Date: 28/04/2021

Practical 10:

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Hash tables are important data structure. However, hash tables are subject to collision. Implement a program with a collision resolution technique with Insert, delete and display operation

Code:

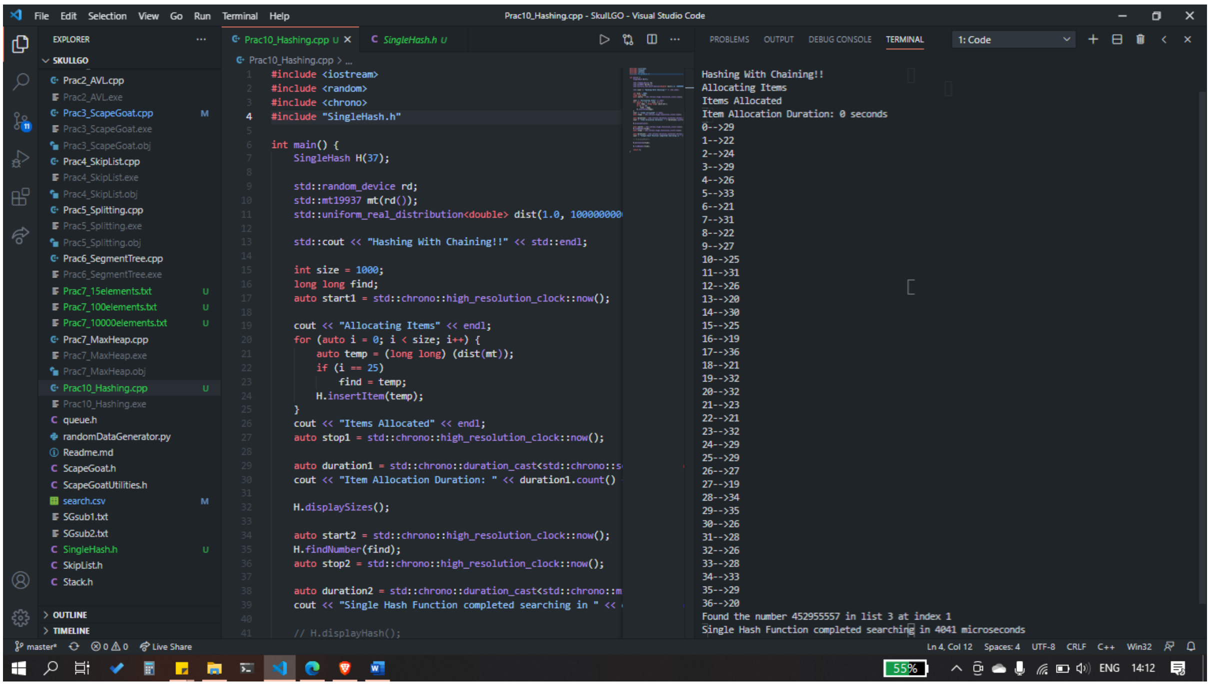
Prac10_Hashing.cpp

```cpp
#include <iostream>
#include <random>
#include <chrono>
#include "SingleHash.h"

int main() {
    SingleHash H(37);

    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist(1.0, 1000000000.0);

    std::cout << "Hashing With Chaining!!" << std::endl;

    int size = 1000;
    long long find;
    auto start1 = std::chrono::high_resolution_clock::now();

    cout << "Allocating Items" << endl;
    for (auto i = 0; i < size; i++) {
        auto temp = (long long) (dist(mt));
        if (i == 25)
            find = temp;
        H.insertItem(temp);
    }
    cout << "Items Allocated" << endl;
    auto stop1 = std::chrono::high_resolution_clock::now();

    auto duration1 = std::chrono::duration_cast<std::chrono::seconds>(stop1 - start1);
    cout << "Item Allocation Duration: " << duration1.count() << " seconds" << endl;

    H.displaySizes();

    auto start2 = std::chrono::high_resolution_clock::now();
    H.findNumber(find);
    auto stop2 = std::chrono::high_resolution_clock::now();

    auto duration2 = std::chrono::duration_cast<std::chrono::microseconds>(stop2 - start2);
    cout << "Single Hash Function completed searching in " << duration2.count() << " microseconds" << endl;

    // H.displayHash();

    H.deleteItem(find);
    H.findNumber(find);

    return 0;
}
```

SingleHash.h

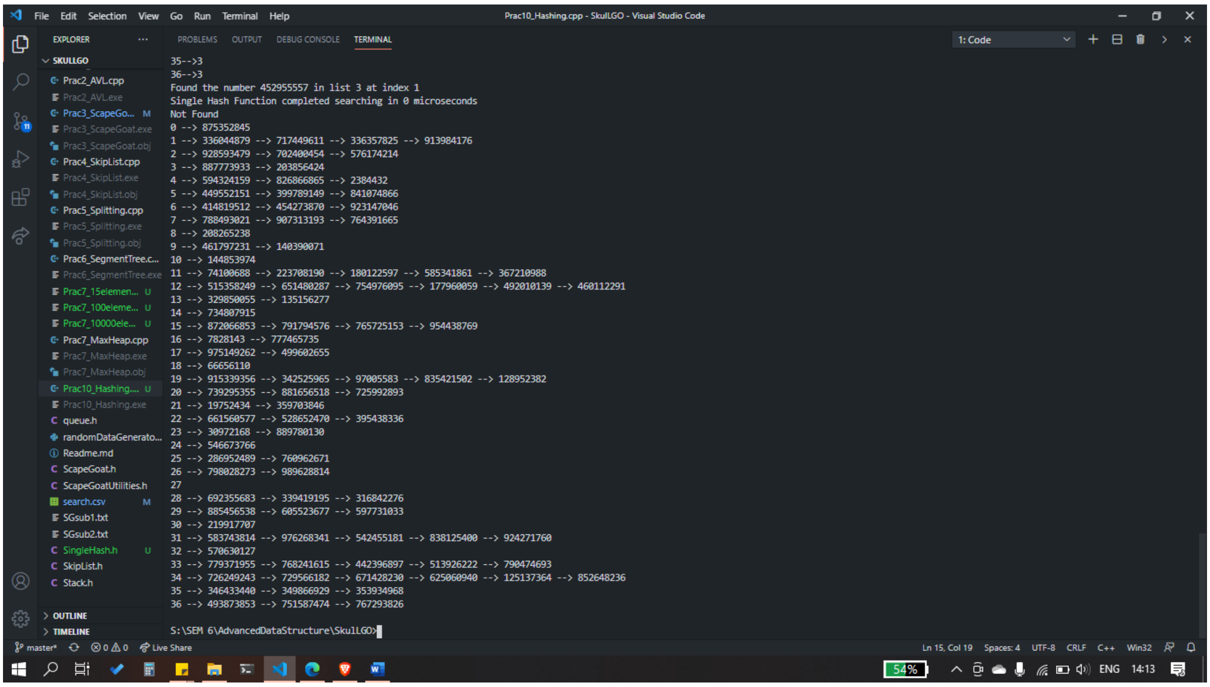
```
#include <iostream>
#include <vector>

using namespace std;

class SingleHash {
    int buckets;
    std::vector<long long> *table;

public:
    explicit SingleHash(int v) {
        buckets = v;
        table = new vector<long long>[buckets];
    }

    void insertItem(long long key) {
        int index = hashFunction(key);
        table[index].push_back(key);
    }

    void deleteItem(long long key) {
        int index = hashFunction(key);

        std::vector<long long>::iterator i;
        for (i = table[index].begin(); i != table[index].end(); i++) {
            if (*i == key)
                break;
        }

        if (i != table[index].end()) {
            table[index].erase(i);
        }
    }

    int hashFunction(long long x) {
        return x % (long long) buckets;
    }

    void displayHash() {
        for (int i = 0; i < buckets; i++) {
            cout << i;
            for (auto x : table[i])
                cout << " --> " << x;
            cout << endl;
        }
    }

    void displaySizes() {
        for (int i = 0; i < buckets; i++) {
            cout << i << "-->" << table[i].size() << '\n';
        }
    }

    void findNumber(long long key) {
        int index = hashFunction(key);
        int k = 0;
        std::vector<long long>::iterator i;
        for (i = table[index].begin(); i != table[index].end(); i++) {
            k++;
            if (*i == key)
                break;
        }

        if (i != table[index].end()) {
            cout << "Found the number " << key << " in list " << index << " at index " << k << '\n';
        } else {
            cout << "Not Found" << '\n';
        }
    }

    long long getNumber(int index, int lst) {
        auto i = table[lst].begin();
        advance(i, index - 1);
        return *i;
    }
};
```

Snapshot of the output:



```cpp
#include <iostream>
#include <random>
#include <chrono>
#include "SingleHash.h"

int main() {
    SingleHash H(37);

    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist(1.0, 100000000

    std::cout << "Hashing With Chaining!!" << std::endl;

    int size = 1000;
    long long find;
    auto start1 = std::chrono::high_resolution_clock::now();

    cout << "Allocating Items" << endl;
    for (auto i = 0; i < size; i++) {
        auto temp = (long long) (dist(mt));
        if (i == 25)
            find = temp;
        H.insertItem(temp);
    }
    cout << "Items Allocated" << endl;
    auto stop1 = std::chrono::high_resolution_clock::now();

    auto duration1 = std::chrono::duration_cast<std::chrono::s
    cout << "Item Allocation Duration: " << duration1.count()

    H.displaySizes();

    auto start2 = std::chrono::high_resolution_clock::now();
    H.findNumber(find);
    auto stop2 = std::chrono::high_resolution_clock::now();

    auto duration2 = std::chrono::duration_cast<std::chrono::m
    cout << "Single Hash Function completed searching in " <<

    // H.displayHash();
```

```
Hashing With Chaining!!
Allocating Items
Items Allocated
Item Allocation Duration: 0 seconds
0-->29
1-->22
2-->24
3-->29
4-->26
5-->33
6-->21
7-->31
8-->22
9-->27
10-->25
11-->31
12-->26
13-->20
14-->30
15-->25
16-->19
17-->36
18-->21
19-->32
20-->32
21-->23
22-->21
23-->32
24-->29
25-->29
26-->27
27-->19
28-->34
29-->35
30-->26
31-->28
32-->26
33-->28
34-->33
35-->29
36-->20
Found the number 452955557 in list 3 at index 1
Single Hash Function completed searching in 4041 microseconds
```



```
35-->3
36-->3
Found the number 452955557 in list 3 at index 1
Single Hash Function completed searching in 0 microseconds
Not Found
0 --> 875352845
1 --> 336044879 --> 717449611 --> 336357825 --> 913984176
2 --> 928593479 --> 702400454 --> 576174214
3 --> 887773933 --> 203856424
4 --> 594324159 --> 826866865 --> 2384432
5 --> 449552151 --> 399789149 --> 841074866
6 --> 414819512 --> 454273870 --> 923147046
7 --> 788493021 --> 907313193 --> 764391665
8 --> 208265238
9 --> 461797231 --> 140390071
10 --> 144853974
11 --> 74100688 --> 223708190 --> 180122597 --> 585341861 --> 367210988
12 --> 515358249 --> 651480287 --> 754976095 --> 177960059 --> 492010139 --> 460112291
13 --> 329850055 --> 135156277
14 --> 734807915
15 --> 872066853 --> 791794576 --> 765725153 --> 954438769
16 --> 7828143 --> 777465735
17 --> 975149262 --> 499602655
18 --> 66656110
19 --> 915339356 --> 342525965 --> 97005583 --> 835421502 --> 128952382
20 --> 739295355 --> 881656518 --> 725992893
21 --> 19752434 --> 359703846
22 --> 661560577 --> 528652470 --> 395438336
23 --> 30972168 --> 889780130
24 --> 546673766
25 --> 286952489 --> 760962671
26 --> 798028273 --> 989628814
27
28 --> 692355683 --> 339419195 --> 316842276
29 --> 885456538 --> 605523677 --> 597731033
30 --> 219917707
31 --> 583743814 --> 976268341 --> 542455181 --> 838125400 --> 924271760
32 --> 570630127
33 --> 779371955 --> 768241615 --> 442396897 --> 513926222 --> 790474693
34 --> 726249243 --> 729566182 --> 671428230 --> 625060940 --> 125137364 --> 852648236
35 --> 346433440 --> 349866929 --> 353934968
36 --> 493873853 --> 751587474 --> 767293826

S:\SEM 6\AdvancedDataStructure\SkulLGO>
```

Conclusion:

Hashing an incredible way to store and retrieve object quickly.