List Implementation Code Practice

CSE 225 - Data Structures and Algorithms

Md. Mahfuzur Rahman
ECE Department
North South University

1 Unsorted List Implementation

1.1 Summary

```
ItemType
  ItemType.h     Specification for items on the list
  ItemType.cpp     Implementation file for items on the list

Array Based Implementation
  unsorted.h     Specification file for UnsortedType class
  unsorted.cpp     Implementation file for UnsortedType class

Linked-List Based Implementation
  unsorted.h     Specification file for UnsortedType class
  unsorted.cpp     Implementation file for UnsortedType class
```

1.2 ItemType

```

// The following declarations and definitions go into file
// ItemType.h.

#include <fstream>
const int MAX_ITEMS = 5;
enum RelationType   {LESS, GREATER, EQUAL};

class ItemType
{
public:
  ItemType();
  RelationType ComparedTo(ItemType) const;
  void Print(std::ostream&) const;
  void Initialize(int number);
private:
  int value;
};
```

```
// The following definitions go into file ItemType.cpp.
#include <fstream>
#include <iostream>
#include "ItemType.h"

ItemType::ItemType()
{
  value = 0;
}
```

```
10
11 RelationType ItemType::ComparedTo(ItemType otherItem) const
12 {
13   if (value < otherItem.value)
14     return LESS;
15   else if (value > otherItem.value)
16     return GREATER;
17   else return EQUAL;
18 }
19
20 void ItemType::Initialize(int number)
21 {
22   value = number;
23 }
24
25 void ItemType::Print(std::ostream& out) const
26 // pre:  out has been opened.
27 // post: value has been sent to the stream out.
28 {
29   out << value;
30 }
```

1.3 Array Based Implementation

```
1 #include "ItemType.h"
2 // File ItemType.h must be provided by the user of this class.
3 //  ItemType.h must contain the following definitions:
4 //  MAX_ITEMS:     the maximum number of items on the list
5 //  ItemType:      the definition of the objects on the list
6 //  RelationType:  {LESS, GREATER, EQUAL}
7 //  Member function ComparedTo(ItemType item) which returns
8 //       LESS, if self "comes before" item
9 //       GREATER, if self "comes after" item
10 //      EQUAL, if self and item are the same
11
12 class UnsortedType
13 {
14 public:
15   UnsortedType();
16   // Constructor
17
18   void MakeEmpty();
19   // Function: Returns the list to the empty state.
20   // Post:  List is empty.
21
22   bool IsFull() const;
23   // Function:  Determines whether list is full.
24   // Pre:  List has been initialized.
25   // Post: Function value = (list is full)
26
27   int GetLength() const;
28   // Function: Determines the number of elements in list.
29   // Pre:  List has been initialized.
```

```
30   // Post: Function value = number of elements in list
31
32   ItemType GetItem(ItemType, bool&);
33   // Function: Retrieves list element whose key matches item's key (if
34   //           present).
35   // Pre:  List has been initialized.
36   //       Key member of item is initialized.
37   // Post: If there is an element someItem whose key matches
38   //       item's key, then found = true and someItem is returned.
39   //   otherwise found = false and item is returned.
40   //       List is unchanged.
41
42   void PutItem(ItemType item);
43   // Function: Adds item to list.
44   // Pre:  List has been initialized.
45   //       List is not full.
46   //       item is not in list.
47   // Post: item is in list.
48
49   void DeleteItem(ItemType item);
50   // Function: Deletes the element whose key matches item's key.
51   // Pre:  List has been initialized.
52   //       Key member of item is initialized.
53   //       One and only one element in list has a key matching item's key.
54   // Post: No element in list has a key matching item's key.
55
56   void ResetList();
57   // Function: Initializes current position for an iteration through the
       list.
58   // Pre:  List has been initialized.
59   // Post: Current position is prior to list.
60
61   ItemType GetNextItem();
62   // Function: Gets the next element in list.
63   // Pre:  List has been initialized and has not been changed since last
       call.
64   //       Current position is defined.
65   //       Element at current position is not last in list.
66   //
67   // Post: Current position is updated to next position.
68   //       item is a copy of element at current position.
69
70 private:
71   int length;
72   ItemType info[MAX_ITEMS];
73   int currentPos;
74 };
```

```
1  // Implementation file for Unsorted.h
2
3  #include "unsorted.h"
4
5  UnsortedType::UnsortedType()
6  {
7
8
9  }
10 bool UnsortedType::IsFull() const
11 {
12   return
13 }
14
15 int UnsortedType::GetLength() const
16 {
17   return
18 }
19
20 ItemType UnsortedType::GetItem(ItemType item, bool& found)
21 // Pre:  Key member(s) of item is initialized.
22 // Post: If found, item's key matches an element's key in the
23 //       list and a copy of that element has been returned;
24 //       otherwise, item is returned.
25 {
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54 }
```

```
55 void UnsortedType::MakeEmpty()
56 // Post: list is empty.
57 {
58
59
60 }
61 void UnsortedType::PutItem(ItemType item)
62 // Post: item is in the list.
63 {
64
65
66
67
68 }
69 void UnsortedType::DeleteItem(ItemType item)
70 // Pre:  item's key has been initialized.
71 //       An element in the list has a key that matches item's.
72 // Post: No element in the list has a key that matches item's.
73 {
74
75
76
77
78
79
80
81
82
83
84
85
86
87 }
88 void UnsortedType::ResetList()
89 // Post: currentPos has been initialized.
90 {
91
92
93 }
94
95 ItemType UnsortedType::GetNextItem()
96 // Pre:  ResetList was called to initialized iteration.
97 //       No transformer has been executed since last call.
98 //       currentPos is defined.
99 // Post: item is current item.
100 //       Current position has been updated.
101 {
102
103
104
105
106 }
```

1.4 Linked-List Based Implementation

```
1
2  #include "ItemType.h"
3  // File ItemType.h must be provided by the user of this class.
4  //  ItemType.h must contain the following definitions:
5  //  MAX_ITEMS:      the maximum number of items on the list
6  //  ItemType:       the definition of the objects on the list
7  //  RelationType:   {LESS, GREATER, EQUAL}
8  //  Member function ComparedTo(ItemType item) which returns
9  //        LESS, if self "comes before" item
10 //        GREATER, if self "comes after" item
11 //        EQUAL, if self and item are the same
12 struct NodeType;
13
14 class UnsortedType
15 {
16 public:
17   UnsortedType(); // Constructor
18   ~UnsortedType(); // Destructor
19   void MakeEmpty();
20   // Function: Returns the list to the empty state.
21   // Post:  List is empty.
22   bool IsFull() const;
23   // Function:  Determines whether list is full.
24   // Pre:  List has been initialized.
25   // Post: Function value = (list is full)
26
27   int GetLength() const;
28   // Function: Determines the number of elements in list.
29   // Pre:  List has been initialized.
30   // Post: Function value = number of elements in list
31
32   ItemType GetItem(ItemType& item, bool& found);
33   // Function: Retrieves list element whose key matches item's key (if
34   //           present).
35   // Pre:  List has been initialized.
36   //       Key member of item is initialized.
37   // Post: If there is an element someItem whose key matches
38   //       item's key, then found = true and someItem is returned;
39   //   otherwise found = false and item is returned.
40   //       List is unchanged.
41
42   void PutItem(ItemType item);
43   // Function: Adds item to list.
44   // Pre:  List has been initialized.
45   //       List is not full.
46   //       item is not in list.
47   // Post: item is in list.
48
49   void DeleteItem(ItemType item);
50   // Function: Deletes the element whose key matches item's key.
51   // Pre:  List has been initialized.
```

```
52    //       Key member of item is initialized.
53    //       One and only one element in list has a key matching item's key.
54    // Post: No element in list has a key matching item's key.
55
56    void ResetList();
57    // Function: Initializes current position for an iteration through the
        list.
58    // Pre:  List has been initialized.
59    // Post: Current position is prior to list.
60
61    ItemType GetNextItem();
62    // Function: Gets the next element in list.
63    // Pre:  List has been initialized and has not been changed since last
        call.
64    //       Current position is defined.
65    //       Element at current position is not last in list.
66    //
67    // Post: Current position is updated to next position.
68    //       item is a copy of element at current position.
69
70  private:
71    NodeType* listData;
72    int length;
73    NodeType* currentPos;
74  };
```

```
1   // This file contains the linked implementation of class
2   // UnsortedType.
3
4   #include "unsorted.h"
5   struct NodeType
6   {
7       ItemType info;
8       NodeType* next;
9   };
10
11  UnsortedType::UnsortedType()  // Class constructor
12  {
13
14
15
16
17  }
18
19  bool UnsortedType::IsFull() const
20  // Returns true if there is no room for another ItemType
21  //  on the free store; false otherwise.
22  {
23
24
25
26
27
28
```

```
29
30
31
32
33
34
35
36 }
37
38 int UnsortedType::GetLength() const
39 // Post: Number of items in the list is returned.
40 {
41
42
43
44 }
45
46 void UnsortedType::MakeEmpty()
47 // Post: List is empty; all items have been deallocated.
48 {
49
50
51
52
53
54
55
56
57
58
59
60
61 }
62 void UnsortedType::PutItem(ItemType item)
63 // item is in the list; length has been incremented.
64 {
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81 }
82
```

```
83 ItemType UnsortedType::GetItem(ItemType& item, bool& found)
84 // Pre:  Key member(s) of item is initialized.
85 // Post: If found, item's key matches an element's key in the
86 //       list and a copy of that element has been stored in item;
87 //       otherwise, item is unchanged.
88 {
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116 }
117
118  void UnsortedType::DeleteItem(ItemType item)
119 // Pre:  item's key has been initialized.
120 //       An element in the list has a key that matches item's.
121 // Post: No element in the list has a key that matches item's.
122 {
123
124
125
126
127
128
129
130
131
132
133
134
135
136
```

```
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151 }
152
153 void UnsortedType::ResetList()
154 // Post: Current position has been initialized.
155 {
156
157
158 }
159
160 ItemType UnsortedType::GetNextItem()
161 // Post:  A copy of the next item in the list is returned.
162 //        When the end of the list is reached, currentPos
163 //        is reset to begin again.
164 {
165
166
167
168
169
170
171
172
173
174
175 }
176
177 UnsortedType::~UnsortedType()
178 // Post: List is empty; all items have been deallocated.
179 {
180
181
182
183
184
185
186
187
188
189
190 }
```

2 Sorted List Implementation

2.1 Summary

```
ItemType
  ItemType.h     Specification for items on the list
  ItemType.cpp     Implementation file for items on the list

Array Based Implementation
  sorted.h     Specification file for SortedType class
  sorted.cpp     Implementation file for SortedType class

Linked-List Based Implementation
  sortedType.h     Specification file for SortedType class
  sortedType.cpp     Implementation file for SortedType class
```

2.2 ItemType

```

// The following declarations and definitions go into file
// ItemType.h.

#include <fstream>
const int MAX_ITEMS = 5;
enum RelationType  {LESS, GREATER, EQUAL};

class ItemType
{
public:
  ItemType();
  RelationType ComparedTo(ItemType) const;
  void Print(std::ostream&) const;
  void Initialize(int number);
private:
  int value;
};


// The following definitions go into file ItemType.cpp.
#include <fstream>
#include <iostream>
#include "ItemType.h"

ItemType::ItemType()
{
  value = 0;
```

```
}

RelationType ItemType::ComparedTo(ItemType otherItem) const
{
  if (value < otherItem.value)
    return LESS;
  else if (value > otherItem.value)
    return GREATER;
  else return EQUAL;
}

void ItemType::Initialize(int number)
{
  value = number;
}

void ItemType::Print(std::ostream& out) const
// pre:  out has been opened.
// post: value has been sent to the stream out.
{
  out << value;
}
```

2.3 Array Based Implementation

```
#ifndef SORTED
#define SORTED

#include "ItemType.h"
// File ItemType.h must be provided by the user of this class.
//  ItemType.h must contain the following definitions:
//  MAX_ITEMS:     the maximum number of items on the list
//  ItemType:      the definition of the objects on the list
//  RelationType:  {LESS, GREATER, EQUAL}
//  Member function ComparedTo(ItemType item) which returns
//       LESS, if self "comes before" item
//       GREATER, if self "comes after" item
//       EQUAL, if self and item are the same

class SortedType
{
public:
  SortedType();

  void MakeEmtpy();
  // Function: Returns list to the empty state
  // Post:  List is empty.

  bool IsFull() const;
  // Function:  Determines whether list is full.
  // Pre:  List has been initialized.
  // Post: Function value = (list is full)

```

```
29    int GetLength() const;
30    // Function: Determines the number of elements in list.
31    // Pre:  List has been initialized.
32    // Post: Function value = number of elements in list
33
34    ItemType GetItem(ItemType item, bool& found);
35    // Function: Retrieves list element whose key matches item's key (if
36    //           present).
37    // Pre:  List has been initialized.
38    //       Key member of item is initialized.
39    // Post: If there is an element someItem whose key matches
40    //       item's key, then found = true and item is returned;
41    //       someItem; otherwise found = false and item is returned.
42    //       List is unchanged.
43
44    void PutItem(ItemType item);
45    // Function: Adds item to list.
46    // Pre:  List has been initialized.
47    //       List is not full.
48    //       item is not in list.
49    //       List is sorted.
50    // Post: item is in list.
51    //       List is sorted
52
53    void DeleteItem(ItemType item);
54    // Function: Deletes the element whose key matches item's key.
55    // Pre:  List has been initialized.
56    //       Key member of item is initialized.
57    //       One and only one element in list has a key matching item's key.
58    // Post: No element in list has a key matching item's key.
59    //       List is sorted.
60
61    void ResetList();
62    // Function: Initializes current position for an iteration through the
        list.
63    // Pre:  List has been initialized.
64    // Post: Current position is prior to list.
65
66    ItemType GetNextItem();
67    // Function: Gets the next element in list.
68    // Pre:  List has been initialized and has not been changed since last
        call.
69    //       Current position is defined.
70    //       Element at current position is not last in list.
71    //
72    // Post: Current position is updated to next position.
73    //       Returns a copy of element at current position.
74
75    void MakeEmpty();
76    // Function: Make the list empty
77    // Pre:  List has been initialized.
78    // Post: The list is empty
79
80  private:
```

```
81    int length;
82    ItemType info[MAX_ITEMS];
83    int currentPos;
84  };
85  #endif
```

```
1   // Implementation file for sorted.h
2
3   #include "sorted.h"
4   SortedType::SortedType()
5   {
6
7
8   }
9
10  void SortedType::MakeEmpty()
11  {
12
13
14  }
15
16
17  bool SortedType::IsFull() const
18  {
19    return
20  }
21
22  int SortedType::GetLength() const
23  {
24    return
25  }
26
27  ItemType SortedType::GetItem(ItemType item, bool& found)
28  {
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
```

```
49
50
51
52
53
54
55
56 }
57
58 void SortedType::DeleteItem(ItemType item)
59 {
60
61
62
63
64
65
66
67
68
69
70
71
72 }
73
74 void SortedType::PutItem(ItemType item)
75 {
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101 }
102
```

```
103 void SortedType::ResetList()
104 // Post: currentPos has been initialized.
105 {
106
107
108 }
109
110 ItemType SortedType::GetNextItem()
111 // Post: item is current item.
112 //       Current position has been updated.
113 {
114
115
116
117
118 }
```

2.4 Linked-List Based Implementation

```
#include "ItemType.h"
// Header file for Sorted List ADT.
struct NodeType;

class SortedType
{
public:
  SortedType();       // Class constructor
  ~SortedType();      // Class destructor

  bool IsFull() const;
  int  GetLength() const;
  void MakeEmpty();
  ItemType GetItem(ItemType& item, bool& found);
  void PutItem(ItemType item);
  void DeleteItem(ItemType item);
  void ResetList();
  ItemType GetNextItem();

private:
  NodeType* listData;
  int length;
  NodeType* currentPos;
};
```

```
#include "sortedType.h"

struct NodeType
{
  ItemType info;
  NodeType* next;
};

SortedType::SortedType()  // Class constructor
{



}

bool SortedType::IsFull() const
{
```

```
28
29
30
31
32
33
34 }
35
36 int SortedType::GetLength() const
37 {
38   return
39 }
40
41 void SortedType::MakeEmpty()
42 {
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57 }
58
59 ItemType SortedType::GetItem(ItemType& item, bool& found)
60 {
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
```

```
82
83
84
85
86
87
88 }
89
90 void SortedType::PutItem(ItemType item)
91 {
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135 }
```

```
136 void SortedType::DeleteItem(ItemType item)
137 {
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166 }
167
168 void SortedType::ResetList()
169 {
170
171
172 }
173
174 ItemType SortedType::GetNextItem()
175 {
176
177
178
179
180
181
182
183
184
185
186
187
188 }
189
```

```
190 SortedType::~SortedType()
191 {
192
193
194
195
196
197
198
199
200
201
202 }
```