

```

1  #include <iostream>
2  #include <stdlib.h> //srand, rand
3  #include <time.h> //clock_t, clock, CLOCKS_PER_SEC
4  using namespace std;
5
6  // implementing hash table as an array of linked lists
7  // and using it to print the unique elements of an array
8
9  class Node{
10
11     private:
12         int data;
13         Node* nextNodePtr;
14
15     public:
16         Node() {}
17
18         void setData(int d){
19             data = d;
20         }
21
22         int getData(){
23             return data;
24         }
25
26         void setNextNodePtr(Node* nodePtr){
27             nextNodePtr = nodePtr;
28         }
29
30         Node* getNextNodePtr(){
31             return nextNodePtr;
32         }
33
34 };
35
36 class List{
37
38     private:
39         Node *headPtr;
40
41     public:
42         List(){
43             headPtr = new Node();
44             headPtr->setNextNodePtr(0);
45         }
46
47         Node* getHeadPtr(){
48             return headPtr;
49         }
50
51         bool isEmpty(){
52
53             if (headPtr->getNextNodePtr() == 0)
54                 return true;
55
56             return false;
57         }
58
59         void insert(int data){
60
61             Node* currentNodePtr = headPtr->getNextNodePtr();
62             Node* prevNodePtr = headPtr;
63
64

```

```

65     while (currentNodePtr != 0){
66         prevNodePtr = currentNodePtr;
67         currentNodePtr = currentNodePtr->getNextNodePtr();
68     }
69
70     Node* newNodePtr = new Node();
71     newNodePtr->setData(data);
72     newNodePtr->setNextNodePtr(0);
73     prevNodePtr->setNextNodePtr(newNodePtr);
74
75 }
76
77 void insertAtIndex(int insertIndex, int data){
78
79     Node* currentNodePtr = headPtr->getNextNodePtr();
80     Node* prevNodePtr = headPtr;
81
82     int index = 0;
83
84     while (currentNodePtr != 0){
85
86         if (index == insertIndex)
87             break;
88
89         prevNodePtr = currentNodePtr;
90         currentNodePtr = currentNodePtr->getNextNodePtr();
91         index++;
92     }
93
94     Node* newNodePtr = new Node();
95     newNodePtr->setData(data);
96     newNodePtr->setNextNodePtr(currentNodePtr);
97     prevNodePtr->setNextNodePtr(newNodePtr);
98
99 }
100
101
102 int read(int readIndex){
103
104     Node* currentNodePtr = headPtr->getNextNodePtr();
105     Node* prevNodePtr = headPtr;
106     int index = 0;
107
108     while (currentNodePtr != 0){
109
110         if (index == readIndex)
111             return currentNodePtr->getData();
112
113         prevNodePtr = currentNodePtr;
114         currentNodePtr = currentNodePtr->getNextNodePtr();
115
116         index++;
117
118     }
119
120     return -1; // an invalid value indicating
121               // index is out of range
122
123 }
124
125
126
127 bool deleteElement(int deleteData){
128

```

```

129 Node* currentNodePtr = headPtr->getNextNodePtr();
130 Node* prevNodePtr = headPtr;
131 Node* nextNodePtr = headPtr;
132
133 while (currentNodePtr != 0){
134
135     if (currentNodePtr->getData() == deleteData){
136         nextNodePtr = currentNodePtr->getNextNodePtr();
137         prevNodePtr->setNextNodePtr(nextNodePtr);
138         return true;
139     }
140
141     prevNodePtr = currentNodePtr;
142     currentNodePtr = currentNodePtr->getNextNodePtr();
143
144 }
145
146 return false;
147
148 }
149
150 int countList(){
151
152     Node* currentNodePtr = headPtr->getNextNodePtr();
153     int numElements = 0;
154
155     while (currentNodePtr != 0){
156
157         numElements++;
158         currentNodePtr = currentNodePtr->getNextNodePtr();
159
160     }
161
162     return numElements;
163 }
164
165
166 void IterativePrint(){
167
168     Node* currentNodePtr = headPtr->getNextNodePtr();
169
170     while (currentNodePtr != 0){
171         cout << currentNodePtr->getData() << " ";
172         currentNodePtr = currentNodePtr->getNextNodePtr();
173     }
174
175     cout << endl;
176
177 }
178
179
180 bool containsElement(int searchData){
181
182     Node* currentNodePtr = headPtr->getNextNodePtr();
183
184     while (currentNodePtr != 0){
185
186         if (currentNodePtr->getData() == searchData)
187             return true;
188
189         currentNodePtr = currentNodePtr->getNextNodePtr();
190     }
191
192     return false;

```

```

193
194     }
195
196
197 };
198
199
200 class Hashtable{
201
202     private:
203         List* listArray;
204         int tableSize;
205
206     public:
207         Hashtable(int size){
208             tableSize = size;
209             listArray = new List[size];
210         }
211
212         int getTableSize(){
213             return tableSize;
214         }
215
216         void insert(int data){
217
218             int hashIndex = data % tableSize;
219             listArray[hashIndex].insert(data);
220
221         }
222
223         void deleteElement(int data){
224
225             int hashIndex = data % tableSize;
226             while (listArray[hashIndex].deleteElement(data));
227
228         }
229
230         bool hasElement(int data){
231
232             int hashIndex = data % tableSize;
233             return listArray[hashIndex].containsElement(data);
234
235         }
236
237         void printHashTable(){
238
239             for (int hashIndex = 0; hashIndex < tableSize; hashIndex++){
240                 cout << "Hash Index: " << hashIndex << " : " ;
241                 listArray[hashIndex].IterativePrint();
242             }
243
244         }
245
246 };
247
248 int main(){
249
250     int numElements;
251     cout << "Enter the number of elements you want to store in the array: ";
252     cin >> numElements;
253
254     int maxValue;
255     cout << "Enter the maximum value for an element: ";
256     cin >> maxValue;

```

```

257
258     int hashCodeSize;
259     cout << "Enter the size of the hash table: ";
260     cin >> hashCodeSize;
261
262
263     srand(time(NULL));
264
265     int array[numElements];
266     cout << "Elements generated: ";
267     for (int index = 0; index < numElements; index++){
268         array[index] = rand() % maxValue;
269         cout << array[index] << " ";
270     }
271
272     cout << endl;
273
274     Hashtable hashCode(hashCodeSize);
275
276     for (int index = 0; index < numElements; index++){
277
278         if (!hashCode.hasElement(array[index])){
279             cout << array[index] << " ";
280             hashCode.insert(array[index]);
281         }
282
283     }
284
285     cout << endl;
286
287     return 0;
288 }

```

```

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

```