Create your own copy of this template to edit. In the menu, select File -> Make a copy

Practicum I Rubric & Self-Evaluation

Team member name(s):	**Team member email(s):**
Lingyu Hu	hu.lingyu@northeastern.edu

Part 1 - Memory Hierarchy	**Exemplary**	**Good**	**Acceptable**	**Points**	**Function names/Code Chunk/Lines**	**Additional notes (if needed)**
Messages contain the minimally required fields and are implemented with an appropriate data structure	6	5	3	6	message.h -> #11	
Function create_msg() that creates a new message with the fields appropriately set and returns a dynamically allocated message "object".	8	7	4	8	message.c -> create_msg(): #20	
Function store_msg() successfully stores the message on disk.	8	7	4	8	message.c -> store_msg(): #50	
Function retrieve_msg() finds and returns a message identified by its identifier.	8	7	4	8	message.c -> retrieve_msg(): #97	

Part 2 - Cache	**Exemplary**	**Good**	**Acceptable**	**Points**	**Function names/Code Chunk/Lines**	**Additional notes (if needed)**
Cache works and is used to find messages	5	4	2	5	cache.h cache.c ->#35 void init_cache() ->#50 int find_msg_in_cache(int id) ->#124 Message* retrieve_msg_cached(int id, bool *msg_in_cache)	See Design details in README
All functions use and update the cache	10	8	5	10	cache.c ->#66 int add_msg_to_cache(Message *msg) ->#96 int store_msg_cached(Message *msg) ->#124 Message* retrieve_msg_cached(int id, bool *msg_in_cache)	
Cache hits and misses are detected and handled properly	10	8	5	10	cache.c -> #124 retrieve_msg_cached() ->#128 Cache hits ->#132 Cache misses	
Messages are being saved to disk	5	4	2	5	cache.c ->#96 int store_msg_cached(Message *msg) ->#101 if (!store_msg(msg)) message.c ->#50 bool store_msg(const Message *msg)	

Part 3 - Page Replacement	**Exemplary**	**Good**	**Acceptable**	**Points**	**Function names/Code Chunk/Lines**	**Additional notes (if needed)**
Random Replacement: a replacement page is chosen at random	5	4	2	5	cache.c ->#246 int store_msg_cached_by_strategy(Message *msg, int use_lru) ->#261 return add_msg_to_cache_by_strategy(cache_copy, use_lru); ->#279 Message* retrieve_msg_cached_by_strategy(int id, bool *msg_in_cache, int use_lru) ->#299 add_msg_to_cache_by_strategy(cache_copy, use_lru); ->#195 int add_msg_to_cache_by_strategy(Message *msg, int use_lru) ->#218 index_to_replace = find_random_replacement_index(); ->#159 int find_random_replacement_index()	

LRU: the least recently used page is replaced	10	8	5	10	cache.c ->#246 int store_msg_cached_by_strategy(Message *msg, int use_lru) ->#261 return add_msg_to_cache_by_strategy(cache_copy, use_lru); ->#279 Message* retrieve_msg_cached_by_strategy(int id, bool *msg_in_cache, int use_lru) ->#299 add_msg_to_cache_by_strategy(cache_copy, use_lru); ->#195 int add_msg_to_cache_by_strategy(Message *msg, int use_lru) ->#218 index_to_replace = find_random_replacement_index(); ->#159 int find_random_replacement_index()	

Part 4 - Evaluation Metrics	**Exemplary**	**Good**	**Acceptable**	**Points**	**Function names/Code Chunk/Lines**	**Additional notes (if needed)**
Cache hits and cache miss metrics	10	8	5	10	test.c ->#57 void random_access_and_metrics(int use_lru) -> #72 & #74 ->#146 cache.c ->#124 Message* retrieve_msg_cached(int id, bool *msg_in_cache) ->#279 Message* retrieve_msg_cached_by_strategy(int id, bool *msg_in_cache, int use_lru)	use bool *msg_in_cache to detect hit or miss
Cache hit ratio	5	4	2	5	test.c ->#79 Hit Ratio= Cache Hits/ Total Lookups The ratio is also related to the cache size.	

Overall	**Out of**			**Points**
Code and comments quality	5			5
Demo / presentation	5			5