```cpp
#ifdef _WIN32
   #include <tchar.h>
   #include <conio.h>
#elif (defined __linux__) || (defined _AIX)
   typedef char _TCHAR;
   #define _tmain main
#endif

#include <stdio.h>
#include <iostream>
using namespace std;

#include "LinkedList2.h"
//------------------------------------------------------------------------
TListNode::TListNode()
{
   Data = 0;
   nextPtr = NULL; // dolezite !!!!!!!!!!!!!!!!!!!!
}
//------------------------------------------------------------------------
TListNode::~TListNode()
{
}
//------------------------------------------------------------------------
TList::TList()
{
   firstPtr = lastPtr = NULL; // dolezite !!!!!!!!!!!!!!
}
//------------------------------------------------------------------------
TList::~TList()
{
   TListNode *currentPtr = firstPtr, *tempPtr;

   if (!isEmpty()) {
      cout << "\nDeallocating the nodes ...\n";

      while (currentPtr != NULL) {
         tempPtr = currentPtr;
         cout << tempPtr->Data << endl;
         currentPtr = currentPtr->nextPtr;
         delete tempPtr;
      }
      cout << "All the nodes are deallocated.\n";
   }
}
//------------------------------------------------------------------------
void TList::insertAtFront(const int &data)
{
   TListNode *newPtr = new TListNode();

   newPtr->Data = data;
   if (isEmpty())
      firstPtr = lastPtr = newPtr;
```

```
54     else {
55        newPtr->nextPtr = firstPtr;
56        firstPtr = newPtr;
57     }
58  }
59  //--------------------------------------------------------------------------
60  void TList::insertBeforeElement(const int &data)
61  {
62     TListNode *newPtr = new TListNode();
63
64     newPtr->Data = data;
65     if (isEmpty())
66        firstPtr = lastPtr = newPtr;
67     else {
68        TListNode *currentPtr = firstPtr, *tempPtr = firstPtr;
69
70        if (data < currentPtr->Data) {
71           newPtr->nextPtr = firstPtr;
72           firstPtr = newPtr;
73        }
74        else {
75           while ((currentPtr != NULL) && (data >= currentPtr->Data)) {
76              tempPtr = currentPtr;
77              currentPtr = currentPtr->nextPtr;
78           }
79
80           newPtr->nextPtr = currentPtr;
81           tempPtr->nextPtr = newPtr;
82           if (currentPtr == NULL)
83              lastPtr = newPtr;
84        }
85     }
86  }
87  //--------------------------------------------------------------------------
88  void TList::insertAtBack(const int &data)
89  {
90     TListNode *newPtr = new TListNode();
91
92     newPtr->Data = data;
93     if (isEmpty())
94        firstPtr = lastPtr = newPtr;
95     else {
96        lastPtr->nextPtr = newPtr;
97        lastPtr = newPtr;
98     }
99  }
100 //--------------------------------------------------------------------------
101 int TList::removeFromFront(int &data)
102 {
103    if (isEmpty())
104       return 0;
105    else {
106       TListNode *tempPtr = firstPtr;
```

```
107
108         if (firstPtr == lastPtr)
109            firstPtr = lastPtr = NULL;
110         else
111            firstPtr = firstPtr->nextPtr;
112
113         data = tempPtr->Data;
114         delete tempPtr;
115         return 1;
116      }
117   }
118   //--------------------------------------------------------------------------
119   int TList::removeFromBack(int &data)
120   {
121      if (isEmpty())
122         return 0;
123      else {
124         TListNode *tempPtr = lastPtr;
125
126         if (firstPtr == lastPtr)
127            firstPtr = lastPtr = NULL;
128         else {
129            TListNode *currentPtr = firstPtr;
130
131            while (currentPtr->nextPtr != lastPtr)
132               currentPtr = currentPtr->nextPtr;
133
134            lastPtr = currentPtr;
135            currentPtr->nextPtr = NULL;
136         }
137         data = tempPtr->Data;
138         delete tempPtr;
139         return 1;
140      }
141   }
142   //--------------------------------------------------------------------------
143   void TList::sortElements()
144   {
145      TListNode *firstTempPtr, *secondTempPtr, *thirdTempPtr, *secondPtr;
146      bool flaw;
147
148      if (firstPtr != NULL)
149         do {
150            flaw = false;
151
152            secondPtr = firstPtr->nextPtr;
153            if (secondPtr != NULL && firstPtr->Data > secondPtr->Data) {
154               firstPtr->nextPtr = secondPtr->nextPtr;
155               secondPtr->nextPtr = firstPtr;
156               firstPtr = secondPtr;
157               flaw = true;
158            }
159
```

```
160          firstTempPtr = firstPtr;
161          secondTempPtr = firstTempPtr->nextPtr;
162          thirdTempPtr = secondTempPtr->nextPtr;
163          while (thirdTempPtr != NULL) {
164             if (secondTempPtr->Data > thirdTempPtr->Data) {
165                firstTempPtr->nextPtr = thirdTempPtr;
166                secondTempPtr->nextPtr = thirdTempPtr->nextPtr;
167                thirdTempPtr->nextPtr = secondTempPtr;
168                flaw = true;
169             }
170             firstTempPtr = firstTempPtr->nextPtr;
171             secondTempPtr = firstTempPtr->nextPtr;
172             thirdTempPtr = secondTempPtr->nextPtr;
173          }
174          lastPtr = secondTempPtr;
175       } while (flaw);
176 }
177 //--------------------------------------------------------------------------
178 void TList::print() const
179 {
180    TListNode *currentPtr = firstPtr;
181
182    if (isEmpty()) {
183       cout << "The list is empty!" << endl << endl;
184       return;
185    }
186
187    cout << "List = ";
188    while (currentPtr != NULL) {
189       cout << currentPtr->Data << ' ';
190       currentPtr = currentPtr->nextPtr;
191    }
192    cout << endl << endl;
193 }
194 //--------------------------------------------------------------------------
195
```