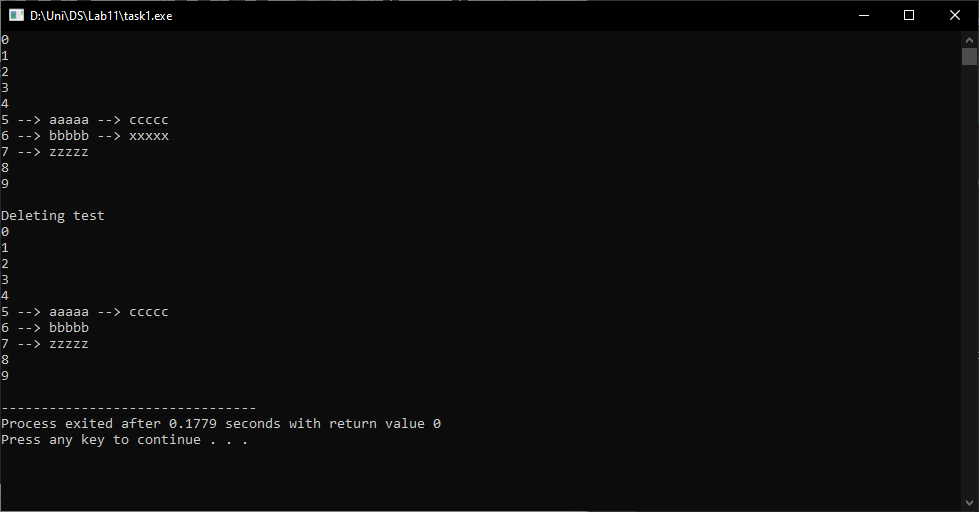
**K226007**

**Syed Yousha**  
**Lab 11 – Hashing**

Task 1:



#include <bits/stdc++.h>

using namespace std;

class Hash

{

int Bucket;

list<string> \*table;

public:

Hash(int B)

{

Bucket = B;

table = new list<string>[Bucket];

}

int hashFunction(char x)

{

return (x % Bucket);

}

void insert(char key, string str)

{

int index = hashFunction(key);

table[index].push\_back(str);

}

void deleteItem(char key, string str)

{

int index = hashFunction(key);

list <string> :: iterator i;

for (i = table[index].begin() ;i != table[index].end(); i++)

{

if (\*i == str)

{

break;

}

}

if (i != table[index].end())

{

table[index].erase(i);

}

}

void display()

{

list<string>::iterator j;

for (int i = 0; i < Bucket; i++)

{

cout << i;

for (j = table[i].begin(); j != table[i].end(); j++)

{

cout<<" --> "<< \*j;

}

cout<<endl;

}

}

};

int main()

{

Hash obj(10);

obj.insert('A', "aaaaa");

obj.insert('B', "bbbbb");

obj.insert('A', "ccccc");

obj.insert('C', "zzzzz");

obj.insert('B', "xxxxx");

obj.display();

cout<<endl<<"Deleting test"<<endl;

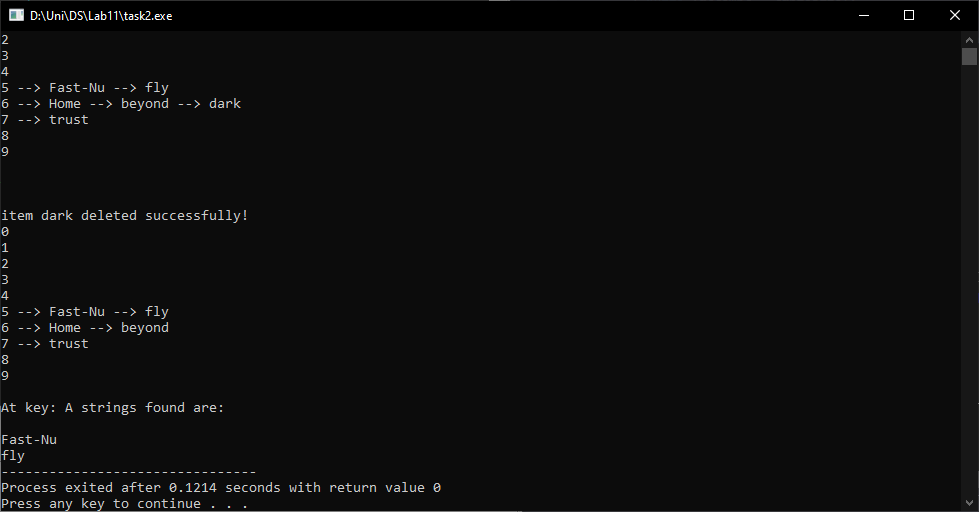
obj.deleteItem('B', "xxxxx");

obj.display();

return 0;

}

Task2:



#include <bits/stdc++.h>

using namespace std;

class Hash

{

int Bucket;

list<string> \*table;

public:

Hash(int B)

{

Bucket = B;

table = new list<string>[Bucket];

}

int hashFunction(char x)

{

return (x % Bucket);

}

void insert(char key, string str)

{

int index = hashFunction(key);

table[index].push\_back(str);

}

void deleteItem(char key, string str)

{

int index = hashFunction(key);

list <string> :: iterator i;

for (i = table[index].begin() ;i != table[index].end(); i++)

{

if (\*i == str)

{

break;

}

}

if (i != table[index].end())

{

table[index].erase(i);

}

cout<<"\nitem "<< str <<" deleted successfully!\n";

}

void word\_search(char key)

{

int index = hashFunction(key);

list<string> :: iterator i;

cout<<"\nAt key: "<<key<<" strings found are: \n";

for(i = table[index].begin(); i != table[index].end(); i++)

{

cout<<endl<<\*i;

}

}

void display()

{

list<string>::iterator j;

for (int i = 0; i < Bucket; i++)

{

cout << i;

for (j = table[i].begin(); j != table[i].end(); j++)

{

cout<<" --> "<< \*j;

}

cout<<endl;

}

}

};

int main()

{

Hash obj(10);

obj.insert('A', "Fast-Nu");

obj.insert('B', "Home");

obj.insert('A', "fly");

obj.insert('C', "trust");

obj.insert('B', "beyond");

obj.insert('B', "dark");

obj.display();

cout<<endl<<endl;

obj.deleteItem('B', "dark");

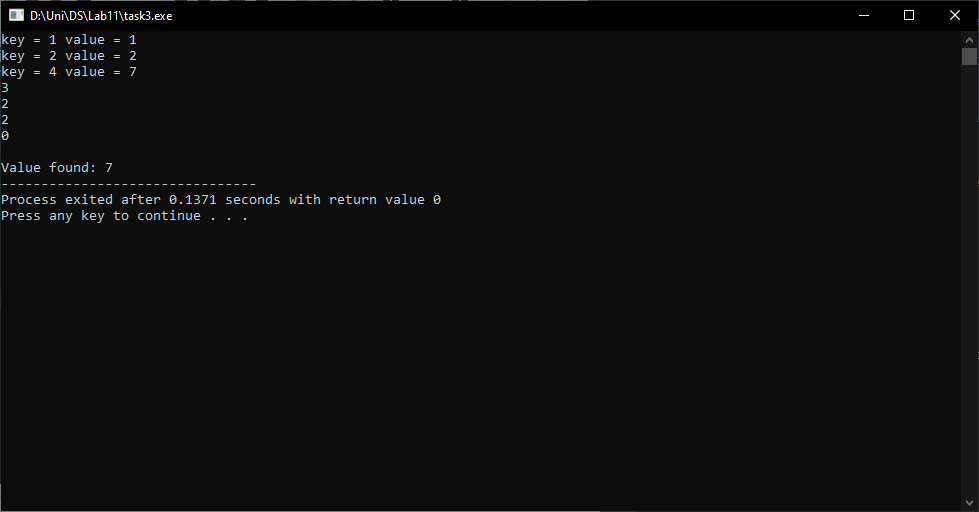
obj.display();

obj.word\_search('A');

return 0;

}

Task 3:



#include <bits/stdc++.h>

using namespace std;

template <typename K, typename V>

class Hash

{

public:

V value;

K key;

Hash(K key, V value)

{

this->key = key;

this->value = value;

}

};

template <typename K, typename V>

class HashMap

{

Hash<K, V>\*\* arr;

int capacity;

int size;

Hash<K, V>\* dummy;

public:

HashMap()

{

capacity = 20;

size = 0;

arr = new Hash<K, V>\*[capacity];

for(int i = 0; i < capacity; i++)

{

arr[i] = NULL;

}

dummy = new Hash<K, V>(-1, -1);

}

int hashCode(K key)

{

return key % capacity;

}

void insertNode(K key, V value)

{

Hash<K, V>\* temp = new Hash<K, V>(key, value);

int hashIndex = hashCode(key);

while(arr[hashIndex] != NULL && arr[hashIndex]->key != key && arr[hashIndex]->key != -1)

{

hashIndex++;

hashIndex %= capacity;

}

if(arr[hashIndex] == NULL || arr[hashIndex]->key == -1)

{

size++;

arr[hashIndex] = temp;

}

}

V deleteNode(int key)

{

int hashIndex = hashCode(key);

while(arr[hashIndex] != NULL)

{

if(arr[hashIndex]->key == key)

{

Hash<K, V>\* temp = arr[hashIndex];

arr[hashIndex] = dummy;

size--;

return temp->value;

}

hashIndex++;

hashIndex %= capacity;

}

return NULL;

}

V search(int key)

{

int hashIndex = hashCode(key);

int counter = 0;

while(arr[hashIndex] != NULL)

{

if(counter++ > capacity)

{

cout<<"Key: "<<key<<" not found!!\n";

return NULL;

}

if(arr[hashIndex]->key == key)

{

cout<<"\nValue found: ";

return arr[hashIndex]->value;

}

hashIndex++;

hashIndex %= capacity;

}

cout<<"Key: "<<key<<" not found!!\n";

return NULL;

}

int sizeofMap()

{

return size;

}

bool isEmpty()

{

return size ==0;

}

void display()

{

for(int i=0; i < capacity; i++)

{

if( arr[i] != NULL && arr[i]->key != -1 )

{

cout<<"key = "<< arr[i]->key << " value = " << arr[i]->value << endl;

}

}

}

};

int main()

{

HashMap<int, int>\* h = new HashMap<int, int>;

h->insertNode(1, 1);

h->insertNode(2, 2);

h->insertNode(2, 3);

h->insertNode(4, 7);

h->display();

cout<< h->sizeofMap() <<endl;

cout<< h->deleteNode(2) <<endl;

cout<< h->sizeofMap() <<endl;

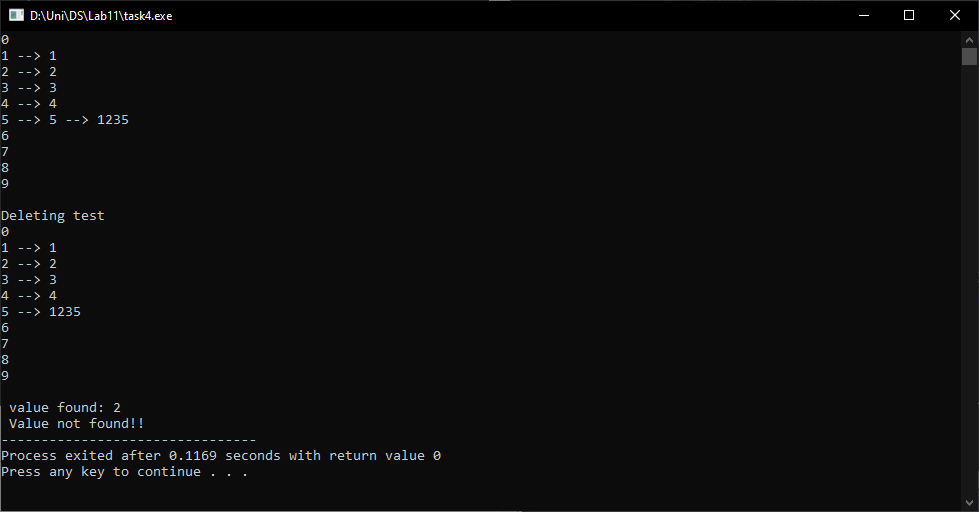
cout<< h->isEmpty() <<endl;

cout<< h->search(4);

return 0;

}

Task 4:



#include <bits/stdc++.h>

using namespace std;

class Hash

{

int Bucket;

list<int> \*table;

public:

Hash(int B)

{

Bucket = B;

table = new list<int>[Bucket];

}

int hashFunction(int x)

{

return (x % Bucket);

}

void insert(int key)

{

int index = hashFunction(key);

table[index].push\_back(key);

}

void deleteItem(int key)

{

int index = hashFunction(key);

list <int> :: 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);

}

}

void search(int key)

{

list<int>::iterator j;

for(int i = 0; i < Bucket; i++)

{

for(j = table[i].begin(); j != table[i].end(); j++)

{

if(\*j == key)

{

cout<<endl<<" value found: "<<\*j;

return;

}

}

}

cout<<"\n Value not found!!";

}

void display()

{

list<int>::iterator j;

for (int i = 0; i < Bucket; i++)

{

cout << i;

for (j = table[i].begin(); j != table[i].end(); j++)

{

cout<<" --> "<< \*j;

}

cout<<endl;

}

}

};

int main()

{

Hash obj(10);

obj.insert(1);

obj.insert(2);

obj.insert(3);

obj.insert(4);

obj.insert(5);

obj.insert(1235);

obj.display();

cout<<endl<<"Deleting test"<<endl;

obj.deleteItem(5);

obj.display();

obj.search(2);

obj.search(7);

return 0;

}

Task 5:

#include <bits/stdc++.h>

#include <iostream>

#include <list>

using namespace std;

class Hash

{

int Bucket;

list<int> \*table;

public:

Hash(int B)

{

Bucket = B;

table = new list<int>[Bucket];

}

int hashFunction(int x)

{

return (x % Bucket);

}

void insert(int key)

{

int index = hashFunction(key);

table[index].push\_back(key);

}

void deleteItem(int key)

{

int index = hashFunction(key);

list <int> :: 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);

}

}

void search(int key)

{

list<int>::iterator j;

for(int i = 0; i < Bucket; i++)

{

for(j = table[i].begin(); j != table[i].end(); j++)

{

if(\*j == key)

{

cout<<endl<<" value found: "<<\*j;

return;

}

}

}

cout<<"\n Value not found!!";

}

void pair\_searching()

{

list<int>::iterator j;

for(int i = 0; i < Bucket - 1; i++ )

{

for(j = table[i].begin(); j != table[i].end(); j++)

{

auto j2 = next(j);

if( (i)+(\*j) == (i+1 ) + (\*j2) )

{

cout<<"\n( "<<i<<", "<<j<<" ) = ( "<<i2<<", "<<j2<<" )";

cout<<"\n Explaination: "<<i<<" + "<<j<<" = "<<i2<<" + "<<j2;

}

}

}

}

void display()

{

cout<<"\n\nSearching pairs: \n";

list<int>::iterator j;

for (int i = 0; i < Bucket; i++)

{

cout << i;

for (j = table[i].begin(); j != table[i].end(); j++)

{

cout<<" --> "<< \*j;

}

cout<<endl;

}

}

};

int main()

{

Hash obj(10);

int arr[] = {1,2,3,124,5,1115};

int size = sizeof(arr)/sizeof(arr[0]);

for(int i =0; i<size; i++)

{

obj.insert(arr[i]);

}

obj.display();

cout<<endl<<"Deleting test"<<endl;

obj.deleteItem(5);

obj.display();

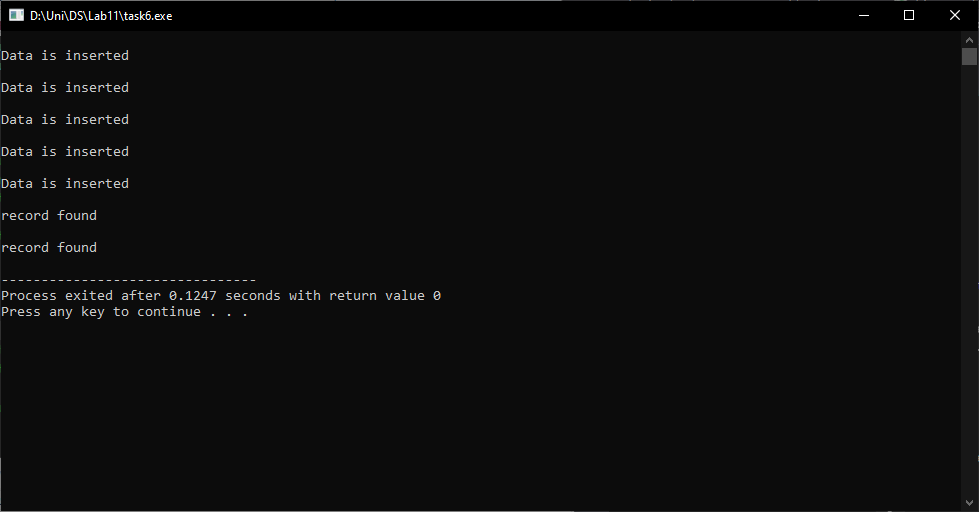
obj.search(2);

obj.search(7);

obj.pair\_searching();

return 0;

}

Task 6:  


#include <iostream>

#include <string>

using namespace std;

class Student

{

public:

int rollNumber;

string name;

Student(int roll, string& n): rollNumber(roll), name(n){}

};

class StudentHashTable

{

static const int size = 15;

Student\* table[size];

public:

StudentHashTable()

{

for (int i=0; i<size; i++)

{

table[i] = NULL;

}

}

int hash(int roll, int attempt)

{

return (roll + attempt \* attempt) % size;

}

void insertRecord(int roll, string name)

{

int atps = 0;

int index;

do

{

index = hash(roll, atps);

if(table[index] == NULL)

{

table[index] = new Student(roll, name);

cout<<"\nData is inserted\n";

return;

}

atps++;

}while(atps < size);

cout<<"\nHashTable is Full!";

}

void search(int roll)

{

int atps = 0;

int index;

do

{

index = hash(roll, atps);

if(table[index]->rollNumber == roll)

{

cout<<"\nrecord found\n";

return;

}

atps++;

}while(atps < size);

cout<<"\nRecord not found!!";

}

};

int main()

{

StudentHashTable obj;

obj.insertRecord(6007, "Yousha");

obj.insertRecord(6111, "Thomas");

obj.insertRecord(6232, "Dazai");

obj.insertRecord(6999, "kiseke");

obj.insertRecord(1111, "Levi");

obj.search(1111);

obj.search(6007);

return 0;

}