```cpp
#include <iostream>
#include <string>
#include <cstring>
#include <stdlib.h> //srand, rand
#include <time.h>//clock_t, clock, CLOCKS_PER_SEC
using namespace std;

// implementing hash tables as an array of linked lists
// and using it to check whether two sequences are permutations of each other

class Node{

    private:
        int data;
        Node* nextNodePtr;

    public:
        Node(){}

        void setData(int d){
            data = d;
        }

        int getData(){
            return data;
        }

        void setNextNodePtr(Node* nodePtr){
                nextNodePtr = nodePtr;
        }

        Node* getNextNodePtr(){
            return nextNodePtr;
        }

};

class List{

    private:
        Node *headPtr;

    public:
        List(){
            headPtr = new Node();
            headPtr->setNextNodePtr(0);
        }

        Node* getHeadPtr(){
            return headPtr;
        }

        bool isEmpty(){

            if (headPtr->getNextNodePtr() == 0)
                return true;

            return false;
        }


        void insert(int data){

            Node* currentNodePtr = headPtr->getNextNodePtr();
            Node* prevNodePtr = headPtr;

            while (currentNodePtr != 0){
                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr->getNextNodePtr();
            }

            Node* newNodePtr = new Node();
            newNodePtr->setData(data);
            newNodePtr->setNextNodePtr(0);
            prevNodePtr->setNextNodePtr(newNodePtr);

        }

        void insertAtIndex(int insertIndex, int data){

            Node* currentNodePtr = headPtr->getNextNodePtr();
            Node* prevNodePtr = headPtr;

            int index = 0;

            while (currentNodePtr != 0){

                if (index == insertIndex)
                    break;

                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr->getNextNodePtr();
                index++;
            }

            Node* newNodePtr = new Node();
            newNodePtr->setData(data);
            newNodePtr->setNextNodePtr(currentNodePtr);
            prevNodePtr->setNextNodePtr(newNodePtr);

        }


        int read(int readIndex){

            Node* currentNodePtr = headPtr->getNextNodePtr();
            Node* prevNodePtr = headPtr;
            int index = 0;

            while (currentNodePtr != 0){

                if (index == readIndex)
                    return currentNodePtr->getData();

                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr->getNextNodePtr();

                index++;

            }

            return -1; // an invalid value indicating
                        // index is out of range

        }


        bool deleteElement(int deleteData){

            Node* currentNodePtr = headPtr->getNextNodePtr();
            Node* prevNodePtr = headPtr;
            Node* nextNodePtr = headPtr;

            while (currentNodePtr != 0){

                if (currentNodePtr->getData() == deleteData){
                    nextNodePtr = currentNodePtr->getNextNodePtr();
                    prevNodePtr->setNextNodePtr(nextNodePtr);
                    return true;
                }

                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr->getNextNodePtr();

            }

            return false;

        }

        int countList(){

            Node* currentNodePtr = headPtr->getNextNodePtr();
            int numElements = 0;

            while (currentNodePtr != 0){

                numElements++;
                currentNodePtr = currentNodePtr->getNextNodePtr();

            }

            return numElements;
        }


        void IterativePrint(){

            Node* currentNodePtr = headPtr->getNextNodePtr();

            while (currentNodePtr != 0){
                cout << currentNodePtr->getData() << " ";
                currentNodePtr = currentNodePtr->getNextNodePtr();
            }

            cout << endl;

        }


        bool containsElement(int searchData){

            Node* currentNodePtr = headPtr->getNextNodePtr();

            while (currentNodePtr != 0){

                if (currentNodePtr->getData() == searchData)
                    return true;

                currentNodePtr = currentNodePtr->getNextNodePtr();
            }
```

```

                return false;

            }


    };


    class Hashtable{

        private:
            List* listArray;
            int tableSize;

        public:
            Hashtable(int size){
                tableSize = size;
                listArray = new List[size];
            }

            int getTableSize(){
                return tableSize;
            }

            void insert(int data){

                int hashIndex = data % tableSize;
                listArray[hashIndex].insert(data);

            }

            void deleteElement(int data){

                int hashIndex = data % tableSize;
                listArray[hashIndex].deleteElement(data);

            }

            bool hasElement(int data){

                int hashIndex = data % tableSize;
                return listArray[hashIndex].containsElement(data);

            }

            void printHashTable(){

                for (int hashIndex = 0; hashIndex < tableSize; hashIndex++){
                    cout << "Hash Index: " << hashIndex << " : " ;
                    listArray[hashIndex].IterativePrint();
                }

            }


            bool isEmpty(){

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

                    if (!listArray[hashIndex].isEmpty())
                        return false;
                }

```

```cpp
                return true;

            }

    };

    int main(){

        string integerSequence;
        cout << "Enter the integer sequence: ";
        getline(cin, integerSequence);

        string testSequence;
        cout << "Enter the test sequence for permutation: ";
        getline(cin, testSequence);

        int hashTableSize;
        cout << "Enter the size of the hash table: ";
        cin >> hashTableSize;
        Hashtable hashTable(hashTableSize);

        char* integerArray = new char[integerSequence.length()+1];
        strcpy(integerArray, integerSequence.c_str());

        char* cptr = strtok(integerArray, ", ");

        while (cptr != 0){

            string token(cptr);
            int value = stoi(token);

            hashTable.insert(value);

            cptr = strtok(NULL, ", ");

        }

        cout << endl;

        hashTable.printHashTable();



        char* testArray = new char[testSequence.length()+1];
        strcpy(testArray, testSequence.c_str());

        char* tptr = strtok(testArray, ", ");

        while (tptr != 0){

            string token(tptr);
            int testValue = stoi(token);

            if (hashTable.hasElement(testValue))
                hashTable.deleteElement(testValue);
            else{
                cout << testSequence << " is not a permuted sequence of " << integerSequence
                 << endl;
                return 0;
            }

            tptr = strtok(NULL, ", ");

        }

        if (hashTable.isEmpty())
            cout << testSequence << " is a permuted sequence of " << integerSequence << endl;
        else
            cout << testSequence << " is not a permuted sequence of " << integerSequence <<
            endl;

    return 0;
    }
```