**Question#1**

#include <iostream>

#include <cstring>

using namespace std;

class HashSystem {

private:

struct Node {

char\* signature;

Node\* next;

};

Node\*\* table;

int tableSize;

int hashFunction(const char\* key) {

int hash = 0;

for (int i = 0; key[i] != '\0'; i++) {

hash = (hash \* 31 + key[i]) % tableSize;

}

return hash;

}

void insert(const char\* key) {

int index = hashFunction(key);

Node\* newNode = new Node;

newNode->signature = new char[strlen(key) + 1];

strcpy\_s(newNode->signature, strlen(key) + 1, key);

newNode->next = table[index];

table[index] = newNode;

}

bool search(const char\* key) {

int index = hashFunction(key);

Node\* current = table[index];

while (current != nullptr) {

if (strcmp(current->signature, key) == 0) {

return true;

}

current = current->next;

}

return false;

}

public:

HashSystem(int size) {

tableSize = size;

table = new Node \* [tableSize];

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

{

table[i] = nullptr;

}

}

~HashSystem()

{

for (int i = 0; i < tableSize; i++) {

Node\* current = table[i];

while (current != nullptr){

Node\* temp = current;

current = current->next;

delete[] temp->signature;

delete temp;

}

}

delete[] table;

}

void processPacket(const char\* packet){

if (!search(packet)){

insert(packet);

}

}

bool checkThreat(const char\* signature){

return search(signature);

}

void displayTable() {

for (int i = 0; i < tableSize; i++) {

cout << i << ": ";

Node\* current = table[i];

while (current != nullptr){

cout << current->signature << " -> ";

current = current->next;

}

cout << "NULL\n";

}

}

};

int main() {

HashSystem hs(1000);

hs.processPacket("packet1");

hs.processPacket("packet2");

hs.displayTable();

if (hs.checkThreat("packet1")){

cout << "Threat Detected\n";

}

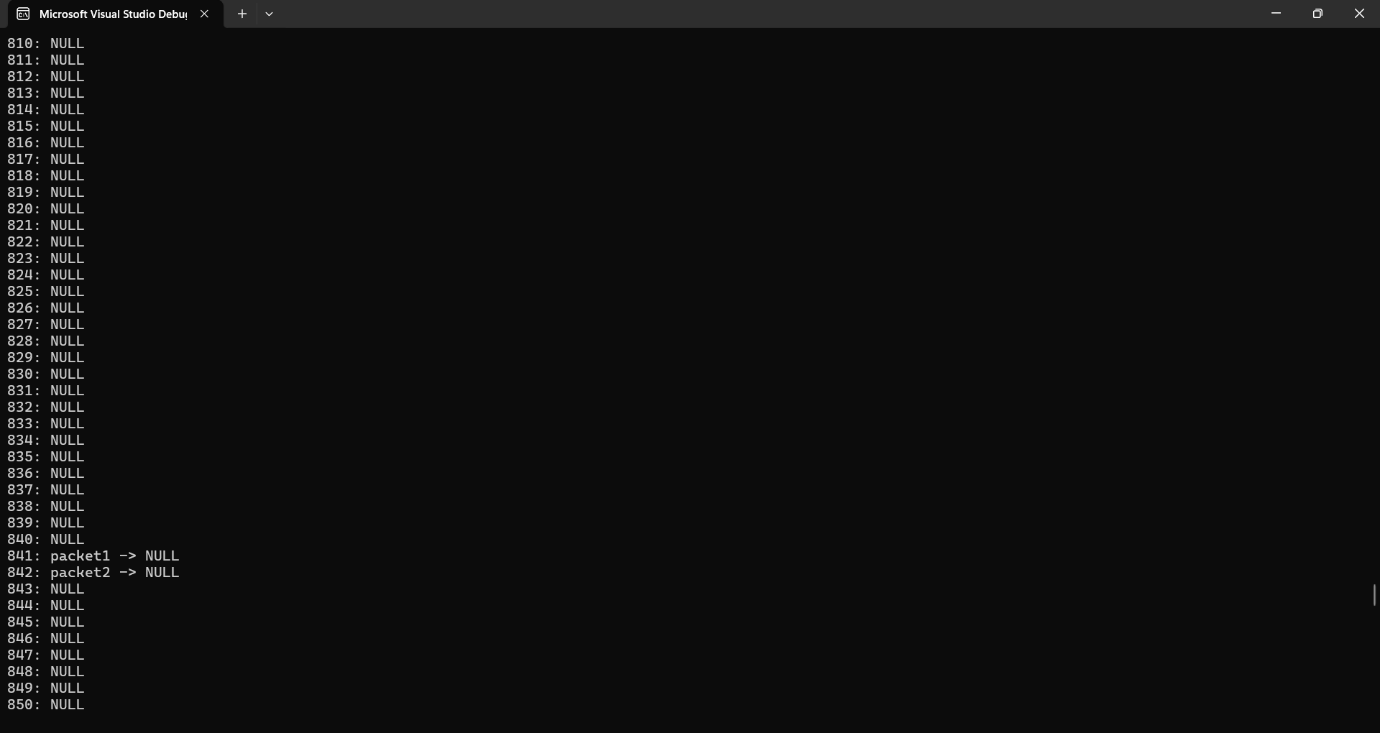
else{

cout << "No Threat\n";

}

return 0;

}

****

**Question#2**

#include <iostream>

using namespace std;

class LibraryHash {

private:

int\* table;

bool\* occupied;

int tableSize;

bool doubleHashing;

int hash1(int key){

return key % tableSize;

}

int hash2(int key){

return 1 + (key % (tableSize - 1));

}

int findSlot(int key){

int index = hash1(key);

int step = doubleHashing ? hash2(key) : 1;

while (occupied[index]){

index = (index + step) % tableSize;

}

return index;

}

public:

LibraryHash(int size, bool useDoubleHashing = false){

tableSize = size;

doubleHashing = useDoubleHashing;

table = new int[tableSize];

occupied = new bool[tableSize];

for (int i = 0; i < tableSize; i++) {

table[i] = -1;

occupied[i] = false;

}

}

~LibraryHash(){

delete[] table;

delete[] occupied;

}

void insert(int key){

int index = findSlot(key);

table[index] = key;

occupied[index] = true;

}

void display(){

for (int i = 0; i < tableSize; i++){

if (occupied[i]){

cout << i << ": " << table[i] << "\n";

}

else{

cout << i << ": EMPTY\n";

}

}

}

};

int main(){

int size = 15;

LibraryHash hashTable1(size, false);

LibraryHash hashTable2(size, true);

int n;

cout << "Enter the number of ISBNs to insert: ";

cin >> n;

cout << "Enter the ISBNs:\n";

for (int i = 0; i < n; i++){

int isbn;

cin >> isbn;

hashTable1.insert(isbn);

hashTable2.insert(isbn);

}

cout << "First strategy (linear probing):\n";

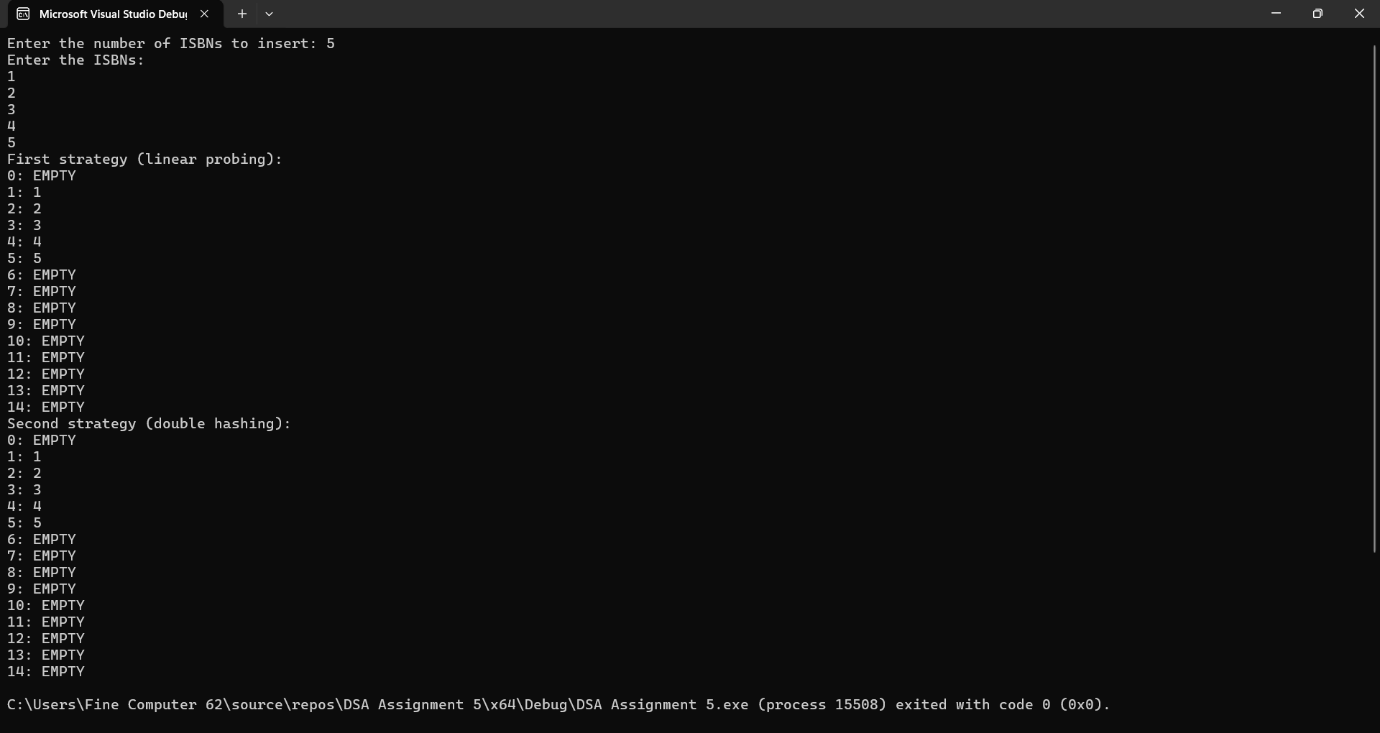
hashTable1.display();

cout << "Second strategy (double hashing):\n";

hashTable2.display();

return 0;

}

****

**Question#3**

#include <iostream>

using namespace std;

class ProductCatalog{

private:

struct Node{

int key;

Node\* next;

};

int\*\* bucketTable;

Node\*\* chainTable;

bool useChaining;

int tableSize;

int bucketCapacity;

int hashFunction(int key){

return key % tableSize;

}

public:

ProductCatalog(int size, bool chaining){

tableSize = size;

useChaining = chaining;

bucketCapacity = 3;

if (useChaining){

chainTable = new Node \* [tableSize];

for (int i = 0; i < tableSize; i++){

chainTable[i] = nullptr;

}

}

else

{

bucketTable = new int\* [tableSize];

for (int i = 0; i < tableSize; i++){

bucketTable[i] = new int[bucketCapacity];

for (int j = 0; j < bucketCapacity; j++)

{

bucketTable[i][j] = -1;

}

}

}

}

~ProductCatalog()

{

if (useChaining)

{

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

{

Node\* current = chainTable[i];

while (current)

{

Node\* temp = current;

current = current->next;

delete temp;

}

}

delete[] chainTable;

}

else

{

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

{

delete[] bucketTable[i];

}

delete[] bucketTable;

}

}

void insert(int key)

{

int index = hashFunction(key);

if (useChaining)

{

Node\* newNode = new Node;

newNode->key = key;

newNode->next = chainTable[index];

chainTable[index] = newNode;

}

else

{

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

{

if (bucketTable[index][i] == -1)

{

bucketTable[index][i] = key;

return;

}

}

}

}

void display()

{

if (useChaining)

{

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

{

cout << i << ": ";

Node\* current = chainTable[i];

while (current)

{

cout << current->key << " -> ";

current = current->next;

}

cout << "NULL\n";

}

}

else

{

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

{

cout << i << ": ";

for (int j = 0; j < bucketCapacity; j++)

{

if (bucketTable[i][j] != -1)

{

cout << bucketTable[i][j] << " ";

}

}

cout << "\n";

}

}

}

};

int main()

{

int skus[] = { 17, 26, 15, 9, 11, 43, 75, 19, 35, 45, 55, 9, 10, 21, 61, 23 };

int n = sizeof(skus) / sizeof(skus[0]);

ProductCatalog catalog1(15, true);

ProductCatalog catalog2(15, false);

cout << "Using Chaining:\n";

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

{

catalog1.insert(skus[i]);

}

catalog1.display();

cout << "\nUsing Bucketing:\n";

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

{

catalog2.insert(skus[i]);

}

catalog2.display();

return 0;

}

