

There are two basic trends in the performance Report.

Comparison between Best Fit logic and First Fit logic:

Since Best Fit logic finds the free space which is the most suited memory block, the fragmentation is better than First Fit logic (Although in Equal size allocation it doesn't matter because both first fit and best fit are one and the same). However this comes at a cost. Since it has to iterate through the free list the execution time is worse when compared to First Fit logic.

Comparison between random allocation and equal size allocation:

During random allocation the memory requested could be of lower size or a higher size. This gives various pocket sizes of free spaces. Therefore it is far more likely to fit a small memory block in existing free space. However, in the equal size allocation it is one to one fit . Therefore in most cases you cannot allocate more than 1 in a free space memory block. Therefore the fragmentation is more in equal size allocation when compared to random allocation.