

```

1  struct Node
2  {
3      int data;
4      Node *pNext;
5  };
6
7  Node *initNode(int value)
8  {
9      Node *p = new Node;
10
11     p->data = value;
12     p->pNext = NULL;
13
14     return p;
15 };
16
17 struct Hashtable
18 {
19     struct Bucket
20     {
21         Node *pHead;
22         Node *pTail;
23     };
24
25     Bucket *buckets;
26     int size;
27 };
28
29 void initHashtable(Hashtable &h, int size)
30 {
31     h.size = size;
32     h.buckets = new Hashtable::Bucket[size];
33     for (int i = 0; i < size; i++)
34     {
35         h.buckets[i].pHead = NULL;
36         h.buckets[i].pTail = NULL;
37     }
38 }
39
40 int hashFunc(int value, int size)
41 {
42     return value % size;
43 }
44
45 int hashFunc(Node *p, int size)
46 {
47     return p->data % size;
48 }
49
50 void add(Hashtable &h, Node *p)
51 {
52     int viTri = hashFunc(p, h.size);
53
54     if (h.buckets[viTri].pHead == NULL)
55     {
56         h.buckets[viTri].pHead = p;
57         h.buckets[viTri].pTail = p;
58     }
59     else
60     {
61         h.buckets[viTri].pTail->pNext = p;
62         h.buckets[viTri].pTail = p;
63     }
64 }
65

```

```

65
66 void add(Hashtable &h, int value)
67 {
68     int viTri = hashFunc(value, h.size);
69
70     Node *p = initNode(value);
71
72     if (h.buckets[viTri].pHead == NULL)
73     {
74         h.buckets[viTri].pHead = p;
75         h.buckets[viTri].pTail = p;
76     }
77     else
78     {
79         h.buckets[viTri].pTail->pNext = p;
80         h.buckets[viTri].pTail = p;
81     }
82 }
83
84 void deleteHashtable(Hashtable &h)
85 {
86     for (int i = 0; i < h.size; i++)
87     {
88         Node *p = h.buckets[i].pHead;
89         while (p != NULL)
90         {
91             Node *temp = p;
92             p = p->pNext;
93             delete temp;
94         }
95     }
96     delete[] h.buckets;
97 }
98
99 void initDataHTAuto(Hashtable &h)
100 {
101     int n = rand() % 51 + 45;
102
103     for (int i = 0; i < n; i++)
104     {
105         int value = rand() % (988 - 856 + 1) + 856;
106         add(h, value);
107     }
108 }
109
110 void initDataHTArr(Hashtable &h, int a[], int n)
111 {
112     for (int i = 0; i < n; i++)
113     {
114         add(h, a[i]);
115     }
116 }
117
118 void printHashtable(Hashtable h)
119 {
120     cout << "Size of Hashtable: " << h.size << endl;
121
122     for (int i = 0; i < h.size; i++)
123     {
124         cout << "Bucket[" << i << "]: ";
125
126         for (Node *p = h.buckets[i].pHead; p != NULL; p = p->pNext)
127         {
128             cout << p->data << "\t";
129         }
130         cout << endl;
131     }
132 }
133

```

```

132 }
133
134 void deleteValue(Hashtable &h, int value)
135 {
136     int viTri = hashFunc(value, h.size);
137
138     Node *p = h.buckets[viTri].pHead;
139     Node *prev = NULL;
140
141     while (p != NULL && p->data != value)
142     {
143         prev = p;
144         p = p->pNext;
145     }
146
147     if (p == NULL)
148     {
149         cout << "Khong tim thay gia tri " << value << " trong bang băm." << endl;
150         return;
151     }
152
153     if (prev == NULL)
154     {
155         h.buckets[viTri].pHead = p->pNext;
156     }
157     else
158     {
159         prev->pNext = p->pNext;
160     }
161
162     delete p;
163     cout << "Da xoa gia tri " << value << " trong bang băm." << endl;
164 }
165
166 bool findValue(Hashtable h, int value)
167 {
168     int viTri = hashFunc(value, h.size);
169
170     for (Node *p = h.buckets[viTri].pHead; p != NULL; p = p->pNext)
171     {
172         if (p->data == value)
173         {
174             return true;
175         }
176     }
177     return false;
178 }
179
180 bool isEmpty(Hashtable h)
181 {
182     for (int i = 0; i < h.size; i++)
183     {
184         if (h.buckets[i].pHead != NULL)
185         {
186             return false;
187         }
188     }
189     return true;
190 }

```