Okusanya Oluwatise
Project 2 Design Document

Class Design

1. Fileblock Class: This class holds the fileblock which contains the ID, payload and checksum

Important Member Variables:

a. Int storedChecksum: Holds checksum value that has been calculated using the payload of the fileblock. Validate_checksum uses it to check if the current recomputed checksum matches the stored checksum.

Important Member Function:

a. compute_checksum(): This function converts the elements(characters) to unsigned char using “static_cast<unsigned char>” . This is done to ensure that all character values are treated as non-negative integers so we get an accurate checksum calculation.

2. Hashing Class: This class handles the fileblocks using either separate chaining(if hashtype==1) or open addressing(if hashtype==0).

Important Member Variables:

a. std::vector<std::vector<Fileblock > Chain: This member variable is a vector of vectors. Outer vector for the hashtable and inner vector for the chains.
b. std::vector<Fileblock> doublehashing: This is the single vector used for doublehashing
c. std::vector<bool > occupied: This vector is updated along with the double hashing vector, to prevent the doublehashing error explained in the lab. It shows taht a deleted spot is still occupied so that a value whose initial probe is that index, isn’t marked as not in the table. Used vectors so I could manage memory efficiently.
d. Int hashtype: Determines with method of hashing is to be used

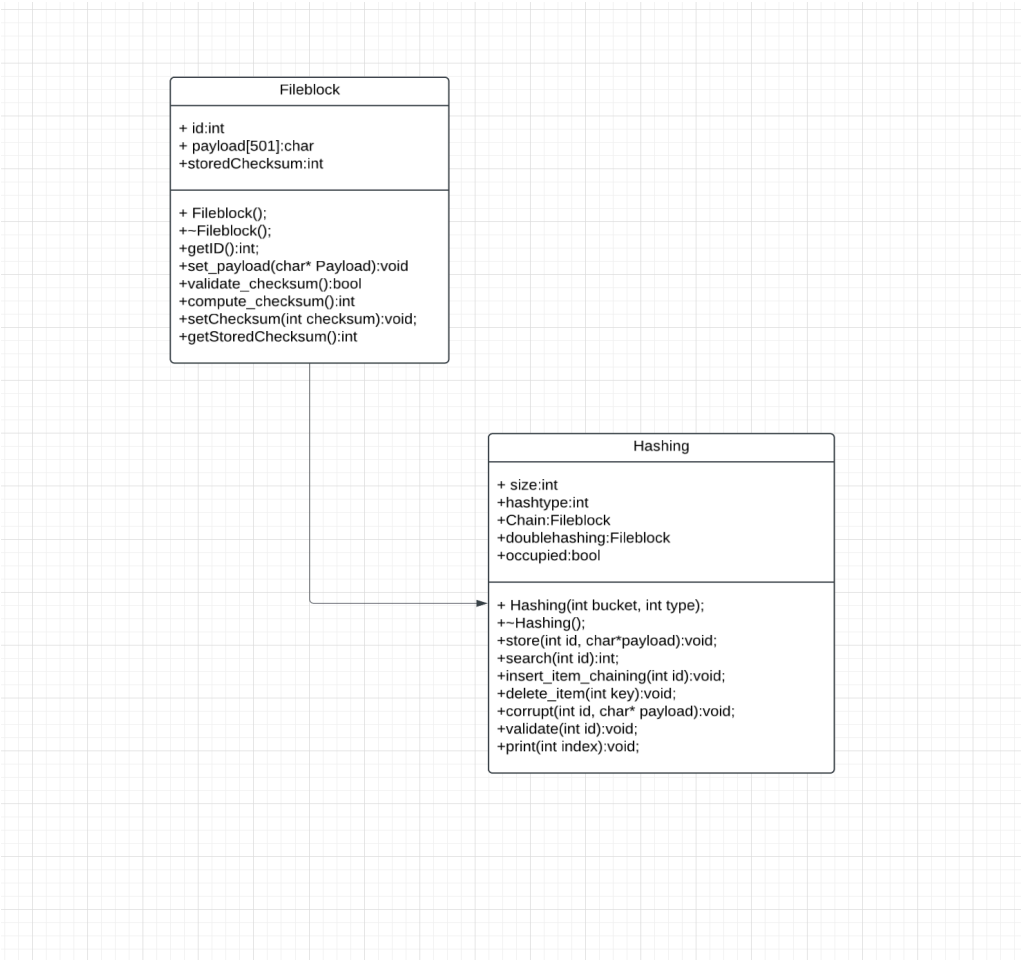
Important Member Functions:

a. Hashing(int bucket, int type): Parameterized constructor that gives a set size of the hashtable(the vector of double hashing, and outer vector of chaining) and the type of hashing to be used.
b. print(int index): Makes use of insertion sort when sorting the IDs, because it is a stable sorting algorithm

Function Design

1. *store(int id, char** payload): Handles collisions when inserting fileblocks in two different ways. Depending on which one user asks for

UML Diagram



Runtime Analysis

1. STORE: For separate chaining, average run timeis O(1) because we can directly push it into the vector that has the index gotten by the primary hashfunction. For double hashing,it is also O(1) because we use a are probing in a way that we only need to check the table the number of times we have a collision, which is O(1)
2. SEARCH: For chaining, we can access the position in the hashtable, by simply placing the id in the hashfunction and searching a small list bound by constant. For double hashing, because we use a are probing in a way that we only need to search the table the number of times we have a collision, which is O(1)
3. DELETE: For Separate Chaining,O(1) on average, since we can directly access the appropriate chain and remove the Fileblock once it is accessed For double hashing,O(1) average run time because we follow the same probing sequence used in the search, making the deletion straightforward