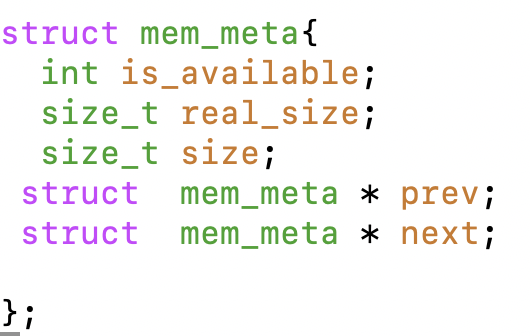
ECE 650 Assignment 1: Malloc Library Part I

Yuchen Yang(yy227)

This report is a study of the implementation of “malloc” function in C library. There are mainly three parts contained, function implementation, experimental results, comparison and analysis.

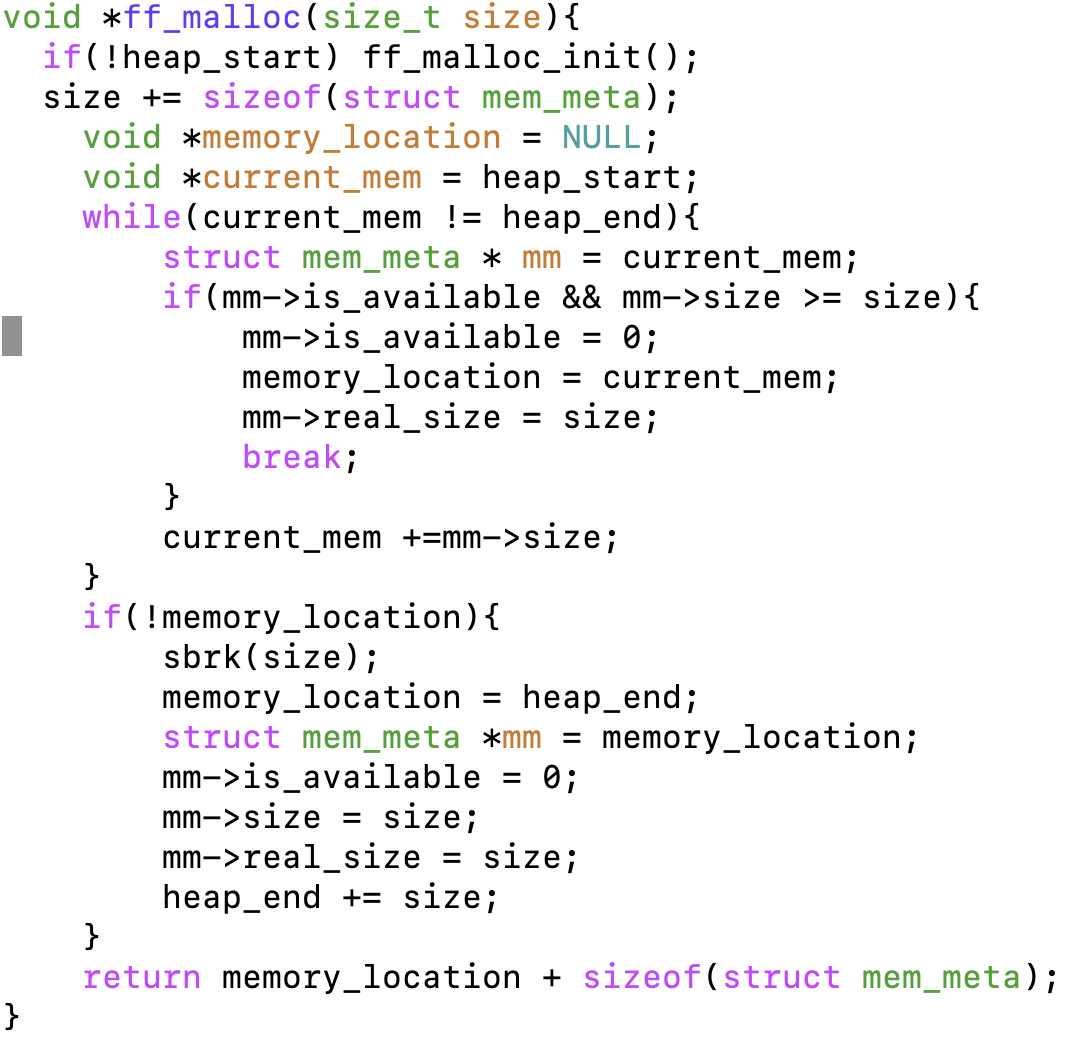
1. **Function Implementation**

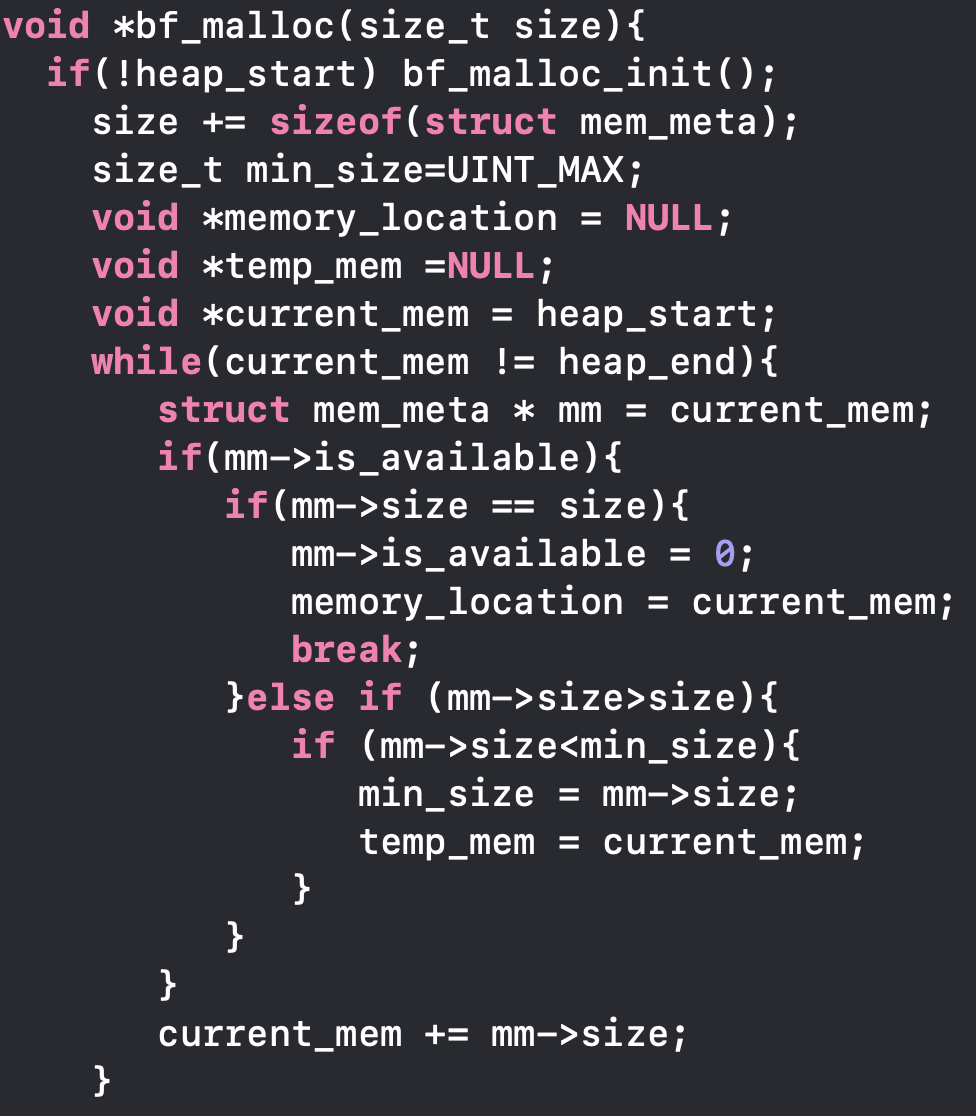
The “malloc” function is designed to allocate memory in the heap until reaching the end of the heap. Regarding adding new chunks freely and repeatedly, linked list would be a good fit to implement the data structure. However, it is necessary to keep track of every chunk of memory allocated and the availability of it. In this case, I added a head node, which contains all related information.

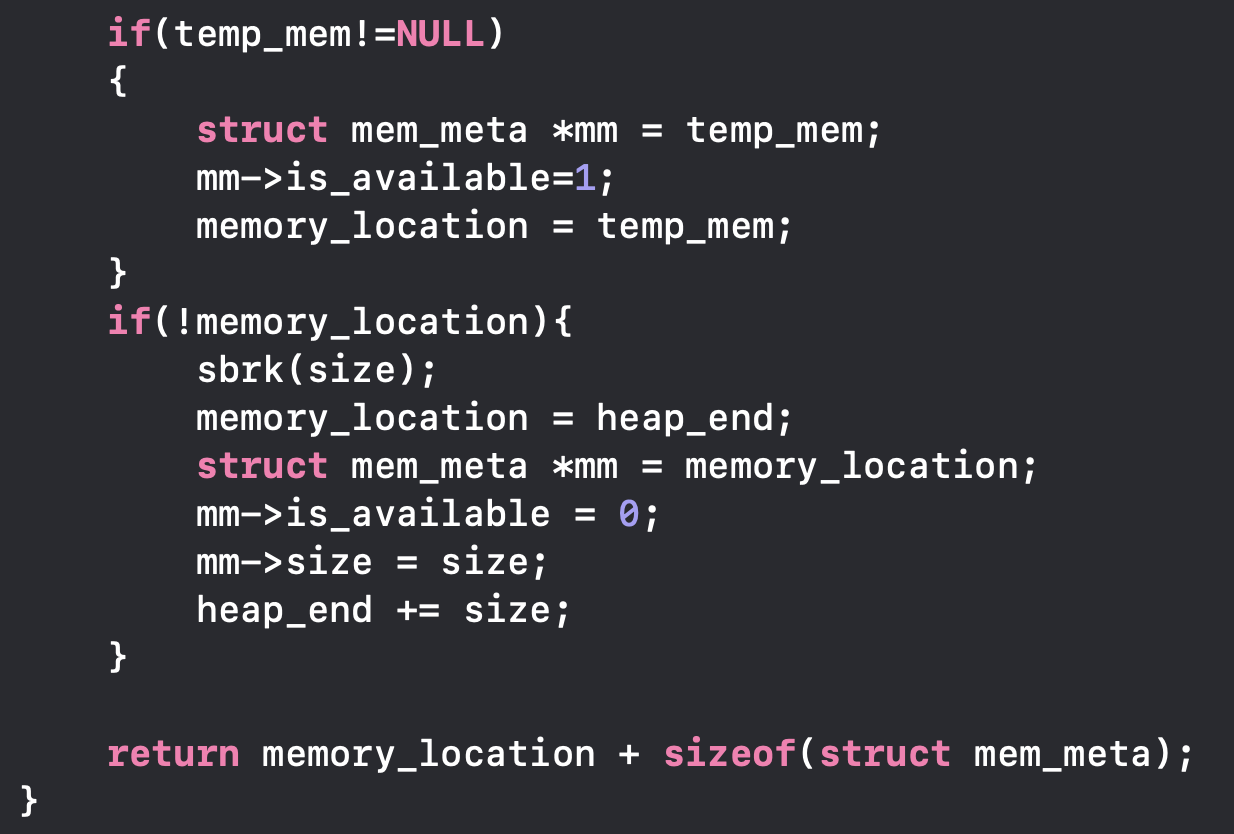


Next step is to find the place to insert the target chunk. First, we should go over the used part—the part of heap which was used by “malloc” before but may be freed later and see if we can fit the target chunk into this area. If nothing available, we invoke “sbrk()” call to make up more space for “malloc” to use.

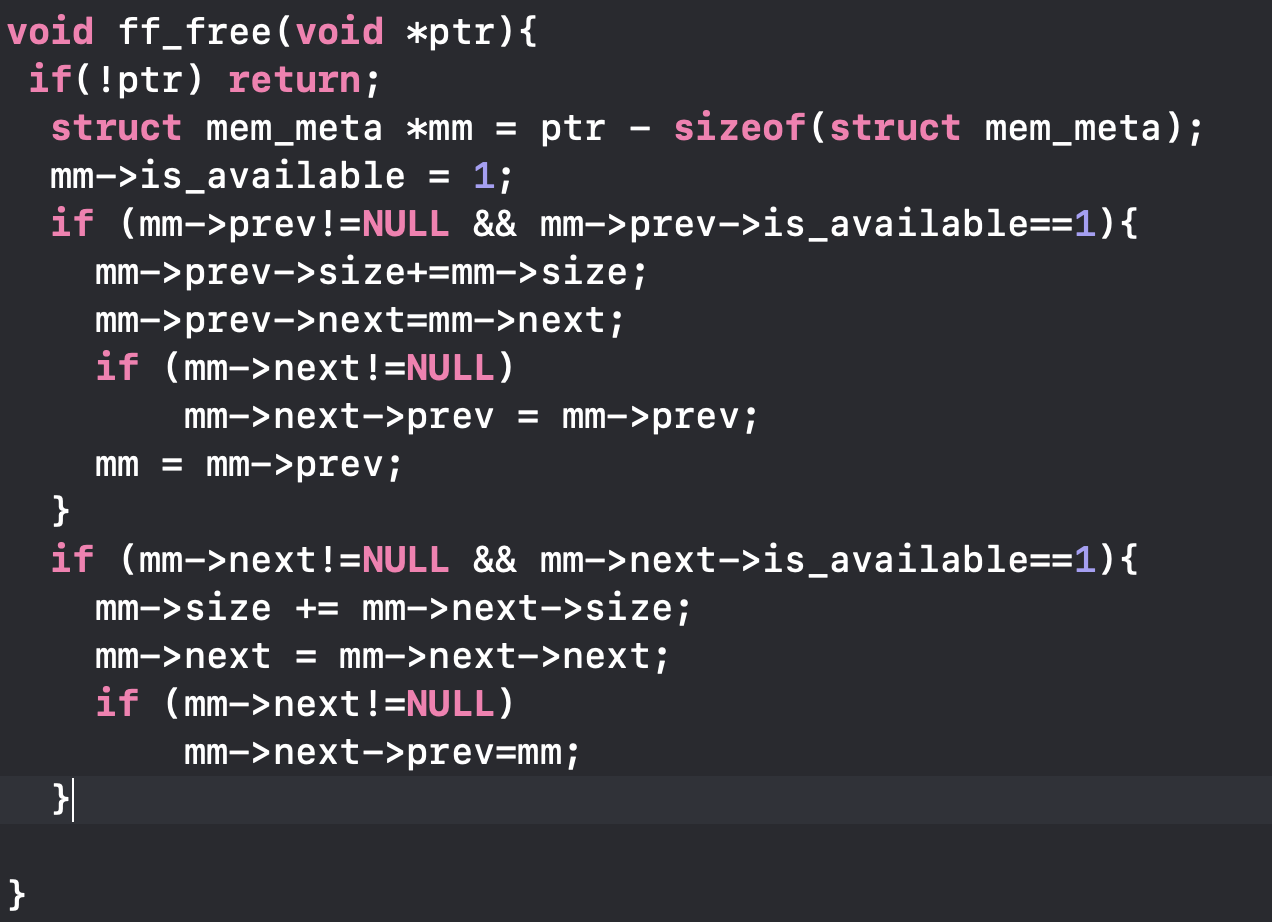
In terms of “First Fit”, we simply go through the list and find the first available seat. As for “Best Fit”, we have to go over the entire list to decide the minimum available seat.

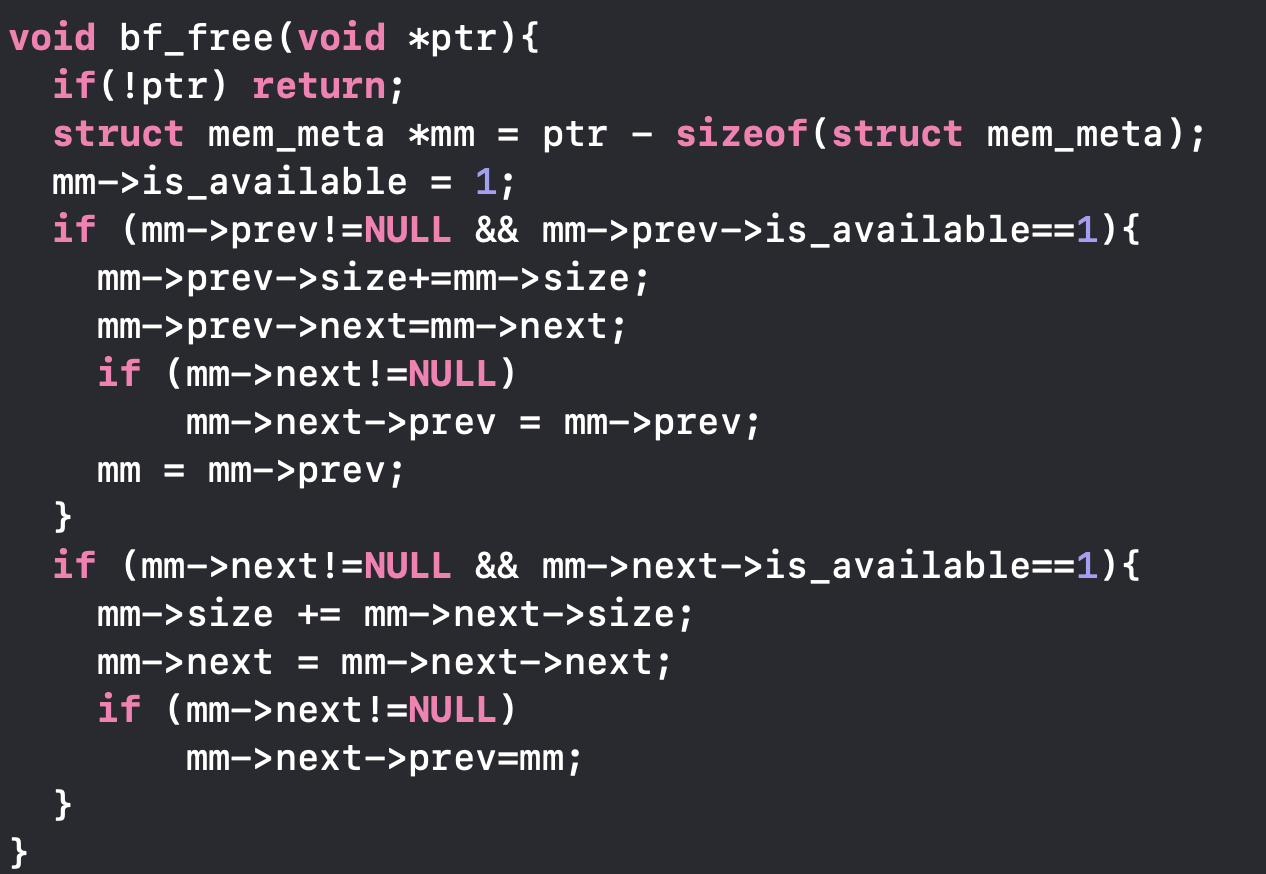






When free, we need to merge the adjacent free chunks by adjusting the linked list.





1. **Experimental Results**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | FF | BF |
| Small\_range\_rand\_allocs | Data\_segment\_space | 3955776 | 3597056 |
| Data\_segment\_free\_space | 376064 | 17344 |
| Execution\_time | 96.830135 s | 96.546725 s |
| Fragmentation | 9.5067% | 0.4822% |
| Large\_range\_rand\_allocs | Execution\_time | 105.425821 s | 187.295578 s |
| Fragmentation | 18.4581% | 1.5857% |
| Equal\_size\_allocs | Execution\_time | 543.727429 s | 527.427586 s |
| Fragmentation | 0% | 0% |

Note that in the table above, the definition of fragmentation is the division of (allocated size – used size)/allocated size.

1. **Analysis and Conclusion**

In terms of space allocation, it could be observed from the table above that for small range random allocation, the data segment space of Best Fit is smaller than First Fit, which proves that Best Fit can make the best use of allocated space. Besides, the fragmentation rate in both small range and large range shows a favor in Best Fit rather than First Fit, which also proves that Best Fit makes the best of allocated space. For equal size allocation, every time the same size of space was allocated and freed, resulting in a zero fragmentation.

For execution time, Best Fit and First Fit should be taken as same scenario as for equal size allocation. Similarly, small range random allocation can also be compared to the equal case, which gave us a close execution time. However, the large range random allocation reveals the fact that runtime for First Fit is much smaller compared to Best Fit.

The last thing to notice is that, the fragmentation for large range is worse than small range. This is because we have a larger range of chunks which make them hard to fit and make use of every bit of space compared to smaller range.

In conclusion, during the implementation of “malloc”, First Fit gives us faster performance, while Best Fit make a better use of heap memory.

Notice that the execution time is relatively large. Better runtime can be achieved if a new linked list is created to keep track of the free chunks. In this way, we can find the free chunks by looking up this new list instead of going through the whole list.