

<input checked="" type="checkbox"/> Gr. 1, Dr. G. Kronberger	Name <u>PAPESH Konstantin</u>	Aufwand in h <u>8</u>
<input type="checkbox"/> Gr. 2, Dr. H. Gruber		
<input type="checkbox"/> Gr. 3, Dr. D. Auer	Punkte <u>20</u>	Kurzzeichen Tutor / Übungsleiter <u>L. S. /</u>

## 1. (De-)Kompression von Dateien

(10 Punkte)

Die sogenannte *Lauf längencodierung* (engl. *run length encoding*, kurz *RLE*) ist eine einfache Datei-Kompressionstechnik, bei der jede Zeichenfolge, die aus mehr als zwei gleichen Zeichen besteht, durch das erste Zeichen und die Länge der Folge codiert wird.

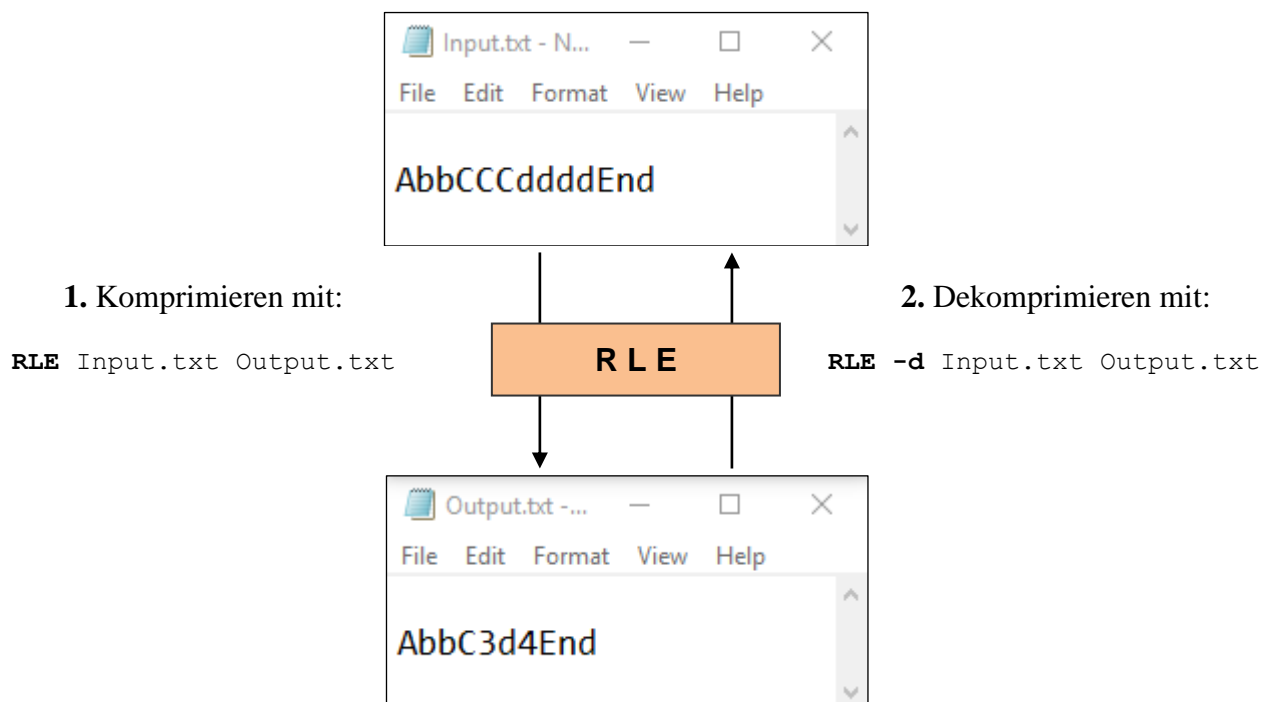
Implementieren Sie eine einfache Variante dieses Verfahrens in Form eines Filterprogramms *RLE*, welches Textdateien, die nur Groß- und Kleinbuchstaben, Satz- und das Leerzeichen (aber keine Ziffern) enthalten, komprimieren und wieder dekomprimieren kann. Ihr Filterprogramm muss folgende Aufrufmöglichkeiten von der Kommandozeile aus bieten (die Metasymbole [...] und ...|... stehen für Option bzw. Alternativen):

```
RLE [ -c | -d ] [ inFile [outFile] ]
```

Bedeutung der Parameter:

-c	die Eingabedatei soll komprimiert werden (Standardannahme),
-d	die Eingabedatei soll dekomprimiert werden,
inFile	Name der Eingabedatei, sonst Standardeingabe und
outFile	Name der Ausgabedatei, sonst Standardausgabe.

Beispiel:



Ostern ist zwar schon vorbei, aber .... ;-)

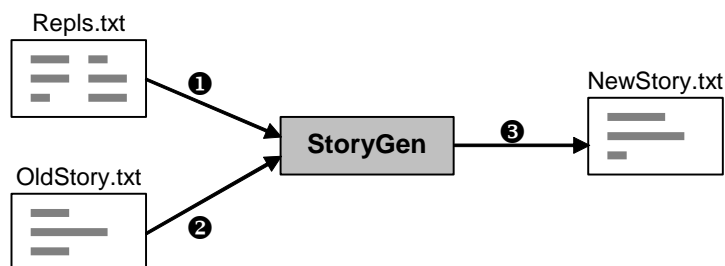


## 2. ... Geschichten vom ...

(14 Punkte)

Heutzutage muss ja alles schnell gehen ;-) Wer hat schon noch die Muse, sich der Jahreszeit entsprechende Geschichten für die kleinen und großen Kinder auszudenken. Gesucht ist deshalb ein Pascal-Programm *StoryGen* (als Abkürzung für *story generator*), welches es ermöglicht, z. B. aus einer Geschichte vom Osterhasen, die gerade noch aktuell war, schnell eine Geschichte vom Krampus oder vom Christkind zu machen, indem spezielle Wörter ausgetauscht werden (z. B. Osterhase durch Christkind und Ostern durch Weihnachten).

Damit man *StoryGen* flexibel anwenden kann, müssen die durchzuführenden Ersetzungen in einer ersten Textdatei (z. B. *Repls.txt* für *Replacements*) definiert sein und die alte Geschichte in einer zweiten Textdatei (z. B. *OldStory.txt*) stehen. Daraus muss die neue Geschichte in einer dritten Textdatei (z. B. *NewStory.txt*) entstehen, so wie in folgender Abbildung dargestellt:



*StoryGen* muss zuerst die Ersetzungen einlesen und in einer geeigneten Datenstruktur speichern. Dann muss die alte Geschichtendatei zeilenweise gelesen, es müssen alle Ersetzungen in der aktuellen Zeile vorgenommen und die geänderte Zeile dann in die neue Geschichtendatei geschrieben werden.

Die Datei mit den Ersetzungen soll beliebig viele Ersetzungen aufnehmen können, in jeder Zeile aber nur eine (mit der Syntax *oldWord newWord*) enthalten. Eine rudimentäre Ersetzungsdatei für den Übergang von Ostern zu Weihnachten könnte dann folgendermaßen aussehen:

```
Osterhase Christkind
Ostern Weihnachten
...
```

*StoryGen* muss von der Kommandozeile mit drei Parametern (den Dateinamen der drei beteiligten Textdateien) versorgt werden, so dass es für obiges Beispiel wie folgt aufgerufen werden kann:

```
StoryGen Repls.txt OldStory.txt NewStory.txt
```

Zum Testen finden Sie in *Ostern.txt* eine Geschichte vom Osterhasen. Machen Sie daraus eine Weihnachtsgeschichte.

# ADF2x & PRO2X Algorithmen & Datenstrukturen und OOP

## – SS 2018

### Übungsabgabe 4

Konstantin Papesh

25. April 2018

#### Zusammenfassung

In dieser Übung wird die Verarbeitung von Files genauer betrachtet. In der ersten Übung wird dabei die Lauflängencodierung betrachtet, wobei ein Programm geschrieben wird, welches die De- und Kompression von Textfiles ermöglicht. In der zweiten Übung wird ein Text aus einem File eingelesen, bestimmte Wörter ersetzt und dann wieder in einen anderen File ausgegeben.

## 4.1 (De-)Kompression von Dateien

### 4.1.1 Lösungsidee

Als erste Funktion wird das Parsen der Kommandozeilenbefehle umgesetzt. Dabei werden die mitgegebenen Variablen *paramStr* und *paramCount* ausgelesen und verarbeitet. Weiters wird dann aufgrund dieser Variablen entschieden, ob komprimiert oder dekomprimiert werden soll. Die jeweilige Subroutine wird dann aufgerufen. In beiden Fällen wird zuerst der eingegebene Name zu einem bestimmten File gemappt und die erste Zeile eingelesen.

Beim Komprimieren wird diese dann Buchstabe für Buchstabe durchgegangen. Ist ein Buchstabe mit dem vorherigen ident, wird ein Zähler erhöht. Dies geschieht so lange, bis ein anderes Zeichen auftritt. Sollte der Zähler über 2 sein, werden alle vorherigen, identen Buchstabe ersetzt mit dieser Zahl angehängt an dem Buchstaben. Beispielsweise steht 'B5' für 'BBBBB'.

Beim Dekomprimieren wird dann wiederum eine komprimierte Datei wieder vergrößert. Dabei wird ähnlich vorgegangen wie beim Komprimieren, nur dass die Zeile nach Zahlen durchgegangen wird. Tritt eine Zahl auf, wird der vorhergehende Buchstabe um diese Anzahl vervielfacht und die Zahl entfernt. Beispielsweise wird aus 'X3' → 'XXX'.

*\clearpage*

### 4.1.2 Implementierung

✓ Name - 0,5

2

Listing 4.1: rle.pas

```
1 program rle;
2
3 uses sysutils;
4
5 function fileExists(fromFileName : string) : boolean;
6 var
7     f : TEXT;
8 begin
9     (*$I-*)
10    assign(f, fromFileName); } uneinheitlich - 0,5
11    Reset(f); //Creates error
12    if IOResult <> 0 then begin
13        fileExists := FALSE;
14    end else +
15        fileExists := TRUE; +
16    (*$I+*)
17 end;
18
19 function isNumber(character : char) : boolean;
20 begin
21     if(ord(character) < 48) or (ord(character) > 57) then
22         isNumber := FALSE
23     else
24         isNumber := TRUE;
25 end;
26
27 function addCharCountAtEnd(previousChar : char; charOccurrences : integer;
28     compressedLine : string) : string;
29 var
30     charOccurrencesString : string;
31 begin
32     if(charOccurrences > 2) then begin
33         str(charOccurrences, charOccurrencesString);
34         compressedLine := compressedLine + previousChar + charOccurrencesString;
35     end else
36         compressedLine := compressedLine + stringOfChar(previousChar, charOccurrences);
37     addCharCountAtEnd := compressedLine;
38 end;
39
40 function compressLine(line : string) : string;
41 var
42     i, lineLength : integer;
43     previousChar, currentChar : char;
44     charOccurrences : integer;
45     compressedLine : string;
46 begin
47     i := 1;
48     lineLength := length(line);
49     charOccurrences := 1;
50     compressedLine := '';
51     while (i <= lineLength) do begin
52         currentChar := line[i];
53         if(isNumber(currentChar)) then begin
54             writeln('Can''t compress text with numbers in it!');
```

```

54      halt; zu lange Funktion für Sprunganweisungen - 0,5
55      end;
56      if(currentChar = previousChar) then begin
57          inc(charOccurrences);
58      end else if (i = 1) then Syntax - Highlighting fehlt - 0,5
59          previousChar := currentChar // there is no prev character so just set
        cur char to prev.
60      else begin
61          compressedLine := addCharCountAtEnd(previousChar, charOccurrences,
        compressedLine);
62          previousChar := currentChar;
63          charOccurrences := 1;
64      end;
65      inc(i);
66  end;
67  compressedLine := addCharCountAtEnd(previousChar, charOccurrences, compressedLine
    );
68  compressLine := compressedLine;
69 end;
70
71 procedure compress(fromFileName, toFileName : string);
72 var
73     inFile, outFile : TEXT;
74     line : string;
75 begin
76     assign(inFile, fromFileName);
77     assign(outFile, toFileName);
78     reset(inFile);
79     rewrite(outFile); // overwrite if file exists
80     while not eof(inFile) do begin
81         readln(inFile, line);
82         line := compressLine(line);
83         writeln(outFile, line);
84     end;
85     close(inFile);
86     close(outFile);
87 end;
88
89 function decompressLine(line : string) : string;
90 var
91     i, lineLength : integer;
92     decompressedLine : string;
93 begin
94     decompressedLine := '';
95     i := 1;
96     lineLength := length(line);
97     while (i <= lineLength) do begin
98         if(isNumber(line[i])) then begin
99             decompressedLine := decompressedLine + stringOfChar(line[i-1], strToInt(
        line[i])-1);
100         end else
101             decompressedLine := decompressedLine + line[i]; wieder uneindeutlich bei
102             inc(i); begin .. end
103         end;
104     decompressLine := decompressedLine;
105 end;
106

```

```

107 procedure decompress(fromFileName, toFileName : string);
108 var
109     inFile, outFile : TEXT;
110     line : string;
111 begin
112     assign(inFile, fromFileName);
113     assign(outFile, toFileName);
114     reset(inFile);
115     rewrite(outFile); // overwrite if file exists
116     while not eof(inFile) do begin
117         readln(inFile, line);
118         line := decompressLine(line);
119         writeln(outFile, line);
120     end;
121     close(inFile);
122     close(outFile);
123 end;
124
125 var
126     option : string;
127     fromFileName, toFileName : string;
128
129 begin
130     option := '';
131     fromFileName := '';
132     toFileName := '';
133
134     case paramCount of
135         0: begin
136             end;
137         1: begin
138             if(paramStr(1)[1] = '-') then
139                 option := paramStr(1)
140             else
141                 fromFileName := paramStr(1);
142             end;
143         2: begin
144             if paramStr(1)[1] = '-' then begin
145                 option := paramStr(1);
146                 fromFileName := paramStr(2);
147             end else begin
148                 fromFileName := paramStr(1);
149                 toFileName := paramStr(2);
150             end;
151         end;
152         3: begin
153             option := paramStr(1);
154             fromFileName := paramStr(2);
155             toFileName := paramStr(3);
156         end;
157     else begin
158         writeln('Usage: rle [ -c | -d ] [ inFile [ outFileName ] ]');
159         halt;
160     end;
161 end;
162
163 if not fileExists(fromFileName) then begin

```

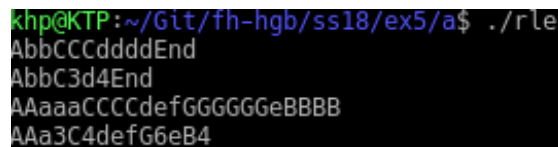
*in Funktion - 0,5*

```

164     writeln('inFile does not exist! Unable to complete operation');
165     halt;
166 end;
167
168 if (option = '-c') or (option = '') then // default -b
169     compress(fromFileName, toFileName)
170 else if (option = '-d') then
171     decompress(fromFileName, toFileName)
172 else begin
173     writeln('Unknown option ', option);
174     halt;
175 end;
176 end.

```

### 4.1.3 Ausgabe

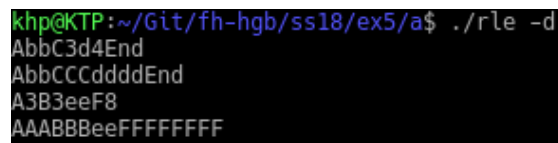


```

khp@KTP:~/Git/fh-hgb/ss18/ex5/a$ ./rle
AbbCCCdddEnd
AbbC3d4End
AAaaaCCCCdefGGGGGeBBBB
AAa3C4defG6eB4

```

Abbildung 4.1: Kompression mit Lauflängencodierung in der Konsole



```

khp@KTP:~/Git/fh-hgb/ss18/ex5/a$ ./rle -d
AbbC3d4End
AbbCCCdddEnd
A3B3eeF8
AAABBBeeFFFFFFFF

```

Abbildung 4.2: Dekompression mit Lauflängencodierung in der Konsole

*keine Leerzeichen, Satzzeichen -0,5*

### 4.1.4 Auswertung


Wird kein Filename angegeben, wird der Standardinput<sup>1</sup> verwendet. Dies ermöglicht auch eine leichte Überprüfung der Funktion des Programms.

## 4.2 Text Replacement

Ein Text ist gegeben, bei diesem sollen bestimmte Wörter, welche in einem zusätzlichen Textfile angegeben sind ersetzt werden.

### 4.2.1 Lösungsidee

Um das Programm auszuführen müssen zuerst folgende Files existieren:

1. replacement.txt - Enthält alle zu ersetzenden Wörter in der Form 'altes Wort'  'neues Wort'.
2. oldStory.txt - Enthält den Text, in welchem die Wörter ersetzt werden sollen.

<sup>1</sup>In diesem Fall die Konsole

Weiters muss der Name des Files gegeben werden, in welchen die neue Story hineingeschrieben werden soll. Dieser File muss jedoch nicht existieren. Sollte er dies dennoch tun, wird der ursprüngliche Inhalt überschrieben.

Werden alle notwendigen Dateinamen angegeben, beginnt das Programm die alte Story zeilenweise einzulesen und in einzelne Wörter aufzusplitten. Diese einzelnen Wörter werden dann mit den ersten Wörtern in jeder Zeile der *replacement.txt* verglichen. Gibt es einen Match, wird das ursprüngliche Wort mit dem zweiten Wort in der Zeile von *replacement.txt* ersetzt. Danach wird das nächste Wort eingelesen und der Prozess wiederholt sich bis zum Ende der Zeile. Danach wird die Zeile in den *newStory* File geschrieben. Danach wird die nächste Zeile eingelesen. Dieser Vorgang wiederholt sich bis zum Ende des Dokuments. *Datenstruktur? - 0,5*

#### 4.2.2 Implementierung

**Listing 4.2:** StoryGen.pas

```

1 program StoryGen;
2
3 type
4     replacementPair = record
5         oldWord : string;
6         newWord : string;
7     end;
8     replacementPairArray = array of replacementPair;
9
10 var
11     replacementWords : replacementPairArray;
12
13 function fileExists(fromFileName : string) : boolean;
14 var
15     f : TEXT;
16 begin
17     (*$I-*)
18     assign(f, fromFileName);
19     Reset(f); //Creates error
20     if IOResult <> 0 then begin
21         fileExists := FALSE;
22     end else
23         fileExists := TRUE;
24     (*$I+*)
25 end;
26
27 function isDelimiter(character : string) : boolean;
28 begin
29     if(character = ' ') or (character = '.') or (character = '!') or (character =
30         '?') then
31         isDelimiter := TRUE
32     else
33         isDelimiter := FALSE;
34 end;
35
36 function isEqual(oldWord, newWord : string) : boolean;
37 var
38     lengthOldWord : integer;
```



```

38     lengthNewWord : integer;
39 begin
40     lengthOldWord := length(oldWord);
41     lengthNewWord := length(newWord);
42     if(isDelimiter(oldWord[lengthOldWord])) then
43         delete(oldWord,lengthOldWord,1);
44     if oldWord = newWord then
45         isEqual := TRUE
46     else
47         isEqual := FALSE;
48 end;
49
50 function getAndRemoveFirstWord(var line : string) : string;
51 var
52     i : integer;
53     finished : boolean;
54     curWord : string;
55 begin
56     i := 1;
57     finished := FALSE;
58     curWord := '';
59     while (i <= length(line)) and (finished = FALSE) do begin
60         if(isDelimiter(line[i])) or (i = length(line)) then begin
61             if not(line[i] = ' ') then
62                 curWord := curWord + line[i];
63             getAndRemoveFirstWord := curWord;
64             Delete(line,1,i);
65             finished := TRUE;
66         end else begin
67             curWord := curWord + line[i];
68         end;
69         inc(i);
70     end;
71 end;
72 procedure splitLine(line : string; var lineWords : array of string);
73 var
74     words : integer;
75     finished : boolean;
76 begin
77     words := 0;
78     finished := FALSE;
79     while (line <> '') and (finished = FALSE) do begin
80         lineWords[words] := getAndRemoveFirstWord(line);
81         if(words >= 2) then
82             finished := TRUE;
83         inc(words);
84     end;
85 end;
86
87 procedure copyText(fromFileName, toFileName : string);
88 var
89     inFile, outFile : TEXT;
90     line : string;
91 begin
92     assign(inFile, fromFileName);
93     assign(outFile, toFileName);
94     reset(inFile);

```

```

95     rewrite(outFile); // overwrite if file exists
96     while not eof(inFile) do begin
97         readln(inFile, line);
98         writeln(outFile, line);
99     end;
100    close(inFile);
101    close(outFile);
102 end;
103
104 procedure parseRepls(fromFileName : string);
105 var
106     inFile : TEXT;
107     line : string;
108     lineWords : array[0..1] of string;
109     i : integer;
110 begin
111     i := 0;
112     assign(inFile, fromFileName);
113     reset(inFile);
114     while not eof(inFile) do begin
115         setLength(replacementWords, length(replacementWords)+1);
116         readln(inFile, line);
117         splitLine(line, lineWords);
118         replacementWords[i].oldWord := lineWords[0];
119         replacementWords[i].newWord := lineWords[1];
120         inc(i);
121     end;
122     close(inFile);
123 end;
124
125 function replaceWordsInLine(line : string) : string;
126 var
127     curWord : string;
128     i : integer;
129     newLine : string;
130     finished : boolean;
131     endSymbol : string;
132 begin
133     newLine := '';
134     finished := FALSE;
135     while (line <> '') do begin
136         i := 0;
137         endSymbol := '';
138         finished := FALSE;
139         curWord := getAndRemoveFirstWord(line);
140         while (i < length(replacementWords)) and (finished = FALSE) do begin
141             if (isDelimiter(curWord[length(curWord)])) then begin
142                 endSymbol := curWord[length(curWord)];
143             end;
144             if (isEqual(curWord, replacementWords[i].oldWord)) then begin
145                 curWord := replacementWords[i].newWord + endSymbol;
146                 finished := TRUE;
147             end;
148             inc(i);
149         end;
150         if not (curWord = ' ') then
151             newLine := newLine + curWord + ' ';

```

*schlechte Effizienz  
- 0,5*

```

152     end;
153     replaceWordsInLine := newLine;
154 end;
155
156 procedure createNewStory(fromFileName, toFileName : string);
157 var
158     inFile, outFile : TEXT;
159     line : string;
160     newLine : string;
161 begin
162     assign(inFile, fromFileName);
163     assign(outFile, toFileName);
164     reset(inFile);
165     rewrite(outFile); // overwrite if file exists
166     while not eof(inFile) do begin
167         readln(inFile, line);
168         newLine := replaceWordsInLine(line);
169         writeln(outFile, newLine);
170     end;
171     close(inFile);
172     close(outFile);
173 end;
174
175
176 var
177     repls : string;
178     fromFileName, toFileName : string;
179     i : integer;
180
181 begin
182     fromFileName := '';
183     toFileName := '';
184     i := 0;
185     if not (paramCount = 3) then begin
186         writeln('Usage: StoryGen REPLACEMENTFILE OLDSTORYFILE NEWSTORYFILE');
187         halt;
188     end;
189
190     repls := paramStr(1);
191     fromFileName := paramStr(2);
192     toFileName := paramStr(3);
193
194     if not fileExists(repls) then begin
195         writeln('Replacement file does not exist!');
196         halt;
197     end;
198     if not fileExists(fromFileName) then begin
199         writeln('Old story file does not exist!');
200         halt;
201     end;
202
203     parseRepls(repls);
204     createNewStory(fromFileName, toFileName);
205     writeln('Wrote story just for you <3');
206
207 end.

```

### 4.2.3 Ausgabe

```
Osterhase Christkind
Osterhasen Christkind
Ostern Weihnachten
Ostereier Geschenke
legten besorgten
Hühnchen Wichtel
Eier Geschenke
Hähnchen Zwerge
Osterfest Weihnachtsfest
Ei Geschenk
gelegt besorgt
bemalen verpacken
```

Abbildung 4.3: Beispiel für einen Textfile mit Ersetzungen

```
Die Geschichte vom Osterhasen Hanni!
Wißt Ihr liebe Kinder, vor langer, langer Zeit gab es noch keinen Osterhasen.
Da legten die Hühnchen die Eier und die Hähnchen bemalten sie. In
einem Körbchen trugen sie vorsichtig die Eier zu den Kindern und
stellten es vor die Tür. Die Kinder warteten schon gespannt auf das
Osterfest. Und wenn es dann endlich so weit war, sprangen sie schnell
aus ihren Betten und öffneten neugierig die Tür. Wie freuten sie sich über
die vielen bunten Ostereier ...
```

Abbildung 4.4: Beispiels für einen Ursprungstext

```
Die Geschichte vom Christkind Hanni!
Wißt Ihr liebe Kinder, vor langer, langer Zeit gab es noch keinen Christkind.
Da besorgten die Hühnchen die Geschenke und die Hähnchen bemalten sie. In
einem Körbchen trugen sie vorsichtig die Geschenke zu den Kindern und
stellten es vor die Tür. Die Kinder warteten schon gespannt auf das
Weihnachtsfest. Und wenn es dann endlich so weit war, sprangen sie schnell
aus ihren Betten und öffneten neugierig die Tür. Wie freuten sie sich über
die vielen bunten Geschenke ...
```

Abbildung 4.5: Beispiel für einen Text mit Ersetzungen