

lab_12.cpp

```
1  //<----Lab 12 - Hashing---->
2
3  #include <iostream>
4  #include <list>
5  #include <algorithm>
6  using namespace std;
7
8  class HashMapTable
9  {
10 public:
11     int table_size;
12     list<int> *table;
13
14     HashMapTable(int key)
15     {
16         this->table_size = key;
17         table = new list<int>[table_size];
18     }
19
20     int hashFunction(int key)
21     {
22         return key % table_size;
23     }
24
25     void insertElement(int key)
26     {
27         int index = hashFunction(key);
28         table[index].push_back(key);
29     }
30
31     void deleteElement(int key)
32     {
33         int index = hashFunction(key);
34         list<int>::iterator i = find(table[index].begin(), table[index].end(), key);
35
36         if (i != table[index].end())
37         {
38             table[index].erase(i);
39         }
40     }
41
42     void displayHashTable()
43     {
44         for (int i = 0; i < table_size; i++)
45         {
46             cout << i;
47             for (auto j : table[i])
48             {
49                 cout << " ==> " << j;
50             }
51             cout << endl;
52         }
53     }
```

```
54 };
55
56 // Q1. Implement a hashing algorithm using Linear Probing.
57
58 class LinearProbingHashTable : public HashMapTable
59 {
60 public:
61     LinearProbingHashTable(int key) : HashMapTable(key) {}
62
63     void insertElementLinearProbing(int key)
64     {
65         int index = hashFunction(key);
66
67         while (!table[index].empty())
68         {
69             index = (index + 1) % table_size;
70         }
71
72         table[index].push_back(key);
73     }
74 };
75
76 // Q2. Implement a hashing algorithm using Quadratic Probing.
77
78 class QuadraticProbingHashTable : public HashMapTable
79 {
80 public:
81     QuadraticProbingHashTable(int key) : HashMapTable(key) {}
82
83     void insertElementQuadraticProbing(int key)
84     {
85         int index = hashFunction(key);
86         int i = 1;
87
88         while (!table[index].empty())
89         {
90             index = (index + i * i) % table_size;
91             i++;
92         }
93
94         table[index].push_back(key);
95     }
96 };
97
98 // Q3. Implement a hashing algorithm using Double Hashing.
99
100 class DoubleHashingHashTable : public HashMapTable
101 {
102 public:
103     DoubleHashingHashTable(int key) : HashMapTable(key) {}
104
105     int hashFunction2(int key)
106     {
107         return 1 + (key % (table_size - 2)); // Ensure step size is relatively prime to
table_size
108     }
```

```
109
110 void insertElementDoubleHashing(int key)
111 {
112     int index = hashFunction(key);
113     int step = hashFunction2(key);
114
115     // Iterate until an empty slot is found or a full cycle is completed
116     while (!table[index].empty() && table[index].front() != key)
117     { // Check for existing key
118         index = (index + step) % table_size;
119         if (index == hashFunction(key))
120         { // Full cycle check
121             cout << "Hash table is full\n";
122             return;
123         }
124     }
125
126     table[index].push_back(key);
127 }
128 };
129
130 int main()
131 {
132     int arr[] = {1,2,3,4,5,6};
133     int n = sizeof(arr) / sizeof(arr[0]);
134
135     // Linear Probing
136     LinearProbingHashTable linearTable(6);
137     for (int i = 0; i < n; i++)
138         linearTable.insertElementLinearProbing(arr[i]);
139
140     cout << "Linear Probing Hash Table:" << endl;
141     linearTable.displayHashTable();
142     cout << endl;
143
144     // Quadratic Probing
145     QuadraticProbingHashTable quadraticTable(6);
146     for (int i = 0; i < n; i++)
147         quadraticTable.insertElementQuadraticProbing(arr[i]);
148
149     cout << "Quadratic Probing Hash Table:" << endl;
150     quadraticTable.displayHashTable();
151     cout << endl;
152
153     // Double Hashing
154     DoubleHashingHashTable doubleHashingTable(6);
155     for (int i = 0; i < n; i++)
156         doubleHashingTable.insertElementDoubleHashing(arr[i]);
157
158     cout << "Double Hashing Hash Table:" << endl;
159     doubleHashingTable.displayHashTable();
160
161     return 0;
162 }
163
```