The Assignment is structured into 3 parts: the system folder has the file key_value2.c which has the implementation of all the functions, the include folder consist of the kv.h file which in turn has all the definitions of the methods used in the implementation file, the shell folder has all the test cases.

The implementation has the following functions:

kv_set(key,value): This function sets the value of that particular key into the cache. If the key already exists in the cache then it is moved to the top of the cache, but if there is no entry for that key then we just simply add that to the top of the cache. Also note that if the cache size is full and the entry for that key is not present then the first element from the cache is removed and the most recent one is added on top of the cache. If the entry for that key is already present, then simply shift the key to the top of the cache.

kv_get(key): This function returns the value for that stored key in the data structure. If the key is not present in the cache then the function returns NULL, however if the node is present in the list then return the value of that node and shift the node to the front of the list making it the most recently used key.

Kv_delete(key): This function deletes a particular key and its corresponding value from the cache. It returns a Boolean value, if its successfully in deleting the key and its corresponding value then it returns true or else it returns false.

kv_init(): This function initializes the cache metadata and calls the xmalloc_init() which in turn initializes the buffer pools.

kv_reset(): This function empties the whole cache of the system and then frees the memory of the buffer pools which were used to allocate the key value stores.

get_cache_info(kind): This functions returns a value based on the input parameter kind. The return types are total_hits, total_accesses, total_set_success, cache_size, num_keys, total_evictions

most_popular_keys(k): This function traverses through the linkedlist and returns the top k values of the least recently used.

The cache eviction technique which I used was LRU. This was done by maintaining a list, if the new key is not present in the list then the then the list which was been least recently used is removed from the list, if the key is present in the list then that key is shifted to the front of the list. This way the most recently used keys are always present in the beginning of the list. The key is being hashed and stored in the data structure and the character limit for the value is 1000 characters.