To compile:

gcc -o message_app main.c message_store.c cache.c

To run:

./message_app

To clean:

rm messages.dat index.dat

Functionality Details

- **Message Creation (`create_msg`)**:
  - Allocates memory for a new message.
  - Populates the message fields, ensuring proper null-termination and avoiding buffer overflows.
  - Sets the `time_sent` field to the current time.
- **Message Storage (`store_msg`)**:
  - Writes the message to the messages file, appending it to the end.
  - Calculates the file offset where the message is stored.
  - Updates the index file with a new entry mapping the message ID to its offset.
  - Adds the message to the cache if caching is enabled.
- **Message Retrieval (`retrieve_msg`)**:
  - Checks the cache for the message using its ID.
    - **Cache Hit**: Returns a copy of the message from the cache.
    - **Cache Miss**: Searches the index file to find the message offset, reads the message from the messages file, and adds it to the cache.
  - Returns a copy of the message to prevent external modifications to cached data.
- **Cache Implementation**:
  - **Hash Table**:
    - An array where each index corresponds to a bucket holding a linked list of cache nodes (for collision handling).
    - Uses a simple hash function to map message IDs to indices.
  - **Doubly Linked List**:
    - Maintains the order of cached messages based on recent usage.
    - Supports quick updates for both the LRU and Random Replacement policies.
  - **Cache Nodes (`CacheNode`)**:

        - Contains the message ID, a pointer to the message, and pointers for the linked list and hash table chaining.
- **Performance Metrics**:
    - **Hit Count**: Number of times a message retrieval found the message in the cache.
    - **Miss Count**: Number of times the message was not in the cache and had to be read from disk.
    - **Hit Ratio**: Calculated as `(Hit Count) / (Hit Count + Miss Count)` to evaluate cache effectiveness.