Programming Exercise

Your programs must:

- Be of production quality
- Build and run on Linux
- Come with unit tests
- Not be written in C++17 as we are unable to compile it in our current environment

Provide a build utility with make or cmake.

Provide a README with instructions on how to build the program, run the program, design documentation as well as any auxiliary data files and scripts needed to run the program. Take about 3 to 4 days of your time to write.

In Memory File Cache Design

Develop a program that performs caching of frequently accessed items. Input and output formats for your program are clarified below. The size of the cache is quite small compared to the size of the items.

- Size of the cache comes from command line.
- There are multiple readers and writers of items. Number of readers and writers comes from Readers and Writers file which is passed to the program from command line.
- Design an efficient mechanism to retrieve items that are cached.
- Design and implement an efficient eviction mechanism when there is a cache miss.
- Handle the case of dirty cache item, where a writer is modifying the item that is already cached.
- Your program should handle all corner cases and robust to failures. If it fails, it should fail gracefully giving out meaningful failure messages.
- List out all the assumptions you have made for any unclear requirements.

Your program will take 4 command line arguments and will be executed as:

```
cache <size_of_cache>  <reader_file>  <writer_file>  <items_file>
```

~ *size_of_cache* is integer value indicating number of elements that the cache can hold at any given time

~ *reader_file* is a file that contains list of file names. Each file in this list is an input to a reader thread. Your program has to concurrently spawn N readers if there are N files in the list. Each file in the list will contain a list of positions in item file that it reads from. Each reader will read the position and write to an output file the item read and if the item is read from cache or from disk.

Sample contents of reader_file:

Reader1.txt
Reader2.txt
Reader3.txt
Reader4.txt

Sample content of Reader1.txt:
1
3
5
7
1
3
5
7

~ Similarly *writer_file* is a file that contains list of file names. Each file in this list is an input to a writer thread. Your Program has to spawn that many writer threads concurrently. Each writer file will contain list of positions and the value to be written. Each writer thread has to write that value into the item file at the specified position.

Sample contents of Writers: (Program has to spawn 3 concurrent writers) Writer1.txt
Writer2.txt
Writer3.txt

Sample content of Writer1.txt:
1 100
2 200
4 300
5 500
8 800

~ items_file is the actual data file for your program. Each line of the file is either blank or contains one number. The positions in the reader and writer files correspond to line numbers in the item file.

Your item file could start out empty or prefilled with values. Sample
contents of items_file:
0
23.5
-33

75.2

45
90.0
100
8
9

10000.0

(Note the blank lines in between)

After executing your program with cache size 4
Cache 4 Readers Writers Items

Reader1 will output Reader1.out.txt
0 or 100 Disk (depending on if reader1 reads first or writer1 writes first) -33
Disk
75.2 or 500 Disk
90.0 Disk
0 or 100 Cache
-33 Cache
75.2 or 500 Cache
90.0 Cache

Your Item file at the termination of the program would look like (assuming there were only Reader1 and Writer1).
100
200
-33
300
500
45
90.0
800
8
9

10000.0

Enunciate your design considerations, for cache when your program is read heavy vs write heavy.