

1 Einfach verkettete Liste

1.1 Lösungsidee

Die Prozedur RemoveMax wird folgendermaßen implementiert.

Mittels Abfrage wird zuerst ausgeschlossen, dass es sich um eine leere Liste handelt.

Dann wird die Liste mithilfe einer Schleife iteriert. In jedem Schleifendurchlauf wird geprüft, ob der aktuelle Datenwert größer als der bisher größte Datenwert ist. Wenn das der Fall ist wird der Zeiger auf den Knoten mit der größeren Zahl und der Zeiger auf den Wert davor gespeichert.

Wenn die Liste vollständig durchlaufen ist, kann man jetzt den Knoten mit dem größten Datenwert von der Liste aushängen, indem man die next Komponente des vorherigen Knotens mit der Adresse des Nachfolgeknotens überschreibt.

Die Adresse des gefundenen Knoten mit dem höchsten Datenwert wird über den Ausgangsparameter maxNode zurückgegeben.

Zum Testen werden noch weitere Prozeduren erstellt, wie etwa zum Hinzufügen eines Knotens, zum Ausgeben einer Liste oder zum Disposed.

```

1  PROGRAM SinglyLinkedList;
2
3  TYPE
4      ListNodePtr = ^ListNode;
5      ListNode = RECORD
6          next: ListNodePtr;
7          data: INTEGER;
8      END;
9      ListPtr = ListNodePtr;
10
11  PROCEDURE AddNode(VAR list: ListPtr; val: INTEGER);
12      VAR
13          newNode: ListNodePtr;
14          last: ListNodePtr;
15  BEGIN
16      New(newNode);
17      newNode^.data := val;
18      newNode^.next := NIL; (*Wert wird hinten angefügt*)
19
20      (*Liste leer*)
21      IF (list = NIL) THEN
22          list := newNode
23      ELSE BEGIN
24          last := list;
25          (*Liste nicht leer*)
26          WHILE (last^.next <> NIL) DO BEGIN
27              last := last^.next;
28          END;
29          (*jetzt ist last^.next = nil. Also Fügen wir dort Knoten ein*)
30          last^.next := newNode;
31      END;
32  END;
33
34  PROCEDURE ReadList(VAR list: ListPtr);
35      VAR
36          val: INTEGER;
37  BEGIN
38      WriteLn('Enter some int values for your list (end with 0): ');
39      Write('> '); ReadLn(val);
40      WHILE (val <> 0) DO BEGIN
41          AddNode(list, val);
42          Write('> '); ReadLn(val);
43      END;
44
45  END;

```

```

46
47     PROCEDURE DisposeList(VAR list: ListPtr);
48         VAR
49             tmpList: ListNodePtr;
50
51         BEGIN
52             WHILE (list <> NIL) DO BEGIN
53                 tmpList := list;
54                 list := list^.next;
55                 Dispose(tmpList);
56             END;
57         END;
58
59     PROCEDURE WriteList(list: ListPtr);
60     BEGIN
61         Write('[');
62         WHILE (list <> NIL) DO BEGIN
63             Write(list^.data:3);
64             list := list^.next;
65         END; (*while*)
66         Write('  ]');
67         WriteLn;
68
69
70     END;
71
72     PROCEDURE RemoveMax(VAR list: ListPtr; VAR maxNode: ListNodePtr);
73         VAR
74             curr: ListNodePtr;
75             maxNodePtr: ListNodePtr;
76             prev: ListNodePtr;
77             prevMaxNodePtr: ListNodePtr;
78     BEGIN
79         (*empty list*)
80         IF (list = NIL) THEN BEGIN
81             maxNode := NIL;
82             Exit;
83         END;
84
85         (*not empty list*)
86         prev := NIL;
87         curr := list;
88         maxNodePtr := curr; (* init für den Fall, dass die while Schleife
überprungen wird weil nur ein Element drin ist*)
89         prevMaxNodePtr := NIL;
90         WHILE (curr <> NIL) DO BEGIN

```

```

91     IF (curr^.data > maxNodePtr^.data) THEN BEGIN
92         maxNodePtr := curr;
93         prevMaxNodePtr := prev;
94     END; (* if*)
95
96     prev := curr;
97     curr := curr^.next;
98 END; (*while*)
99 (*jetzt: maxnodePtr ist der ptr zum node mit dem höchsten datenwert*)
100 (*jetzt muss dieser node gelöscht werden*)
101
102 (*Fall 1: 1. wert ist größter Wert (zu verändernder Wert ist im Compound
einer Node sondern in der Startvariable list)*)
103 IF prevMaxNodePtr = NIL THEN BEGIN (*Bedingung in der while Schleife nie
erfüllt*)
104     list := list^.next;
105 END ELSE BEGIN
106     prevMaxNodePtr^.next := maxNodePtr^.next; (* "Aushängen" des größten
Wertes. Hier könnte man dispoen, aber weil die Adresse zurückgegeben wird,
dispose ich nicht, damit man noch was damit machen kann*)
107 END; (*if*)
108 maxNode := maxNodePtr; (*Zuweisen vom Rückgabewert*)
109 END;
110
111 VAR
112     l: ListPtr;
113     maxValPtr: ListNodePtr;
114 BEGIN
115     ReadList(l);
116     WriteLn('list entered: ');
117     WriteList(l);
118     WriteLn('max val removed:');
119     RemoveMax(l, maxValPtr);
120     WriteList(l);
121     IF (maxValPtr <> NIL) THEN
122         WriteLn('Value removed: ', maxValPtr^.data)
123     ELSE
124         WriteLn('Emty List');
125     Dispose(maxValPtr);
126     DisposeList(l);
127 END.

```

1.3 Tests

Eingabe	Beschreibung	Ausgabe
1 3 5 4	Standardeingabe	<pre>Enter some int values for your list (end with 0): > 1 > 3 > 5 > 4 > 0 list entered: [1 3 5 4] max val removed: [1 3 4] Value removed: 5</pre>
5 3 4 1 2	Erster Wert am Größten	<pre>Enter some int values for your list (end with 0): > 5 > 3 > 4 > 1 > 2 > 0 list entered: [5 3 4 1 2] max val removed: [3 4 1 2] Value removed: 5</pre>
1 3 4 5	Letzter Wert am Größten	<pre>list entered: [1 3 4 5] max val removed: [1 3 4] Value removed: 5</pre>
1 5 4 5	Wenn der Wert zweimal vorkommt, wird der erste entfernt	<pre>list entered: [1 5 4 5] max val removed: [1 4 5] Value removed: 5</pre>
4	nur 1 Knoten	<pre>list entered: [4] max val removed: [] Value removed: 4</pre>
-	Leere Liste	<pre>Enter some int values for your list (end with 0): > 0 list entered: [] max val removed: [] Empty List</pre>

2 WWW-Zugriffszahlen

2.1 Lösungsidee

Die Hauptprozedur verwendet mehrere Funktionen.

Die Funktion String2IPAddr verwandelt eine als string übergebene IP-Adresse in ein Array of Bytes mit 4 Elementen. Dazu wird in einer Schleife der string iteriert. Wenn ein Char eine Ziffer ist, wird er zu einem Substring hinzugefügt. Wenn ein Punkt kommt, wird dieser Substring mit der Standardprozedur Val in einen Integer konvertiert und der Substring wird neu initialisiert.

Die Funktion SortedInsert ist das Herz des Programmes. Sie sorgt dafür, dass eine IP-Adresse in Form eines Arrays an der richtigen Stelle in eine Liste eingefügt wird. Dabei wird die Liste iteriert und immer der Datenwert des Elements in der Liste mit dem Datenwert des einzufügenden Knoten verglichen. Ist der einzufügende Datenwert plötzlich nicht mehr größer als die Werte der Liste muss der Knoten dort eingefügt werden. Durch Abfragen und Vergleiche etc. werden auch Edge-Cases berücksichtigt.

Die Hauptprozedur userInteraction lest in einer Schleife die IP-Adressen solange ein, bis das 'End of file' erreicht ist. Die Adressen werden mit der Funktion String2IPAddr in Arrays konvertiert und mit der Prozedur SortedInsert in die Liste eingefügt. Danach wird die Liste iteriert und eine Countervariable verwendet um zu prüfen, Wieviele IP-Adressen mehrmals vorkommen.

2.2 Quellcode

```
1  PROGRAM WWWZugriffszahlen;
2
3  CONST
4      ip4max = 4;
5
6  TYPE
7      IPAddr = ARRAY[1..ip4max] OF BYTE;
8      IPAddrNodePtr = ^IPAddrNode;
9      IPAddrNode = RECORD
10         next: IPAddrNodePtr;
11         addr: IPAddr;
12         count: INTEGER; (*number of accesses*)
13     END; (*record*)
14     IPAddrListPtr = IPAddrNodePtr;
15
16 FUNCTION IPGreaterThen(ip1, ip2: IPAddr): BOOLEAN;
17     VAR
18         i: INTEGER;
19         bEqualValues: BOOLEAN;
20         bResult: BOOLEAN;
21     BEGIN
22         i := 1;
23         bEqualValues := TRUE;
24         WHILE (i <= high(ip1)) AND (i <= high(ip2)) AND bEqualValues DO BEGIN
25             bResult := (ip1[i] > ip2[i]);
26             bEqualValues := (ip1[i] = ip2[i]);
27             inc(i);
28         END;
29         IPGreaterThen := bResult;
30     END;
31
32 FUNCTION IPSEqual(ip1, ip2: IPAddr): BOOLEAN;
33     VAR
34         bResult: BOOLEAN;
35         i: INTEGER;
36     BEGIN
37         i := 1;
38         bResult := TRUE;
39         WHILE (i <= high(ip1)) AND (i <= high(ip2)) AND bResult DO BEGIN
40             bResult := ip1[i] = ip2[i];
41             inc(i);
42         END;
43         IPSEqual := bResult;
```

```

44     END;
45
46
47     FUNCTION String2IPAddr(strAddress: STRING): IPAddr;
48     VAR
49         i: INTEGER;
50         subCount: INTEGER;
51         subString: STRING;
52         resultArr: IPAddr;
53
54         (*vars for val procedure*)
55         nStringVal: INTEGER;
56         errCode: WORD;
57     BEGIN
58         subString := '';
59         subCount := 1;
60
61         nStringVal := 0;
62         errCode := 0;
63         FOR i := 1 TO Length(strAddress) DO BEGIN
64             IF strAddress[i] IN ['0'..'9'] THEN BEGIN
65                 subString := subString + strAddress[i];
66             END ELSE BEGIN
67                 val(subString, nStringVal, errCode);
68                 resultArr[subCount] := nStringVal;
69
70                 inc(subCount);
71                 subString := ''; (*reset*)
72             END;
73         END; (*for*)
74         IF (subString <> '') THEN BEGIN
75             val(subString, nStringVal, errCode);
76             resultArr[subCount] := nStringVal;
77         END;
78         String2IPAddr := resultArr;
79     END;
80
81     PROCEDURE PrintIP(addr: IPAddr);
82     VAR
83         i: INTEGER;
84     BEGIN
85         FOR i := 1 TO high(addr) DO
86             Write(addr[i]:5);
87         WriteLn;
88     END;
89

```



```

90      (*procedure SortedInsert() und Insert besser in einer Funktion, damit die
Liste nicht mehrmals durchlaufen wird*)
91
92      PROCEDURE SortedInsert(VAR list: IPAddrListPtr; adress: IPAddr);
93      VAR
94          pNewNode: IPAddrNodePtr;
95          curr: IPAddrNodePtr;
96          prev: IPAddrNodePtr;
97
98      BEGIN
99          New(pNewNode);
100         pNewNode^.addr := adress;
101         pNewNode^.next := NIL;
102         pNewNode^.count := 1;
103
104         curr := list; (*init*)
105         prev := list;
106
107
108         IF (list = NIL) THEN BEGIN
109             list := pNewNode;
110             Exit; (*list empty*)
111         END; (*if*)
112
113         WHILE (curr <> NIL) AND IPGreaterThan(pNewNode^.addr, curr^.addr) DO
114             BEGIN
115                 prev := curr;
116                 curr := curr^.next;
117             END;
118             (*Möglichkeiten: curr = Nil --> Knoten hinten einfügen - prev^.next :=
n*)
119             (* oder: curr <> nil --> Einfügeposition gefunden, dann ist
not(pNewNode^.addr > curr^.addr), also (pNewNode^.address <= curr^.addr)*)
120             (*Eingefügt werden muss zwischen prev und curr*)
121             IF (curr <> NIL) AND IPSEqual(pNewNode^.addr, curr^.addr) THEN BEGIN
122                 (*wenn gleich, counter erhöhen, Neue Node wieder dispoen*)
123                 inc(curr^.count);
124                 Dispose(pNewNode);
125             END ELSE BEGIN
126                 IF (curr = NIL) THEN BEGIN (*hinten einfügen*)
127                     prev^.next := pNewNode;
128                 END ELSE
129                 IF (curr = list) THEN BEGIN
130                     pNewNode^.next := list;
131                     list := pNewNode;
132                 END ELSE BEGIN (* curr <> nil*)

```

```

131         pNewNode^.next := prev^.next; (*Adr von curr*)
132         prev^.next := pNewNode; (*fertig eingehängt*)
133     END; (*if*)
134 END; (*if*)
135 END;
136
137 PROCEDURE DisposeList(VAR list: IPAddrListPtr);
138 VAR
139     tmpList: IPAddrNodePtr;
140
141 BEGIN
142     WHILE (list <> NIL) DO BEGIN
143         tmpList := list;
144         list := list^.next;
145         Dispose(tmpList);
146     END;
147 END;
148
149 PROCEDURE userInteraction;
150 VAR
151     strIP: STRING;
152     arrIPAddr: IPAddr;
153     list: IPAddrListPtr;
154     multiAddrCount: INTEGER;
155
156     (*zum Liste iterieren:*)
157     curr: IPAddrNodePtr;
158 BEGIN
159     list := NIL; (*init*)
160     WHILE NOT eof DO BEGIN
161         ReadLn(strIP);
162         arrIPAddr := String2IPAddr(strIP);
163         SortedInsert(list, arrIPAddr);
164     END;
165
166     (*Liste iterieren*)
167     multiAddrCount := 0;
168     curr := list;
169
170     WriteLn;
171     WHILE (curr <> NIL) DO BEGIN
172         PrintIP(curr^.addr);
173         IF (curr^.count > 1) THEN
174             inc(multiAddrCount);
175         curr := curr^.next;
176     END; (*while *)

```

```
177
178     DisposeList(list);
179     list := NIL;
180     WriteLn('Number of Adresses with more than 1 access: ', multiAddrCount);
181     WriteLn;
182     END;
183
184 BEGIN
185     userInteraction;
186     END.
```

2.3 Tests

Eingabe	Beschreibung	Ausgabe
ip3.txt	Standardfall -- funktioniert, auch Speicher ist wieder freigegeben	<pre> 216 178 32 48 Number of Addresses with more than 1 access: 8 Heap dump by heaptrc unit of G:\Meine Ablage\Obsidian \UE6\Abgabe\www-zugriff_style.exe 163 memory blocks allocated : 1956/2608 163 memory blocks freed : 1956/2608 0 unfreed memory blocks : 0 True heap size : 98304 (144 used in System startup) True free heap : 98160 </pre>
ip1.txt: 128.112.128.15 128.112.136.35 128.112.18.11 140.247.50.127 128.103.60.24	keine Adresse kommt zweimal vor, IP-Adressen sind sortiert	<pre> 128 103 60 24 128 112 18 11 128 112 128 15 128 112 136 35 140 247 50 127 Number of Addresses with more than 1 access: 0 </pre>
10.0.0.2 10.0.0.1 10.00.00.01	führende Nullen werden ignoriert	<pre> 10 0 0 1 10 0 0 2 Number of Addresses with more than 1 access: 1 </pre>
-	leere file	<pre> le.exe < IPAddr\ipTest.txt Number of Addresses with more than 1 access: 0 </pre>
199.106.69.11 128.36.229.30 66.94.234.13 128.36.229.30 128.36.229.30	Adressen kommen mehrmals vor, nur Counter wird erhöht	<pre> 66 94 234 13 128 36 229 30 199 106 69 11 Number of Addresses with more than 1 access: 1 </pre>
255.255.255.255	Grenzwert 255 und nur 1 Element	<pre> 255 255 255 255 Number of Addresses with more than 1 access: 0 </pre>

3 Bibliotheksverwaltung

3.1 Lösungsidee

Die Idee besteht darin, verschachtelt in Listen sortiert einzufügen.

In der Prozedur `InsertBook` wird zunächst geprüft, ob die Bücherliste noch leer ist. Ist das der Fall, wird der erste Knoten erstellt und eingehängt. Ansonsten wird die Liste solange mit einem Hilfszeiger durchlaufen, bis der einzufügende Wert lexikographisch größer als der Datenwert des jeweiligen Knoten in der Liste ist. Je nachdem, auf welchen Knoten der Hilfszeiger jetzt zeigt, müssen verschiedene Fälle berücksichtigt werden. Im Allgemeinen Fall (Einfügen in der Mitte, Datenwert des Hilfszeigers ist nicht gleich der einzufügenden Daten) wird für den einzufügenden Buchtitel ein neuer Knoten erstellt und der Knoten wird vor dem Knoten, auf den der Hilfszeiger zeigt, eingefügt.

Beim Erstellen eines neuen Knoten in der Bücherliste, muss auch eine neue Liste für die Autoren des Buches erstellt werden. Dies geschieht mittels Aufruf einer dafür vorgesehenen Prozedur (`SortedInsertAuth`). Sie geht ähnlich vor wie die Hauptprozedur `InsertBook`: Wenn die übergebene Liste noch leer ist, wird ein neuer Knoten erstellt. Ansonsten wird die Liste solange durchlaufen, bis die Einfügeposition gefunden wurde. Wenn der Name des Authors in der Liste noch nicht existiert, wird er eingefügt.

Die Funktion `NrOfBooksOf` durchläuft die ganze Bücherliste. Wenn in einer zu einem Buch gehörenden Autorenliste der jeweilige Name gefunden wird (Funktionsaufruf `inAuthorList()`), wird eine Countervariable inkrementiert. Die Funktion `inAuthorList` durchläuft dabei immer die übergebene Liste der Autoren. Wird der Author darin gefunden, ist der Rückgabewert `True`, ansonsten `False`.

Die Prozedur `PrintAll` durchläuft erneut die ganze Liste und darin immer (in einer eigenen Prozedur) die Liste der Autoren. In jedem Schleifendurchlauf wird der Datenwert einmal ausgegeben.

`DisposeAll` verwendet wieder eine ähnliche Logik.

3.2 Quellcode

```
1  PROGRAM Bibliotheksverwaltung;
2
3  TYPE
4      (*AuthorNode*)
5      AuthorNodePtr = ^AuthorNode;
6      AuthorNode = RECORD
7          next: AuthorNodePtr;
8          name: STRING;
9      END;
10
11     (*BookNodes*)
12     BookNodePtr = ^BookNode;
13     BookNode = RECORD
14         prev, next: BookNodePtr;
15         title: STRING;
16         authors: AuthorNodePtr;
17     END;
18     BookListPtr = BookNodePtr;
19
20     VAR (*nötig, weil die Schnittstelle InsertBook keinen Platz für einen
21         Listenparameter hat*)
22         pList: BookListPtr;
23
24     PROCEDURE SortedInsertAuth(VAR auList: AuthorNodePtr; author: STRING);
25     VAR
26         NewNodePtr: AuthorNodePtr;
27         curr, prev: AuthorNodePtr;
28     BEGIN
29         NewNodePtr := NIL;
30         curr := auList;
31         prev := NIL;
32
33         IF (auList = NIL) THEN BEGIN
34             New(NewNodePtr);
35             auList := NewNodePtr;
36             NewNodePtr^.next := NIL;
37             NewNodePtr^.name := author;
38         END ELSE BEGIN
39             WHILE (curr <> NIL) AND (author > curr^.name) DO BEGIN
40                 prev := curr;
41                 curr := curr^.next;
42             END; (*while*)
```

```

43     IF (curr = NIL) THEN BEGIN (*hinten einfügen*)
44         New(NewNodePtr);
45         NewNodePtr^.next := NIL;
46         NewNodePtr^.name := author;
47         prev^.next := NewNodePtr;
48     END ELSE
49         IF (author <> curr^.name) THEN BEGIN (*nichts tun wenn bereits
drin*)
50             IF (prev = NIL) THEN BEGIN (*Schleife nie betreten -- vorne
einfügen*)
51                 New(NewNodePtr);
52                 NewNodePtr^.next := auList;
53                 auList := NewNodePtr;
54                 NewNodePtr^.name := author;
55             END ELSE BEGIN (*irgendwo dazwischen einfügen (zwischen prev und
curr)*)
56                 New(NewNodePtr);
57                 NewNodePtr^.name := author;
58                 NewNodePtr^.next := curr;
59                 prev^.next := NewNodePtr;
60             END; (* fertig - eingefügt oder Author bereits in der Liste*)
61         END; (*if (author <> curr^.name)*)
62     END;
63 END; (*procedure*)
64
65
66 PROCEDURE InsertBook(title: STRING; author: STRING);
67     VAR
68         newBookPtr: BookNodePtr;
69         curr: BookNodePtr;
70 BEGIN
71     (*init*)
72     newBookPtr := NIL;
73     curr := pList;
74
75     IF (pList = NIL) THEN BEGIN
76         New(newBookPtr);
77         newBookPtr^.prev := NIL;
78         newBookPtr^.next := NIL;
79         newBookPtr^.title := title;
80         newBookPtr^.authors := NIL;
81         SortedInsertAuth(newBookPtr^.authors, author);
82
83         pList := newBookPtr;
84     END ELSE BEGIN
85         (*Liste nach dem Wert durchsuchen*)

```

```

86     WHILE (curr^.next <> NIL) AND (title > curr^.title) DO BEGIN
87         curr := curr^.next;
88     END;
89     (* Möglichkeiten: Knoten enthalten oder nicht*)
90     IF (curr <> NIL) AND (curr^.title = title) THEN BEGIN
91         (*Knoten gibts – curr ist der Knoten*)
92         SortedInsertAuth(curr^.authors, author);
93     END ELSE BEGIN
94         (*Knoten gibts noch nicht -> curr^.title > newBookPtr^.title*)
95         New(newBookPtr);
96         newBookPtr^.title := title;
97         newBookPtr^.authors := NIL;
98         SortedInsertAuth(newBookPtr^.authors, author);
99
100        (*einhängen*)
101        IF (curr^.title > title) THEN BEGIN (*vor curr einfügen*)
102            newBookPtr^.prev := curr^.prev;
103            newBookPtr^.next := curr;
104            IF (curr^.prev = NIL) THEN BEGIN (*curr ist erster Knoten*)
105                pList := newBookPtr;
106            END ELSE BEGIN
107                curr^.prev^.next := newBookPtr;
108            END;
109            curr^.prev := newBookPtr;
110        END ELSE BEGIN (* curr^.title < title --> letztes Element, nach curr
einfügen*)
111            newBookPtr^.prev := curr;
112            newBookPtr^.next := NIL;
113            curr^.next := newBookPtr;
114
115            END;
116        END; (*else*)
117    END;
118 END;
119
120 FUNCTION inAuthorList(authorList: AuthorNodePtr; author: STRING): BOOLEAN;
121 VAR
122     curr: AuthorNodePtr;
123     bResult: BOOLEAN;
124 BEGIN
125     bResult := FALSE;
126     curr := authorList;
127     WHILE (curr <> NIL) AND (author <> curr^.name) DO BEGIN
128         curr := curr^.next;
129     END; (*while*)
130

```



```

131     IF (curr <> NIL) THEN BEGIN
132         bResult := TRUE;
133     END;
134
135     inAuthorList := bResult;
136 END;
137
138 FUNCTION NrOfBooksOf(author: STRING): INTEGER;
139     VAR
140         curr: BookNodePtr;
141         bookCount: INTEGER;
142 BEGIN
143     curr := pList;
144     bookCount := 0;
145
146     WHILE (curr <> NIL) DO BEGIN
147         IF inAuthorList(curr^.authors, author) THEN BEGIN
148             inc(bookCount);
149         END;
150         curr := curr^.next;
151     END;
152
153     NrOfBooksOf := bookCount;
154
155 END;
156
157
158 PROCEDURE PrintAuthors(aList: AuthorNodePtr);
159     VAR
160         curr: AuthorNodePtr;
161 BEGIN
162     curr := aList;
163     WHILE (curr <> NIL) DO BEGIN
164         WriteLn(' ', curr^.name);
165         curr := curr^.next;
166     END;
167 END;
168
169
170 PROCEDURE PrintAll;
171     VAR
172         curr: BookNodePtr;
173 BEGIN
174     curr := pList;
175     WHILE (curr <> NIL) DO BEGIN
176         WriteLn(curr^.title, ':');

```

```

177         PrintAuthors(curr^.authors);
178         curr := curr^.next;
179     END;
180 END;
181
182 PROCEDURE DisposeAuthors(aList: AuthorNodePtr);
183     VAR
184         curr, prev: AuthorNodePtr;
185 BEGIN
186     curr := aList;
187     prev := aList;
188     WHILE (curr <> NIL) DO BEGIN
189         prev := curr;
190         curr := curr^.next;
191         Dispose(prev);
192     END;
193 END;
194
195 PROCEDURE DisposeAll;
196     VAR
197         curr: BookNodePtr;
198         prev: BookNodePtr;
199 BEGIN
200     curr := pList;
201     WHILE (curr <> NIL) DO BEGIN
202         prev := curr;
203         curr := curr^.next;
204         DisposeAuthors(prev^.authors);
205         Dispose(prev);
206     END;
207     pList := NIL;
208 END;
209
210 VAR
211     title, author: STRING;
212 BEGIN
213     pList := NIL;
214     title := 'title';
215     author := 'author';
216     WHILE (length(title) > 0) AND (length(author) > 0) DO BEGIN
217         Write('title (Enter to proceed) > ');
218         ReadLn(title);
219         IF (length(title) > 0) THEN BEGIN
220             Write('author (Enter to proceed) > ');
221             ReadLn(author);
222             IF (length(author) > 0) THEN

```

```
223         InsertBook(title, author);
224     END;
225 END;
226
227 PrintAll;
228
229 WriteLn;
230 Write('The number of Books of which author do you want to print (Enter to
exit) > ');
231 ReadLn(author);
232 IF (length(author) > 0) THEN BEGIN
233     WriteLn('Nr. of Books written by ', author, ':', NrofBooksOf(author):2);
234 END;
235 WriteLn;
236
237 DisposeAll;
238 END.
```

3.3 Tests

Beschreibung	Eingabe / Ausgabe
Einfügen von 2 Autoren in ein Buch, selber Author in zwei Büchern	<pre>title (Enter to proceed) > Buch0 author (Enter to proceed) > Klemens title (Enter to proceed) > Buch0 author (Enter to proceed) > Max title (Enter to proceed) > Buch1 author (Enter to proceed) > Max title (Enter to proceed) > Buch2 author (Enter to proceed) > Gustav title (Enter to proceed) > Buch0: Klemens Max Buch1: Max Buch2: Gustav The number of Books of which author do you want to print (Enter to exit) > Max Nr. of Books written by Max: 2</pre>
Nichts eingegeben, kein Fehler	<pre>title (Enter to proceed) > The number of Books of which author do you want to print (Enter to exit) ></pre>
Duplikat eingeben --> sollen nur einmal in der Liste aufgenommen werden	<pre>title (Enter to proceed) > Buch0 author (Enter to proceed) > Gustav title (Enter to proceed) > Buch0 author (Enter to proceed) > Gustav title (Enter to proceed) > Buch1 author (Enter to proceed) > Max title (Enter to proceed) > Buch0: Gustav Buch1: Max</pre>
lex. Sortierung der Bücher	<pre>title (Enter to proceed) > BBuch author (Enter to proceed) > Gustav title (Enter to proceed) > CBuch author (Enter to proceed) > Max title (Enter to proceed) > ABuch author (Enter to proceed) > Gustav title (Enter to proceed) > ABuch: Gustav BBuch: Gustav CBuch: Max</pre>

Beschreibung	Eingabe / Ausgabe
lex. Sortierung der Autoren	<pre> title (Enter to proceed) > Buch author (Enter to proceed) > Max title (Enter to proceed) > Buch author (Enter to proceed) > Dorothea title (Enter to proceed) > Buch author (Enter to proceed) > Annelie title (Enter to proceed) > Buch author (Enter to proceed) > Berta title (Enter to proceed) > Buch: Annelie Berta Dorothea Max </pre>