**ADF2x & PRO2x** **Übungen zu Fortgeschrittenen Algorithmen & Datenstrukturen und OOP** **SS 24, Übung 1**

**Abgabetermin: Sa, 23.03.2024**

☒ **Gr. 1,** **S. Schöberl, MSc** **Name** Elias Leonhardsberger **Aufwand in h** 9

☐ **Gr. 2,** **DI (FH) G. Horn-Völlenkle, MSc**

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

## 1. Worthäufigkeiten mit Hash-Tabellen (12 + 12 Punkte)

Entwickeln Sie ein Pascal-Programm *WordCounter*, das für eine Textdatei die Häufigkeiten der darin vorkommenden Wörter ermittelt und folgende Ergebnisse ausgibt: (1) die Anzahl aller vorkommenden Wörter, (2) die Anzahl der Wörter, die öfter als einmal vorkommen und (3) das am häufigsten vorkommende Wort mit seiner Häufigkeit. Zwischen Groß- und Kleinschreibung ist bei dieser Aufgabe nicht zu unterscheiden.

Zur Verwaltung der Wörter und ihrer Häufigkeiten verwenden Sie:

a) eine *Hash-Tabelle* mit Kollisionsbehandlungs-Strategie *Verkettung* (engl. *chaining*) und

b) eine *Hash-Tabelle* mit Kollisionsbehandlungs-Strategie *offene Adressierung*, z.B. *lineare* oder *quadratische* Kollisionsbehandlung bzw. *doppeltes Hashing*.

Hinweis für Ergebnis (2): Ermitteln Sie die Anzahl der Wörter, die öfter als einmal vorkommen, indem sie alle Wörter mit Häufigkeit 1 aus der Hash-Tabelle entfernen und danach alle verbleibenden Wörter zählen.

Untersuchen Sie für beide Varianten die Laufzeiteffizienz (mittels *Timer.pas*) und diskutieren Sie die Vor- und Nachteile der beiden Varianten.

Da das Thema Dateibearbeitung noch nicht behandelt wurde, finden Sie im Moodle-Kurs in der Datei *WordCounter.zip* mit *WordReader.pas* ein Modul, das eine einfache Schnittstelle zum Lesen von Wörtern aus Textdateien zur Verfügung stellt und mit *WordCounter.pas* eine Vorlage für die von Ihnen zu erstellenden Programmversionen, je eine für a) und b).

Testen Sie Ihre Programme mit kleineren und größeren Textdateien ausführlich.

### Hinweise:

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

# 1. Worthäufigkeiten mit Hash-Tabellen

## Lösungsidee

Zuerst muss eine string GetHashCode Funktion in einem Unit erstellt werden, diese rechnet einen Hashcode aus, indem die Summe jedes Ordinalwertes mal 31 hoch der Position gerechnet wird. Dieser Hasing Algorithmus wurde in der Übung kurz als die Lösung, die Java verwendet, angeschnitten.

Für die zwei verschiedenen Hash-Tabellen Arten werden 2 verschiedene Units erstellt (ChainedWordCounter und OpenAdressedWordCounter).

Das Hinzufügen der Wörter und die Kollisionsvermeidung wird gleich der Übung gemacht, wobei verschiedene Hashtabellengrößen getestet werden (die offen Adressierte Variante hat eine Untergrenze an gültiger Größe. Für die Testfälle wird eine Tabelle mit Zeiten erstellt um den Einfluss der Größe zur Laufzeit zu zeigen.

Um die mehrfach vorkommenden Wörter zu zählen müssen zu jedem Wort die Anzahl an Vorkommnisse gespeichert werden. Dadurch kann später über alle Worte itteriert werden, um alle einmal vorkommenden Worte zu finden, die überbliebenen Worte zu Zählen und das Maximum zu finden.

Timer.pas wurde nicht verändert, in WordReader.pas wurde WinCrt entfernt, um das Unit Linux kompatibel zu machen, und der Word Datentyp wurde auf WordString geändert um Namenskonflikte mit dem Pascal Datentyp Word zu verhindern.

Für die spätere Auswertung der Tests kann man jeden WordCounter in einem HumanReadable und in einem CSV Format aufrufen.

Per KI Richtlinie gebe ich bekannt CoPilot zu verwenden, wobei es zur Formatierung und als Autocomplete genutzt wurde.

# Code

## WordCounterTest.pas

```
1   PROGRAM WordCounterTest;
2
3   USES ChainedWordCounter, OpenAddressedWordCounter;
4
5   TYPE
6     WordCounter = PROCEDURE (path : STRING; printHumanReadable: BOOLEAN);
7
8   PROCEDURE TestWordCounter(name: STRING; printHumanReadable: BOOLEAN; counter: WordCounter);
9   BEGIN (*TestWordCounter*)
10    WriteLn('Testing ', name);
11    counter('../TestFiles/empty.txt', printHumanReadable);
12    counter('../TestFiles/se.txt', printHumanReadable);
13    counter('../TestFiles/metamorphosis.txt', printHumanReadable);
14    counter('../TestFiles/verwandlung.txt', printHumanReadable);
15    counter('../TestFiles/mobyDick.txt', printHumanReadable);
16    counter('../TestFiles/romeoAndJuliet.txt', printHumanReadable);
17    counter('../TestFiles/bible.txt', printHumanReadable);
18    counter('../TestFiles/loremIpsum.txt', printHumanReadable);
19    counter('../TestFiles/a.txt', printHumanReadable);
20    counter('../TestFiles/b.txt', printHumanReadable);
21    counter('../TestFiles/space.txt', printHumanReadable);
22  END;(*TestWordCounter*)
23
24  BEGIN (*WordCounter*)
25    TestWordCounter('ChainedWordCounter', TRUE, StartChainedWordCounter);
26    TestWordCounter('OpenAddressedWordCounter', TRUE, StartOpenAddressedWordCounter);
27    TestWordCounter('ChainedWordCounter', FALSE, StartChainedWordCounter);
28    TestWordCounter('ChainedWordCounter', FALSE, StartChainedWordCounter);
29    TestWordCounter('ChainedWordCounter', FALSE, StartChainedWordCounter);
30    TestWordCounter('OpenAddressedWordCounter', FALSE, StartOpenAddressedWordCounter);
31    TestWordCounter('OpenAddressedWordCounter', FALSE, StartOpenAddressedWordCounter);
32    TestWordCounter('OpenAddressedWordCounter', FALSE, StartOpenAddressedWordCounter);
33  END. (*WordCounter*)
```

**ChainedWordCounter.pas**

```
1
2   UNIT ChainedWordCounter;
3
4   INTERFACE
5
6   PROCEDURE StartChainedWordCounter(filePath: STRING; printHumanReadable: BOOLEAN);
7
8   IMPLEMENTATION
9
10  USES Crt, Timer, WordReader, UStringHash;
11
12  CONST
13    HASH_TABLE_SIZE = 10000;
14
15  TYPE
16    NodePtr = ^Node;
17    Node = RECORD
18      word: WordString;
19      count: Longword;
20      next: NodePtr;
21    END;
22    Hash = 0..HASH_TABLE_SIZE-1;
23    HashTable = ARRAY [Hash] OF NodePtr;
24
25  PROCEDURE InitHashTable(VAR ht: HashTable);
26  VAR
27    i: Hash;
28  BEGIN (*InitHashTable*)
29    FOR i := low(Hash) TO High(Hash) DO
30      BEGIN (*FOR*)
31        ht[i] := NIL;
32      END; (*FOR*)
33  END; (*InitHashTable*)
34
35  PROCEDURE DisposeHashTable(ht: HashTable);
```

```
35  PROCEDURE DisposeHashTable(ht: HashTable);
36  VAR
37    i: Hash;
38    next: NodePtr;
39  BEGIN (*DisposeHashTable*)
40    FOR i := low(Hash) TO High(Hash) DO
41      BEGIN (*FOR*)
42        WHILE ht[i] <> NIL DO
43          BEGIN (*WHILE*)
44            next := ht[i]^.next;
45            dispose(ht[i]);
46            ht[i] := next;
47          END; (*WHILE*)
48      END; (*FOR*)
49  END; (*DisposeHashTable*)
50
51  PROCEDURE AddWord(VAR ht: HashTable; word: WordString; VAR ok: BOOLEAN);
```

```
51  PROCEDURE AddWord(VAR ht: HashTable; word: WordString; VAR ok: BOOLEAN);
52  VAR
53    h: Hash;
54    curr, prev: NodePtr;
55  BEGIN (*AddWord*)
56    ok := FALSE;
57    h := GetHashCode(word, HASH_TABLE_SIZE);
58    prev := NIL;
59    curr := ht[h];
60
61    WHILE (curr<>NIL) AND (curr^.word <> word) DO
62      BEGIN (*WHILE*)
63        prev := curr;
64        curr := curr^.next;
65      END; (*WHILE*)
66
67    IF (curr = NIL) THEN
68      BEGIN (*IF*)
69        new(curr);
70
71        IF (curr <> NIL) THEN
72          BEGIN (*IF*)
73            curr^.word := word;
74            curr^.count := 1;
75            curr^.next := NIL;
76
77            IF (prev = NIL) THEN
78              BEGIN (*IF*)
79                ht[h] := curr;
80              END (*IF*)
81            ELSE
82              BEGIN (*ELSE*)
83                prev^.next := curr;
84              END; (*ELSE*)
85
86            ok := TRUE;
87          END; (*IF*)
88      END (*IF*)
89    ELSE
90      BEGIN (*ELSE*)
91        curr^.count := curr^.count + 1;
92        ok := TRUE;
93      END; (*ELSE*)
94  END; (*AddWord*)
95
96  PROCEDURE RemoveUniqueWords(VAR ht: HashTable);
```

```
96  PROCEDURE RemoveUniqueWords(VAR ht: HashTable);
97  VAR
98    i: Hash;
99    curr, prev: NodePtr;
100 BEGIN (*RemoveUniqueWords*)
101   FOR i := low(hash) TO high(hash) DO
102     BEGIN (*FOR*)
103       curr := ht[i];
104       prev := NIL;
105
106       WHILE (curr <> NIL) DO
107         BEGIN (*WHILE*)
108           IF (curr^.count = 1) THEN
109             BEGIN (*IF*)
110               IF (prev <> NIL) THEN
111                 BEGIN (*IF*)
112                   prev^.next := curr^.next;
113                   dispose(curr);
114                   curr := prev^.next;
115                 END (*IF*)
116               ELSE
117                 BEGIN (*ELSE*)
118                   ht[i] := curr^.next;
119                   dispose(curr);
120                   curr := ht[i];
121                 END; (*ELSE*)
122             END; (*IF*)
123         END; (*WHILE*)
124     END; (*FOR*)
125 END; (*RemoveUniqueWords*)
126
127 PROCEDURE GetAggregate(ht: HashTable; VAR moreThanOneCount, maxCount : Longword; VAR maxWord : WordString);
```

```
127  PROCEDURE GetAggregate(ht: HashTable; VAR moreThanOneCount, maxCount : Longword; VAR maxWord : WordString);
128  VAR
129    i: Hash;
130    curr: NodePtr;
131  BEGIN (*GetAggregate*)
132    moreThanOneCount := 0;
133    maxCount := 0;
134    maxWord := '';
135
136    FOR i := low(hash) TO high(hash) DO
137      BEGIN (*FOR*)
138        curr := ht[i];
139
140        WHILE (curr <> NIL) DO
141          BEGIN (*WHILE*)
142            moreThanOneCount := moreThanOneCount +1;
143
144            IF (curr^.count > maxCount) THEN
145              BEGIN (*IF*)
146                maxCount := curr^.count;
147                maxWord := curr^.word;
148              END; (*IF*)
149
150            curr := curr^.next;
151          END; (*WHILE*)
152      END; (*FOR*)
153  END; (*GetAggregate*)
154
155  PROCEDURE PrintResult(printHumanReadable: BOOLEAN;
156                        totalCount, moreThanOneCount, maxCount: Longword;
157                        maxWord: WordString;
158                        ElapsedTime: STRING);
159  BEGIN (*PrintResult*)
160    IF (printHumanReadable)THEN
161      BEGIN (*IF*)
162        WriteLn('Number of words: ', totalCount);
163        WriteLn('Number of words with more than one count: ', moreThanOneCount);
164        WriteLn('Word with the most counts: "', maxWord, '" ', maxCount, ' times');
165        WriteLn('Elapsed time: ', ElapsedTime);
166        WriteLn();
167      END (*IF*)
168    ELSE
169      BEGIN (*ELSE*)
170        WriteLn(totalCount, ';', moreThanOneCount, ';', maxCount, ';', maxWord, ';', ElapsedTime);
171      END; (*ELSE*)
```

```
174  PROCEDURE StartChainedWordCounter(filePath: STRING; printHumanReadable: BOOLEAN);
175  VAR
176    ht: HashTable;
177    ok: BOOLEAN;
178    totalCount, moreThanOneCount, maxCount: Longword;
179    maxWord, w: WordString;
180  BEGIN (*StartChainedWordCounter*)
181    IF (printHumanReadable) THEN
182      BEGIN (*IF*)
183        writeln('Path: ', filePath);
184      END (*IF*)
185    ELSE
186      BEGIN (*ELSE*)
187        write(filePath, ';')
188      END; (*ELSE*)
189
190    InitHashTable(ht);
191    OpenFile(filePath, toLower);
192    totalCount := 0;
193
194    StartTimer();
195    ReadWord(w);
196
197    WHILE (w <> '') DO
198      BEGIN (*WHILE*)
199        totalCount := totalCount + 1;
200        AddWord(ht, w, ok);
201
202        IF (NOT ok) THEN
203          BEGIN (*IF*)
204            WriteLn('There was an issue adding word "', w, '" to the hash table.');
205          END; (*IF*)
206
207        ReadWord(w);
208      END; (*WHILE*)
209
210    GetAggregate(ht, moreThanOneCount, maxCount, maxWord);
211    StopTimer();
212    CloseFile();
213
214    PrintResult(printHumanReadable, totalCount, moreThanOneCount, maxCount, maxWord, ElapsedTime);
215
216    DisposeHashTable(ht);
217  END; (*StartChainedWordCounter*)
218
219  BEGIN
220  END.
```

**OpenAddressedWordCounter.pas**

```
1
2   UNIT OpenAddressedWordCounter;
3
4   INTERFACE
5
6   PROCEDURE StartOpenAddressedWordCounter(filePath: STRING; printHumanReadable: BOOLEAN);
7
8   IMPLEMENTATION
9
10  USES Crt, Timer, WordReader, UStringHash;
11
12  CONST
13    HASH_TABLE_SIZE = 1000000;
14
15  TYPE
16    NodePtr = ^Node;
17    Node = RECORD
18      word: WordString;
19      count: Longword;
20      deleted: BOOLEAN;
21    END;
22    Hash = 0..HASH_TABLE_SIZE-1;
23    HashTable = ARRAY [Hash] OF NodePtr;
24
25  PROCEDURE InitHashTable(VAR ht: HashTable);
26  VAR
27    i: Hash;
28  BEGIN (*InitHashTable*)
29    FOR i := low(Hash) TO High(Hash) DO
30      BEGIN (*FOR*)
31        ht[i] := NIL;
32      END; (*FOR*)
33  END; (*InitHashTable*)
34
35  PROCEDURE DisposeHashTable(ht: HashTable);
36  VAR
37    i: Hash;
38  BEGIN (*DisposeHashTable*)
39    FOR i := low(Hash) TO High(Hash) DO
40      BEGIN (*FOR*)
41        IF ht[i] <> NIL THEN
42          BEGIN (*IF*)
43            dispose(ht[i]);
44            ht[i] := NIL;
45          END; (*IF*)
46      END; (*FOR*)
47  END; (*DisposeHashTable*)
48
49  PROCEDURE AddWord(VAR ht: HashTable; word: WordString; VAR ok: BOOLEAN);
```

```
49  PROCEDURE AddWord(VAR ht: HashTable; word: WordString; VAR ok: BOOLEAN);
50  VAR
51    h: Hash;
52    steps: Longword;
53  BEGIN (*AddWord*)
54    ok := true;
55    h := GetHashCode(word, HASH_TABLE_SIZE);
56    steps := 0;
57
58    WHILE (steps < HASH_TABLE_SIZE) AND (ht[h]<>NIL) AND (ht[h]^.word <> word) AND (NOT ht[h]^.deleted) DO
59      BEGIN (*WHILE*)
60        h := (h + 1) MOD HASH_TABLE_SIZE;
61        steps := steps + 1;
62      END; (*WHILE*)
63
64    IF (steps >= HASH_TABLE_SIZE) THEN
65      BEGIN (*IF*)
66        ok := FALSE;
67      END (*IF*)
68    ELSE
69      IF (ht[h] = NIL) THEN
70        BEGIN (*ELSE IF*)
71          new(ht[h]);
72
73          IF (ht[h] <> NIL) THEN
74            BEGIN (*IF*)
75              ht[h]^.word := word;
76              ht[h]^.count := 1;
77              ht[h]^.deleted := FALSE;
78            END (*IF*)
79          ELSE
80            BEGIN (*ELSE*)
81              ok := FALSE;
82            END; (*ELSE*)
83        END (*ELSE IF*)
84    ELSE
85      BEGIN (*ELSE*)
86        IF (ht[h]^.deleted) THEN
87          BEGIN (*IF*)
88            ht[h]^.count :=  1;
89            ht[h]^.word := word;
90            ht[h]^.deleted := FALSE;
91          END (*IF*)
92        ELSE
93          BEGIN (*ELSE*)
94            ht[h]^.count := ht[h]^.count + 1;
95          END; (*ELSE*)
96      END; (*ELSE*)
97  END; (*AddWord*)
```

```
99  PROCEDURE RemoveUniqueWords(VAR ht: HashTable);
100 VAR
101   i: Hash;
102 BEGIN (*RemoveUniqueWords*)
103   FOR i := low(hash) TO high(hash) DO
104     BEGIN (*FOR*)
105       IF ((ht[i]<> NIL) AND (NOT ht[i]^.deleted) AND (ht[i]^.count = 1)) THEN
106         BEGIN (*IF*)
107           ht[i]^.deleted := TRUE;
108         END; (*IF*)
109     END; (*FOR*)
110 END; (*RemoveUniqueWords*)
111
112 PROCEDURE GetAggregate(ht: HashTable; VAR moreThanOneCount, maxCount : Longword; VAR maxWord : WordString);
113 VAR
114   i: Hash;
115 BEGIN (*GetAggregate*)
116   moreThanOneCount := 0;
117   maxCount := 0;
118   maxWord := '';
119
120   FOR i := low(hash) TO high(hash) DO
121     BEGIN (*FOR*)
122       IF ((ht[i]<> NIL) AND (NOT ht[i]^.deleted)) THEN
123         BEGIN (*IF*)
124           moreThanOneCount := moreThanOneCount +1;
125
126           IF (ht[i]^.count > maxCount) THEN
127             BEGIN (*IF*)
128               maxCount := ht[i]^.count;
129               maxWord := ht[i]^.word;
130             END; (*IF*)
131         END; (*IF*)
132     END; (*FOR*)
133 END; (*GetAggregate*)
134
135 PROCEDURE PrintResult(printHumanReadable: BOOLEAN;
```

```
135 PROCEDURE PrintResult(printHumanReadable: BOOLEAN;
136                       totalCount, moreThanOneCount, maxCount: Longword;
137                       maxWord: WordString;
138                       ElapsedTime: STRING);
139 BEGIN (*PrintResult*)
140   IF (printHumanReadable) THEN
141     BEGIN (*IF*)
142       WriteLn('Number of words: ', totalCount);
143       WriteLn('Number of words with more than one count: ', moreThanOneCount);
144       WriteLn('Word with the most counts: "', maxWord, '" ', maxCount, ' times');
145       WriteLn('Elapsed time: ', ElapsedTime);
146       WriteLn();
147     END (*IF*)
148   ELSE
149     BEGIN (*ELSE*)
150       WriteLn(totalCount, ';', moreThanOneCount, ';', maxCount, ';', maxWord, ';', ElapsedTime);
151     END; (*ELSE*)
152 END; (*PrintResult*)
153
154 PROCEDURE StartOpenAddressedWordCounter(filePath: STRING; printHumanReadable: BOOLEAN);
```

```
154  PROCEDURE StartOpenAddressedWordCounter(filePath: STRING; printHumanReadable: BOOLEAN);
155  VAR
156    ht: HashTable;
157    ok: BOOLEAN;
158    totalCount, moreThanOneCount, maxCount: Longword;
159    maxWord, w: WordString;
160  BEGIN (*StartChainedWordCounter*)
161    IF (printHumanReadable) THEN
162      BEGIN (*IF*)
163        writeln('Path: ', filePath);
164      END (*IF*)
165    ELSE
166      BEGIN (*ELSE*)
167        write(filePath, ';')
168      END; (*ELSE*)
169
170    InitHashTable(ht);
171    OpenFile(filePath, toLower);
172    totalCount := 0;
173
174    StartTimer();
175    ReadWord(w);
176
177    WHILE (w <> '') DO
178      BEGIN (*WHILE*)
179        totalCount := totalCount + 1;
180        AddWord(ht, w, ok);
181
182        IF (NOT ok) THEN
183          BEGIN (*IF*)
184            WriteLn('There was an issue adding word "', w, '" to the hash table.');
185          END; (*IF*)
186
187        ReadWord(w);
188      END; (*WHILE*)
189
190    GetAggregate(ht, moreThanOneCount, maxCount, maxWord);
191    StopTimer();
192    CloseFile();
193
194    PrintResult(printHumanReadable, totalCount, moreThanOneCount, maxCount, maxWord, ElapsedTime);
195
196    DisposeHashTable(ht);
197  END; (*StartChainedWordCounter*)
198
199  BEGIN
200  END.
```

**Timer.pas**(unverändert)

```
(* Timer:                                              HDO, 2005-04-01
   -----
   Simple utility for run-time measurement.
========================================================================*)

UNIT Timer;

INTERFACE

PROCEDURE StartTimer;
PROCEDURE StopTimer;

FUNCTION TimerIsRunning: BOOLEAN;

FUNCTION Elapsed:      STRING;    (*same as ElapsedTime*)
FUNCTION ElapsedTime:  STRING;    (*time format: mm:ss.th*)
FUNCTION ElapsedSecs:  STRING;    (*secs format: sssss.th*)
FUNCTION ElapsedTicks: LONGINT;   (*1 tick = 10 ms*)


IMPLEMENTATION

USES
    {$IFDEF FPC}
Dos;    (*for targets 'real mode' and 'protected mode'*)
    {$ELSE}
      {$IFDEF MSDOS}
Dos;    (*for targets 'real mode' and 'protected mode'*)
      {$ENDIF}
      {$IFDEF WINDOWS}
WinDos; (*for target 'windows'*)
      {$ENDIF}
    {$ENDIF}


VAR
  running: BOOLEAN;
  startedAt, stoppedAt, ticksElapsed: LONGINT;


PROCEDURE Assert(cond: BOOLEAN; msg: STRING);
```

```
41  PROCEDURE Assert(cond: BOOLEAN; msg: STRING);
42  BEGIN
43    IF NOT cond THEN
44      BEGIN
45        WriteLn('ERROR : ', msg);
46        HALT;
47      END; (*IF*)
48  END; (*Assert*)
49
50  PROCEDURE GetTicks(VAR ticks: LONGINT);
51  VAR
52    h, m, s, hs: WORD;
53  BEGIN
54    GetTime(h, m, s, hs);
55    ticks := h     * 60 +  m;
56    ticks := ticks * 60 +  s;
57    ticks := ticks * 100 + hs;
58  END; (*GetTicks*)
59
60  FUNCTION DigitFor(n: INTEGER): CHAR;
61  BEGIN
62    Assert((n >= 0) AND (n <= 9), 'invalid digit value in DigitFor');
63    DigitFor := CHR(ORD('0') + n);
64  END; (*DigitFor*)
65
66
67    (* StartTimer: start the timer
68    ----------------------------------------------------------------*)
69  PROCEDURE StartTimer;
70  BEGIN
71    Assert(NOT running, 'StartTimer called while timer is running');
72    GetTicks(startedAt);
73    ticksElapsed := 0;
74    running := TRUE;
75  END; (*StartTimer*)
76
77
78    (* StopTimer: stop the timer
79    ----------------------------------------------------------------*)
80  PROCEDURE StopTimer;
```

```
80  PROCEDURE StopTimer;
81  BEGIN
82    Assert(running, 'StopTimer called when timer not running');
83    GetTicks(stoppedAt);
84    ticksElapsed := stoppedAt - startedAt;
85    running := FALSE;
86  END; (*StopTimer*)
87
88
89    (* TimerIsrunning: is the timer running?
90    ----------------------------------------------------------------*)
91  FUNCTION TimerIsRunning: BOOLEAN;
92  BEGIN
93    TimerIsRunning := running;
94  END; (*TimerIsRunning*)
95
96
97    (* Elapsed: same ElapsedTime
98    ----------------------------------------------------------------*)
99  FUNCTION Elapsed: STRING;
100 BEGIN
101   Elapsed := ElapsedTime;
102 END; (*ElapsedTime*)
103
104
105   (* ElapsedTime: return elapsed time in format "mm:ss.th"
106   ----------------------------------------------------------------*)
107 FUNCTION ElapsedTime: STRING;
```

```
105   (* ElapsedTime: return elapsed time in format "mm:ss.th"
106   ------------------------------------------------------------*)
107 FUNCTION ElapsedTime: STRING;
108 VAR
109   ticks: LONGINT;
110   m, s, t, h: INTEGER;
111   timeStr: STRING[8];
112 BEGIN
113   Assert(NOT running, 'ElapsedTime called while timer is running');
114   ticks  := ticksElapsed;
115   h      := ticks MOD 10;
116   ticks  := ticks DIV 10;
117   t      := ticks MOD 10;
118   s      := ticks DIV 10;
119   m      := s     DIV 60;
120   s      := s     MOD 60;
121                  (*12345678*)
122   timeStr := 'mm:ss.th';
123   timeStr[1] := DigitFor(m DIV 10);
124   timeStr[2] := DigitFor(m MOD 10);
125   timeStr[4] := DigitFor(s DIV 10);
126   timeStr[5] := DigitFor(s MOD 10);
127   timeStr[7] := DigitFor(t);
128   timeStr[8] := DigitFor(h);
129   ElapsedTime := timeStr;
130 END; (*ElapsedTime*)
131
132
133   (* ElapsedSecs: return elapsed seconds in format "sssss.th"
134   ------------------------------------------------------------*)
135 FUNCTION ElapsedSecs: STRING;
```

```
133     (* ElapsedSecs: return elapsed seconds in format "ssssss.th"
134     -------------------------------------------------------------*)
135   FUNCTION ElapsedSecs: STRING;
136   VAR
137     ticks: REAL;
138     secsStr: STRING[8];
139   BEGIN
140     Assert(NOT running, 'ElapsedSecs called while timer is running');
141     ticks := elapsedTicks;
142     Str((ticks / 100.0): 8: 2, secsStr);
143     ElapsedSecs := secsStr;
144   END; (*ElapsedSecs*)
145
146
147     (* ElapsedTicks: return elapsed time in ticks
148     -------------------------------------------------------------*)
149   FUNCTION ElapsedTicks: LONGINT;
150   BEGIN
151     Assert(NOT running, 'ElapsedTicks called while timer is running');
152     ElapsedTicks := ticksElapsed;
153   END; (*ElapsedTicks*)
154
155
156   BEGIN (*Timer*)
157
158     running := FALSE;
159
160   END. (*Timer*)
```

**UStringHash.pas**

```
1
2   UNIT UStringHash;
3
4   INTERFACE
5
6   FUNCTION ModPower(base, exponent, modulus: Longword): Longword;
7
8   FUNCTION GetHashCode(word:STRING; modulus: Longword): Longword;
9
10  IMPLEMENTATION
11
12  FUNCTION ModPower(base, exponent, modulus: Longword): Longword;
13  VAR
14    result: Longword;
15  BEGIN
16    result := 1;
17    WHILE exponent > 0 DO
18      BEGIN
19        result := (result * base) MOD modulus;
20        exponent := exponent - 1;
21      END;
22    ModPower := result;
23  END;
24
25  FUNCTION GetHashCode(word:STRING; modulus: Longword): Longword;
26  VAR i, result: Longword;
27  BEGIN
28    result := 0;
29    FOR i := 1 TO Length(word) DO
30      BEGIN
31        result := (result + ((ord(word[i])* ModPower(31, i, modulus)) MOD modulus)) MOD modulus;
32      END;
33    GetHashCode := result;
34  END;
35
36  END.
```

**WordReader.pas(kleine Änderungen)**

```
1  (* WordReader:                                    HDO, 03-02-27 *)
2  (* ----------                                                   *)
3  (* Reads a text file word (= seq. of characters) by word.       *)
4  (*=============================================================*)
5
6  UNIT WordReader;
7
8  INTERFACE
9
10 CONST
11   maxWordLen = 20;
12
13 TYPE
14   Conversion = (noConversion, toLower, toUpper);
15   WordString = STRING[maxWordLen]; (*to save memory, longer words are stripped*)
16
17 PROCEDURE OpenFile(fileName: STRING; c: Conversion);
18 PROCEDURE ReadWord(VAR w: WordString); (*w = '' on end of file*)
19 PROCEDURE CloseFile;
20
21 IMPLEMENTATION
22
23 CONST
24   characters = ['a' .. 'z', 'ä', 'ö', 'ü', 'ß',
25                 'A' .. 'Z', 'Ä', 'Ö', 'Ü'];
26   EF = CHR(0);         (*end of file character*)
27
28 VAR
29   txt: TEXT;           (*text file*)
30   open: BOOLEAN;       (*file opened?*)
31   line: STRING;        (*current line*)
32   ch: CHAR;            (*current character*)
33   cnr: INTEGER;        (*column number of current character*)
34   conv: Conversion;    (*kind of conversion*)
35
36
37 PROCEDURE ConvertToLower(VAR w: WordString);
```

```
37  PROCEDURE ConvertToLower(VAR w: WordString);
38  VAR
39    i: INTEGER;
40  BEGIN
41    FOR i := 1 TO Length(w) DO
42      BEGIN
43        CASE w[i] OF
44          'A'..'Z': w[i] := CHR(ORD(w[i]) + 32) ;
45          'Ä':      w[i] := 'ä';
46          'Ö':      w[i] := 'ö';
47          'Ü':      w[i] := 'ü';
48        END; (*CASE*)
49      END; (*FOR*)
50  END; (*ConvertToLower*)
51
52  PROCEDURE ConvertToUpper(VAR w: WordString);
53  VAR
54    i: INTEGER;
55  BEGIN
56    FOR i := 1 TO Length(w) DO
57      BEGIN
58        CASE w[i] OF
59          'a'..'z': w[i] := UpCase(w[i]);
60          'ä':      w[i] := 'Ä';
61          'ö':      w[i] := 'Ö';
62          'ü':      w[i] := 'Ü';
63        END; (*CASE*)
64      END; (*FOR*)
65  END; (*ConvertToUpper*)
66
67  PROCEDURE NextChar;
```

```
67  PROCEDURE NextChar;
68  BEGIN
69    IF cnr < Length(line) THEN
70      BEGIN
71        cnr := cnr + 1;
72        ch := line[cnr]
73      END (*THEN*)
74    ELSE
75      BEGIN
76        IF NOT Eof(txt) THEN
77          BEGIN
78            ReadLn(txt, line);
79            cnr := 0;
80            ch := ' '; (*separate lines by ' '*)
81          END (*THEN*)
82        ELSE
83          ch := EF;
84      END; (*ELSE*)
85  END; (*NextChar*)
86
87
88    (* OpenFile: opens text file named fileName                    *)
89    (*-------------------------------------------------------------*)
90  PROCEDURE OpenFile(fileName: STRING; c: Conversion);
91  BEGIN
92    IF open THEN
93      CloseFile;
94    Assign(txt, fileName);
95      (*$I-*)
96    Reset(txt);
97      (*$I+*)
98    IF IOResult <> 0 THEN
99      BEGIN
100       WriteLn('ERROR in WordReader.OpenFile: file ', fileName, ' not found');
101       HALT;
102     END; (*IF*)
103   open := TRUE;
104   conv := c;
105   line := '';
106   cnr := 1; (*1 >= Length('') => force reading of first line*)
107   NextChar;
108 END; (*OpenFile*)
109
110
111   (* NextWord: reads next word from file, returns '' on endfile  *)
112   (*-------------------------------------------------------------*)
113 PROCEDURE ReadWord(VAR w: WordString);
```

```
111   (* NextWord: reads next word from file, returns '' on endfile  *)
112   (*-------------------------------------------------------------*)
113 PROCEDURE ReadWord(VAR w: WordString);
114 VAR
115   i: INTEGER;
116 BEGIN
117   w := '';
118   WHILE (ch <> EF) AND NOT (ch IN characters) DO
119     BEGIN
120       NextChar;
121     END; (*WHILE*)
122   IF ch = EF THEN
123     EXIT;
124   i := 0;
125   REPEAT
126     IF i < maxWordLen THEN
127       BEGIN
128         i := i + 1;
129         (*$R-*)
130         w[i] := ch;
131         (*$R+*)
132       END; (*IF*)
133     NextChar;
134   UNTIL (ch = EF) OR NOT (ch IN characters);
135   w[0] := Chr(i);
136   CASE conv OF
137     toUpper: ConvertToUpper(w);
138     toLower: ConvertToLower(w);
139   END; (*CASE*)
140 END; (*ReadWord*)
141
142
143   (* CloseFile: closes text file                                 *)
144   (*-------------------------------------------------------------*)
145 PROCEDURE CloseFile;
```

```
143   (* CloseFile: closes text file                                 *)
144   (*-------------------------------------------------------------*)
145 PROCEDURE CloseFile;
146 BEGIN
147   IF open THEN
148     BEGIN
149       Close(txt);
150       open := FALSE;
151     END; (*IF*)
152 END; (*CloseFile*)
153
154
155 BEGIN (*WordReader*)
156   open := FALSE;
157 END. (*WordReader*)
```

## Testfälle

Um verschieden große Texte zu testen wird Project Gutenberg(https://www.gutenberg.org/) genutzt. Dadurch ergeben sich Testfälle für:

- Eine leere Datei
- Die Datei aus der Übung(se.txt)
- Mehrere Bücher verschiedener Größe(24030 Wörter – 794780 Wörter)
- Lorem Ipsum
- Eine Datei mit nur einem Zeichen ohne Leerzeichen(a.txt)
- Eine Datei mir nur einem Zeichen und Leerzeichen dazwischen(b.txt)
- Eine Datei nur aus Leerzeichen

**Konsolenausgabe der gesamten Testfälle:**

```
Testing ChainedWordCounter
Path: ../TestFiles/empty.txt
Number of words: 0
Number of words with more than one count: 0
Word with the most counts: "" 0 times
Elapsed time: 00:00.00

Path: ../TestFiles/se.txt
Number of words: 142
Number of words with more than one count: 110
Word with the most counts: "und" 7 times
Elapsed time: 00:00.00

Path: ../TestFiles/metamorphosis.txt
Number of words: 25501
Number of words with more than one count: 2972
Word with the most counts: "the" 1348 times
Elapsed time: 00:00.03

Path: ../TestFiles/verwandlung.txt
Number of words: 24030
Number of words with more than one count: 4367
Word with the most counts: "die" 631 times
Elapsed time: 00:00.02

Path: ../TestFiles/mobyDick.txt
Number of words: 222094
Number of words with more than one count: 17131
Word with the most counts: "the" 14727 times
Elapsed time: 00:00.09

Path: ../TestFiles/romeoAndJuliet.txt
Number of words: 29909
Number of words with more than one count: 3995
Word with the most counts: "the" 878 times
Elapsed time: 00:00.01

Path: ../TestFiles/bible.txt
Number of words: 794780
Number of words with more than one count: 12877
Word with the most counts: "the" 64117 times
Elapsed time: 00:00.29

Path: ../TestFiles/loremIpsum.txt
```

```
Path: ../TestFiles/loremIpsum.txt
Number of words: 41344
Number of words with more than one count: 117
Word with the most counts: "et" 2128 times
Elapsed time: 00:00.02

Path: ../TestFiles/a.txt
Number of words: 1
Number of words with more than one count: 1
Word with the most counts: "aaaaaaaaaaaaaaaaaaaa" 1 times
Elapsed time: 00:00.00

Path: ../TestFiles/b.txt
Number of words: 128
Number of words with more than one count: 1
Word with the most counts: "b" 128 times
Elapsed time: 00:00.00

Path: ../TestFiles/space.txt
Number of words: 0
Number of words with more than one count: 0
Word with the most counts: "" 0 times
Elapsed time: 00:00.00

Testing OpenAddressedWordCounter
Path: ../TestFiles/empty.txt
Number of words: 0
Number of words with more than one count: 0
Word with the most counts: "" 0 times
Elapsed time: 00:00.01

Path: ../TestFiles/se.txt
Number of words: 142
Number of words with more than one count: 110
Word with the most counts: "und" 7 times
Elapsed time: 00:00.01

Path: ../TestFiles/metamorphosis.txt
Number of words: 25501
Number of words with more than one count: 2972
Word with the most counts: "the" 1348 times
Elapsed time: 00:00.01

Path: ../TestFiles/verwandlung.txt
```

```
Path: ../TestFiles/verwandlung.txt
Number of words: 24030
Number of words with more than one count: 4367
Word with the most counts: "die" 631 times
Elapsed time: 00:00.01

Path: ../TestFiles/mobyDick.txt
Number of words: 222094
Number of words with more than one count: 17131
Word with the most counts: "the" 14727 times
Elapsed time: 00:00.09

Path: ../TestFiles/romeoAndJuliet.txt
Number of words: 29909
Number of words with more than one count: 3995
Word with the most counts: "the" 878 times
Elapsed time: 00:00.01

Path: ../TestFiles/bible.txt
Number of words: 794780
Number of words with more than one count: 12877
Word with the most counts: "the" 64117 times
Elapsed time: 00:00.29

Path: ../TestFiles/loremIpsum.txt
Number of words: 41344
Number of words with more than one count: 117
Word with the most counts: "et" 2128 times
Elapsed time: 00:00.02

Path: ../TestFiles/a.txt
Number of words: 1
Number of words with more than one count: 1
Word with the most counts: "aaaaaaaaaaaaaaaaaaaa" 1 times
Elapsed time: 00:00.00

Path: ../TestFiles/b.txt
Number of words: 128
Number of words with more than one count: 1
Word with the most counts: "b" 128 times
Elapsed time: 00:00.00

Path: ../TestFiles/space.txt
```

```
Testing ChainedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.00
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.01
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.01
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.09
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.01
../TestFiles/bible.txt;794780;12877;64117;the;00:00.30
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.03
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
Testing ChainedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.00
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.01
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.01
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.11
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.01
../TestFiles/bible.txt;794780;12877;64117;the;00:00.30
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.02
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
Testing ChainedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.00
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.01
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.01
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.09
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.01
../TestFiles/bible.txt;794780;12877;64117;the;00:00.30
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.02
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
Testing OpenAddressedWordCounter
```

```
Testing OpenAddressedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.00
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.02
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.02
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.11
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.02
../TestFiles/bible.txt;794780;12877;64117;the;00:00.30
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.02
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
Testing OpenAddressedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.01
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.01
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.01
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.09
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.01
../TestFiles/bible.txt;794780;12877;64117;the;00:00.29
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.03
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
Testing OpenAddressedWordCounter
../TestFiles/empty.txt;0;0;0;;00:00.00
../TestFiles/se.txt;142;110;7;und;00:00.00
../TestFiles/metamorphosis.txt;25501;2972;1348;the;00:00.01
../TestFiles/verwandlung.txt;24030;4367;631;die;00:00.02
../TestFiles/mobyDick.txt;222094;17131;14727;the;00:00.10
../TestFiles/romeoAndJuliet.txt;29909;3995;878;the;00:00.02
../TestFiles/bible.txt;794780;12877;64117;the;00:00.29
../TestFiles/loremIpsum.txt;41344;117;2128;et;00:00.02
../TestFiles/a.txt;1;1;1;aaaaaaaaaaaaaaaaaaaa;00:00.00
../TestFiles/b.txt;128;1;128;b;00:00.00
../TestFiles/space.txt;0;0;0;;00:00.00
```

In dieser Tabelle ist die Auswirkung, der verschiedenen Hashtable Größen jeweils für jede Datei, auf die Laufzeit und Speichernutzung dargestellt.

Dafür wurde die HASH_TABLE_SIZE vor jeder Ausführung angepasst und jede Datei nur einmal mit dem Flag *printHumanReadable* auf FALSE gezählt.

| Art | HASH_TABLE_SIZE | Laufzeit(Summe) |
|---|---|---|
| Chained | 1 | 00:09.45 |
| Chained | 1000 | 00:00.95 |
| Chained | 10000 | 00:00.78 |
| Chained | 100000 | 00:00.94 |
| Chained | 1000000 | 00:00.86 |
| OpenAddressed | 1000 | - |
| OpenAddressed | 10000 | - |
| OpenAddressed | 100000 | 00:00.72 |
| OpenAddressed | 1000000 | 00:00.79 |

Die zeitlichen Unterschiede sind vernachlässigbar. Die Tabelle mit der Offenen Adressierung benötigt immer mindestens die Anzahl der Elemente die man hinzufügen möchte.