Documentation

The commands supported by the program are:

- **SET –** This command will insert a value of type "string" to the corresponding given key value. This will overwrite any key present or will insert in case key hasn't been used yet.
  Time Complexity – O(1)

- **GET –** This command will retrieve value corrseponding to the key if and only if key is already present and holds a value of type "string".
  Time Complexity – O(1)

- **ZADD –** This command will add pair of **<score, value>** to the **sorted set** corresponding to the key if and only if key is present with value of type sorted set or if key is absent, in which case it will make a new sorted set.
  Time Complexity – O(log n), per pair where **n** is the size of sorted set.

- **ZRANK –** This command will return rank of provided "string" within the sorted set in case set and value is present.
  Time Complexity – O(log n), where **n** is the size of sorted set.

- **ZRANGE –** This command will return elements of sorted set within the given range provided the range given is valid.
  Time Complexity – O(log n + m), where **m** is expected number of elements in range and **n** is size of sorted set.

- **EXPIRE –** This command will set a timer for the key if holding "string" type value**.** After timer is up the key and value corresponding to it will be removed.

- **SAVE NOW –** This command will save the current database into a file.

Language Used: C++

The implementation of Redis itself is in **C.** Therefore, this is not far from it. The main reason for using C/C++ is for the high speed and performance they provide. Although, there are not bundle pre-implemented libraries available like in other High-Level languages still nothing comes closer if database is to be used as a cache and specially if queries are to be processed rapidly.

Data Structures: Hash Tables, Self-Balancing AVL Trees (sorted set)

Persistence:

The implementation here is persistent like original Redis i.e, when the program is closed the data present in memory is saved to be used later on. To achieve persistence, the data is **written to a .txt file after program is closed.** Also, command can be used to save current data to avoid losing it if program, system or power happens to crash. A custom **serializer and de-serializer is implemented to write/read AVL Tress to/from file.**

Time Complexity: O(n), where **n** is number of values in data.

Improvements:

- **Flags/Arguments –** Extra flag or arguments can be provided with the commands to further defince operations on database.
- **Delete –** Currently, the value only gets updated or removed using EXPIRE, but a delete/remove command can be implemented to remove key then and there from the database.
- **Checkpointing –** Currently, the database is only saved using command or on exit. But, this may cause in loss of data in case program has been running for a long time and simultaneous system crash. Therefore, a cron Job can be implemented to save database om regular intervals.
- **Error/Exception Handling –** The queries are being performed with the assumption that there will be no error of any kind with the provided input. This may pose some problem and cause a crash if input is not according to the syntax.
- **Encryption –** The data saved in .txt file is exposed and has a risk of leak or unauthorised access. Even if someone should gain access they should not be able to read and infer its contents. For this very purpose an encryption scheme can be used to protect data better. Encryption will secure the data. But, this security will come at the cost of time as encryption is process and time consuming and so is decryption when data is to be loaded.
- **Swapping –** The idea of writing/reading entire data is time and process consuming. Swapping can be implemented to read/write only certain amount of data compared to whole database.
- **Multi-threading –** Multi-threading can be used to perform operations such as EXPIRE where another thread dedicated to only removing keys improve performance and main program does not need to check for it. But, multi-threading is complex and may also call for locking mechanisms. Without locking and proper implementation program can crash and data can be lost.