Cache Simulator Report

1. Introduction

This report provides details about the coding approach used in making the Cache Simulator, which simulates the working of a cache memory.

2. Coding Approach

The Cache Simulator Project is structured around a set of classes representing different components of a cache memory. Each class is designed to simulate the working of a cache memory.

- **Cache**: The base class for all cache types. It has methods and attributes to simulate the working of a cache memory.
- **LRUCache, FIFOCache, RandomCache**: These classes inherit from the Cache class and implement the LRU, FIFO, and RANDOM replacement policies respectively.
- **CacheBlock**: Represents a block in the cache memory. It has methods and attributes to simulate the working of a cache block.
- **LRUCacheBlock, FIFOCacheBlock, RandomCacheBlock**: These classes inherit from the CacheBlock class and implement the LRU, FIFO, and RANDOM replacement policies respectively.

Workflow of the Cache Simulator

1. The Cache Simulator reads the cache configuration from `cache.config` file and creates a cache object of the corresponding type by calling the `createCache` method of the Cache class, which returns the pointer to a cache object of the corresponding type.
2. Then the Cache Simulator reads the memory accesses from `cache.access` file and passes them to the `access` method of the Cache class, which is the general handler for all memory accesses.
3. Each Cache class has custom find and insert methods which are called by the `access` method to find and insert a memory address in the cache memory based on the replacement policy of the respective cache type.
4. All the Cache type classes have vector of vectors of CacheBlock objects of the respective type, which represent the cache memory. Each CacheBlock object has a boolean `valid` attribute which is set to true if the block is valid and false otherwise. It also has a `tag` attribute which stores the tag of the memory address stored in the block. The `tag` attribute is set to the tag of the memory address when the block is inserted in the cache memory. And the block may also have other attributes depending on the replacement policy of the cache type.
5. The find and insert methods of the Cache type classes handle the cache hits and misses and also the replacement of blocks in case of a miss.
6. The access method then outputs the result of the memory access in the `output.txt` file.

3. Testing Strategy

For testing, I tried with different cache configurations and different memory access patterns, including both read and write instructions. I tried to simulate the testcases on RIPES and checking the outputs from my program side-by-side. I tried some small custom testcases to check the specific behaviours of write access and no write access and also to check whether the replacement policies were working fine. I also tried some large

testcases and checked the results. I have added some example test cases in `testcases.txt` file. Also the `cache.config` and `cache.access` files are initially populated with a sample testcase, with their output in `output.txt` file.