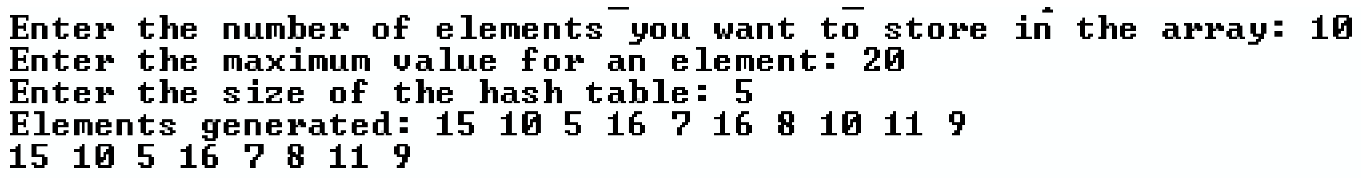
```
#include <iostream>
#include <stdlib.h> //srand, rand
#include <time.h>//clock_t, clock, CLOCKS_PER_SEC
using namespace std;

// implementing hash table as an array of linked lists
// and using it to print the unique elements of an array

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;

```

```cpp
            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;
```

```cpp

        }


};


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;
            while (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();
            }

        }

};

int main(){

    int numElements;
    cout << "Enter the number of elements you want to store in the array: ";
    cin >> numElements;

    int maxValue;
    cout << "Enter the maximum value for an element: ";
    cin >> maxValue;
```

```cpp

        int hashTableSize;
        cout << "Enter the size of the hash table: ";
        cin >> hashTableSize;


        srand(time(NULL));

        int array[numElements];
        cout << "Elements generated: ";
        for (int index = 0; index < numElements; index++){
            array[index] = rand() % maxValue;
            cout << array[index] << " ";
        }

        cout << endl;

        Hashtable hashTable(hashTableSize);

        for (int index = 0; index < numElements; index++){

            if (!hashTable.hasElement(array[index])){
                cout << array[index] << " ";
                hashTable.insert(array[index]);
            }

        }

        cout << endl;

    return 0;
    }
```



```
Enter the number of elements you want to store in the array: 10
Enter the maximum value for an element: 20
Enter the size of the hash table: 5
Elements generated: 15 10 5 16 7 16 8 10 11 9
15 10 5 16 7 8 11 9
```