Program to implement Rehashing:

#include <iostream>

#include <vector>

#include <functional>

class Map {

private:

class MapNode {

public:

int key;

int value;

MapNode\* next;

MapNode(int key, int value) {

this->key = key;

this->value = value;

this->next = NULL;

}

};

// The bucket array where

// the nodes containing K-V pairs are stored

std::vector<MapNode\*> buckets;

// No. of pairs stored - n

int size;

// Size of the bucketArray - b

int numBuckets;

// Default loadFactor

double DEFAULT\_LOAD\_FACTOR = 0.75;

int getBucketInd(int key) {

// Using the inbuilt function from the object class

int hashCode = std::hash<int>()(key);

// array index = hashCode%numBuckets

return (hashCode % numBuckets);

}

public:

Map() {

numBuckets = 5;

buckets.resize(numBuckets);

std::cout << "HashMap created" << std::endl;

std::cout << "Number of pairs in the Map: " << size << std::endl;

std::cout << "Size of Map: " << numBuckets << std::endl;

std::cout << "Default Load Factor : " << DEFAULT\_LOAD\_FACTOR << std::endl;

}

void insert(int key, int value) {

// Getting the index at which it needs to be inserted

int bucketInd = getBucketInd(key);

// The first node at that index

MapNode\* head = buckets[bucketInd];

// First, loop through all the nodes present at that index

// to check if the key already exists

while (head != NULL) {

// If already present the value is updated

if (head->key == key) {

head->value = value;

return;

}

head = head->next;

}

// new node with the K and V

MapNode\* newElementNode = new MapNode(key, value);

// The head node at the index

head = buckets[bucketInd];

// the new node is inserted

// by making it the head

// and it's next is the previous head

newElementNode->next = head;

buckets[bucketInd] = newElementNode;

std::cout << "Pair(" << key << ", " << value << ") inserted successfully." << std::endl;

// Incrementing size

// as new K-V pair is added to the map

size++;

// Load factor calculated

double loadFactor = (1 \* size) / numBuckets;

std::cout << "Current Load factor = " << loadFactor << std::endl;

// If the load factor is > 0.75, rehashing is done

if (loadFactor > DEFAULT\_LOAD\_FACTOR) {

std::cout << loadFactor << " is greater than " << DEFAULT\_LOAD\_FACTOR << std::endl;

std::cout << "Therefore Rehashing will be done." << std::endl;

// Rehash

rehash();

std::cout << "New Size of Map: " << numBuckets << std::endl;

}

std::cout << "Number of pairs in the Map: " << size << std::endl;

}

void rehash() {

std::cout << "\n\*\*\*Rehashing Started\*\*\*\n" << std::endl;

// The present bucket list is made temp

std::vector<MapNode\*> temp = buckets;

// New bucketList of double the old size is created

buckets.resize(2 \* numBuckets);

for (int i = 0; i < 2 \* numBuckets; i++) {

// Initialised to null

buckets[i] = NULL;

}

// Now size is made zero

// and we loop through all the nodes in the original bucket list(temp)

// and insert it into the new list

size = 0;

numBuckets \*= 2;

for (int i = 0; i < temp.size(); i++) {

// head of the chain at that index

MapNode\* head = temp[i];

while (head != NULL) {

int key = head->key;

int val = head->value;

// calling the insert function for each node in temp

// as the new list is now the bucketArray

insert(key, val);

head = head->next;

}

}

std::cout << "\*\*\*Rehashing Done\*\*\*\n" << std::endl;

}

};

int main() {

Map map;

// Inserting elements

map.insert(1, 1);

map.insert(2, 2);

map.insert(3, 3);

map.insert(4, 4);

map.insert(5, 5);

map.insert(6, 6);

map.insert(7, 7);

map.insert(8, 8);

map.insert(9, 9);

map.insert(10, 10);

return 0;

}