Project 2 Design Document
Kelly Kim
ECE 250
4 November 2024

This is the design document for ECE250 Project 2 Hashing Checksum.
Using a single header file (hashTable.h) for the project keeps things organized and straightforward, reducing complexity. Since the components are closely related, multiple header files can create confusion about which methods interact with which member variables. Additionally, this approach can lead to faster compilation times, as the compiler processes fewer files.

Public:

- hash(int N, int T)
- ~hash()

Both constructor and destructor are public, allowing objects of the class to be created and destroyed outside of class. The constructor allows external code to create instances with specific parameters. The destructor cleans up the resources when they are done using.

• void newTable (int N, int T) • void store (int ID, std::string charstring) • void search (int ID) • void deleteID (int ID)	• void corrupt (int ID, std::String newCharstring) • void validate (int ID) • void print (int i) • bool exit();

By making these methods public, users will be able to easily manage tasks like storing, searching, deleting, etc. Each function will have an if-else statement, checking if it is using open addressing or a separate chaining method.
newTable function will have a void return type, printing out “success” when the hash table has been created. For open addressing, it uses arrays, and for separate chaining, it uses 2D Vectors. Using 2D vectors is preferable than arrays for separate chaining because it provides flexibility to grow or shrink as needed without manual memory management.
Store functions will have a void return type, printing out “success” when it has been successfully stored, or “failure” if there are no more spots/key already exists.
Search function will have a void return type, printing out the location where the ID has been found or “not found” if the ID does not exist.
deleteID function will have a void return type, printing out “success” or “failure” depending on if the file block has been deleted or not.
Corrupt functions will have a void return type, printing out “success” or “failure” depending on if the ID exists or not.
Validate functions will have a void return type, printing out “valid”, “invalid”, or “failure” depending on checksum calculation.
Print functions will print the IDs in order for separate chaining, unless the table is empty.
Exit function will be used to control when the program stops. Returning true for the program to exit, returning false for the program to keep going.

- bool exitsign = true;

This variable changes to false when the exit function is called to exit the program.

- int calculateCheckSum(const std::string& charstring)

calculateCheckSum function will be used to calculate the checksum for charstring. It will take the charstring (without the '!' at the end) and compute checksum. This will be later used in store function as well as validate function.

Private:

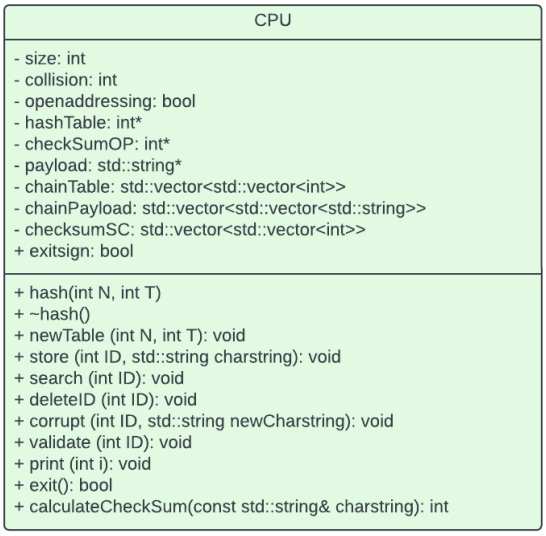
• int size • int collision • bool openaddressing	• int* hashTable • int* checksumOP • std::string* payload	• std::vector<std::vector<int>> chainTable • std::vector<std::vector<std::string>> chainPayload • std::vector<std::vector<int>> checksumSC

Size/Collision/Openaddressing is used to determine the set up of the hash table. The size will represent the slots of the hash table and collision/openaddressing will be used to determine if it uses open addressing or separate chaining. Making them private prevents accidental modification, ensuring the hash table's behaviour and structure remains consistent.

hashTable/ChecksumOP/Payload manage the main storage for keys, checksums, and payloads when using open addressing. Making them private prevents the data structure from unauthorized access and ensures that only the class's functions can modify them directly, reducing the risk or memory errors.

chainTable/chainPayload/checksumSC store the elements, payloads, and checksums for separate chaining. Making them private, the user can control how chains are modified or accessed ensuring that only certain functions can access (change) them.

UML DIAGRAM:



Running Time:

Hash Table size remains constant, assuming that there are at most m=O(1) collisions. It will iterate through m times and since the number is constant, the run time for the run function would be O(1).

Open Addressing	Separate Chaining
Store (Insertion): When a key is inserted, we use the primary hash function first. If the index is empty, insertion is O(1). If the index is not empty, we use a secondary hash function to determine the increment, probing until we find an empty slot, taking 'm' most condition checks. **Search/Delete:** For both search and delete functions, we first locate the element. Calculating its primary hash and checking if it matches the ID. If it does, the search and delete is O(1). It will iterate through m times and since the number is constant, the run time for the run function would be O(1).	**Store (Insertion):** To insert a key, we first calculate the primary hash and place the element in the chain at that index. It takes at most 'm' condition checks. **Search/Delete:** For both search and deletion functions, we first go to the chain and iterate over the chain elements to find the ID. Chain's length remains constant, assuming that there are at most m=O(1) collisions. It will iterate through m times and since the number is constant, the run time for the run function would be O(1).