REPORT

NAME: Paavaneeswar reddy

Roll no: cs22btech11014

Code Overview and Approach:

1. **Cache Structure:**
   - Utilized a struct (`CacheSet`) to represent the cache with arrays for validity, tag, FIFO counters, LRU counters, and dirty bits.
2. **Cache Initialization:**
   - Implemented a function (`initializeCache`) to initialize the cache, dynamically allocating memory for each array and setting initial values.
3. **Cache Access Simulation:**
   - Wrote a function (`simulateCacheAccess`) to simulate cache access based on specified parameters such as replacement policy (FIFO, LRU, RANDOM) and write policy (Write-Back, Write-Through).
   - Calculated block offset, index, and tag based on the provided address.
   - Checked for cache hits by comparing tags and valid bits.
   - If a cache miss occurred, determined the replacement index using the specified replacement policy.
   - Updated cache information based on the replacement policy and write policy.
   - Printed access information, including hit/miss status, address, set, and tag.
4. **File Handling:**
   - Implemented a function (`countLines`) to count the number of lines in a file, used for determining the number of cache lines in the main simulation.
5. **Main Functionality:**
   - Read cache configuration from `cache.config` and access sequence from `cache.access`.
   - Validated input parameters and policies.
   - Dynamically allocated memory for the cache sets and initialized the cache.
   - Simulated cache access for each line in the access sequence, updating the cache accordingly.
   - Printed the access information for each operation.

Testing Approach:

1. **Input Validation:**
   - Checked for valid cache parameters such as size, block size, and associativity.
   - Verified the correctness of replacement policies (FIFO, LRU, RANDOM) and write policies (Write-Back, Write-Through).
2. **Functional Testing:**
   - Executed the program with various input files and configurations to ensure correct simulation results.
   - Manually verified output against expected behavior for different scenarios, including cache hits and misses.
3. **Edge Cases:**
   - Tested edge cases, such as minimum and maximum values for cache parameters and policies, to ensure the program handles extreme scenarios gracefully.
4. **Randomness Testing:**
   - Specifically tested the RANDOM replacement policy to ensure it behaves as expected and introduces randomness into the cache replacement process.