glib malloc implementation:

1. Maintaining separate heap and freelist data structures for each thread called per thread arena.

2. Allocated memory region is released only to ‘glibc malloc’ library, which adds this freed block to main arenas bin.

3. When user requests memory, ‘glibc malloc’ doesnt get new heap memory from kernel, instead it will try to find a free block in bin. And only when no free block exists, it obtains memory from kernel.

4. Application’s arena limit is based on number of cores present in the system.

5. Data structures:
    heap_info – Heap Header – A single thread arena can have multiple heaps. Each heap has its own header.
    malloc_state – Arena Header – A single thread arena can have multiple heaps, but for all those heaps only a single arena header exists. Arena header contains information about bins, top chunk, last remainder chunk
    malloc_chunk – Chunk Header – A heap is divided into many chunks based on user requests. Each of those chunks has its own chunk header.

6. No two free chunks can be adjacent together. When both the chunks are free, its gets combined into one single free chunk.

mm.c (naive implementation):

1. In this naive approach, a block is allocated by simply incrementing the brk pointer.
2. A block is pure payload. There are no headers or footers.
3. Blocks are never coalesced or reused.
4. Realloc is implemented directly using mm_malloc and mm_free.

Performance of mm after running on different trace files:

Reading tracefile: traces/binary-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	56%	12000	0.000065	183767
Total		56%	12000	0.000065	183767

Perf index = 34 (util) + 40 (thru) = 74/100

Reading tracefile: traces/cp-decl-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	30%	6648	0.000044	152128
Total		30%	6648	0.000044	152128

Perf index = 18 (util) + 40 (thru) = 58/100

Reading tracefile: traces/random-bal.rep
Checking mm_malloc for correctness, ERROR [trace 0, line 1666]: Payload address (0xffffffff) not aligned to 8 bytes

trace	valid	util	ops	secs	Kops
0	no	-	-	-	-
Total		-	-	-	-

Reading tracefile: traces/realloc-bal.rep
Checking mm_malloc for correctness, ERROR [trace 0, line 1705]: Payload address (0xffffffff) not aligned to 8 bytes

trace	valid	util	ops	secs	Kops
0	no	-	-	-	-
Total		-	-	-	-

```
Reading tracefile: traces/short1-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes   50%      12  0.000000          120000
Total          50%      12  0.000000          120000
Perf index = 30 (util) + 40 (thru) = 70/100
```

```
Reading tracefile: traces/short2-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes  100%      12  0.000000          60000
Total        100%      12  0.000000          60000
Perf index = 60 (util) + 40 (thru) = 100/100
```

```
Reading tracefile: traces/short3-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes   49%      10  0.000000          100000
Total          49%      10  0.000000          100000
Perf index = 30 (util) + 40 (thru) = 70/100
```

mm1.c (our solution as per given in the problem statement):

In mm1 implementation:

1. When a request for allocation comes, it first searches for an available free block in the list of blocks using best fit strategy.

2. If no sufficient size of block is found, a new block is allocated using mem_sbrk(). This block is then added to the list of blocks.

3. On free, two-sided coalescing is performed.

4. The head pointer (free_list_start) and tail pointer of the list is maintained (free_list_end).

5. The insertion takes O(1) time due to the insertion of block at the end of list usign tail pointer.

6. Searching is done using best fit and takes O(n) to search the list.

7. Each block has its metadata stored in its header: size, is_free, next pointer and previous pointer. It's size is 24 bytes.

Performance of mm1 after running on different trace files:

```
Reading tracefile: traces/binary-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes   53%   12000  0.175935          68
Total          53%   12000  0.175935          68
Perf index = 32 (util) + 5 (thru) = 36/100
```

```
Reading tracefile: traces/cp-decl-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes   99%    6648  0.025435          261
Total          99%    6648  0.025435          261
Perf index = 59 (util) + 17 (thru) = 77/100
```

```
Reading tracefile: traces/random-bal.rep
trace  valid  util     ops      secs     Kops
 0       yes   95%    4800  0.022180          216
```

Total		95%	4800	0.022180	216

Perf index = 57 (util) + 14 (thru) = 72/100

Reading tracefile: traces/realloc-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	26%	14401	0.133523	108
Total		26%	14401	0.133523	108

Perf index = 15 (util) + 7 (thru) = 23/100

Reading tracefile: traces/short1-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	66%	12	0.000000	40000
Total		66%	12	0.000000	40000

Perf index = 40 (util) + 40 (thru) = 80/100

Reading tracefile: traces/short2-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	99%	12	0.000000	40000
Total		99%	12	0.000000	40000

Perf index = 60 (util) + 40 (thru) = 100/100

Reading tracefile: traces/short3-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	50%	10	0.000000	50000
Total		50%	10	0.000000	50000

Perf index = 30 (util) + 40 (thru) = 70/100

mm2.c(our own design and implemetation):

Improvements in mm2 over mm1 implementation:

1. Header size reduced from 24 bytes to 16 bytes by removing is_free variable to check block is free or not. Instead, now free checking is done using LSB of the size variable (Header -> actual_size) of the header.

2. Using Modified first fit statergery for better throughput along with fragmentation if needed.

3. Added two way traversal (i.e. from head as well as tail) in alternate fashion, so that (used) block congestion doesn't occur at the start of the block list.

4. Implemented deferred coalescing so that searching and coalescing happens simultaneously and space utilisation is improved.

5. In realloc part, added next free block coalescing function. If the adjacent block is free, then some portion of it gets assigned and other half gets fragmented. Hence space as well as throughput is improved.

The reason for our design being optimal for widest variety of workloads is:

1. We are using a modified first first statergy in which the traversal occurs in both the direction (i.e. from start to tail and vice versa) after every request. Because of this there is no congestion of used blocks at one end and hence it's distributed across the list.

2. The malloc request has buffer depending on the request inorder improve the throughput of realloc.

3. Also deffered coalescing technique (coalescing while searching for the block) helps optimise the space simulataneously. Hence space utilisation is improved.

4. Performing periodic coalescing throughout the list (after particular number of free requests) helps maintain the blocks in defragmented state.

Performance of mm2 after running on different trace files:

Reading tracefile: traces/binary-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	54%	12000	0.153750	78
Total		54%	12000	0.153750	78

Perf index = 32 (util) + 5 (thru) = 37/100

Reading tracefile: traces/cp-decl-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	98%	6648	0.011143	597
Total		98%	6648	0.011143	597

Perf index = 59 (util) + 40 (thru) = 98/100

Reading tracefile: traces/random-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	90%	4800	0.008535	562
Total		90%	4800	0.008535	562

Perf index = 54 (util) + 37 (thru) = 91/100

Reading tracefile: traces/realloc-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	73%	14401	0.000310	46440
Total		73%	14401	0.000310	46440

Perf index = 44 (util) + 40 (thru) = 84/100

Reading tracefile: traces/short1-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	98%	12	0.000000	60000
Total		98%	12	0.000000	60000

Perf index = 59 (util) + 40 (thru) = 99/100

Reading tracefile: traces/short2-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	99%	12	0.000000	60000
Total		99%	12	0.000000	60000

Perf index = 59 (util) + 40 (thru) = 99/100

Reading tracefile: traces/short3-bal.rep

trace	valid	util	ops	secs	Kops
0	yes	99%	10	0.000000	50000
Total		99%	10	0.000000	50000

Perf index = 59 (util) + 40 (thru) = 99/100

References:

1. https://moss.cs.iit.edu/cs351/slides/slides-malloc.pdf
2. https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/
3. https://arjunsreedharan.org/post/148675821737/memory-allocators-101-write-a-simple-memory