Approach Rationale

This project implements two different hash table approaches for storing file blocks: open addressing with double hashing and separate chaining. The implementation uses dynamic arrays (vectors).

Open addressing was implemented using a vector of FileBlock pointers to allow for efficient storage and retrieval while maintaining flexibility for deletions. Separate chaining was implemented using a vector of vectors, where each chain can dynamically grow, providing efficient collision handling without need for pre-allocated space.

The project was implemented this way over using a linked list as the chains will not need to be resized, memory will be allocated as needed (no pre-allocation), and will allow easier insertion and deletion. A linked list will have required additional memory for node pointers, more complex memory management, as well as manual memory tracking.

Class Design

Class FileBlock

This class manages individual file blocks containing data and associated checksums

Public variables and functions:

FileBlock(unsigned int ID, const std::string& data);
Parameters: unsigned int ID, const std::string& data
Return Type: n/a
Rationale: The constructor initializes a file block with given ID and data, creates a 500-byte payload initialized to zeros, and computes initial checksum.

unsigned int compute_checksum() const;
Parameters: n/a
Return Type: unsigned int
Rationale: This function computes checksum of payload data according to the formula provided for all 500 bytes and returns it as an unsigned integer.

bool validate_checksum() const;
Parameters: n/a
Return Type: bool (T/F)
Rationale: This function validates by comparing stored and computed checksums, T if matching/valid, F if not matching.

void set_payload(const std::string& data, bool recompute_checksum = true);
Parameters: const std::string& data, bool recompute_checksum
Return Type: n/a
Rationale: This function sets payload data and optionally updates checksum when required.

Private Variables:

unsigned int ID; //stores unique identifier for the file block
ID should not be modifiable after creation to maintain data integrity

std::vector<char> payload; //fixed-size 500-byte array to store file data
Direct access could corrupt data/checksum, therefore made private

unsigned int checksum; //stores computed checksum value for data validation
Must be managed internally cannot be changed by public, therefore private

Class HashTable
This class implements the hash table with both collision resolution methods

Public variables and functions:
bool store(unsigned int id, const std::string& data);
Parameters: unsigned int id, const std::string& data
Return Type: bool
Rationale: Stores a new file block with collision handling based on selected method, T if stored, F if not or if already stored

bool search(unsigned int id, unsigned int& position) const;
Parameters: unsigned int id, unsigned int& position
Return Type: bool
Rationale: Searches for file block and returns T if found.

bool remove(unsigned int id);
Parameters: unsigned int id
Return Type: bool
Rationale: This function removes file block, returns T if removed

bool corrupt(unsigned int id, const std::string& data);
Parameters: ID and new data
Return Type: bool indicating success
Rationale: This function corrupts block data without updating checksum value

bool validate(unsigned int id) const;
Parameters: block ID
Return Type: bool indicating validity
Rationale: This function checks if block exists and has valid checksum

bool is_chaining() const;
Parameters: n/a
Return Type: bool
Rationale: returns T if chaining used

bool print_chain(int index) const;
Parameters: index
Return Type: bool

Rationale: Prints contents of specified chain, T for successful print

Private variables and functions:

unsigned int table_size; //stores size of hash table

Size should not be modifiable after creation, therefore private

bool use_chaining; //determines collision resolution method

Strategy should not change after initialization, therefore private

std::vector<FileBlock> table* //array for open addressing implementation

Direct access could break hash table invariants, therefore private

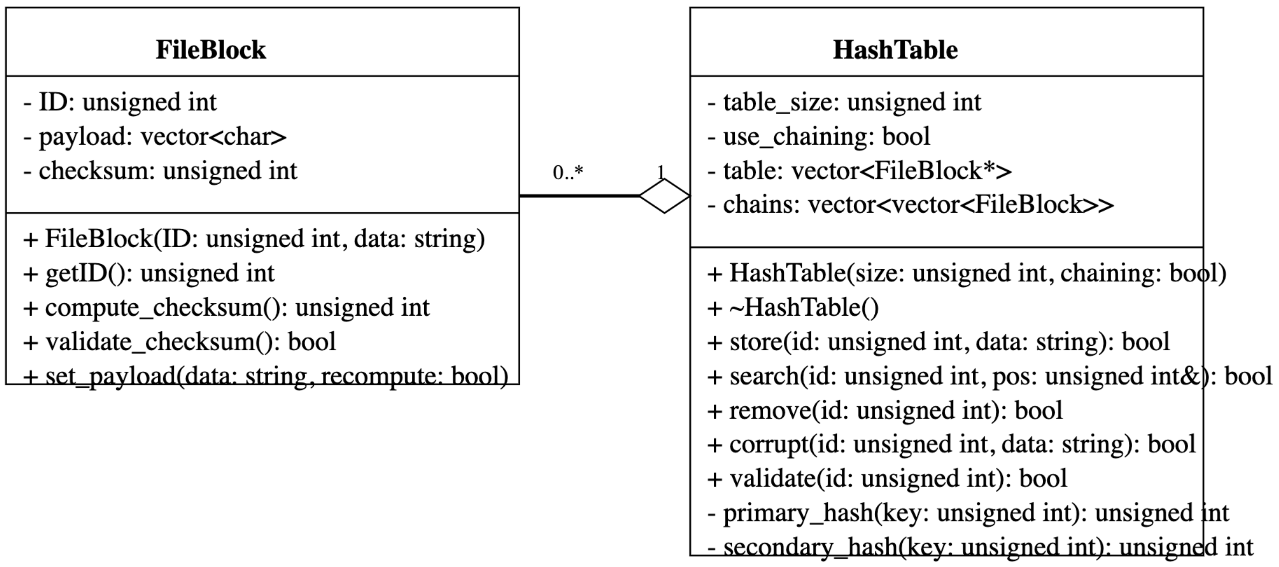
std::vector<std::vector<FileBlock>> chains //array of chains for separate chaining

Direct manipulation could corrupt hash structure, therefore private

unsigned int primary_hash(unsigned int key) const;
unsigned int secondary_hash(unsigned int key) const;

Both are private since they should not me modifiable by the public

<u>UML Diagram</u>


FileBlock
- ID: unsigned int
- payload: vector<char>
- checksum: unsigned int
+ FileBlock(ID: unsigned int, data: string)
+ getID(): unsigned int
+ compute_checksum(): unsigned int
+ validate_checksum(): bool
+ set_payload(data: string, recompute: bool)
0..*
1
HashTable
- table_size: unsigned int
- use_chaining: bool
- table: vector<FileBlock*>
- chains: vector<vector<FileBlock>>
+ HashTable(size: unsigned int, chaining: bool)
+ ~HashTable()
+ store(id: unsigned int, data: string): bool
+ search(id: unsigned int, pos: unsigned int&): bool
+ remove(id: unsigned int): bool
+ corrupt(id: unsigned int, data: string): bool
+ validate(id: unsigned int): bool
- primary_hash(key: unsigned int): unsigned int
- secondary_hash(key: unsigned int): unsigned int

Runtime Analysis

Primary hash computation: O(1)
Secondary hash computation: O(1)

Chaining

STORE Command
Computing hash: O(1)
Checking for duplicates in chain: O(m)
Inserting at end of chain: O(1)
Total: O(m) where m << T and m is O(1), therefore O(1)

SEARCH Command
Computing hash: O(1)
Searching chain: O(m)
Total: O(m) where m << T and m is O(1), therefore O(1)

DELETE Command
Computing hash: O(1)
Finding element: O(m)
Removing from vector: O(m)
Total: O(m) where m << T and m is O(1), therefore O(1)

Double Hashing - Open Addressing

STORE Command
Computing hashes: O(1)
Probing sequence: O(m)
Total: O(m) where m << T and m is O(1), therefore O(1)

SEARCH Command
Computing hashes: O(1)
Probing sequence: O(m)
Total: O(m) where m << T and m is O(1), therefore O(1)

DELETE Command
Computing hashes: O(1)
Probing sequence: O(m)
Total: O(m) where m << T and m is O(1), therefore O(1)