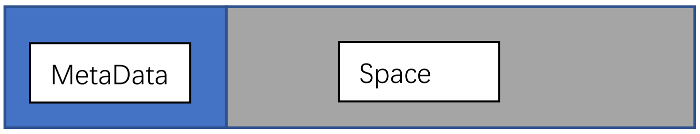
1. Implementation

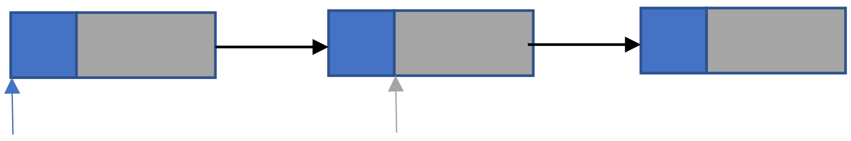
1.1. High Level

In the high level design, I use a singly LinkedList to implement the FreeList and Use sbrk() to increase the heap size. In the FreeList Node, I name it as node_t, which is composed of a MetaData and the useable Space.



1.2. FreeList

In the FreeList, the List will be sorted by the blue pointer, which replace the address in the heap. And the grey pointer is what we will return to the user.



When I want to free a block, I firstly, add it to the very front of my Free List, And Iterate over the whole List to find a suitable position. Once I find the position, I will return the pointer to the former node. Then I will check if I can merge the newly added node with the former one or the next one.

1.3. Malloc

When I want to malloc a space, I will first check if I can find a usable node. If I can find one, I will consider returning the block or splitting the block. If after splitting, the remaining space is not enough for the sizeof(node_t), I will just return the node, else, I will split the node and split from the tail of the node, then I will return newly split node pointer.
For the FF: I will just return the first found node.
For the BF: I will iterate to find the best, if I find one which can not be split, I will return immediately return this node.

2. Result

The Test Result is just like the following Figure. (you can find the Res.txt in the GitHub Repository).

And the Test Environment is i7-6700HQ, in the Virtual Box virtual machine. The Operating System is Ubuntu 20.04, using gcc, make to compile the code. The editor is Emacs, which has the same configure with ECE 651 VM.

```
FF
cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 37.260579 seconds
Fragmentation  = 0.450000

cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 115.189351 seconds
Fragmentation  = 0.094559

cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3707312, data_segment_free_space = 275504
Execution Time = 35.572876 seconds
Fragmentation  = 0.074314


BF

cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./equal_size_allocs
Execution Time = 36.448824 seconds
Fragmentation  = 0.450000

cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./small_range_rand_allocs
data_segment_size = 3543328, data_segment_free_space = 93872
Execution Time = 8.637845 seconds
Fragmentation  = 0.026493

cales@cales-VirtualBox:~/Desktop/ECE_650/hw1/my_malloc/alloc_policy_tests$ ./large_range_rand_allocs
Execution Time = 139.230253 seconds
Fragmentation  = 0.042504
```

For the Equal one:
The result in BF and FF is almost the same. As the Equal Test will malloc the same size node and it may not result in any difference.
For the Small one:
The result in BF is much better than the FF. In my opinion, I believe this arises from that every newly malloc size range is very little, so FF will just return the first found one, but this strategy will result in the too many fragments. However, the Best will avoid this problem and so it leads to less time and Fragmentation.
For the Large one:
The result in BF is slower than FF, while it has less Fragmentation. The less Fragmentation is trivial, as BF will obviously lead to this result. For the slower time, I believe this is because BF strategy will not result in the faster find process. As the range between different malloc is very large so when I want to find a suitable node, I will not save the time. And As the BF process will lead to more time, So the BF will leads to the More Time.

In Real Application, I think when I want to malloc some space with large range, I will use the FF strategy. When I want to malloc some spaces with small range, I will surely choose the BF strategy. When I want to malloc some spaces with equal range, it seems no different between two strategy. And I will implement FF strategy as this strategy is more easily to code and there is less error.