Design Choices

1. Intermediate files: I am using two sets of intermediate files, one each for mappers and reducers.
2. The file formats are fixed, and the procedure for running this code is to first take an input file, chunk it using `file_chunker.py` which will generate the files in the appropriate format.

Running Details

1. The master needs to be run before the workers are run. To run the master, run `go run . --mappers <num> --reducers <num> --task <task>` wherein we are specifying the numbers of mappers and reducers via the commandline.
2. The number of mappers will be equal to the number of chunks created. Hence, it is easily possible to automate that part, but for convenience, I am including a commandline flag for both.
3. To run the workers, run `go run . --id <num>` wherein the id is used to identify the workers. The workers will then connect to the master and get an ID in return. The files are chunked in the following format: `<filename>_<chunk>_mapper_<assign_id>.txt` and each mapper reads the files with the same id as the one assigned by the master, i.e. if a master assigns the worker having commandline id (user input) with an assign_id of 6, this mapper worker will read the file with that particular ending id and process it

Implementation Details

1. Each mapper then generates intermediate files using hashing. The hashing used is FNV hash algorithm followed by taking modulus with the number of reducers. As a result, every word that the mapper reads is given a hash value and is stored in the appropriate intermediate file (the filename contains information such as which reducer it is assigned to)
2. Note that the reducers will operate only after the mappers are done with their task. This means that if a reducer waits while some mappers are still working, these reducers are put into a queue. Once all the mappers are done working, the first `num_reducer` reducers are popped from the queue and set to work.
3. Again, the reducer will go and operate only on the files it has been assigned. This ensures efficiency and a clear division of labour. The reducer stores the intermediate files in `reducer_intermediate`. Once all the reducers have completed their task, the master sends out kill signals to all the workers to terminate their execution.
4. To ensure that the reducers start only after mappers are done and similarly kills are sent out only after the reducers are done executing is ensured using a variety of queues and locks. The extensive implementation helps keep track of which reducers were assigned to which commandline ids and so on.