# ADF2x & PRO2x — Übungen zu Fortgeschrittenen Algorithmen & Datenstrukturen und OOP — SS 24, Übung 8

**Abgabetermin: Sa, 06.07.2024**

☒ **Gr. 1,** S. Schöberl, MSc
☐ **Gr. 2,** DI (FH) G. Horn-Völlenkle, MSc

**Name** Elias Leonhardsberger **Aufwand in h** 3

**Punkte** _____ **Tutor*in / Übungsleiter*in** ____ / ____

## 1. Ein neuer Behälter für die MiniLib (14 + 2 + 8 Punkte)

Die MiniLib bietet bereits die Behälterklasse `MLVector` mit entsprechendem Iterator. Eine Behälterklasse auf Basis einer dynamischen Liste fehlt jedoch noch.

Analysieren Sie zunächst die Klassen `MLCollection` und `MLIterator` und leiten Sie davon Ihre Lösung für eine dynamische Liste ab.

a) Implementieren Sie eine neue Behälterklasse `MLList`, die eine einfach-verkettete, nichtzyklische Liste realisiert. Implementieren Sie alle notwendigen Methoden (siehe `MLCollection`).

b) Ergänzen Sie zudem eine Methode `Prepend`.

c) Entwickeln Sie einen Iterator `MLListIterator`, der die Liste vom ersten bis zum letzten Knoten durchläuft.

Testen Sie die Liste und den Iterator ausführlich (d.h. alle Methoden) und füllen Sie verschiedene Listen mit Objekten verschiedener MiniLib-Klassen.

### Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
2. Dokumentieren und kommentieren Sie Ihre Algorithmen.
3. Bei Programmen: Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Programm funktioniert, und dass es auch in Fehlersituation entsprechend reagiert.

# ADF2/PRO2 UE08

Elias Leonhardsberger

5. Juli 2024, Hagenberg

## Inhaltsverzeichnis

# 1 Ein neuer Behälter für die MiniLib

## 1.1 Lösungsidee

Die Angabe beschreibt bereits ziemlich genau den Ablauf der Implementierung und außerdem wurden die gefragen Methoden schon bei früheren Übungen gemacht. Die Tests werden wieder wie Unittests umgesetzt.

Der MiniLib Sourcecode, bis auf MLColl.pas, wird nicht in der Abgabe angehängt, da sich kein Code geändert hat. MLColl wird angehängt da es sich hier um die Basisklasse der Liste handelt.

## 1.2 Souce Code

### 1.2.1 MLColl.pas

```
1
2  (*MLColl:                                                MiniLib V.4, 2004
3    -----
4    Abstract classes MLCollection and MLIterator,
5    the root classes for all MiniLib collection and iterator classes.
6  ========================================================================*)
7
8  UNIT MLColl;
9
10 INTERFACE
11
12 USES
13 MLObj;
14
15 TYPE
16
17   MLIterator = ^MLIteratorObj; (*full declaration below MLCollection*)
18
19 (*=== class MLCollection ===*)
20
21   MLCollection = ^MLCollectionObj;
22   MLCollectionObj = OBJECT(MLObjectObj) (*treat like an abstract class*)
23
24     CONSTRUCTOR Init;
25     DESTRUCTOR Done; VIRTUAL;
26
27 (*--- overridden methods ---*)
28
29     FUNCTION AsString: STRING; VIRTUAL;
30       (*returns string representation of collention with some elements*)
31
32     PROCEDURE WriteAsString; VIRTUAL;
33       (*writes collection to output with all elements*)
```

```
34
35  (*--- new "abstract" methods that have to be overridden ---*)
36
37      FUNCTION Size: INTEGER; VIRTUAL;
38        (*returns number of elements in collection*)
39
40      PROCEDURE Add(o: MLObject); VIRTUAL;
41        (*adds element o to collection*)
42
43      FUNCTION Remove(o: MLObject): MLObject; VIRTUAL;
44        (*removes first element = o and returns that element*)
45
46      FUNCTION Contains(o: MLObject): BOOLEAN; VIRTUAL;
47        (*returns whether collection contains an element = o*)
48
49      PROCEDURE Clear; VIRTUAL;
50        (*removes all elements WITHOUT disposing them*)
51
52      FUNCTION NewIterator: MLIterator; VIRTUAL;
53        (*returns an iterator which has to be deleted after usage*)
54
55  (*--- new method ---*)
56
57      PROCEDURE DisposeElements; VIRTUAL;
58        (*removes all elements (like Clear) AND disposes them*)
59
60    END; (*OBJECT*)
61
62
63  (*=== class MLIterator ===*)
64
65    (*MLIterator = ^MLIteratorObj;         see declaration above*)
66    MLIteratorObj = OBJECT(MLObjectObj) (*treat like an abstract class*)
67
68      CONSTRUCTOR Init;
69      DESTRUCTOR Done; VIRTUAL;
70
71  (*--- new abstract method ---*)
72
73      FUNCTION Next: MLObject; VIRTUAL;
74        (*returns next element or NIL if "end of" collection reached*)
75
76    END; (*OBJECT*)
77
78
79  (*=========================================================================*)
80
```

```
81  IMPLEMENTATION
82
83  USES
84  MetaInfo;
85
86
87  (*=== MLCollection ===*)
88
89
90  CONSTRUCTOR MLCollectionObj.Init;
91  BEGIN
92    INHERITED Init;
93    Register('MLCollection', 'MLObject');
94  END; (*MLCollectionObj.Init*)
95
96  DESTRUCTOR MLCollectionObj.Done;
97  BEGIN
98    INHERITED Done;
99  END; (*MLCollectionObj.Done*)
100
101
102 (*--- overridden method ---*)
103
104 FUNCTION MLCollectionObj.AsString: STRING;
105 LABEL (*with GOTO ;-)*)
106   999;
107 VAR
108   it:    MLIterator;
109   o:     MLObject;
110   s, os: STRING;
111   first: BOOLEAN;
112 BEGIN
113   Str(Size, s);
114   s := CLASS + ' with ' + s + ' Element(s): { ';
115   it := NewIterator;
116   o := it^.Next;
117   first := TRUE;
118   WHILE o <> NIL DO
119     BEGIN
120       os := o^.AsString;
121       IF Length(s) + Length(os) > 230 THEN
122         BEGIN
123           s := s + ' ... TRUNCATED!';
124           GOTO 999;
125         END; (*IF*)
126       IF NOT first THEN
127         s := s + ', ';
```

```
128         first := FALSE;
129         s := s + os;
130         o := it^.Next;
131       END; (*WHILE*)
132     999:
133         DISPOSE(it, Done);
134     s := s + ' }';
135     AsString := s;
136   END; (*MLCollectionObj.AsString*)
137
138   PROCEDURE MLCollectionObj.WriteAsString;
139   VAR
140     it:    MLIterator;
141     o:     MLObject;
142     first: BOOLEAN;
143   BEGIN
144     Write(CLASS, ' with ', Size, ' Element(s): { ');
145     it := NewIterator;
146     o := it^.Next;
147     first := TRUE;
148     WHILE o <> NIL DO
149       BEGIN
150         IF NOT first THEN
151           Write(', ');
152         first := FALSE;
153         Write(o^.AsString);
154         o := it^.Next;
155       END; (*WHILE*)
156     DISPOSE(it, Done);
157     WriteLn(' }');
158   END; (*MLCollectionObj.WriteAsString*)
159
160
161   (*--- default implementations of "abstract" methods ---*)
162
163   FUNCTION MLCollectionObj.Size: INTEGER;
164   BEGIN
165     AbstractMethodError('MLCollectionObj.Size');
166     Size := 0;
167   END; (*MLCollectionObj.Size*)
168
169   PROCEDURE MLCollectionObj.Add(o: MLObject);
170   BEGIN
171     AbstractMethodError('MLCollectionObj.Add');
172   END; (*MLCollectionObj.Add*)
173
174   FUNCTION MLCollectionObj.Remove(o: MLObject): MLObject;
```

```
175 BEGIN
176   AbstractMethodError('MLCollectionObj.Remove');
177   Remove := NIL;
178 END; (*MLCollectionObj.Remove*)
179
180 FUNCTION MLCollectionObj.Contains(o: MLObject): BOOLEAN;
181 BEGIN
182   AbstractMethodError('MLCollectionObj.Contains');
183   Contains := FALSE;
184 END; (*MLCollectionObj.Contains*)
185
186 PROCEDURE MLCollectionObj.Clear;
187 BEGIN
188   AbstractMethodError('MLCollectionObj.Clear');
189 END; (*MLCollectionObj.Clear*)
190
191 FUNCTION MLCollectionObj.NewIterator: MLIterator;
192 BEGIN
193   AbstractMethodError('MLCollectionObj.Iterator');
194   NewIterator := NIL;
195 END; (*MLCollectionObj.NewIterator*)
196
197
198 (*--- new method ---*)
199
200 PROCEDURE MLCollectionObj.DisposeElements;
201 VAR
202   it: MLIterator;
203   o:  MLObject;
204 BEGIN
205   it := NewIterator;
206   o := it^.Next;
207   WHILE o <> NIL DO
208     BEGIN
209       IF o^.IsA('MLCollection') THEN  (*dispose its elements rec.*)
210         MLCollection(o)^.DisposeElements;
211       DISPOSE(o, Done);
212       o := it^.Next;
213     END; (*WHILE*)
214   DISPOSE(it, Done);
215   Clear;
216 END; (*MLCollectionObj.DisposeElements*)
217
218
219 (*=== MLIterator ===*)
220
221
```

```
222 CONSTRUCTOR MLIteratorObj.Init;
223 BEGIN
224   INHERITED Init;
225   Register('MLIterator', 'MLObject');
226 END; (*MLIteratorObj.Init*)
227
228 DESTRUCTOR MLIteratorObj.Done;
229 BEGIN
230   INHERITED Done;
231 END; (*MLIteratorObj.Done*)
232
233
234 (*--- default implementation of "abstract" method ---*)
235
236 FUNCTION MLIteratorObj.Next: MLObject;
237 BEGIN
238   AbstractMethodError('MLIteratorObj.Next');
239   Next := NIL;
240 END; (*MLIteratorObj.Next*)
241
242
243 END. (*MLColl*)
```

### 1.2.2 MLLi.pas

```
1
2  UNIT MLLi;
3
4  INTERFACE
5
6  USES
7  MLObj, MLColl;
8
9  TYPE
10
11   MLObjectNodePtr = ^MLObjectNode;
12   MLObjectNode = RECORD
13     obj: MLObject;
14     next: MLObjectNodePtr;
15   END;
16
17   MLListIterator = ^MLListIteratorObj;
18
19   MLList = ^MLListObj;
20   MLListObj = OBJECT(MLCollectionObj)
21
22     CONSTRUCTOR Init;
23
```

```
24      DESTRUCTOR Done; VIRTUAL;
25
26      FUNCTION Size: INTEGER; VIRTUAL;
27
28      PROCEDURE Add(o: MLObject); VIRTUAL;
29
30      FUNCTION Remove(o: MLObject): MLObject; VIRTUAL;
31
32      FUNCTION Contains(o: MLObject): BOOLEAN; VIRTUAL;
33
34      PROCEDURE Clear; VIRTUAL;
35
36      FUNCTION NewIterator: MLIterator; VIRTUAL;
37
38      PROCEDURE Prepend(o: MLObject); VIRTUAL;
39
40      PRIVATE
41
42        head: MLObjectNodePtr;
43        curSize: INTEGER;
44
45    END;
46
47    MLListIteratorObj = OBJECT(MLIteratorObj)
48
49      CONSTRUCTOR Init(list: MLList);
50
51      DESTRUCTOR Done; VIRTUAL;
52
53      FUNCTION Next: MLObject; VIRTUAL;
54
55      FUNCTION GetList: MLList; VIRTUAL;
56
57      PRIVATE
58
59        cur: MLObjectNodePtr;
60
61        l: MLList;
62
63    END;
64
65
66  FUNCTION NewMLList: MLList;
67
68  IMPLEMENTATION
69
70  USES
```

```
71  MetaInfo;
72
73  FUNCTION NewMLList: MLList;
74  VAR
75    l: MLList;
76  BEGIN (* NewMLList *)
77    NEW(l, Init);
78    NewMLList := l;
79  END; (* NewMLList *)
80
81  CONSTRUCTOR MLListObj.Init;
82  BEGIN (* MLListObj *)
83    INHERITED Init();
84    Register('MLList', 'MLCollection');
85    curSize := 0;
86    head := NIL;
87  END; (* MLListObj *)
88
89  DESTRUCTOR MLListObj.Done;
90  BEGIN (* MLListObj *)
91    Clear();
92    INHERITED Done();
93  END; (* MLListObj *)
94
95  FUNCTION MLListObj.Size: INTEGER;
96  BEGIN (* MLListObj.Size *)
97    Size := curSize;
98  END; (* MLListObj.Size *)
99
100 PROCEDURE MLListObj.Add(o: MLObject);
101 VAR
102   temp, prev: MLObjectNodePtr;
103 BEGIN (* MLListObj.Add *)
104   curSize := curSize + 1;
105   temp := head;
106   prev := NIL;
107
108   WHILE temp <> NIL DO
109     BEGIN (* WHILE *)
110       prev := temp;
111       temp := temp^.next;
112     END; (* WHILE *)
113
114   NEW(temp);
115   temp^.obj := o;
116   temp^.next := NIL;
117
```

```
118     IF prev = NIL THEN
119       BEGIN (* IF *)
120         head := temp
121       END (* IF *)
122     ELSE
123       BEGIN (* ELSE *)
124         prev^.next := temp;
125       END; (* ELSE *)
126   END; (* MLListObj.Add *)
127
128   FUNCTION MLListObj.Remove(o: MLObject): MLObject;
129   VAR
130     temp, prev: MLObjectNodePtr;
131     found: BOOLEAN;
132   BEGIN (* MLListObj.Remove *)
133     temp := head;
134     prev := NIL;
135     found := FALSE;
136
137     WHILE ((temp <> NIL) AND (NOT found)) DO
138       BEGIN (* WHILE *)
139         IF temp^.obj^.IsEqualTo(o) THEN
140           BEGIN (* IF *)
141             found := TRUE;
142           END (* IF *)
143         ELSE
144           BEGIN (* ELSE *)
145             prev := temp;
146             temp := temp^.next;
147           END; (* ELSE *)
148       END; (* WHILE *)
149
150     IF found THEN
151       BEGIN (* IF *)
152         curSize := curSize - 1;
153
154         IF prev = NIL THEN
155           BEGIN (* IF *)
156             head := temp^.next;
157           END (* IF *)
158         ELSE
159           BEGIN (* ELSE *)
160             prev^.next := temp^.next;
161           END; (* ELSE *)
162
163         Remove := temp^.obj;
164         DISPOSE(temp);
```

```
165       END (* IF *)
166     ELSE
167       BEGIN (* ELSE *)
168         Remove := NIL;
169       END; (* ELSE *)
170   END; (* MLListObj.Remove *)
171
172   FUNCTION MLListObj.Contains(o: MLObject): BOOLEAN;
173   VAR
174     temp: MLObjectNodePtr;
175     found: BOOLEAN;
176   BEGIN (* MLListObj.Contains *)
177     temp := head;
178     found := FALSE;
179
180     WHILE ((temp <> NIL) AND (NOT found)) DO
181       BEGIN (* WHILE *)
182         IF temp^.obj^.IsEqualTo(o) THEN
183           BEGIN (* IF *)
184             found := TRUE;
185           END (* IF *)
186         ELSE
187           BEGIN (* ELSE *)
188             temp := temp^.next;
189           END; (* ELSE *)
190       END; (* WHILE *)
191
192     Contains := found;
193   END; (* MLListObj.Contains *)
194
195
196   PROCEDURE MLListObj.Clear;
197   VAR
198     temp: MLObjectNodePtr;
199   BEGIN (* MLListObj.Clear *)
200     WHILE head <> NIL DO
201       BEGIN (* WHILE *)
202         temp := head^.next;
203         DISPOSE(head);
204         head := temp;
205       END; (* WHILE *)
206
207     curSize := 0;
208   END; (* MLListObj.Clear *)
209
210   FUNCTION MLListObj.NewIterator: MLIterator;
211   VAR
```

```
212   it: MLListIterator;
213 BEGIN (* MLListObj.NewIterator *)
214   NEW(it, Init(@SELF));
215   NewIterator := it;
216 END; (* MLListObj.NewIterator *)
217
218 PROCEDURE MLListObj.Prepend(o: MLObject);
219 VAR
220   temp: MLObjectNodePtr;
221 BEGIN (* MLListObj.Prepend *)
222   curSize := curSize + 1;
223   NEW(temp);
224   temp^.obj := o;
225   temp^.next := head;
226   head := temp;
227 END; (* MLListObj.Prepend *)
228
229 CONSTRUCTOR MLListIteratorObj.Init(list: MLList);
230 BEGIN (* MLListIteratorObj *)
231   INHERITED Init();
232   Register('MLListIterator', 'MLIterator');
233   l := list;
234   cur := l^.head;
235 END; (* MLListIteratorObj *)
236
237 DESTRUCTOR MLListIteratorObj.Done;
238 BEGIN (* MLListIteratorObj *)
239   INHERITED Done;
240 END; (* MLListIteratorObj *)
241
242 FUNCTION MLListIteratorObj.Next: MLObject;
243 VAR
244   o: MLObject;
245 BEGIN (* MLListIteratorObj *)
246   IF cur <> NIL THEN
247     BEGIN (* IF *)
248       o := cur^.obj;
249       cur := cur^.next;
250       Next := o;
251     END (* IF *)
252   ELSE
253     BEGIN (* ELSE *)
254       Next := NIL;
255     END; (* ELSE *)
256 END; (* MLListIteratorObj.Next *)
257
258 FUNCTION MLListIteratorObj.GetList: MLList;
```

```
259 BEGIN (* MLListIteratorObj.GetList *)
260   GetList := l;
261 END; (* MLListIteratorObj.GetList *)
262
263 END.
```

## 1.3 Tests

### 1.3.1 TestMLList.pas

```
1  PROGRAM TestMLList;
2
3  USES
4  MLLi, MLInt, MLColl, MLObj;
5
6  TYPE
7    test = PROCEDURE (l: MLList; VAR success: BOOLEAN; i1, i2, i3:
   ↪  MLInteger);
8
9  PROCEDURE NewList_IsEmpty(l: MLList; VAR success: BOOLEAN; i1, i2, i3:
   ↪  MLInteger);
10 BEGIN (* NewList_IsEmpty *)
11   success := (l^.Size() = 0);
12 END; (* NewList_IsEmpty *)
13
14 PROCEDURE Clear_EmptiesList(l: MLList; VAR success: BOOLEAN; i1, i2, i3:
   ↪  MLInteger);
15 BEGIN (* Clear_EmptiesList *)
16   l^.Add(i1);
17   l^.Add(i2);
18   l^.Add(i3);
19
20   success := (l^.Size() = 3);
21   l^.Clear();
22
23   success := success
24              AND (l^.Size() = 0);
25 END; (* Clear_EmptiesList *)
26
27 PROCEDURE Add_AddsCorrectElement(l: MLList; VAR success: BOOLEAN; i1, i2,
   ↪  i3: MLInteger);
28 VAR
29   o: MLObject;
30   it: MLIterator;
31 BEGIN (* Add_AddsCorrectElement *)
32   l^.Add(i1);
33
34   it := l^.NewIterator();
35   o := it^.Next();
36   DISPOSE(it, Done);
37
38   success := (i1^.IsEqualTo(o))
39              AND (l^.Size() = 1);
40 END; (* Add_AddsCorrectElement *)
```

```
41
42  PROCEDURE Remove_RemovesCorrectElement(l: MLList; VAR success: BOOLEAN;
    ↪  i1, i2, i3: MLInteger);
43  VAR
44    o1, o2, o3: MLObject;
45    it: MLIterator;
46  BEGIN (* Remove_RemovesCorrectElement *)
47    l^.Add(i1);
48    l^.Add(i2);
49
50    o1 := l^.Remove(i1);
51    o3 := l^.Remove(i3);
52
53    it := l^.NewIterator();
54    o2 := it^.Next();
55    DISPOSE(it, Done);
56
57    success := (i1^.IsEqualTo(o1))
58               AND (i2^.IsEqualTo(o2))
59               AND (o3 = NIL)
60               AND (l^.Size() = 1);
61  END; (* Remove_RemovesCorrectElement *)
62
63  PROCEDURE Contains_ReturnsCorrectValue(l: MLList; VAR success: BOOLEAN;
    ↪  i1, i2, i3: MLInteger);
64  BEGIN (* Contains_ReturnsCorrectValue *)
65    l^.Add(i1);
66
67    success := l^.Contains(i1)
68               AND NOT l^.Contains(i2);
69  END; (* Contains_ReturnsCorrectValue *)
70
71  PROCEDURE Prepend_AddsElementAtBeginning(l: MLList; VAR success: BOOLEAN;
    ↪  i1, i2, i3: MLInteger);
72  VAR
73    o1, o2, o3: MLObject;
74    it: MLIterator;
75  BEGIN (* Prepend_AddsElementAtBeginning *)
76    l^.Add(i1);
77    l^.Prepend(i2);
78    l^.Prepend(i3);
79
80    it := l^.NewIterator();
81    o1 := it^.Next();
82    o2 := it^.Next();
83    o3 := it^.Next();
84    DISPOSE(it, Done);
```

```
85
86    success := (i3^.IsEqualTo(o1))
87               AND (i2^.IsEqualTo(o2))
88               AND (i1^.IsEqualTo(o3))
89               AND (l^.Size() = 3);
90  END; (* Prepend_AddsElementAtBeginning *)
91
92  PROCEDURE RunTest(NAME: STRING; t: test; i1, i2, i3: MLInteger);
93  VAR
94    l: MLList;
95    success: BOOLEAN;
96  BEGIN (* RunTest *)
97    l := NewMLList();
98    t(l, success, i1, i2, i3);
99    DISPOSE(l, Done);
100
101   IF (success) THEN
102     BEGIN (* IF *)
103       WriteLn('PASSED - ', name);
104     END (* IF *)
105   ELSE
106     BEGIN (* ELSE *)
107       WriteLn('FAILED - ', name);
108     END; (* ELSE *)
109 END; (* RunTest *)
110
111 VAR
112   i1, i2, i3: MLInteger;
113 BEGIN (* TestMLList *)
114   i1 := NewMLInteger(1);
115   i2 := NewMLInteger(2);
116   i3 := NewMLInteger(3);
117
118   RunTest('NewList_IsEmpty', NewList_IsEmpty, i1, i2, i3);
119   RunTest('Clear_EmptiesList', Clear_EmptiesList, i1, i2, i3);
120   RunTest('Add_AddsCorrectElement', Add_AddsCorrectElement, i1, i2, i3);
121   RunTest('Remove_RemovesCorrectElement', Remove_RemovesCorrectElement,
      ↪ i1, i2, i3);
122   RunTest('Contains_ReturnsCorrectValue', Contains_ReturnsCorrectValue,
      ↪ i1, i2, i3);
123   RunTest('Prepend_AddsElementAtBeginning',
      ↪ Prepend_AddsElementAtBeginning, i1, i2, i3);
124
125   DISPOSE(i1, Done);
126   DISPOSE(i2, Done);
127   DISPOSE(i3, Done);
128   WriteLn('All tests passed');
```

```
129 END. (* TestMLList *)
```

## 1.4 Testergebnisse

```
PS C:\Users\leonh\Repos\SEbaBB2\pascal\PascalWorkspace\UE8\bin> .\TestMLList.exe
PASSED - NewList_IsEmpty
PASSED - Clear_EmptiesList
PASSED - Add_AddsCorrectElement
PASSED - Remove_RemovesCorrectElement
PASSED - Contains_ReturnsCorrectValue
PASSED - Prepend_AddsElementAtBeginning
All tests passed
PS C:\Users\leonh\Repos\SEbaBB2\pascal\PascalWorkspace\UE8\bin>
```

Abbildung 1: Ausgabe des Testprogramms *TestMLList*