```cpp
#include <iostream>
using namespace std;

// implementing the dynamic List ADT using Linked List

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();
```

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

```
129
130                 if (index == modifyIndex){
131                     currentNodePtr->setData(data);
132                     return;
133                 }
134
135                 prevNodePtr = currentNodePtr;
136                 currentNodePtr = currentNodePtr->getNextNodePtr();
137
138                 index++;
139             }
140
141
142         }
143
144
145         void deleteElement(int deleteIndex){
146
147             Node* currentNodePtr = headPtr->getNextNodePtr();
148             Node* prevNodePtr = headPtr;
149             Node* nextNodePtr = headPtr;
150             int index = 0;
151
152             while (currentNodePtr != 0){
153
154                 if (index == deleteIndex){
155                     nextNodePtr = currentNodePtr->getNextNodePtr();
156                     break;
157                 }
158
159                 prevNodePtr = currentNodePtr;
160                 currentNodePtr = currentNodePtr->getNextNodePtr();
161
162                 index++;
163             }
164
165             prevNodePtr->setNextNodePtr(nextNodePtr);
166
167         }
168
169
170         void IterativePrint(){
171
172             Node* currentNodePtr = headPtr->getNextNodePtr();
173
174             while (currentNodePtr != 0){
175                 cout << currentNodePtr->getData() << " ";
176                 currentNodePtr = currentNodePtr->getNextNodePtr();
177             }
178
179             cout << endl;
180
181         }
182
183
184 };
185
186 int main(){
187
188     int listSize;
189
190     cout << "Enter the number of elements you want to insert: ";
191     cin >> listSize;
192
```

```
193       List integerList; // Create an empty list
194
195       for (int i = 0; i < listSize; i++){
196
197           int value;
198           cout << "Enter element # " << i << " : ";
199           cin >> value;
200
201           integerList.insertAtIndex(i, value);
202       }
203
204       cout << "Contents of the List: ";
205       integerList.IterativePrint();
206
207       // to read an element at a particular index (before delete)
208
209       int readIndex;
210       cout << "Enter an index to read (before delete): ";
211       cin >> readIndex;
212       cout << "Value at " << readIndex << " is: " << integerList.read(readIndex) << endl;
213
214       // to delete an element at a particular index
215
216       int deleteIndex;
217       cout << "Enter an index to delete: ";
218       cin >> deleteIndex;
219       integerList.deleteElement(deleteIndex);
220
221       cout << "Contents of the List: ";
222       integerList.IterativePrint();
223
224       // to read an element at a particular index (after delete)
225
226       cout << "Enter an index to read (after delete): ";
227       cin >> readIndex;
228       cout << "Value at " << readIndex << " is: " << integerList.read(readIndex) << endl;
229
230
231
232       // to insert an element at a particular index
233       int insertIndex, insertValue;
234       cout << "Enter an index to insert: ";
235       cin >> insertIndex;
236       cout << "Enter a value to insert: ";
237       cin >> insertValue;
238       integerList.insertAtIndex(insertIndex, insertValue);
239
240       cout << "Contents of the List: ";
241       integerList.IterativePrint();
242
243       // to read an element at a particular index (after insert)
244
245       cout << "Enter an index to read (after insert): ";
246       cin >> readIndex;
247       cout << "Value at " << readIndex << " is: " << integerList.read(readIndex) << endl;
248
249
250       // to insert at the end of the list
251       cout << "Enter the element you want to insert at the end of the list: ";
252       cin >> insertValue;
253       integerList.insert(insertValue);
254
255       cout << "Contents of the List: ";
256       integerList.IterativePrint();
```

```
257
258   return 0;
259   }
```

```
Enter the number of elements you want to insert: 5
Enter element # 0 : 12
Enter element # 1 : 45
Enter element # 2 : 78
Enter element # 3 : 96
Enter element # 4 : 22
Contents of the List: 12 45 78 96 22
Enter an index to read (before delete): 2
Value at 2 is: 78
Enter an index to delete: 2
Contents of the List: 12 45 96 22
Enter an index to read (after delete): 2
Value at 2 is: 96
Enter an index to insert: 2
Enter a value to insert: 77
Contents of the List: 12 45 77 96 22
Enter an index to read (after insert): 1
Value at 1 is: 45
Enter the element you want to insert at the end of the list: 55
Contents of the List: 12 45 77 96 22 55
```