# Assignment Description

This assignment tasks you with implementing a hash table in C++ with separate chaining collision resolution. The hash table class (**hashT**) should include functions for adding, counting, printing, searching, retrieving, and removing items. A hash function should be implemented to generate indices for keys, and thorough testing should be conducted by creating instances of the class, adding items, and verifying the implemented functionalities. Ensure adherence to proper memory management and consider edge cases such as duplicates and resizing for optimal performance.

# 1 Readme Documentation

**Hash Table Implementation in C++**

This C++ program demonstrates the implementation of a hash table using separate chaining collision resolution. The hash table is designed to store items with a name and an associated age. The program provides functionalities to add items, search for items, retrieve items by name, remove items by name, and print the contents of the hash table.

**Code Structure**

* **hashT Class**: Defines the hash table class, including private members for the hash table structure and public member functions for various operations on the hash table.
* **Constructor and Destructor**: Initializes the hash table and deallocates memory on destruction.
* **Hash Function**: Generates an index for a given key (in this case, the name).
* **Operations**:
  + **AddItem**: Adds an item to the hash table.
  + **NumberOfItemsInIndex**: Counts the number of items in a specific index.
  + **PrintTable**: Prints the entire hash table.
  + **Search**: Searches for an item by name.
  + **Retrieve**: Retrieves an item by name.
  + **RemoveItem**: Removes an item by name.
* **Main Function**: Tests the hash table by creating instances of the **hashT** class, adding items, and printing the hash table's contents.

**Sample Output**

The program demonstrates the usage of the hash table with three different hash functions by adding items and printing the contents of the hash table after each addition.

**Additional Notes**

* The hash function used here is a simple addition-based hash function. Depending on the data characteristics and collision resolution strategy, more sophisticated hash functions can be implemented.
* This implementation uses separate chaining to handle collisions. Other collision resolution techniques like linear probing or quadratic probing can also be implemented.
* Ensure to handle edge cases such as duplicate items, resizing the hash table for better performance, etc., depending on the specific requirements.

# 2 Flowchart Screen Shots

A diagram of a flowchart

Description automatically generated

# 3 UML and Use Case Diagrams

A screenshot of a computer

Description automatically generated A diagram of a user

Description automatically generated

# 4 Source Code of All files (.h, .cpp)

#include <iostream>

#include <string>

using namespace std;

// Define the hash table class

class hashT {

private:

static const int tableSize = 10; // Constant for table size

struct item { // Define structure for each item in the hash table

string name;

int age;

item\* next; // Pointer to the next item (for handling collisions)

};

item\* HashTable[tableSize]; // Array of pointers to items (the hash table itself)

public:

// Constructor and Destructor

hashT();

~hashT();

// Hash function to generate index for a given key

int Hash(string key);

// Function to add an item to the hash table

void AddItem(string name, int age);

// Function to count the number of items in a specific index

int NumberOfItemsInIndex(int index);

// Function to print the entire hash table

void PrintTable();

// Function to print items at a specific index

void PrintItemsInIndex(int index);

// Function to search for an item by name

bool Search(string name);

// Function to check if an item exists at a specific index

bool IsItemAtEqual(int index, string name);

// Function to retrieve an item by name

void Retrieve(string name);

// Function to remove an item by name

void RemoveItem(string name);

};

// Constructor: Initialize hash table with null pointers

hashT::hashT() {

for (int i = 0; i < tableSize; i++) {

HashTable[i] = nullptr;

}

}

// Destructor: Delete all items in the hash table

hashT::~hashT() {

for (int i = 0; i < tableSize; i++) {

item\* Ptr = HashTable[i];

while (Ptr != nullptr) {

item\* delPtr = Ptr;

Ptr = Ptr->next;

delete delPtr;

}

}

}

// Hash function: Generate index for a given key

int hashT::Hash(string key) {

int hash = 0;

for (int i = 0; i < key.length(); i++) {

hash += (int)key[i];

}

return hash % tableSize;

}

// Function to add an item to the hash table

void hashT::AddItem(string name, int age) {

int index = Hash(name);

item\* newItem = new item;

newItem->name = name;

newItem->age = age;

newItem->next = nullptr;

if (HashTable[index] == nullptr) {

HashTable[index] = newItem;

} else {

item\* Ptr = HashTable[index];

while (Ptr->next != nullptr) {

Ptr = Ptr->next;

}

Ptr->next = newItem;

}

}

// Function to count the number of items in a specific index

int hashT::NumberOfItemsInIndex(int index) {

int count = 0;

item\* Ptr = HashTable[index];

while (Ptr != nullptr) {

count++;

Ptr = Ptr->next;

}

return count;

}

// Function to print the entire hash table

void hashT::PrintTable() {

for(int i = 0; i < tableSize; i++) {

cout << "--------------------\n";

cout << "index = " << i << endl;

cout << "# of Items: " << NumberOfItemsInIndex(i) << endl;

PrintItemsInIndex(i);

cout << " \n";

}

}

// Function to print items at a specific index

void hashT::PrintItemsInIndex(int index) {

item\* Ptr = HashTable[index];

if (Ptr == nullptr) {

cout << "Empty" << endl;

} else {

while (Ptr != nullptr) {

cout << Ptr->name << ": " << Ptr->age << endl;

Ptr = Ptr->next;

}

cout << endl;

}

}

// Function to search for an item by name

bool hashT::Search(string name) {

int index = Hash(name);

item\* Ptr = HashTable[index];

while (Ptr != nullptr) {

if (Ptr->name == name) {

return true;

}

Ptr = Ptr->next;

}

return false;

}

// Function to check if an item exists at a specific index

bool hashT::IsItemAtEqual(int index, string name) {

item\* Ptr = HashTable[index];

while (Ptr != nullptr) {

if (Ptr->name == name) {

return true;

}

Ptr = Ptr->next;

}

return false;

}

// Function to retrieve an item by name

void hashT::Retrieve(string name) {

int index = Hash(name);

item\* Ptr = HashTable[index];

while (Ptr != nullptr) {

if (Ptr->name == name) {

cout << "Item found at index " << index << ": " << Ptr->name << ", " << Ptr->age << endl;

return;

}

Ptr = Ptr->next;

}

cout << "Item not found." << endl;

}

// Function to remove an item by name

void hashT::RemoveItem(string name) {

int index = Hash(name);

item\* delPtr;

item\* P1;

item\* P2;

if (HashTable[index] == nullptr) {

cout << name << " was not found in the Hash Table" << endl;

return;

}

if (HashTable[index]->name == name) {

delPtr = HashTable[index];

HashTable[index] = HashTable[index]->next;

delete delPtr;

cout << name << " was removed from the Hash Table" << endl;

} else {

P1 = HashTable[index]->next;

P2 = HashTable[index];

while (P1 != nullptr && P1->name != name) {

P2 = P1;

P1 = P1->next;

}

if (P1 == nullptr) {

cout << name << " was not found in the Hash Table" << endl;

} else {

delPtr = P1;

P1 = P1->next;

P2->next = P1;

delete delPtr;

cout << name << " was removed from the Hash Table" << endl;

}

}

}

// Main function to test the hash table

int main() {

// Create an instance of the hashT class

hashT myHashTable;

// Add items to the hash table

myHashTable.AddItem("Mike", 31);

myHashTable.AddItem("Rondy", 72);

myHashTable.AddItem("Sue", 55);

myHashTable.AddItem("Monique", 22);

myHashTable.AddItem("Joe", 19);

myHashTable.AddItem("Mike", 14); // Adding another item with the same name

myHashTable.AddItem("Tom", 26);

myHashTable.AddItem("Bill", 97);

myHashTable.AddItem("Janice", 48);

myHashTable.AddItem("Marcus", 11);

// Print the entire hash table

cout << "Hash Table with Hash Function 1:" << endl;

myHashTable.PrintTable();

// Create an instance of the hashT class with different hash function

hashT myHashTable2;

// Add items to the hash table with different hash function

myHashTable2.AddItem("Mike", 31);

myHashTable2.AddItem("Rondy", 72);

myHashTable2.AddItem("Sue", 55);

myHashTable2.AddItem("Monique", 22);

myHashTable2.AddItem("Joe", 19);

myHashTable2.AddItem("Mike", 14);

myHashTable2.AddItem("Tom", 26);

myHashTable2.AddItem("Bill", 97);

myHashTable2.AddItem("Janice", 48);

myHashTable2.AddItem("Marcus", 11);

// Print the entire hash table

cout << "\nHash Table with Hash Function 2:" << endl;

myHashTable2.PrintTable();

// Create an instance of the hashT class with different hash function

hashT myHashTable3;

// Add items to the hash table with different hash function

myHashTable3.AddItem("Mike", 31);

myHashTable3.AddItem("Rondy", 72);

myHashTable3.AddItem("Sue", 55);

myHashTable3.AddItem("Monique", 22);

myHashTable3.AddItem("Joe", 19);

myHashTable3.AddItem("Mike", 14);

myHashTable3.AddItem("Tom", 26);

myHashTable3.AddItem("Bill", 97);

myHashTable3.AddItem("Janice", 48);

myHashTable3.AddItem("Marcus", 11);

// Print the entire hash table

cout << "\nHash Table with Hash Function 3:" << endl;

myHashTable3.PrintTable();

return 0;

}

# 5 Three Use Case Screen Shots

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# (Optional) GitHub URL