Name: **Kashif Ali**
Roll No: **20P-0648**
Section: **3D**

Lab-11 Tasks
Hashing

C 01_task.cpp > HashMapTable

```cpp
#include<iostream>

using namespace std;
class HashTableEntry {
public:
    int k;
    int v;
    HashTableEntry *next;
    HashTableEntry(int key): k(key) { }
    HashTableEntry(): next(NULL) { }
    HashTableEntry(int k, int v) {
        this->k = k;
        this->v = v;
    }
};

class HashMapTable {
private:
    HashTableEntry *head;
    HashTableEntry *tail;
public:
    HashTableEntry* Get_Head() { return head; }
    HashMapTable(): head(NULL), tail(NULL) { }
    int HashFun(int);
    void Sorted_Insert(HashTableEntry **H, int);
    void Insert(HashTableEntry *H[], int);
    HashTableEntry* Search(HashTableEntry *p, int);
    void Delete_Value(int);
    ~HashMapTable() { }
};
```

Task 1 code part-a

C 01_task.cpp > HashMapTable

```cpp
// Getting index for inserting values in Hash Table
int HashMapTable::HashFun(int k) {
    return (k % 10);
}

// For insertion in Sorted Order
void HashMapTable::Sorted_Insert(HashTableEntry **H, int k) {
    HashTableEntry *temp = new HashTableEntry(k);
    HashTableEntry *p = *H;
    HashTableEntry *q = NULL;
    temp->next = NULL;
    if(*H == NULL)
        *H = temp;
    else {
        while(p && p->k < k) {
            q = p;
            p = p->next;
        }
        if(p == *H) {
            temp->next = head;
            head = temp;
        }
        else {
            temp->next = *H;
            *H = temp;
        }
    }
}

// For Getting index from Hash Function and passing value to Sorted_Insert Function
void HashMapTable::Insert(HashTableEntry *H[], int k) {
    int index = HashFun(k);
    Sorted_Insert(&H[index], k);
}


// For searching specific Value
HashTableEntry* HashMapTable::Search(HashTableEntry *p, int k) {
    while(p) {
        if(k == p->k) return p;

        p = p->next;
    }
    return NULL;
}
```

Task 1 code part-b

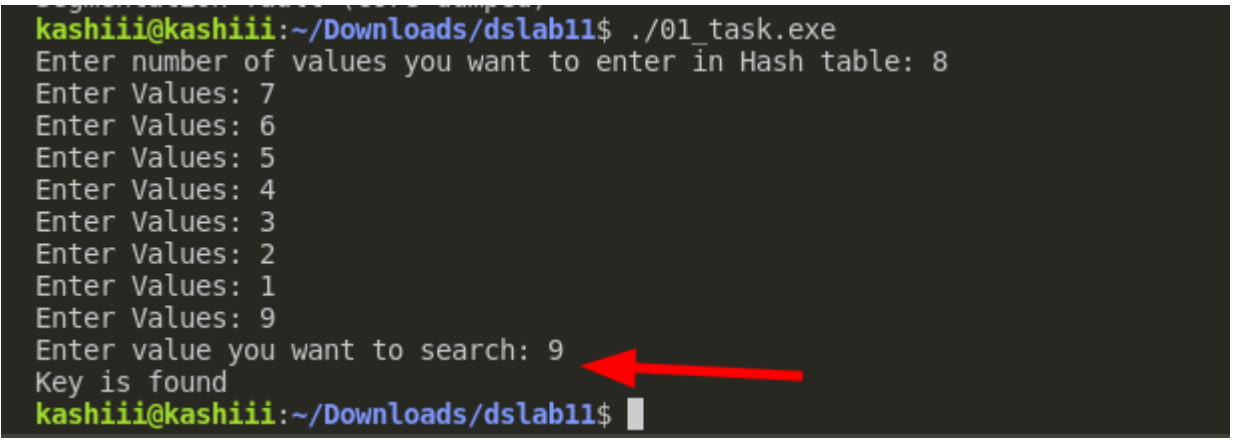
01_task.cpp ●

01_task.cpp > main()

```cpp
// For deleting/removing value
void HashMapTable::Delete_Value(int k) {
    HashTableEntry *p = head, *q = NULL;
    while(p) {
        if(k == p->k) {
            q->next = p->next;
            delete p;
        }
        q = p;
        p = p->next;
    }
}

// Driver Program
int main() {

    int num;
    int value = 0;
    cout << "Enter number of values you want to enter in Hash table: ";  cin >> num;

    HashTableEntry *table[num];
    HashTableEntry *temp;
    for(int i = 0;i < num; i++)
        table[i] = NULL;

    HashMapTable ob1;

    for(int i = 0; i < num; i++) {
        fflush(stdin);
        cout << "Enter Values: ";
        cin >> value;
        ob1.Insert(table, value);
        value = 0;
    }


    cout << "Enter value you want to search: "; cin >> num;
    temp = ob1.Search(table[ob1.HashFun(num)], num);
    if(temp == NULL) {
        cout << "Key is not present in Hash Table\n";
    }
    else {
        cout << "Key is found\n";
    }


    return 0;
}
```

Task 1 code part-c

output


kashiii@kashiii:~/Downloads/dslab11$ ./01_task.exe
Enter number of values you want to enter in Hash table: 8
Enter Values: 7
Enter Values: 6
Enter Values: 5
Enter Values: 4
Enter Values: 3
Enter Values: 2
Enter Values: 1
Enter Values: 9
Enter value you want to search: 9
Key is found
kashiii@kashiii:~/Downloads/dslab11$

02_searching.cpp > ...

```cpp
#include<iostream>
#include<list>
using namespace std;
class Hashing
{
    int hash_bucket; // No. of buckets
// Pointer to an array containing buckets
    list<int> *hashtable;
    public:
        Hashing(int V); // Constructor
// inserts a key into hash table
    void insert_key(int val);
// deletes a key from hash table
    void delete_key(int key);
// hash function to map values to key
    int hashFunction(int x) {
    return (x % hash_bucket);
}
    void displayHash();
    void searching(int val);
};
Hashing::Hashing(int b)
{
    this->hash_bucket = b;
    hashtable = new list<int>[hash_bucket];
}
//insert to hash table
void Hashing::insert_key(int key)
{
    int index = hashFunction(key);
    hashtable[index].push_back(key);
}
void Hashing::delete_key(int key)
{

// get the hash index for key
    int index = hashFunction(key);
// find the key in (inex)th list
    list <int> :: iterator i;  ////////////////////////////////////////////////////////////////
for (i = hashtable[index].begin(); i != hashtable[index].end(); i++) {
    if (*i == key)
        break;
}
// if key is found in hash table, remove it
if (i != hashtable[index].end())
        hashtable[index].erase(i);
}
```

Task 2 code part-a
Searching in hashing

C 02_searching.cpp > ...

```cpp
// display the hash table
void Hashing::displayHash() {
    for (int i = 0; i < hash_bucket; i++) {
        cout<< i;
    for (auto x : hashtable[i])  ////////////////////////////////////////////////////////////////////
        cout<<" > "<<x;
        cout<<endl;
    }
}

void Hashing::searching(int key)
{

// get the hash index for key
    int index = hashFunction(key);
// find the key in (inex)th list
    list <int> :: iterator i;
for (i = hashtable[index].begin();
    i != hashtable[index].end(); i++) {
    if (*i == key)
        break;
}
// if key is found in hash table, remove it
if (i != hashtable[index].end())
        cout<<"Value found: "<<endl;
}
// main program
int main() {
// array that contains keys to be mapped
    int hash_array[] = {11,12,21, 14, 15};
    int n = sizeof(hash_array)/sizeof(hash_array[0]);
    Hashing h(7); // Number of buckets = 7
//insert the keys into the hash table
    for (int i = 0; i< n; i++)
        h.insert_key(hash_array[i]);
// display the Hash table
    cout<<"Hash table created: "<<endl;
    h.displayHash();
// delete 12 from hash table
    h.delete_key(12);
// display the Hash table
cout<<"After Deletion: "<<endl;
    h.displayHash();

cout<<"Searching ... "<<endl;
    h.searching(11);
return 0;
}
```

Task 2 code part-b
Searching in hashing

```
    h.delete_key(12);
// display the Hash table
cout<<"After Deletion: "<<endl;
    h.displayHash();

cout<<"Searching ... "<<endl;
    h.searching(11);
return 0;
}
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
Enter Values: 1
Enter Values: 0
Enter value you want to search: 8
Segmentation fault (core dumped)
kashiii@kashiii:~/Downloads/dslab11$ g++ 02_searching.cpp -o 02_searching.exe
kashiii@kashiii:~/Downloads/dslab11$ ./02_searching.exe
Hash table created:
0 > 21 > 14
1 > 15
2
3
4 > 11
5 > 12
6
After Deletion:
0 > 21 > 14
1 > 15
2
3
4 > 11
5
6
Searching ...
Value found:
kashiii@kashiii:~/Downloads/dslab11$
```

output

C 03_Mul_Div&Midsq.cpp > Folding_Hash(HashTableEntry * [], int)

```cpp
#include <iostream>
#include <cmath>
using namespace std;
class HashTableEntry {
public:
    int k;
    int v;
    HashTableEntry *next;
    HashTableEntry(int key): k(key) { }
    HashTableEntry(): next(NULL) { }
    HashTableEntry(int k, int v) {
        this->k = k;
        this->v = v;
    }
};

class HashMapTable {
private:
    HashTableEntry *head;
    HashTableEntry *tail;
public:
    HashTableEntry* Get_Head() { return head; }
    HashMapTable(): head(NULL), tail(NULL) { }


    int HashFun(int);
    int Sorted_Insert(HashTableEntry **H, int);
    int Insert_Division_Method(HashTableEntry *H[], int);

    int Multiplication_Hash(HashTableEntry *H[], int, int);

    int Mid_Square_Hash(HashTableEntry *H[], int);
    int Middle_Value(int);

    int Folding_Hash(HashTableEntry *H[], int);

    HashTableEntry* Search(HashTableEntry *p, int);
    void Delete_Value(int);
    ~HashMapTable() { }
};
```

Hashing
Important functions
Implementations
part-a

```cpp
// Getting index for inserting values in Hash Table
int HashMapTable::HashFun(int k) {
    return (k % 10);
}

// For insertion in Sorted Order
int HashMapTable::Sorted_Insert(HashTableEntry **H, int k) {
    HashTableEntry *temp = new HashTableEntry(k);
    HashTableEntry *p = *H;
    HashTableEntry *q = NULL;
    temp->next = NULL;
    if(*H == NULL)
        *H = temp;
    else {
        while(p && p->k < k) {
            q = p;
            p = p->next;
        }
        if(p == *H) {
            temp->next = head;
            head = temp;
        }
        else {
            temp->next = *H;
            *H = temp;
        }
    }
}

// For Getting index from Hash Function and passing value to Sorted_Insert Function
int HashMapTable::Insert_Division_Method(HashTableEntry *H[], int k) {
    int index = HashFun(k);
    Sorted_Insert(&H[index], k);
}


// Inserting Values in Hash Table using Multiplication Method
int HashMapTable::Multiplication_Hash(HashTableEntry *H[], int k, int size) {
    int index = HashFun(size*k);
    Sorted_Insert(&H[index], k);
}

// Inserting Values in Hash tabel using Mid Square Hash Method
int HashMapTable::Mid_Square_Hash(HashTableEntry *H[], int k) {
    int index = Middle_Value(k);
    Sorted_Insert(&H[index], k);
}
```

Hashing
Important functions
Implementations
part-b

C 03_Mul_Div&Midsq.cpp > Folding_Hash(HashTableEntry * [], int)

```
89
90  // Calculating Middle digit of the value
91  int HashMapTable::Middle_Value(int k) {
92      int dig = (int)log10(k) + 1;
93
94      k = (int)(k / pow(10, dig / 2)) % 10;
95
96      return k;
97  }
98
99
100 // Inserting Values in Hash tabel using Folding Hash Method
101 int HashMapTable::Folding_Hash(HashTableEntry *H[], int) {
102
103 }
104
105 // For searching specific Value
106 HashTableEntry* HashMapTable::Search(HashTableEntry *p, int k) {
107     while(p) {
108         if(k == p->k) return p;
109
110         p = p->next;
111     }
112     return NULL;
113 }
114
115 // For deleting/removing value
116 void HashMapTable::Delete_Value(int k) {
117     HashTableEntry *p = head, *q = NULL;
118     while(p) {
119         if(k == p->k) {
120             q->next = p->next;
121             delete p;
122         }
123         q = p;
124         p = p->next;
125     }
126 }
```

Hashing
Important functions
Implementations
part-c

```
128
129 // Driver Program
130 int main() {
131
132     int num;
133     int value = 0;
134     int check;
135     cout << "Press respective key for Different Insertion Method\n"
136             << "1. Dividion Mehtod\n2.Multiplication Method\n"
137             << "3.Mid Square Method\n4.Folding Hash\n5.Radix Hash\nEnter you Choice: ";
138             cin >> check;
139
140
141     cout << "Enter number of values you want to enter in Hash table: ";  cin >> num;
142
143     HashTableEntry *table[num];
144     HashTableEntry *temp;
145     for(int i = 0;i < num; i++)
146         table[i] = NULL;
147
148     HashMapTable ob1;
149
150     for(int i = 0; i < num; i++) {
151         fflush(stdin);
152         cout << "Enter Values: ";
153         cin >> value;
154         switch(check) {
155             case 1:
156                 ob1.Insert_Division_Method(table, value);
157                 break;
158             case 2:
159                 ob1.Multiplication_Hash(table, value, num);
160                 break;
161             case 3:
162                 ob1.Mid_Square_Hash(table, value);
163                 break;
164 //          case 4:
165
166         }
167         value = 0;
168     }
169
170
171     cout << "Enter value you want to search: "; cin >> num;
172     temp = ob1.Search(table[ob1.HashFun(num)], num);
173     if(temp == NULL) {
174         cout << "Key is not present in Hash Table\n";
175     }
176     else {
177         cout << "Key is found\n";
178     }
179
180
181     return 0;
182 }
183
```

Hashing
Important functions
Implementations
part-d

C 04_HashtableEntry.cpp > ...

```cpp
#include <iostream>
using namespace std;

const int tablesize = 10;

class HashTableEntry{
    public:
        int key;
        int value;

        HashTableEntry(){
            key=0;
            value=0;
        }
        HashTableEntry(int key, int value){
            this->key = key;
            this->value = value;
        }
};

class HashTable{

    HashTableEntry **table;

    public:
        HashTable(){
            table = new HashTableEntry *[tablesize];
            for(int i=0; i<tablesize; i++){
                table[i] = NULL;
            }
        }

        int hashFunction(int val)
        {
            int a=val;
            int b=0, c=0, d=0, sum;
            b = a% 10;

            c = a % 100;
            c = c/10;

            d = a /100;

            sum = d+b+c;
            return sum;
        }

        void insert(int key, int val){
```

Hashtable Entry Code part-a

C 04_HashtableEntry.cpp > ...

```
48          void insert(int key, int val){
49              int hash = hashFunction(key);
50              table[hash] = new HashTableEntry(key, val);
51              cout << "Element inserted successfully!" << endl;
52          }
53
54          int search(int key){
55              int hash = hashFunction(key);
56              if(table[hash] != NULL){
57                  return table[hash]->value;
58              }
59
60              cout << "No element found at given key!" << endl;
61              return -1;
62
63          }
64
65          void remove(int key){
66              int hash = hashFunction(key);
67              if(table[hash] == NULL){
68                  cout << "No data at key!" << endl;
69                  return;
70              }
71              delete table[hash];
72              cout << "Element deleted successfully!";
73          }
74
75          void display(){
76              cout << "Elements stored in hash table are as follows: ";
77              for(int i=0; i<tablesize; i++){
78                  if(table[i] != NULL){
79                      cout << table[i]->value << " ";
80                  }
81                  else{
82                      cout << "__ ";
83                  }
84              }
85          }
86  };
```

Hashtable Entry
Code part-b

```
88  int main()
89  {
90      HashTable h;
91      int key;
92      int n=5;
93      int arr[5] = {108,151,122,193,15};
94      for(int i=0; i<n; i++){
95          key = h.hashFunction(arr[i]);
96          h.insert(key, arr[i]);
97      }
98
99      h.display();
100     cout << endl;
101
102     cout << "Enter a key to search in hash table: ";
103     cin >> key;
104
105     if(h.search(key) != -1){
106         cout << "Data stored in the key = " << h.search(key) << endl;
107     }
108 }
109
```

Hashtable Entry
Code part-c

Output

```
kashiii@kashiii:~/Downloads/dslab11$ ./04_HashtableEntry.exe
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Elements stored in hash table are as follows: __ __ __ __ 193 122 15 151 __ 108
Enter a key to search in hash table: 0
No element found at given key!
kashiii@kashiii:~/Downloads/dslab11$ ./04_HashtableEntry.exe
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Elements stored in hash table are as follows: __ __ __ __ 193 122 15 151 __ 108
Enter a key to search in hash table: 4
Data stored in the key = 193
kashiii@kashiii:~/Downloads/dslab11$ ./04_HashtableEntry.exe
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Element inserted successfully!
Elements stored in hash table are as follows: __ __ __ __ 193 122 15 151 __ 108
Enter a key to search in hash table: 9
Data stored in the key = 108
```

Thank You

...