

Name : Prem Sah
PRN : 123B1B234
Batch : D-1

Assignment No.11

Consider an employee database of N employees. Make use of a hash table implementation to quickly look up the employer's id number.

Code :

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
public:
    int id;
    string name;
    float salary;
    Employee* next; // Pointer to the next employee (for chaining)

    Employee(int id, string name, float salary)
        : id(id), name(name), salary(salary), next(nullptr) {}
};

class HashTable {
    int bucket_count;
    Employee** table; // Array of pointers to Employee objects (chains)

public:
    HashTable(int size) : bucket_count(size) {
        table = new Employee*[bucket_count];
        for (int i = 0; i < bucket_count; i++) {
            table[i] = nullptr; // Initialize all buckets to nullptr
        }
    }

    ~HashTable() {
        for (int i = 0; i < bucket_count; i++) {
            Employee* current = table[i];
            while (current != nullptr) {
                Employee* to_delete = current;
                current = current->next;
            }
            delete to_delete;
        }
    }
};
```

```

        delete to_delete;
    }
}
delete[] table;
}

int hash_function(int key) {
    return key % bucket_count;
}

void insert(int id, string name, float salary) {
    Employee* new_record = new Employee(id, name, salary);
    int index = hash_function(id);

    // Insert at the beginning of the chain (linked list) for simplicity
    if (table[index] == nullptr) {
        table[index] = new_record;
    } else {
        new_record->next = table[index];
        table[index] = new_record;
    }

    cout << "Inserted Employee " << name << " ID " << id << " Index " << index <<
    ".\n";
}

void display() {
    for (int i = 0; i < bucket_count; i++) {
        cout << "Bucket " << i << ": ";
        Employee* current = table[i];
        while (current != nullptr) {
            cout << "[" << current->id << ": " << current->name << ", " <<
current->salary << "]" << " ";
            current = current->next;
        }
        cout << "\n";
    }
}

};

int main() {
    HashTable ht(10);
    ht.insert(123, "Prem", 50000);
    ht.insert(456, "Niraj", 60000);
}

```

```
ht.insert(272, "Sujit", 20000);
ht.insert(385, "Sahil", 35000);
ht.insert(100, "Nitesh", 16000);
ht.insert(601, "Siddharth", 80000);
ht.insert(110, "Dhiraj", 30000);
ht.insert(116, "Bhaskar", 75000);

ht.display();

return 0;
}
```

Output :

Inserted Employee Prem ID 123 Index 3.
Inserted Employee Niraj ID 456 Index 6.
Inserted Employee Sujit ID 272 Index 2.
Inserted Employee Sahil ID 385 Index 5.
Inserted Employee Nitesh ID 100 Index 0.
Inserted Employee Siddharth ID 601 Index 1.
Inserted Employee Dhiraj ID 110 Index 0.
Inserted Employee Bhaskar ID 116 Index 6.
Bucket 0: [110: Dhiraj, 30000] [100: Nitesh, 16000]
Bucket 1: [601: Siddharth, 80000]
Bucket 2: [272: Sujit, 20000]
Bucket 3: [123: Prem, 50000]
Bucket 4:
Bucket 5: [385: Sahil, 35000]
Bucket 6: [116: Bhaskar, 75000] [456: Niraj, 60000]
Bucket 7:
Bucket 8:
Bucket 9:

main.cpp

Share

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Employee {
5 public:
6     int id;
7     string name;
8     float salary;
9     Employee* next; // Pointer to the next employee (for chaining)
10
11     Employee(int id, string name, float salary)
12         : id(id), name(name), salary(salary), next(nullptr) {}
13 };
14
15 class HashTable {
16     int bucket_count;
17     Employee** table; // Array of pointers to Employee objects (chains)
18
19 public:
20     HashTable(int size) : bucket_count(size) {
21         table = new Employee*[bucket_count];
22         for (int i = 0; i < bucket_count; i++) {
23             table[i] = nullptr; // Initialize all buckets to nullptr
24         }
25     }
26
27     ~HashTable() {
28         for (int i = 0; i < bucket_count; i++) {
29             Employee* current = table[i];
30             while (current != nullptr) {
31                 Employee* to_delete = current;
32                 current = current->next;
33                 delete to_delete;
34             }
35         }
36         delete[] table;
37     }
38
39     int hash_function(int key) {
40         return key % bucket_count;
41     }
42
43     void insert(int id, string name, float salary) {
44         Employee* new_record = new Employee(id, name, salary);
45         int index = hash_function(id);
46
47         // Insert at the beginning of the chain (linked list) for simplicity
48         if (table[index] == nullptr) {
49             table[index] = new_record;
50         } else {
51             new_record->next = table[index];
52             table[index] = new_record;
53         }
54
55         cout << "Inserted Employee " << name << " ID " << id << " Index " << index << ".\n";
56     }
57
58     void display() {
59         for (int i = 0; i < bucket_count; i++) {
60             cout << "Bucket " << i << ": ";
61             Employee* current = table[i];
62             while (current != nullptr) {
63                 cout << "[" << current->id << ": " << current->name << ", " << current->salary << "]" << " ";
64                 current = current->next;
65             }
66             cout << "\n";
67         }
68     }
69 };
70
71 int main() {
72     HashTable ht(10);
73     ht.insert(123, "Prem", 50000);
74     ht.insert(456, "Niraj", 60000);
75     ht.insert(272, "Sujit", 20000);
76     ht.insert(385, "Sahil", 35000);
77     ht.insert(100, "Nitesh", 16000);
78     ht.insert(601, "Siddharth", 80000);
79     ht.insert(110, "Dhiraj", 30000);
80     ht.insert(116, "Bhaskar", 75000);
81
82     ht.display();
83
84     return 0;
85 }
86
```

Output

```
/tmp/Ni5kL2QNIF.o
Inserted Employee Prem ID 123 Index 3.
Inserted Employee Niraj ID 456 Index 6.
Inserted Employee Sujit ID 272 Index 2.
Inserted Employee Sahil ID 385 Index 5.
Inserted Employee Nitesh ID 100 Index 0.
Inserted Employee Siddharth ID 601 Index 1.
Inserted Employee Dhiraj ID 110 Index 0.
Inserted Employee Bhaskar ID 116 Index 6.
Bucket 0: [110: Dhiraj, 30000] [100: Nitesh, 16000]
Bucket 1: [601: Siddharth, 80000]
Bucket 2: [272: Sujit, 20000]
Bucket 3: [123: Prem, 50000]
Bucket 4:
Bucket 5: [385: Sahil, 35000]
Bucket 6: [116: Bhaskar, 75000] [456: Niraj, 60000]
Bucket 7:
Bucket 8:
Bucket 9:

=== Code Execution Successful ===
```