Bundeswettbewerb Informatik,

1. Runde

Andreas Albert
Online-Einsendenummer 348
5. November 2006

<Inhaltsverzeichnis>

1. Vorwort

In diesem Jahr nehme ich das erste Mal am Bundeswettbewerb der Informatik teil und hoffe daher, dass die Dokumentation im Großen und Ganzen richtig aufgebaut wurde von mir.

Als Programmiersprachen zum Lösen der Aufgaben wurden von mir C++ und Delphi gewählt, um meine Programmierkenntnisse auf diese Art und Weise auf zwei unterschiedlichen Gebieten anzueignen bzw. zu vertiefen.

Hier nun Vorab eine Übersicht über die Verzeichnisstruktur auf der beigelegten CD:

Aufgabe:	Art der Bearbeitung:
Närrische Wirtschaft	Vollständig; C++; in /Aufgabe 1
Robot Dressing	Vollständig; C++; in /Aufgabe 2
HTML-Mobiles	Vollständig; Delphi; in /Aufgabe 3
Supermarkt	Vollständig; C++; in /Aufgabe 4
Die Paderbox	Vollständig; Delphi; in /Aufgabe 5

Der Quelltext des Programms hat im Falle von C++ immer den gleichen Namen, also findet man ihn immer unter „main.cpp“ im entsprechenden Verzeichnis.

Im Falle von Delphi bei der Aufgabe „HTML-Mobiles“ findet man die Projektdatei unter „projekt_html_mobiles.dpr“, sowie den Quellcode unter „unit_main.pas“.

Für die Aufgabe 5, die ebenfalls mit Delphi programmiert wurde, findet man die Projektdatei unter „projekt_paderbox.dpr“, sowie den Quellcode aufgteilt auf die Dateien „paderbox_1.pas“, „config_1.pas“ und „paderbox_class.pas“.

Alle Programme wurden von mir unter Windows XP entwickelt.

2. Aufgabe 1: „Närrische Wirtschaft“

2.1. Lösungsidee

In meiner Lösung verwende ich das Wort Artikel gleichbedeutend mit dem Wort Gegenstand aus der Aufgabenstellung.
Die Bezeichnung Artikelkombination bezeichnet nichts anderes als eine Zusammenfassung von mehreren unterschiedlichen Artikeln.

Da man nur ein feststehendes monatliches Budget zur Verfügung hat, aber damit in einer minimalen Anzahl von Monaten alle benötigten Artikel durch geschicktes Kaufen und Verkaufen von vorhandenen und benötigten Gegenständen erreichen soll, ist die Grundvoraussetzung eigentlich, die Verlustquote des Geldes, dass für Eisessen verschwendet wird, so gering wie möglich zu halten.

Daher lässt sich meiner Meinung nach folgende vereinfacht Lösungsidee herleiten:

Max. Monatsbudget = Verkauf aller vorhandenen Artikel + Monatsbudget

Verlustquote 1 = Artikel 1 + Artikel 2 + ... Artikel n
Verlustquote 2 = Artikel 2 + Artikel 3 + ... Artikel n
...
Verlustquote n = Artikel n

Geringste Verlustquote = Minimum.(Verlustquote 1 bis Verlustquote n)
***für welche gelten muss:** Gesamtkosten <= Max. Monatsbudget*

Anders ausgedrückt müsste man sich den Artikelbestand des Vormonates merken, dann ein maximales Monatsbudget errechnen, dass sich aus den Verkauf aller vorhanden Artikel ergibt. Nachfolgend müsste man alle möglichen Artikelstrukturen in einem Algorithmus durchlaufen und alle Verlustquoten, für welche die Regel Gesamtkosten <= Max. Monatsbudget gilt, miteinander zu vergleich, um so die geringste Verlustquote für jeden Monat errechnen zu könne. Anschließend müsste man die Artikelstruktur mit der geringsten Verlustquote übernehmen und diese mit der Liste des Artikelbestandes des Vormonates abgleichen, um so für jeden Monat ausgeben zu könne, welcher Artikel Verkauft bzw. Gekauft werden muss.

2.2. Programm–Dokumentation

Die Lösungsidee wurde mit der Programmiersprache C++ in Form einer Konsolenanwendung realisiert. Es besteht aus der Datei „*Aufgaben 1/main.cpp*“.
Das aktuelle Programm wurde auf eine maximale Anzahl von 20.000 zu besorgende Artikel beschränkt, kann jedoch jederzeit auf einfachste Weise erweitert werden, indem man die Begrenzung in der Zeile „*#define MAX 20000*“ auf einen gewünschten Wert verändert.
Zu beginn des Programms hat der Benutzer die Möglichkeit sich zwischen zwei Arten der Dateneingabe zu entschieden. Durch Auswahl des Menüpunktes „*Werte importieren aus Textdatei*“ kann der Benutzer größere Datenmengen direkt über eine Textdatei importieren. Der Aufbau der Textdatei um Daten zu importieren muss allerdings wie folgt aufgebaut sein:

```
<Monatliches Budget>
<ITEM 1>
<ITEM 2>
<ITEM 3>
::::
<ITEM n>
```

(allgemeiner Aufbau)

Beispiel:

Bei den zu kaufenden Gegenständen von Gegenstände 340 €, 670 €, 790 €, 1320 €, 2100 €, 5200 € und einem monatlichen Budget von 1000 € würde der Aufbau der Textdatei wie folgt aussehen:

```
1000
340
670
790
1320
2100
5200
```

(Aufbau des Beispiels)

Die zweite Möglichkeit, um Daten manuell einzugeben, stellt der Menüpunkt „manuelle Eingabe“ da. Wurde dieser ausgewählt, so muss der Benutzer zunächst das zur Verfügung stehenden monatlichen Budget angeben. Im Anschluss daran gibt das Programm dem Benutzer die Möglichkeit über ein Menü bis zu 20.000 Artikelpreise von Artikeln einzugeben, die besorgt werden sollen. Durch anschließendes auswählen des Menüpunktes „*Artikeleingabe beenden*“ durch drücken der Taste „*e*“ wird zunächst eine Prüfung durchgeführt, ob der Preiswerteste Artikel überhaupt mit dem Monatsbudget gekauft werden kann. Sollte dies nicht der Fall sein, so wird das Programm sofort durch Ausgebe eines Hinweises, „*Ihr Monatsbudget liegt unterhalb des niedrigsten Artikelpreises*“ beendet, da keiner der Artikel überhaupt besorgt werden kann.
Sollte es durch die oben beschrieben Prüfung möglich sein auf jeden Fall den billigsten aller Artikel mithilfe des Monatsbudgets zu kaufen, so wird erst nun die eigentliche Berechnung des Anschaffungsplans in Gang gesetzt.

Zunächst wird die Artikelkombination des Vormonates Zwischengespeichert in einem Array mit der Bezeichnung „*alt_anzahl*“. Nun folgt die Berechnung des Maximalen Monatsbudgets, dass sich wie folgt errechnet:

Max. Monatsbudget = Verkauf aller vorhandenen Artikel + Monatsbudget

Nun wird die von mir Selbstgeschriebene rekursive Funktion „*berechne()*“ aufgerufen, in der alle weitren Berechnungen stattfinden.
In dieser Funktion wird durch einen Algorithmus aus allen Artikeln sämtliche mögliche Artikelkombinationen gebildet für die folgendes gilt:

Gesamtkosten der Artikelkombination <= Max. Monatsbudget

Nach der erfolgreichen Bildung einer Artikelkombination wird deren Verlustquote wie folgt errechnet:

Verlustquote = Max. Monatsbudget - Gesamtkosten der Artikelkombination

Anschließend wird die Verlustquote mit der Variable „*best_differenz*“ verglichen. Sollte die Verlustquote geringer sein, als der Wert der Variable „*best_differenz*“, so wird die Artikelkombination als bisher beste in das Array „*best*“ übernommen und gleichzeitig der Wert der Verlustquote in die Variable „*best_ differenz*“ geschrieben.

Nachdem alle möglichen Artikelkombinationen durchlaufen und miteinander verglichen wurden, liegt am Ende die Artikelkombination mit der geringsten Verlustquote im Array „*best*“ vor.

Durch den vergleich des Arrays „*alt_anzahl“* in dem sich die Artikelkombination des Vormonates befindet mit dem Array „*best*“ in dem sich die aktuelle Artikelkombination befindet, kann nun ermittelt werden, welche Artikel im aktuellen Monat Gekauft und Verkauft werden müssen. Dies wird dem Benutzer dann auf dem Bildschirm ausgegeben.

Nun wird durch eine Schleife geprüft, ob bereits alle benötigten Artikel in der aktuellen Artikelstruktur vorhanden sind, da in diesem Falle dies der letzte zu berechnenden Monat wäre und somit die Variable „*status*“ auf den Wert „*false*“ gesetzt werden muss, um dem Programm zu signalisieren, dass kein weiterer Monat mehr berechnet werden muss.

Sollte der eben beschriebene Fall nicht zutreffen, so wird die aktuelle Artikelkombination Zwischengespeichert in einem Array mit der Bezeichnung „*alt_anzahl*“. Anschließend wird das Maximalen Monatsbudgets wieder berechnet wie anfangs oben beschrieben und ein weiterer Monat berechnet durch das ausführen der rekursiven Funktion „*berechne()*“ Diese Vorgänge wiederholen sich für jeden Monat so lange, bis der beschriebene Fall eintritt, dass bereits alle benötigten Artikel in der aktuellen Artikelstruktur vorhanden sind und somit die Variable „*status*“ auf den Wert „*false*“ gesetzt werden kann , um dem Programm zu signalisieren, dass kein weiterer Monat mehr berechnet werden muss.

Nun erfolgt noch eine abschließende Ausgabe auf dem Bildschirm, die dem Benutzer mitteilt, nach wie vielen Monaten alle benötigten Artikel frühestens besorgt werden können. Nach der drücken einer beliebigen Taste wird abschließend das Programm beendet.

Im gesamten Programm gibt es eigentlich nur 1 Nutzungsgrenze, sowie ein verlangtes Eingabeformat. Die Nutzungsgrenze betrifft die Eingabe der zu besorgenden Artikel, die durch die Zeile *„#define MAX 20000“* auf 20.000 Artikel begrenzt wurde. Diese Nutzungsgrenze kann jedoch jederzeit auf einfachste Weise beseitigt werden, indem man die Begrenzung in der Zeile *„#define MAX 20000“* auf einen gewünschten Wert verändert.

Das verlangte Eingabeformat bezieht sich auf sämtliche Eingaben, bei denen ein Preis einzugeben ist. In diesen Fällen darf die Eingabe nur Ziffern enthalten und es ist auch nicht möglich irgendwelche Kommabeträge einzugeben.

2.3. Programm-Ablaufsprotokoll

2.3.1. Probelauf mit den Daten aus der „*Aufgabenstellung*“

Gegenstände: 340 €, 670 €, 790 €, 1320 €, 2100 €, 5200 €

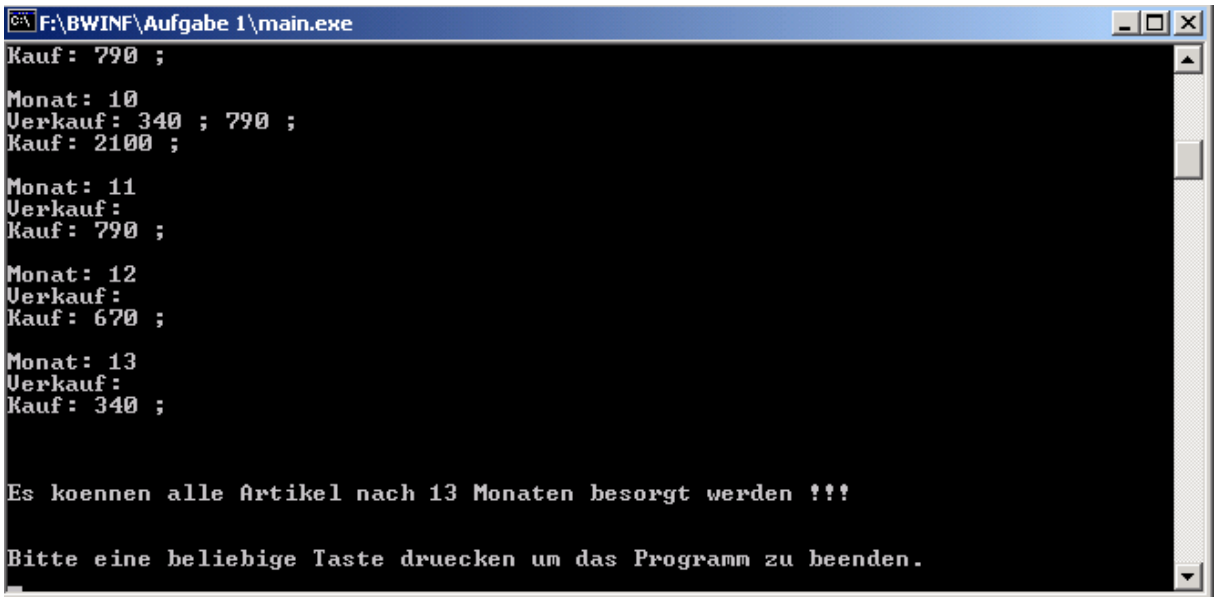
```
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

m = manuelle Eingabe
i = Werte importieren aus Textdatei
m

Bitte geben Sie ihr monatliches Budget an: 1000
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 340
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 670
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 790
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 1320
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 2100
++++++++++++++++++++++
Bitte waehlen Sie aus:
```

```
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
a
Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: 5200
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Neuen Artikelwert eingeben (maximal: 20000)
e = Artikeleingabe beenden
e
Monat: 1
Verkauf:
Kauf: 790 ;

Monat: 2
Verkauf: 790 ;
Kauf: 340 ; 1320 ;

Monat: 3
Verkauf:
Kauf: 790 ;

Monat: 4
Verkauf: 340 ; 790 ;
Kauf: 2100 ;

Monat: 5
Verkauf:
Kauf: 790 ;

Monat: 6
Verkauf: 790 ; 1320 ; 2100 ;
Kauf: 5200 ;

Monat: 7
Verkauf:
Kauf: 790 ;

Monat: 8
Verkauf: 790 ;
Kauf: 340 ; 1320 ;

Monat: 9
Verkauf:
Kauf: 790 ;

Monat: 10
Verkauf: 340 ; 790 ;
Kauf: 2100 ;

Monat: 11
Verkauf:
Kauf: 790 ;

Monat: 12
Verkauf:
Kauf: 670 ;
```

Monat: 13
Verkauf:
Kauf: 340 ;

Es koennen alle Artikel nach 13 Monaten besorgt werden !!!

Bitte eine beliebige Taste druecken um das Programm zu beenden.



2.3.2. Probelauf mit den Daten aus „*beispie1l.txt*“

Gegenstände: Siehe Auflistung in der Datei „*beispiel1.txt*“ mit Ausnahme des ersten Wertes

```
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

m = manuelle Eingabe
i = Werte importieren aus Textdatei
i

Bitte geben Sie nun den korrekten Pfad zur Textdatei an:
(z.B. C:\import.txt oder wenn die Datei im selben Ordner liegt import.txt)
beispiel1.txt

Monat: 1
Verkauf:
Kauf: 1 ;

Monat: 2
Verkauf: 1 ;
Kauf: 2 ;

Monat: 3
Verkauf:
Kauf: 1 ;

Monat: 4
Verkauf: 1 ; 2 ;
Kauf: 4 ;

Monat: 5
Verkauf:
Kauf: 1 ;
```

←

Die Bildschirmausgaben für die Monate nach 5 und vor 131067 wurden weggelassen bei dieser Dokumentation, da diese die Dokumentation sonst sprengen würden

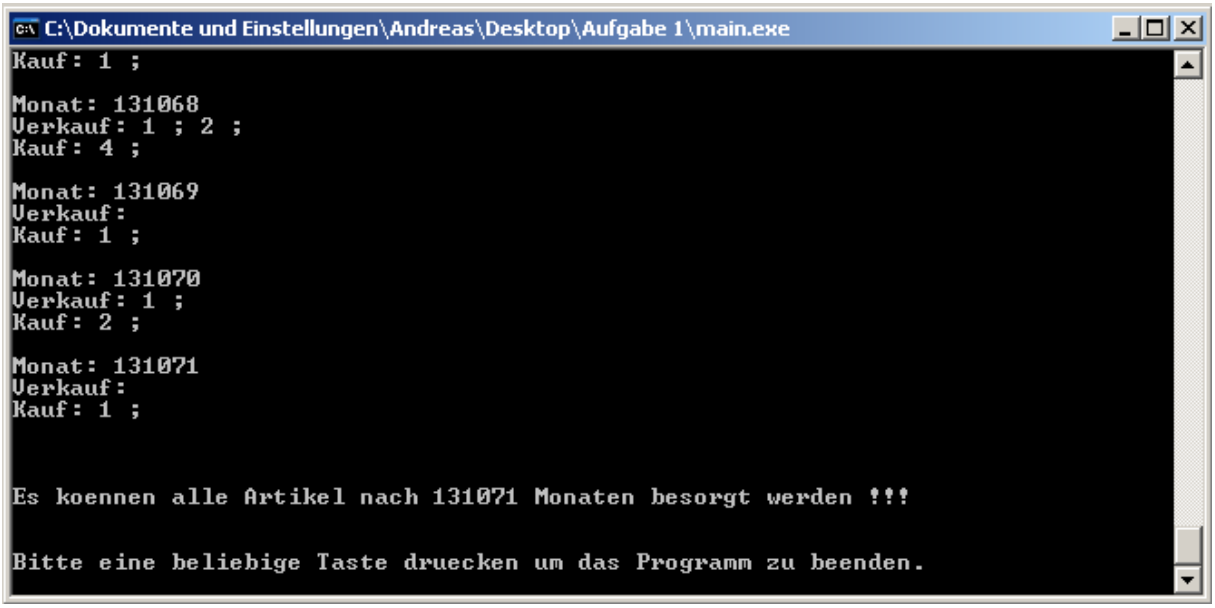
←

```
Monat: 131067
Verkauf:
Kauf: 1 ;

Monat: 131068
Verkauf: 1 ; 2 ;
Kauf: 4 ;

Monat: 131069
Verkauf:
Kauf: 1 ;

Monat: 131070
Verkauf: 1 ;
Kauf: 2 ;
```

Monat: 131071
Verkauf:
Kauf: 1 ;

Es koennen alle Artikel nach 131071 Monaten besorgt werden !!!

Bitte eine beliebige Taste druecken um das Programm zu beenden.



2.3.3. Probelauf mit den Daten aus „*beispie12.txt*“

Gegenstände: Siehe Auflistung in der Datei „*beispiel2.txt*“ mit Ausnahme des ersten Wertes

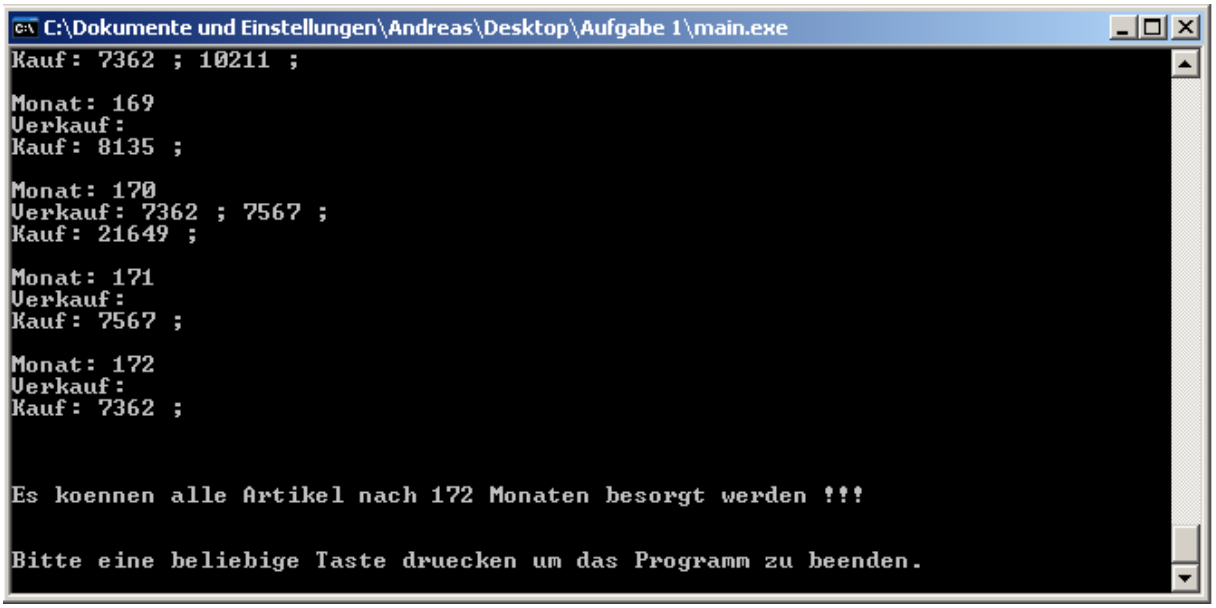
```
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

m = manuelle Eingabe
i = Werte importieren aus Textdatei
i

Bitte geben Sie nun den korrekten Pfad zur Textdatei an:
(z.B. C:\import.txt oder wenn die Datei im selben Ordner liegt import.txt)
beispiel2.txt
```

←

Die Bildschirmausgaben für die Monate 1 bis 169 wurden weggelassen bei dieser Dokumentation, da diese die Dokumentation sonst sprengen würden

←

```
Monat: 170
Verkauf: 7362 ; 7567 ;
Kauf: 21649 ;

Monat: 171
Verkauf:
Kauf: 7567 ;

Monat: 172
Verkauf:
Kauf: 7362 ;

Es koennen alle Artikel nach 172 Monaten besorgt werden !!!

Bitte eine beliebige Taste druecken um das Programm zu beenden.
```



2.3.4. Probelauf mit den Daten aus „*beispiel3.txt*“

Gegenstände: Siehe Auflistung in der Datei „*beispiel3.txt*“ mit Ausnahme des ersten Wertes

```
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

m = manuelle Eingabe
i = Werte importieren aus Textdatei
i

Bitte geben Sie nun den korrekten Pfad zur Textdatei an:
(z.B. C:\import.txt oder wenn die Datei im selben Ordner liegt import.txt)
beispiel3.txt

Monat: 1
Verkauf:
Kauf: 50 ;

Monat: 2
Verkauf: 50;
Kauf: 100 ;

Monat: 3
Verkauf:
Kauf: 50 ;

Monat: 4
Verkauf: 50;
Kauf: 100 ;

Monat: 5
Verkauf:
Kauf: 50 ;
```

←

Die Bildschirmausgaben für die Monate nach 5 wurden weggelassen bei dieser Dokumentation, da diese die Dokumentation sonst sprengen würden

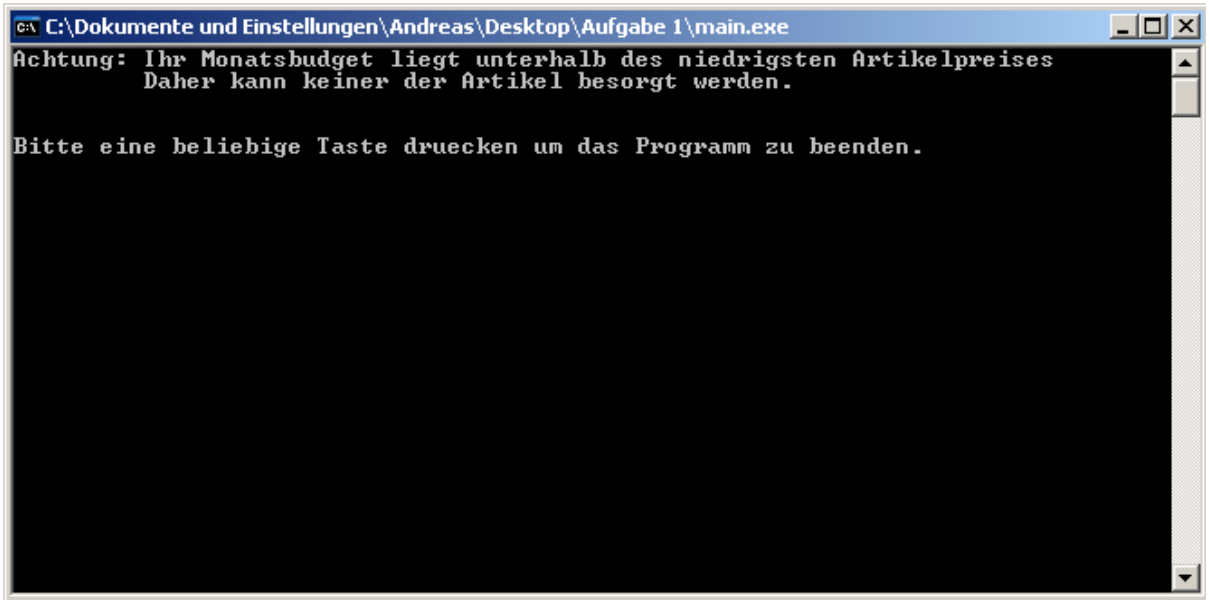
←

```
Es koennen alle Artikel nach 1026 Monaten besorgt werden !!!

Bitte eine beliebige Taste druecken um das Programm zu beenden.
```

2.3.5. Probelauf mit den Daten aus „*beispiel14.txt*“

Gegenstände: Siehe Auflistung in der Datei „*beispiel4.txt*“ mit Ausnahme des ersten Wertes

```
++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

m = manuelle Eingabe
i = Werte importieren aus Textdatei
i

Bitte geben Sie nun den korrekten Pfad zur Textdatei an:
(z.B. C:\import.txt oder wenn die Datei im selben Ordner liegt import.txt)
beispiel4.txt

Achtung: Ihr Monatsbudget liegt unterhalb des niedrigsten Artikelpreises
         Daher kann keiner der Artikel besorgt werden.


Bitte eine beliebige Taste druecken um das Programm zu beenden.
```



2.4. Programm-Text

```
// 25. Bundeswettbewerb Informatik 2006/2007
// Aufgabe 1: Naerrische Wirtschaft
//
// (c) Programmierung Andreas Albert

// Einbinden der benoetigten Programmbibliotheken
#include <conio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;

// Definierte Variable, anhand deren man die maximale Anzahl der Artikel leicht
// und unkompliziert aendern kann
#define MAX 20000

#define BUF 255

// ++++++++++++++++++++++++++++
// Globale Variablendeklaration
// ++++++++++++++++++++++++++++

// Array, in dem die Preise der Artikel gespeichert werden
int preis[MAX];
// Buffer-Array, in dem die Artikelstruktur Zwischengespeichert wird, bei der die
// geringste Preisdifferenz vorliegt
int best[MAX];
// Diese Variable wird benoetigt um die geringste Preisdifferenz zu speichern und
// so die Artikelkombination spaeter mit der geringsten Preisdifferenz in jedem
// Monat zu finden
int best_differenz=0;

// Selbst geschrieben rekursive Prozedur, um alle moeglichen Artikelkombinationen
// inklusive Preisdifferenz zu berechnen und sich jeweils die Artikelkombination
// mit der geringsten Verlustdifferenz zu merken...
void berechne(int ebene,int c_anzahl[MAX],int c_rest_budget, int max)
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// ebene        -> Anzahl der zu besorgenen Artikel
// c_anzahl     -> Merkvariable die notwenidig ist um alle Artikelkombinationen zu
//                 durchlaufen
// c_rest_budget -> Hier wird sich gemerkt, wie hoch das noch zur Verfuegung stehende
//                Budget ist, nachdem ein Artikel gekauft wurde...
// max          -> Merkvariable in der die Anzahl der verschiedenen Artikel steht, und die
//                für die Schleifen notwendig ist
{
  // Algorithmus zur Berechnug der optimalen Anschaffungen, so dass der Verlustbetrag
  // im Monat so gering wie möglich ausfällt...
  cout << "EBENE:" << ebene<< endl;
```

```
// Zunaechst wird versucht den teuersten Artikel zu kaufen ...
if(c_rest_budget-preis[ebene-1]>=0)
{
  // Artikel kaufen...
  c_anzahl[ebene-1]++;
  // Berechnung des verbleibenden Restbudget nach einem Artikelkauf...
  c_rest_budget = c_rest_budget - preis[ebene-1];
  // Pruefung ob bei diesem Fall (=Artikelkobination) eine bessere Preisdifferent
  // erzielt wurde als bei einer bereits vorher ausprobierten Artikelkobination...
  if((best_differenz>c_rest_budget) )
  {
    // Uebernehmen der neuen Artikelstruktur...
    for(int j=0;j<max;j++)
    {
      best[j] = c_anzahl[j];
    }
    // ...und uebernehmen der aktuellen Verlustdifferenz
    best_differenz = c_rest_budget;
  }
}
// Wenn nicht die unterste Ebene erreicht wurde, ruft sich die Funktion nochmals
// rekursiv auf, um zu prüfen, ob noch Artikel einer unteren Preiseben gekauft
// werden können, um eine geringere Verlustdifferenz zu erhalten...
if(ebene>1 && best_differenz>0)
{
  berechne(ebene-1,c_anzahl,c_rest_budget,max);
}
// Wenn die unterste Ebene erreicht wurde, also der billigste Artikel,
// wird nochmals geprüft, ob eine eine geringere Verlustdifferenz
// erzielt wurde als bei einer bereits vorher ausprobierten Artikelkombination...
if(ebene==1)
{
  // Prüfung ob bei diesem Fall (=Artikelkombination) eine geringere Verlustdifferent
  // erzielt wurde als bei einer bereits vorher ausprobierten Artikelkombination...
  if((best_differenz>c_rest_budget) && (c_rest_budget-preis[ebene-1]<preis[ebene-1]))
  {
    // Übernehmen der neuen Artikelstruktur...
    for(int j=0;j<ebene;j++)
    {
      best[j] = c_anzahl[j];
    }
    // Die geringere Verlustdifferenz übernehmen als bisher geringste Verlustdifferenz...
    best_differenz = c_rest_budget;
  }
  // Beenden der Funktion (= wichtig, da es eine rekursive Funktion ist, in der man sich
  // befindet)
  return;
}
```

```
  // Prüfung ob in dieser Ebene der Artikelkombination ein Artikel vorhanden ist,
  // da nun durch Verkauf des Artikels dieser Ebene noch Artikelkombinationen
  // niedriger Ebenen ausporbiert werden müssen...
  while(c_anzahl[ebene-1]!=0 && best_differenz>0)
  {
    // Artikel dieser Ebene verkaufen...
    c_anzahl[ebene-1]--;
    // ... und Gurschrift des Betrages durch den Verkauf auf das aktuelle Rest-Budget
    c_rest_budget = c_rest_budget + preis[ebene-1];
    // Zuruecksetzen der Artikelkombination (=clearen), da nun eine neue Artikelkombination
    // ausprobiert wird...
    for(int p=0; p<ebene-1;p++)
    {
      c_anzahl[p] = 0;
    }
    // erneuter Aufruf der Funktion...
    berechne(ebene-1,c_anzahl,c_rest_budget,max);
  }
  // Beenden der Funktion (= wichtig, da es eine rekursive Funktion ist, in der man sich
befindet)
  return;
}

int main()
{
  // Array, in dem die Artikelkombination des Vormonats gespeichert wird, um spaeter
  // durch einen Verrgleich mit der Monatsaktuellen Artikelkombination vergleichen zu
  // koennen -> Wichtig fuer die spatere Ausgabe des Plans welcher Artikel verkauft bzw.
  // gekauft werden muss im entsprechenden Monat
  int alt_anzahl[MAX];

  // Array, in dem die gespeichert wird, ob der entsprechende Artikel
  // aktuell vorliegt, oder nicht
  int anzahl[MAX];

  // anzahl_items -> Zählvariable für die Anzahl der zu besorgendenen Artikel
  int anzahl_items=0;

  // budget -> Montliches Verfügbares Budget
  int budget;

  // max_budget -> Merkvariable in der das monatlich maximal verfügbare Budget gespeichert
  //               wird... -> notwendig um die geringste Verlustdiffernez zu berechnen
  int max_budget=0;

  // round -> Zaehlvariable, welche die Anzahl der Monate mitzählt...
  int round = 0;

  // eingabe -> Variable, die für Menüauswahl des Benutzers benötigt wird
  char eingabe;

  // status -> Notwendig um Schleife zu verlassen, wenn alle Artikel besorgt werden konnten
  bool status = true;
```

```
// Buffervariable fuer Schleifenpruefungen
bool imp_status = false;

// Buffervariable für das spaetere Zeilenweise einlesen der Textdatei
char puffer[BUF];

// Deklaration der Variable i
int i=0;

// Deklaration eines File-Pointers um spaeter die Textdatei ansprechen zu koennen
FILE *datei;

// Loescht (=cleart) die Konsolenausgabe
system("CLS");
cout << "++++++++++++++++++++++" << endl;
cout << "Bitte waehlen Sie aus:" << endl;
cout << "++++++++++++++++++++++" << endl;
cout << endl;
cout << "m = manuelle Eingabe" << endl;
cout << "i = Werte importieren aus Textdatei" << endl;
// So lange nicht die Taste "m" oder "i" gedrueckt wurde, wird das Auswahlmenue
// weiterhin auf der Konsole angezeigt (= fussgesteuerte Schleife)
do
{
  eingabe = getch();
  if(eingabe=='m')
  {
    imp_status = true;
    // So lange nicht die Taste "e" gedrueckt wurde, wird das Auswahlmenue wieder
    // neu angezeigt auf der Konsole (= fussgesteuerte Schleife)
    do
    {
      // Loescht (=cleart) die Konsolenausgabe
      system("CLS");
      cout << "Bitte geben Sie ihr monatliches Budget an: ";
      // Einlesen des zur Verfuegung stehenden monatlichen Budgets
      cin >> budget;
      // Loescht (=cleart) die Konsolenausgabe
      system("CLS");
      cout << "++++++++++++++++++++++" << endl;
      cout << "Bitte waehlen Sie aus:" << endl;
      cout << "++++++++++++++++++++++" << endl;
      cout << endl;
      cout << "a = Neuen Artikelwert eingeben (maximal: " << MAX << ")" << endl;
      cout << "e = Artikeleingabe beenden" << endl;
      // Wartet darauf, dass eine Taste gedrueckt wird
      eingabe = getch();
      // Prüfung ob die Taste "a" gedrueckt wurde um einen neuer Artikel einzugeben
      if(eingabe=='a')
      {
        // Loescht (=cleart) die Konsolenausgabe
        system("CLS");
        cout << "Bitte geben Sie nun den Preis des zu beschaffenden Gegenstandes an: ";
        // Einlesen des Preises fuer den neuen Artikel
        cin >> preis[anzahl_items];
```

```
        // Zaehlvariable fuer die Gesamtanzahl der Artikel um 1 erhoehen
        anzahl_items++;
      }
    }
    // So lange nicht die Taste "e" gedrueckt wurde, wird das Auswahlmenue wieder
    // neu angezeigt auf der Konsole
    while(eingabe != 'e');
  }
  // Pruefung ob die Taste "i" gedrueckt wurde
  if(eingabe=='i')
  {
    // Setzen des Wertes der Variable imp_status auf true
    imp_status = true;
    // Loescht (=cleart) die Konsolenausgabe
    system("CLS");
    cout << "Bitte geben Sie nun den korrekten Pfad zur Textdatei an:" << endl;
    cout << "(z.B. C:\\import.txt oder wenn die Datei im selben Ordner liegt import.txt)" << endl;
    // Einlesen des Pfades der Textdatei
    cin >> puffer;
    // Oeffnen der Textdatei im Lesemodus
    datei = fopen(puffer,"r");
    // Zeilenweise einlesen aus der Textdatei,bis das Dateiende erreicht wurde
    while(!feof(datei))
    {
      // Zeile aus der Textdatei einlesen
      fgets(puffer,BUF,datei);
      // Pruefung ob es die 1.Zeile ist, da dort das monatliche Budget steht...
      if(i==0)
      {
        // Einlesen des monatlichen Budgets fuer die Artikel
        budget = atoi(puffer);
        // Wert der Variable i um den Wert 1 erhoehen
        i++;
      }
      // ...andernfalls den Artikelpreis einlesen
      else
      {
        // Einlesen des Preises fuer den neuen Artikel
        preis[anzahl_items] = atoi(puffer);
        // Zaehlvariable fuer die Gesamtanzahl der Artikel um 1 erhoehen
        anzahl_items++;
      }
    }
  }
}
while(!imp_status);
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++
// +Ab hier beginnt die eigentliche Berechnung
// ++++++++++++++++++++++++++++++++++++++++++++++++
// Algorithmus zum sortieren aller eingegebenen Betraege in aufsteigender Reihenfolge...
for(int j=0; j<anzahl_items-1; j++)
{
  for(int k=0; k<anzahl_items-1-j; k++)
  {
    // Pruefung ob der Preis des Artikels der rechts (=also weiter hinten) steht
    // geringer ist, da nun beide Artikel getauscht werden muessen
    if(preis[k]>preis[k+1])
    {
      // In den folgenden 3 Zeilen werden einfach nur die beiden Artikel in der
      // Reihenfolge getauscht
      int buffer = preis[k+1];
      preis[k+1] = preis[k];
      preis[k] = buffer;
    }
  }
}
// Pruefung ob mit dem monatliche Budget ueberhaupt der billigste Artikel gekauft
// werden kann...ist dies nicht der Fall, so kann keiner der Artikel besorgt werden
// -> Programm kann beendet werden....
if(budget<preis[0])
{
  // Loescht (=cleart) die Konsolenausgabe
  system("CLS");
  cout << "Achtung: Ihr Monatsbudget liegt unterhalb des niedrigsten Artikelpreises" << endl;
  cout << "         Daher kann keiner der Artikel besorgt werden." << endl;
  cout << endl << endl;
  cout << "Bitte eine beliebige Taste druecken um das Programm zu beenden." << endl;
  // Beenden des Programmes nach druecken einer beliebigen Taste
  getch();
  return 0;
}
// ...verlaeuft die Pruefung negativ, so koennen auf jeden Fall alle Artikel besorgt
// werden und die Berechnung kann beginnen...
else
{
  do
  {
    // Sichern der Artikelstruktur des Vormonats...
    for(int j=0;j<anzahl_items;j++)
    {
      alt_anzahl[j] = best[j];
    }
    // Monate mitzaehlen (also pro Durchlauf die Moantszahl hier um 1 erhoehen
    round++;
    // Berechnung des Maximalen neuen Budgets für den aktuellen Monat durch
    // den Verkauf aller Artikel, in deren Besitz man ist...
    max_budget = budget;
    for(int j=0; j<anzahl_items; j++)
    {
      max_budget = max_budget + best[j] * preis[j];
    }
```

```
// Zuruecksetzen, also clearen der Variablen...
for(int j=0; j<anzahl_items; j++)
{
  best[j] = 0;
  anzahl[j] = 0;
}
best_differenz = max_budget;
// Aufruf der rekursiven Funktion, um alle moeglichen Artikelkombinationen
// inklusive Preisdifferenz zu berechnen und sich jeweils die Artikelkombination
// mit der geringsten Verlustdifferenz zu merken...
// ++++++++++++++++++++++++++++++
// Benoetigte Uebergabeparameter:
// ++++++++++++++++++++++++++++++
// anzahl_items  -> Anzahl der zu besorgenen Artikel
// anzahl      -> Merkvariable die notwenidig ist um alle Artikelkombinationen zu
//                  durchlaufen
// max_budget    -> Uebergabe des maximalen Budgets fuer diesen Monatsdurchlauf
// anzahl_items  -> Anzahl der zu besorgenen Artikel
berechne(anzahl_items,anzahl,max_budget,anzahl_items);
// Variable des Typs Integer(=Ganzzahl) die als Zwischenspeicher (=Buffer) benoetigt
// wird und in der in der folgenden Schleife mitgezaehlt wird, wieviel Artikel in diesem
// Monatsdurchlauf bereits gekauft werden konnten
int buffer=0;
for(int j=0; j<anzahl_items; j++)
{
  // Liegt dieser Artikel vor in diesem Monatsdurchlauf...
  if(best[j]==1)
  {
    //... so wird die Variable "buffer" um den Wert 1 erhoeht
    buffer++;
  }
}
// Ausgabe des Plans auf dem Bildschirm fuer jeden Monat in der Reihenfolge...
// Monatsnummer -> zurueckzugebene Artikel -> zu kaufende Artikel
cout << "Monat: " << round << endl;
cout << "Verkauf: ";
// Durchlauf aller Artikel
for(int j=0; j<anzahl_items; j++)
{
  // Pruefung ob im Vormonat der Artikel in der Artikelstruktur vorhanden war,
  // aber im aktuellen Monat in der Artikelstruktur nicht mehr vorkommt...
  if((alt_anzahl[j] == 1) && (best[j]==0))
  {
    // ...Ausgabe des Artikelpreises der verkauft werden muss in diesem Monat
    cout << preis[j] << " ; ";
  }
}
cout << endl << "Kauf: ";
// Durchlauf aller Artikel
for(int j=0; j<anzahl_items; j++)
{
  // Pruefung ob im Vormonat der Artikel in der Artikelstruktur nicht vorhanden war,
  // aber im aktuellen Monat in der Artikelstruktur vorkommt...
  if((best[j]==1) && (alt_anzahl[j]==0))
  {
```

```
          // ...Ausgabe des Artikelpreises der gekauft werden muss in diesem Monat
          cout << preis[j] << " ; ";
        }
      }
      cout << endl << endl;
      // Pruefung ob schon jeder Artikel einmal gekauft werden konnte in diesem
Monatsdurchlauf
      if(buffer==anzahl_items)
      {
        // Liegen alle Artikel bereits in diesem Monatsdurchlauf vor, so wird der status
        // aus "false" gesetzt damit die Schleife fuer den naechsten Monat weiter unten
        // im Programm abgebrochen wird
        status = false;
      }
    }
    // Wurden in diesem Monatsdurchlauf noch nicht alle Artikel besorgt, so behinhaltet die
    // Boolche Variable "status" immer noch den Wert "true" und somit wird ein weiterer
    // Monatsdurchlauf ausgefuehrt
    while(status);
  cout << endl << endl;
  cout << "Es koennen alle Artikel nach " << round << " Monaten besorgt werden !!!" << endl;
  cout << endl << endl;
  cout << "Bitte eine beliebige Taste druecken um das Programm zu beenden." << endl;
  // Beenden des Programmes nach druecken einer beliebigen Taste
  getch();
  }
  return 0;
}
```

3. Aufgabe 2: Robot Dressing

3.1. Lösungsidee

Dem Computer muss auf irgendeine Art und Weise klar gemacht werden, dass alle Kleidungsstücke die eingegeben wurden, untereinander in einer bestimmten Abhängigkeit voneinander stehen bzw. stehen können. Also ist es meiner Meinung nach Ratsam sich eine eigene Struktur zu erstellen, mit der man ein Kleidungsstück am besten beschreiben kann.
So sollte eine Struktur vom Typ „*kleidungsstueck*“ auf alle Fälle den Namen des Kleidungsstückes enthalten und eine Vorrichtung, mit deren man die Kleidungsstücke in Relation zueinander setzen kann.
Weiterhin währe es sehr empfehlenswert sich in dieser Struktur zu merken, ob das Kleidungsstück später bereits ausgegeben wurde oder nicht, da hiervon abhängig sein wird, ob das Kleidungsstück bei der Anziehreihenfolge noch beachtet werden muss oder nicht, da es bereits ausgegeben wurde.
Um die Kleidungsstücke untereinander überhaupt eindeutig in Relation setzen zu können, wäre eine weitere Vorrichtung notwendig, durch die man ein Kleidungsstück eindeutig identifizieren kann, wie z.B. die Identifikation durch eine einmalig verwendete Zahl.

Würde man eine solche Struktur vom Typ „*kleidungsstueck*“ erstellen, so wäre es auch sehr einfach die Angaben der Reihenfolge vom Benutzer einzulesen, da dieser alle Kleidungsstücke durch die Angabe von 2 Zahlen untereinander in Relation setzen könnte. Die Ermittlung der Gesamtreihenfolge wäre hierbei einfach zu realisieren, indem man die Kleidungsstücke ohne Vorgänger einfach ausgibt und in den verbleibenden Kleidungsstücken alle Relationen zu bereits ausgegebenen Kleidungsstücken löscht. Würde man diesen Vorgang oft genug wiederholen, so werden nach und nach immer weitere Kleidungsstücke ohne Vorgänger sein, die dann ausgegeben werden können, bis die Gesamtreihenfolge fertig gestellt ist.
An dieser Stelle würde auch die Merkvorrichtung der Struktur ins Spiel kommen, durch die einsichtig ist, ob ein Kleidungsstück bereits ausgegeben wurde, da sich nur durch diese Vorrichtung unterschieden ließe, ob ein vorhandenes Kleidungsstück ohne Vorgänger schon ausgegeben wurden oder noch ausgegeben werden muss.

3.2. Programm–Dokumentation

Die Lösungsidee wurde mit der Programmiersprache C++ in Form einer Konsolenanwendung realisiert. Es besteht aus der Datei „*Aufgaben 2/main.cpp*“.
Das aktuelle Programm wurde auf eine maximale Anzahl von 9 verschiedenen Kleidungsstücken beschränkt, kann jedoch jederzeit auf einfachste Weise erweitert werden, indem man die Begrenzung in der Zeile „*#define MAX 9*“ auf einen gewünschten Wert verändert.

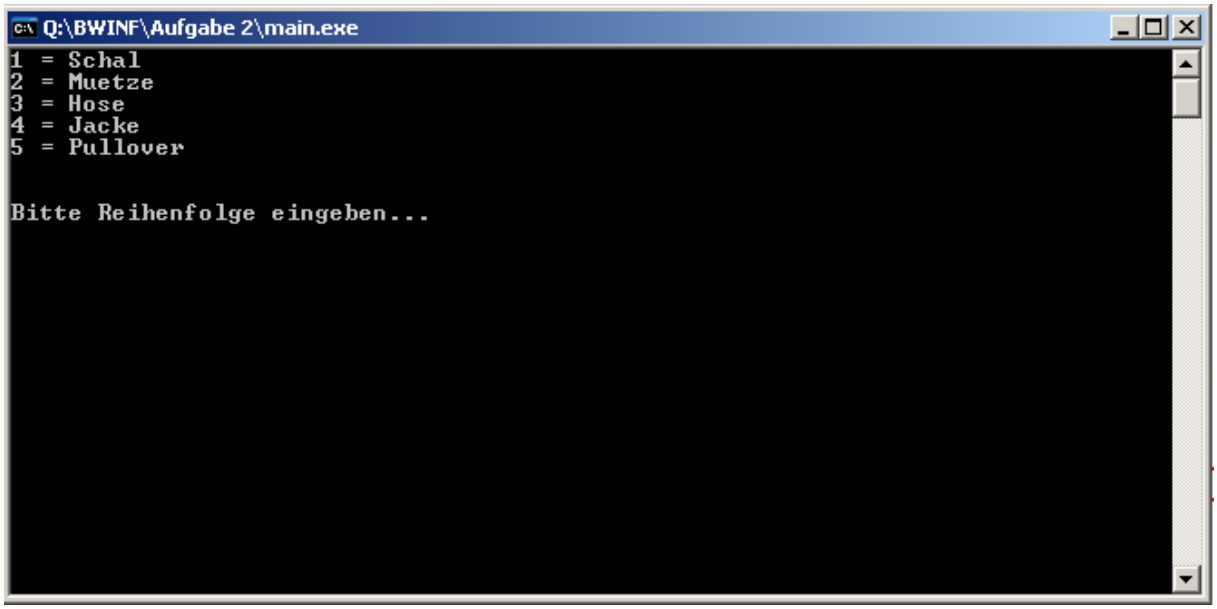
Nach reiflicher Überlegung habe ich mich dazu entschlossen ein Kleidungsstück aufgrund des Lösungsansatzes von oben durch nachfolgende Struktur des Typs „*kleidungsstueck*“ zu beschreiben:

```
struct kleidungsstueck
{
        char name[30];
        int nummer;
        int vorgaenger[MAX];
        bool ausgegeben;
};
```

Durch diese Struktur werden alle in der Lösungsidee gestellten Bedingungen berücksichtigt und abgedeckt.
Der Name eines Kleidungsstückes wurde auf 30 Zeichen inklusive dem abschließenden ’\0’ beschränkt.
Damit ein Kleidungsstück eindeutig identifizierbar ist und somit in Relation zu einem anderen Kleidungsstück gesetzt werden kann, wurde die Variable „*nummer*“ der Struktur hinzugefügt, die eine eindeutige Nummer enthält.
Das Array „*vorgaenger*“ wird im Programmverlauf von vorne beginnend mit den eindeutigen Nummern anderer Kleidungsstücke gefüllt, sobald diese in Relation mit dem aktuellen Kleidungsstück gesetzt werden. Dadurch kann im späteren Programmverlauf genau festgestellt werden, ob ein Kleidungsstück bereits in der Gesamtreihenfolge ausgegeben werden darf.

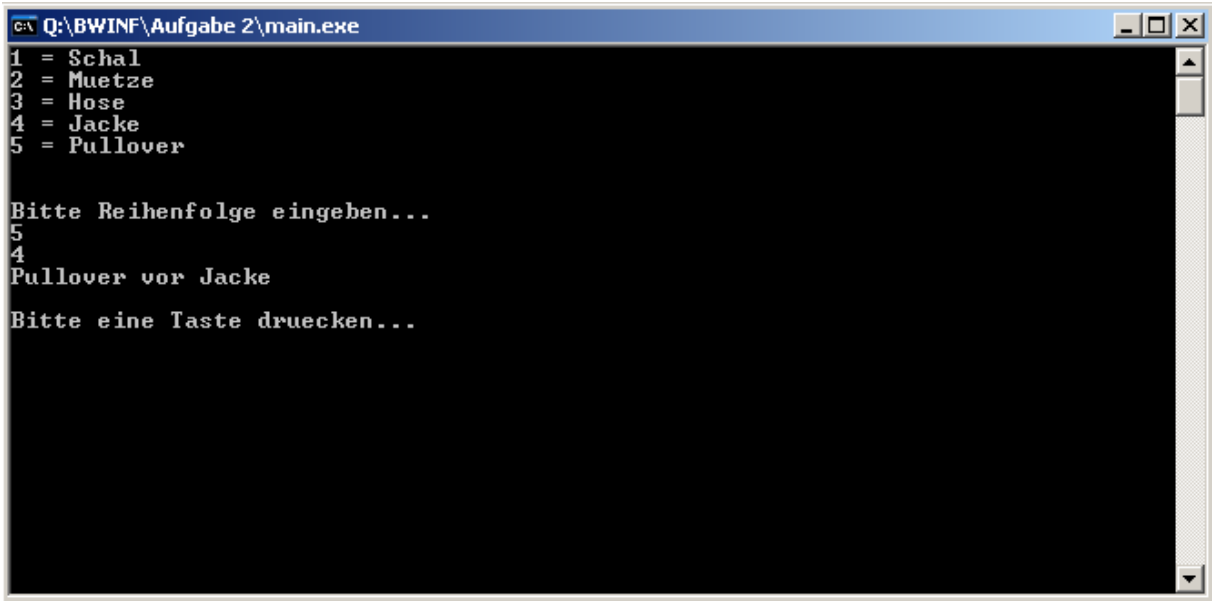
Zu Beginn des Programms muss der Benutzer angeben, wie viele verschiedene Kleidungsstücke er eingeben möchte. Das Programm ist hierbei darauf beschränkt maximal 9 Eingaben zuzulassen. Im Anschluss daran wird zunächst sichergestellt durch die Funktion „*clear_all_data()*“, dass auch alle Datensätze leer sind. Nun folgt über eine Programmschleife das einlesen der Kleidungsstücke, wobei gleichzeitig jedem eingegebenen Kleidungsstück eine eindeutige Identifikationsnummer zugewiesen wird.
Wurden alle Kleidungsstücke korrekt durch den Benutzer eingegeben, so erscheint ein erstes Auswahlmenü, in dem der Benutzer sich wahlweise durch drücken der Taste „a“ anzeigen lassen kann, wie er nun fortzufahren hat, um die Kleidungsstücke in Relation setzen zu könne, d.h. die Angaben zur Reihenfolge eingeben kann. Durch drücken der Taste „b“ gelangt der Benutzer in den Programmteil, wo er nun die Angaben zur Reihenfolge eingeben kann. Um ein einfach einzulesendes Beschreibungsformat zur Eingabe der Relationen zu gewährleisten, wird zunächst eine Übersicht eingeblendet, in der die Identifikationsnummern mit dazugehörigem Namen des entsprechenden Kleidungsstückes zu sehen sind. Nun braucht der Benutzer nur noch die beiden Identifikationsnummern derjenigen Kleidungsstücke einzugeben, die er in Relation zueinander setzen möchte.

Hier nun ein Screenshot bzw. eine kurze Beschreibung zur Eingabeprozedur der Relationen:



(Programmteil zur Eingabe der Relationen)

Möchte der Benutzer nun z.B. die Relation eingeben, dass zuerst der Pullover angezogen werden muss, bevor man die Jacke anzieht, so müsste hier die Eingabe der Zahlen „5“ und „4“ erfolgen und es würde folgendes auf dem Bildschirm anschließend zu sehen sein:



(Programmteil bei der Eingabe der Relationen / Pullover vor Jacke)

Nach dem drücken einer beliebigen Taste übernimmt das Programm nun selbständig die Relationsreihenfolge und der Benutzer kann im folgenden Menü auswählen, ob er eine weitere Reihenfolge eingeben möchte, oder ob die Berechnung der Gesamtreihenfolge beginnen soll.
Die Berechnung der Gesamtreihenfolge ist eigentlich nicht schwer. Über eine rekursive Funktion werden nacheinander alle Kleidungsstücke durchlaufen. Es wird geprüft, ob ein Kleidungsstück ohne Vorgänger ist, und somit ausgegeben werden könnte. Da aber

womöglich noch andere Kleidungsstücke vorhanden sein könnten, die ebenfalls ohne Vorgänger sind, und mit deren man eine weitere Ankleidungsreihenfolge bilden könnte, ebenfalls berücksichtig werden, ruft sich die Funktion in rekursiver Weise nochmals selbst auf, um mithilfe dieser Vorrichtung auch wirklich alle möglichen Ankleidereihenfolgen zu ermitteln.
Befindet man sich in der letzten Ebene, d.h. wenn nur noch ein Kleidungsstück übrig geblieben ist, so wird die gesamte gerade ermittelte Ankleidereihenfolge in einen Buffer geschrieben um später alle ermittelten Ankleidreihenfolgen ausgeben zu können. Die Funktion wird daraufhin beendet und kehrt zur Überfunktion zurück, wo nun nochmals geprüft wird, ob noch weitere Kleidungsstücke ohne Vorgänger vorhanden sind. So werden nach und nach alle möglichen Ankleidereihenfolgen ermittelt.
Wurden alle Ankleidereihenfolgen ermittelt, ist natürlich auch automatisch die Funktion beendet und es werden nun alle gefunden Ankleidereihenfolgen die im Buffer stehen nacheinander durchlaufen, um sie auf dem Bildschirm auszugeben.
Nun sieht der Benutzer also alle möglichen Ankleidreihenfolgen auf dem Bildschirm und kann durch drücken einer weiteren beliebigen Taste das Programm verlassen.

3.3. Programm-Ablaufsprotokoll

3.3.1. Probelauf mit den Daten aus der Aufgabenstellung

Kleidungsstücke: Bluse, Handschuhe, Hose, Jacke, Mütze, Pullover, Schal, Schuhe, Strümpfe

Angaben zur Reihenfolge:
die Strümpfe vor den Schuhen
Die Strümpfe vor der Hose
die Hose vor den Schuhen
die Hose vor dem Pullover
die Bluse vor der Hose
die Bluse vor dem Pullover
der Pullover vor der Jacke
die Hose vor der Jacke
der Schal vor der Jacke
die Jacke vor den Handschuhen
der Pullover vor dem Schal
keine Angabe für die Mütze

```
Wie viele Kleidungsstuecke wollen Sie eingeben ?
(Es sind maximal 9 moeglich)

9

Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Bluse
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Handschuhe
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Hose
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Jacke
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Muetze
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Pullover
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Schal
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Schuhe
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Struempfe

++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe
```

```
Bitte Reihenfolge eingeben...
9
8
Struempfe vor Schuhe

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
9
3
Struempfe vor Hose

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
3
8
Hose vor Schuhe
```

```
Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
3
6
Hose vor Pullover

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
1
3
Bluse vor Hose

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
```

```
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
1
6
Bluse vor Pullover

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
6
4
Pullover vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
```

```
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
3
4
Hose vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe


Bitte Reihenfolge eingeben...
7
4
Schal vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe
```

Bitte Reihenfolge eingeben...
4
2
Jacke vor Handschuhe

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Bluse
2 = Handschuhe
3 = Hose
4 = Jacke
5 = Muetze
6 = Pullover
7 = Schal
8 = Schuhe
9 = Struempfe

Bitte Reihenfolge eingeben...
6
7
Pullover vor Schal

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
c

Gesamtreihenfolgen:

Bluse vor Muetze vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Bluse vor Muetze vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Bluse vor Muetze vor Struempfe vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Bluse vor Muetze vor Struempfe vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Bluse vor Muetze vor Struempfe vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Muetze vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Bluse vor Struempfe vor Hose vor Muetze vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Bluse vor Struempfe vor Hose vor Muetze vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Muetze vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Muetze vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Muetze vor Schal vor Jacke vor Handschuhe vor Schuhe

Bluse vor Struempfe vor Hose vor Pullover vor Muetze vor Schal vor Jacke vor Schuhe vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Muetze vor Schal vor Schuhe vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Muetze vor Schuhe vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Muetze vor Schuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe vor Muetze

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Muetze vor Handschuhe vor Schuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Muetze vor Schuhe vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe vor Muetze

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Muetze vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Muetze vor Jacke vor Handschuhe vor Schuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Muetze vor Jacke vor Schuhe vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Muetze vor Schuhe vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe vor Muetze

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Muetze vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Schuhe vor Muetze vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schuhe vor Muetze vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe vor Muetze

Bluse vor Struempfe vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Muetze vor Handschuhe

Bluse vor Struempfe vor Hose vor Pullover vor Schuhe vor Schal vor Muetze vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Schuhe vor Muetze vor Pullover vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Schuhe vor Pullover vor Muetze vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe vor Muetze

Bluse vor Struempfe vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Muetze vor Handschuhe

Bluse vor Struempfe vor Hose vor Schuhe vor Pullover vor Schal vor Muetze vor Jacke vor Handschuhe

Bluse vor Struempfe vor Muetze vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Bluse vor Struempfe vor Muetze vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Bluse vor Struempfe vor Muetze vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Bluse vor Struempfe vor Muetze vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Bluse vor Struempfe vor Muetze vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Muetze vor Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Muetze vor Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Muetze vor Bluse vor Struempfe vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Muetze vor Bluse vor Struempfe vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Muetze vor Bluse vor Struempfe vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Muetze vor Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Muetze vor Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Muetze vor Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Muetze vor Struempfe vor Bluse vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Muetze vor Struempfe vor Bluse vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Muetze vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Struempfe vor Bluse vor Hose vor Muetze vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Struempfe vor Bluse vor Hose vor Muetze vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Muetze vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Muetze vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Muetze vor Schal vor Jacke vor Handschuhe vor Schuhe

Struempfe vor Bluse vor Hose vor Pullover vor Muetze vor Schal vor Jacke vor Schuhe vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Muetze vor Schal vor Schuhe vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Muetze vor Schuhe vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Muetze vor Schuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe vor Muetze

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Muetze vor Handschuhe vor Schuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Muetze vor Schuhe vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe vor Muetze

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Muetze vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Muetze vor Jacke vor Handschuhe vor Schuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Muetze vor Jacke vor Schuhe vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Muetze vor Schuhe vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe vor Muetze

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Muetze vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schal vor Schuhe vor Muetze vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schuhe vor Muetze vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe vor Muetze

Struempfe vor Bluse vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Muetze vor Handschuhe

Struempfe vor Bluse vor Hose vor Pullover vor Schuhe vor Schal vor Muetze vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Schuhe vor Muetze vor Pullover vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Schuhe vor Pullover vor Muetze vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe vor Muetze

Struempfe vor Bluse vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Muetze vor Handschuhe

Struempfe vor Bluse vor Hose vor Schuhe vor Pullover vor Schal vor Muetze vor Jacke vor Handschuhe

Struempfe vor Bluse vor Muetze vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Struempfe vor Bluse vor Muetze vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Struempfe vor Bluse vor Muetze vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Struempfe vor Bluse vor Muetze vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Struempfe vor Bluse vor Muetze vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Struempfe vor Muetze vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Handschuhe vor Schuhe

Struempfe vor Muetze vor Bluse vor Hose vor Pullover vor Schal vor Jacke vor Schuhe vor Handschuhe

Struempfe vor Muetze vor Bluse vor Hose vor Pullover vor Schal vor Schuhe vor Jacke vor Handschuhe

Struempfe vor Muetze vor Bluse vor Hose vor Pullover vor Schuhe vor Schal vor Jacke vor Handschuhe

Struempfe vor Muetze vor Bluse vor Hose vor Schuhe vor Pullover vor Schal vor Jacke vor Handschuhe

Bitte eine beliebige Taste druecken um das Programm zu beenden.

Probelauf mit folgenden Daten

Kleidungsstücke: T-Shirt, Pullover, Jacke, Schal, Mütze

Angaben zur Reihenfolge: das T-Shirt vor dem Pullover
der Pullover vor der Jacke
die Jacke vor dem Schal
das T-Shirt vor der Jacke
keine Angabe für die Mütze

```
Wie viele Kleidungsstuecke wollen Sie eingeben ?
(Es sind maximal 9 moeglich)

5

Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: T-Shirt
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Pullover
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Jacke
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Schal
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Muetze

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = T-Shirt
2 = Pullover
3 = Jacke
4 = Schal
5 = Muetze


Bitte Reihenfolge eingeben...
1
2
T-Shirt vor Pullover

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = T-Shirt
2 = Pullover
3 = Jacke
```

```
4 = Schal
5 = Muetze


Bitte Reihenfolge eingeben...
2
3
Pullover vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = T-Shirt
2 = Pullover
3 = Jacke
4 = Schal
5 = Muetze


Bitte Reihenfolge eingeben...
3
4
Jacke vor Schal

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = T-Shirt
2 = Pullover
3 = Jacke
4 = Schal
5 = Muetze


Bitte Reihenfolge eingeben...
1
3
T-Shirt vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++
```

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
c

Gesamtreihenfolgen:

T-Shirt vor Pullover vor Jacke vor Schal vor Muetze

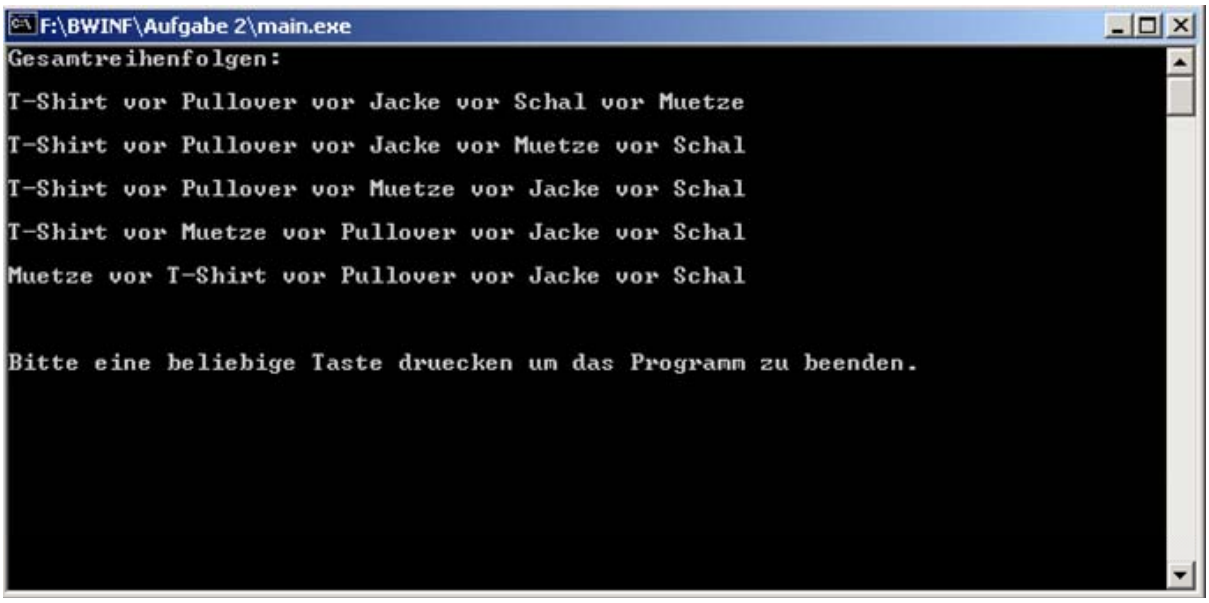
T-Shirt vor Pullover vor Jacke vor Muetze vor Schal

T-Shirt vor Pullover vor Muetze vor Jacke vor Schal

T-Shirt vor Muetze vor Pullover vor Jacke vor Schal

Muetze vor T-Shirt vor Pullover vor Jacke vor Schal

Bitte eine beliebige Taste druecken um das Programm zu beenden.



(Screenshot für die Bildschirmausgabe der 3.3.2)

3.3.2. Probelauf mit folgenden Daten

Kleidungsstücke: Schuhe, Hose, Pullover, Struempfe, Jacke

Angaben zur Reihenfolge: die Strümpfe vor der Hose
die Hose vor der Jacke
der Pullover vor der Jacke
Schuhe vor der Jacke
die Strümpfe vor den Schuhen
die Hose vor den Schuhen

```
Wie viele Kleidungsstuecke wollen Sie eingeben ?
(Es sind maximal 200 moeglich)

5

Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Schuhe
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Hose
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Pullover
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Struempfe
Bitte geben Sie den Namen des 1. Kleidungsstueckes ein: Jacke

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke


Bitte Reihenfolge eingeben...
4
2
Struempfe vor Hose

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke
```

```
Bitte Reihenfolge eingeben...
2
5
Hose vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke


Bitte Reihenfolge eingeben...
3
5
Pullover vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke


Bitte Reihenfolge eingeben...
1
5
Schuhe vor Jacke

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
```

c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke

Bitte Reihenfolge eingeben...
4
1
Struempfe vor Schuhe

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
b

1 = Schuhe
2 = Hose
3 = Pullover
4 = Struempfe
5 = Jacke

Bitte Reihenfolge eingeben...
2
1
Hose vor Schuhe

Bitte eine Taste druecken...

++++++++++++++++++++++
Bitte waehlen Sie aus:
++++++++++++++++++++++

a = Beispiel fuer Eingabeformat
b = Reihenfolge eingeben
c = Gesamtreihenfolge berechnen
c

Gesamtreihenfolgen:

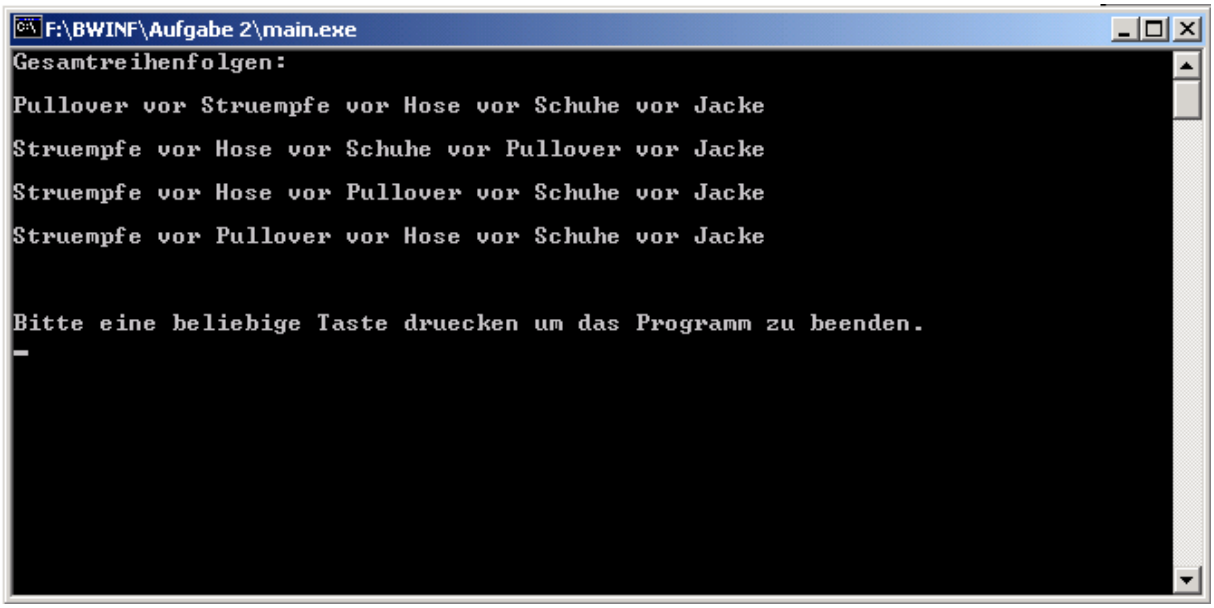
Pullover vor Struempfe vor Hose vor Schuhe vor Jacke

Struempfe vor Hose vor Schuhe vor Pullover vor Jacke

Struempfe vor Hose vor Pullover vor Schuhe vor Jacke

Struempfe vor Pullover vor Hose vor Schuhe vor Jacke

Bitte eine beliebige Taste druecken um das Programm zu beenden.



(Screenshot für die Bildschirmausgabe der 3.3.3)

3.4. Programm-Text

```
// 25. Bundeswettbewerb Informatik 2006/2007
// Aufgabe 2: Robot Dressing
//
// (c) Programmierung Andreas Albert

// Einbinden der benoetigten Programmbibliotheken
#include <conio.h>
#include <iostream>
using namespace std;

// Definierte Variable, anhand deren man die maximale Anzahl der Kleidungsstuecke leicht
// und unkompliziert aendern kann
#define MAX 9

// Definierte Variable, anhand deren man die maximale Anzahl der errechenbaren
// Kombinationsmoeglichkeiten leicht und unkompliziert aendern kann
#define MAX2 57000

// Definition einer eigenen Struktur, die sich besser eignet zur Speicherung der
// einzelnen Kleidungsstuecke
struct kleidungsstueck{
  // Name des Kleidungsstueckes
  char name[30];

  // Einmalige Nummer durch die das Kleidungsstueck eindeutig identifizierbar ist
  int nummer;

  // Array, in dem spaeter die einmaligen Nummern der Kleidungsstuecke gespeichert
  // werden,die angezogen werden muessen, bevor dieses Kleidungsstueck angezogen
  //werden kann
  int vorgaenger[MAX];

  // Status, der gesetzt wird, um spaeter abgleichen zu koennen, ob das entsprechende
  // Kleidungsstueck schon ausgeben wurde
  bool ausgegeben;
};

struct kombination{
  int array[MAX];
  //bool korrekt;
};
```

```
// Funktion, mit deren Hilfe man alle Datensaetze loeschen kann
void clear_all_data(kleidungsstueck *array)
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// *array -> alle Kleiderstuecke mit den Angaben zur Anziehreihenfolge
{
  for(int i=0; i<MAX; i++)
  {
    for(int j=0; j<=MAX-1; j++)
    {
      // Setzt alle Vorgaenger-Identifikationsnummern Standardmaessig auf den Wert "0",
      // da alle Datensaetze geloescht werden
      array[i].vorgaenger[j] = 0;
      // Status, der auf "false" gesetzt wird, da alle Datensaetze geloescht werden
      array[i].ausgegeben = false;
    }
  }
}

// Selbst geschrieben rekursive Funktion, um alle moeglichen Anziehreihenfolgen
// zu ermitteln und diese jeweils in der Speichervariablen "kombi" zu speichern
int berechne_kombi(kleidungsstueck kleider[MAX],int ebene,int anzahl,kombination*
kombi,int nr, int count)
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// kleider        -> alle Kleiderstuecke mit den Angaben zur Anziehreihenfolge
// ebene           -> aktuelle Suchebene (wichtig, da auf der letzten Ebenen die
//                 nr erhoeht werden muss um den Wert 1, da eine komplette
//                 Anziehreihenfolge dann vorliegt
// anzahl          -> Anzahl der verschiedenen Kleidungsstuecke
// kombi           -> Speicher in dem die einzelnen Anziehreihenfolgen gespeichert
//                 werden
// nr             -> gibt an in der wievielten Anziehreihenfolge man sich befindet
// count           -> Zaehler der die Nummer der untersten Ebenen angibt und nie
//                 veraendert wird
//
// ++++++++++++++++++++
// Rueckgabeparameter:
// ++++++++++++++++++++
// return         -> Aktuelle Nr der Reihenfolge
{
  // Buffer 1 fuer Kleidungsstuecke
  kleidungsstueck copy_kleider_1[MAX];

  // Schleife zum Kopieren der original Kleiderstuecke in...
  for(int i=0;i<anzahl;i++)
  {
    // ...Buffer 1 fuer Kleidungsstuecke
    copy_kleider_1[i] = kleider[i];
  }
```

```
// Alle Kleidungsstuecke der aktuellen Ebenen durchlaufen
for(int i=0;i<anzahl;i++)
{
  // Pruefung ob aktuelles Kleidungsstueck einen Vorgaenger besitzt...
  if(copy_kleider_1[i].vorgaenger[0]==0)
  {
    // ...wenn nicht, Kleidungsstucknummer uebernehmen in Kombination
    kombi[nr].array[ebene-1] = copy_kleider_1[i].nummer;
    // Alle Kleidungsstuecke der aktuellen Ebenen durchlaufen
    for(int j=0;j<anzahl;j++)
    {
      // Alle moeglichen Position der Vorgaenger durchlaufen
      for(int k=0;k<MAX;k++)
      {
        // Pruefung ob gerade verwendetes Kleidungstueck als Vorgaenger in dem
        // geraden geprueften Kleidungsstueck vorkommt...
        if(copy_kleider_1[j].vorgaenger[k] == copy_kleider_1[i].nummer)
        {
          // ...wenn ja, alle Vorgaengerpositionen durchlaufen...
          for(int l=k;l<=MAX;l++)
          {
            // ...und entsprechende Vorgaengernummer loeschen, indem die nachfolgenden
            // Vorgaengernummern um eine Stelle nach vorne kopiert werden
            copy_kleider_1[j].vorgaenger[l] = copy_kleider_1[j].vorgaenger[l+1];
          }
        }
      }
    }
    // Alle Kleidungsstuecke der aktuellen Ebenen durchlaufen...
    for(int j=i;j<anzahl-1;j++)
    {
      // Gerade verwendetes Kleidungsstueck ueberschreiben, indem alle ihm
      // folgenden Kleidungsstuecke um eine Position nach vorne verschoben werden
      copy_kleider_1[j] = copy_kleider_1[j+1];
    }
    // Pruefung ob mehr als 1 Rest-Kleidungsstueck vorhanden ist
    if(anzahl>1)
    {
      // Rekursiver Funktionsaufruf
      nr = berechne_kombi(copy_kleider_1,ebene+1,anzahl-1,kombi,nr,count);
    }
    // Alle Kleidungsstuecke der aktuellen Ebenen durchlaufen...
    for(int i=0;i<anzahl;i++)
    {
      // ...und in von Sicherungs-Buffer in den Bearbeitungs-Buffer uebertragen
      // da durch rekursivitaet die Werte in Unterfunktionen eventuell veraendert
      // wurden
      copy_kleider_1[i] = kleider[i];//copy_kleider_2[i];
    }
    // Pruefung ob die unterste Ebene erreicht wurde, da nun die Ankleidekombination
    // komplett ist
    if(ebene==count)
    {
```

```
      for(int l=0;l<ebene;l++)
      {
        if(kombi[nr].array[l]==0)
        {
          // Alle Vorgaengernummern der ermittelten Ankleidereihenfolge uebernehmen
          // aus der vorangehenden Anziehreihenfolge, da durch die rekursivitaet
          // der Wert 0 immer noch bei unveraenderten Positionen steht
          kombi[nr].array[l] = kombi[nr-1].array[l];
        }
      }
      // Aktuelle Kombinationsnummer um den Wert 1 erhoehen
      nr=nr+1;

      // Uebergabe der aktuellen Kombinationsnummer an die Oberfunktion
      return nr;
    }
  }
 }
 // Uebergabe der aktuellen Kombinationsnummer an die Oberfunktion
 return nr;
}

int main()
{
 // Array, das spaeter alle Kleidungsstuecke enthalten wird, die eingegeben wurden
 kleidungsstueck kleider[MAX];

 // Array, das spaeter alle Kleidungsstuecke-Kombinationen enthalten wird
 kombination kombi[MAX2];

 // Hier wird mitgezaehlt, wie viele Kleidungsstuecke bereits ausgegeben wurden
 int ausgabe = 0;

 // Die Menge der verschiedenen Kleidungsstuecke
 int anzahl;

 // Variable zur Abfrage einer gedrueckten Taste
 char eingabe;

 // Buffer-Variable die zur Zwischenspeicherung der 1.Identifikationsnummer benoetigt wird
 int eingabe_a;

 // Buffer-Variable die zur Zwischenspeicherung der 2.Identifikationsnummer benoetigt wird
 int eingabe_b;

 // Zuerst alle Datensaetze loeschen, um sicher zu stellen, dass keine Daten
 // vorhanden sind
 clear_all_data(kleider);

 // Schleife so lange wiederholen bis die Anzahl der einzugebenen Kleidungsstuecke
 // innerhalb des Maximal angebbaren Bereiches liegt
 do
 {
  // Loescht (=cleart) die Konsolenausgabe
  system("CLS");
```

```
  cout << "Wie viele Kleidungsstuecke wollen Sie eingeben ?" << endl;
  cout << "(Es sind maximal " << MAX << " moeglich)" << endl << endl;
  // Einlesen der Menge der Kleidungsstuecke
  cin >> anzahl;
 }
 while(anzahl>MAX);

 // Schleife, um alle Arten der Kleidungsstuecke einzulesen
 for(int i=0; i<anzahl; i++)
 {
  // Loescht (=cleart) die Konsolenausgabe
  system("CLS");
  cout << "Bitte geben Sie den Namen des " << i+1 << ". Kleidungsstueckes ein: ";
  // Einlesen des Namens eines Kleidungsstueckes
  cin >> kleider[i].name;
  // Zuteilung einer eindeutigen Identifikationsnummer fuer das gerade eingegebene
  // Kleidungsstueck
  kleider[i].nummer = i+1;
 }

 // Anzeigen des Menues, so lange die Gesamtreihenfolge noch nicht berechnet werden soll,
 // da der Benutzer noch weitere Ankleide-Reihenfolgen angeben moechte
 do
 {
  // Loescht (=cleart) die Konsolenausgabe
  system("CLS");
  cout << "++++++++++++++++++++++" << endl;
  cout << "Bitte waehlen Sie aus:" << endl;
  cout << "++++++++++++++++++++++" << endl;
  cout << endl;
  cout << "a = Beispiel fuer Eingabeformat" << endl;
  cout << "b = Reihenfolge eingeben" << endl;
  cout << "c = Gesamtreihenfolge berechnen" << endl;
  // Einlesen der Benutzereingabe
  eingabe = getch();
  // Pruefung ob der Benutzer die Taste "a" gedrueckt hat
  if(eingabe=='a')
  {
    // Loescht (=cleart) die Konsolenausgabe
    system("CLS");
    // Ausgabe des Beispiels, damit der benutzer sich ansehen kann, wie die
    // Steuerung zur Eingabe einer Ankleide-Reihenfolge funktioniert
    cout << "++++++++++++++++++++++++++" << endl;
    cout << "Beispiel fuer die Eingabe:" << endl;
    cout << "++++++++++++++++++++++++++" << endl << endl;
    cout << "Beziehungen bitte folgendermassen eingeben:"<< endl;
    cout << "1 = Hose" << endl;
    cout << "2 = Pullover" << endl << endl;
    cout << "Fuer die Eingabe \"Hose vor dem Pullover bitte 1 und dann 2 eingeben\"";
    cout << endl;
    cout << "Fuer die Eingabe \"Pullover vor der Hose bitte 2 und dann 1 eingeben\"";
    cout << endl;
    cout << endl;
    cout << "Bitte eine Taste druecken...";
```

```
    // Auf eine Tastureingabe durch den Benutzer warten
    getch();
  }
  // Pruefung ob der Benutzer die Taste "b" gedrueckt hat und somit eine Angabe zur
  // Reihenfolge eingelesen werden soll
  if(eingabe=='b')
  {
    // Loescht (=cleart) die Konsolenausgabe
    system("CLS");
    // Ausgebe der vorhandenen Kleidungsstuecke in der Form:
    // Identifikationsnummer = Name des Kleidungsstueckes
    for(int i=0; i<anzahl; i++)
    {
      cout << kleider[i].nummer << " = " << kleider[i].name << endl;
    }
    cout << endl << endl;
    cout << "Bitte Reihenfolge eingeben..." << endl;
    // Zwischenspeicherung der 1. Identifikationsnummer des Benutzers
    cin >> eingabe_a;
    eingabe_a = eingabe_a-1;

    // Zwischenspeicherung der 2. Identifikationsnummer des Benutzers
    cin >> eingabe_b;
    eingabe_b = eingabe_b-1;

    // Schleife um die erste Position innerhalb der Vorgaengerreihenfolge zu finden,
    // welche noch nicht mit einem Nachfolge-Kleiderstueck belegt wurde
    for(int j=0; j<anzahl; j++)
    {
      // Suche nach der naechsten freien Position innerhalb der Vorgaengerreihenfolge
      if(kleider[eingabe_b].vorgaenger[j]==0)
      {
        // an der ersten freien Position innerhalb der Vorgaengerreihenfolge die
        // Identifikationsnummerdes Nachfolge-Kleiderstueckes eintragen
        kleider[eingabe_b].vorgaenger[j]=eingabe_a+1;
        // Da eine freie Position bereits gefunden wurde innerhalb der Vorgaengerreihenfolge
        // kann an dieser Stelle aus der Schleife ausgestiegen werden
        break;
      }
    }
    // Ausgabe des Namens der zur 1.Identifikationsnummer gehoert
    cout << kleider[eingabe_a].name;
    cout << " vor ";
    // Ausgabe des Namens der zur 2.Identifikationsnummer gehoert
    cout << kleider[eingabe_b].name << endl;
    cout << endl << "Bitte eine Taste druecken...";
    // Auf eine Tastureingabe durch den Benutzer warten
    getch();
  }
}
// Nochmaliges Anzeigen des Menues, da die Gesamtreihenfolge noch nicht berechnet
// werden soll, da der Benutzer noch nicht die Taste "c" gedrueckt hat, um die Berechnung
// zu starten und womoeglich noch weitere Ankleide-Reihenfolgen angeben moechte
while(eingabe != 'c');
```

```
  // Loescht (=cleart) die Konsolenausgabe
  system("CLS");
  cout << "Gesamtreihenfolgen:" << endl << endl;
  // Buffervariable die den letzten Wert der rekursiven Berechnung auffaengt
  int buffer;
  // Berechnung aller moeglichen Anziehreihenfolgen ueber eine rekursive Funktion
  // (wichtig um zu wissen, wie viele Anziehreihenfolgen gefunden wurden)
  buffer = berechne_kombi(kleider,1,anzahl,kombi,0,anzahl);
  // Alle ermittelten Ankleidereihenfolgen durchlaufen...
  for(int z=0;z<buffer;z++)
  {
    for(int k=0;k<anzahl-1;k++)
    {
      // ... und ausgeben
      cout << kleider[kombi[z].array[k]-1].name << " vor ";
    }
    // ... und ausgeben
    cout << kleider[kombi[z].array[anzahl-1]-1].name;
    cout << endl << endl;
  }
  cout << endl << endl;
  cout << "Bitte eine beliebige Taste druecken um das Programm zu beenden." << endl;
  // Beenden des Programmes nach druecken einer beliebigen Taste
  getch();
  return 0;
}
```

4. Aufgabe3: HTML-Mobiles

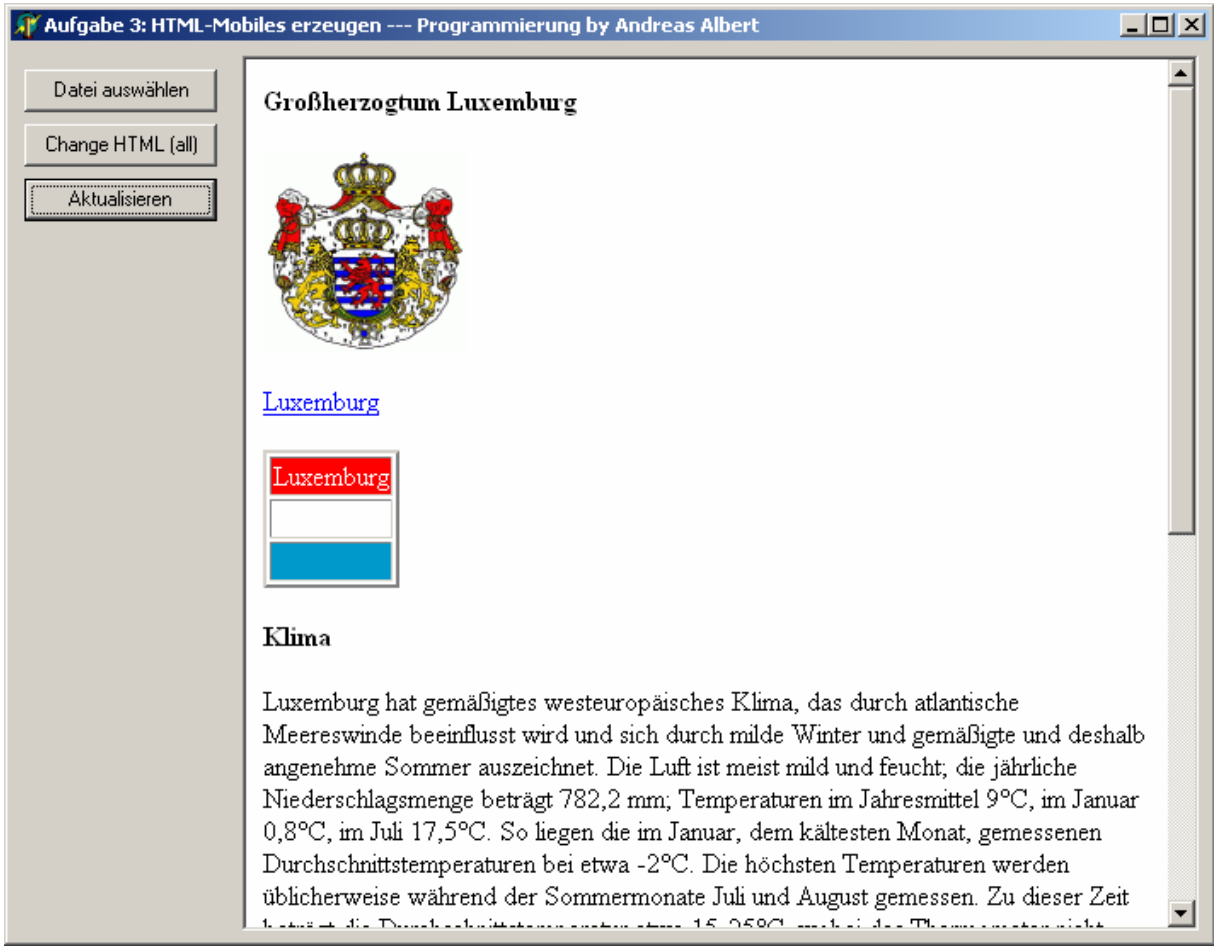
4.1. Lösungsidee

Mein erster Gedanke war es von Anfang an, sich klar zu überlegen, in welcher Art und Form die Änderungen in einem reinen HTML Dokument vorgenommen werden sollen. Dabei bin ich zu dem Entschluss gekommen, dass man eine HTML Datei im Prinzip ganz einfach in einen Art Texteditor laden könnte, um auf diese Art und Weise direkt auf den reinen HTML Quellcode zugreifen zu können. So müsse es sich doch programmierbar sein, anhand gewisser Kriteriumspunkte gewisse Änderungsarten realisieren zu lassen.
Elementare Änderungen, die sich auf den Inhalt beziehen sollten recht einfach und schlicht gehalten werden meiner Meinung nach. Daher dachte ich mir, mich darauf festzulegen, in diesem Bezug die Tabellenstärke und die Hintergrundfarbe einer Tabelle als zufällig Änderbar zu realisieren und gleichzeitig eine Änderung der Formatierung von Zeilenabschnitten vorzunehmen. Änderungen der Formatierung von Zeilenabschnitten sollen sich darauf beschränken, anhand eines Zufallswertes die Formatierung so zu ändern, dass der Zeilenabschnitt später entweder als Kursiv oder Fett formatiert erscheinen wird.
In Bezug auf Elementare Änderungen. Die sich auf die Struktur beziehen, habe ich mich dazu entschlossen die Position von Grafiken nach einem Zufallsprinzip zu verändern. Zwei weitere Strukturelle Änderungen sollten sein, dass ebenfalls die Position einer ganzen Tabelle oder auch die Position von Hyperlinks nach einem Zufallsprinzip verändert werden.

4.2. Programm-Dokumentation

Für die Umsetzung meiner Lösungsidee habe ich mich für die Programmiersprache Delphi entschieden, da ich die vorangegangenen Aufgaben jeweils in der Programmiersprache C++ gelöst habe und mir durch die Wahl einer 2.Programmiersprache mit Delphi ein neues Wissensgebiet erschließen möchte.
Als Entwicklungsumgebung viel meine Wahl auf den Borland Builder für Delphi, da dieser von Haus aus bereits wichtige Objektelemente mitbringt. Speziell für die Umsetzung meiner Lösungsidee für diese Aufgabe war es daher schon eine Erleichterung, dass ich auf Objektelemente zugreifen konnte wie z.B. Memofelder, Datei-Dialoge, Buttons usw.
Bei der Umsetzung der Lösungsidee bin ich zunächst hingegangen und habe mir ein vernünftiges und ansprechendes Oberflächendesign erstellt. So habe ich auf dem Formularelement eine Browserfläche erstellt, in welcher später die Webseite angezeigt werden wird bzw. kann. Des weiteren habe ich mir der Hilfe zweier Momofelder bedient und deren Sichtbarkeit für den späteren Benutzer auf „unsichtbar“ gestellt, da diese im Programm nur als Buffer-Elemente dienen werden um den Quelltext bearbeiten zu können und somit inhaltliche und strukturelle Änderungen schneller realisieren zu können.
Ein weiteres Element, welches ich einfügen musste war der ein Dialog-Element namens „OpenDialog“, damit später im Programm der Benutzer eine zu bearbeitende Webseite auswählen kann über den üblichen Windows Datei-Öffnen-Dialog.

Daraus ergab sich folgende Programmoberfläche:



(Programmoberfläche des fertigen Programmes)

Grundlegend hat das von mir entwickelte programmierte Programm 3 Funktionen, die durch drei Buttons realisiert wurden.
Zunächst der Button „Datei auswählen“. Durch drücken dieser Taste wird der „OpenDialog“ ausgeführt, wo der Benutzer eine Datei auswählen kann. Um es dem Benutzer zu vereinfachen nur HTML Webseiten im Auswahlfenster angezeigt zu bekommen, wurde die Eigenschaft „Filter“ der Komponente „OpenDialog1“ so eingerichtet, dass nur Dateien mit der für HTML Webseiten üblichen Dateiendungen „*.html“ und „*.htm“ angezeigt werden bzw. auswählbar sind.
Im nächsten Schritt kann dann über den Button „Change HTML (all)“ die elementaren Änderungen der HTML Webseite in zufälliger Reihenfolge eingeleitet werden. Hierbei wurde darauf verzichtet, vom Benutzer zufallswerte zu erfragen und dies in den Programmquelltext auszulagern, indem an entsprechender Stelle Zufalls-Generatoren implementiert wurden.
Da es sich um insgesamt 6 elementare Änderungen handelt, wird über einen Zufalls-Generator nun eine zufällige Reihenfolge ermittelt, in welcher die 6 Änderungsarten nacheinander ausgeführt werden sollen.
Prinzipiell ist es so, dass bei jeder Änderungsart zunächst die Webseite komplett in das „Memofeld 1“ geladen wird, dann die entsprechende Änderung durchgeführt wird, so dass der Quelltest der geänderten Webseite im „Memofeld 2“ erscheint und anschließend wieder in die ursprüngliche Datei gespeichert wird.
Nach Abschluss aller 6 Änderungsarten wird noch geprüft, ob während oder vor der Bearbeitung irgendwelche Leerzeilen im Quelltext vorhanden sind, um diese zu löschen.

Nun werde ich noch die Umsetzung der 6 elementaren Änderungen kurz erläutern. Zunächst möchte ich hier die 3 inhaltlichen Änderungen erläutern.

Die Änderung der Tabellenfarbe wird aufgrund der Prozedur „*inhalt_change_bgcolor()*“ vorgenommen. In dieser Prozedur wird der HTML Quellcode in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „*bgcolor="#*“ vorkommt. Sollte dies der Fall sein, so werden die nach dem „#“ folgenden 6 Ziffern (= Hexzahl) gelöscht und über einen Zufallsgenerator wird dann eine neue Zahl berechnet. Hierbei ist zu beachten, dass bewusst für diesen Zufallsgenerator die Zahl „16777217“ als Grenzwert gewählt wurde, da bei der Umrechnung vom Dezimalsystem in das Hexadezimalsystem dies der letzte Wert wäre, der im Hexadezimalsystem mit 6 Ziffern darstellbar ist.
Zum Abschluss wird die neue zufällige Hexadezimalzahl wieder eingefügt nach dem „#“ und diese Änderung ist mit dem abspeichern des Quellcodes erfolgreich abgeschlossen.

Eine weitere Änderung stellt die Veränderung der Tabellenstärke da. Dies geschieht im Programm mit Hilfe der Prozedur „*inhalt_change_border()*“.In dieser Prozedur wird der HTML Quellcode in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „*border="*“ vorkommt. Sollte dies der Fall sein, so wird die nach dem „*border="*“ folgende Ziffer gelöscht und über einen Zufallsgenerator wird dann eine neue Zahl berechnet. Hierbei ist zu beachten, dass bewusst für diesen Zufallsgenerator die Zahl „6“ als Grenzwert gewählt wurde, da bei HTML eine größere Zahl recht unsinnig wäre und in der Regel nicht gebraucht wird.
Zum Abschluss wird die neue zufällige Zahl wieder an entsprechender Stelle nach dem „*border="*“ eingefügt und diese Änderung ist ebenfalls mit dem abspeichern des Quellcodes erfolgreich abgeschlossen.

Eine weitere Änderung stellt die Veränderung des Schrifttyps für einen Abschnitt da. Dies geschieht im Programm mit Hilfe der Prozedur „*inhalt_change_schrifttype()*“.In dieser Prozedur wird der HTML Quellcode zunächst in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „<*p*>“ vorkommt. Sollte dies der Fall sein, so wird über einen Zufallsgenerator ermittelt, ob der Schrifttyp für diesen Abschnitt in „Fett“, „Kursiv“ oder „Standard“ angewandt werden soll. Gleichzeitig wird eine Variable um eins erhöht, damit das Programm nun weiß, dass der vor dem nächsten HTML-Tag „</*p*>“ diese eben angewandte Schrifttypänderung noch abgeschlossen werden muss durch das entsprechende schließende HTML-Tag. Ist dies geschehen, wird die Variable wieder um eins verringert, damit bei einem nun eventuell noch folgenden Abschnitt abermals eine Schrifttypänderung durchgeführt werden kann.
Wurde der gesamte Quellcode abgearbeitet und anschließend der geänderte Quellcode abgespeichert, so ist auch diese Änderung erfolgreich abgeschlossen.

Nun folgen noch die 3 strukturellen Änderungen.

Die Änderung der Position einer Grafik wird aufgrund der Prozedur „*struckt_change_image()*“realisiert. In dieser Prozedur wird der HTML Quellcode in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „<*img*“ vorkommt. Sollte dies der Fall sein, so wird eine Variable um den Wert 1 erhöht, damit das Programm weiß, dass alle Zeichen ausgeschnitten und gleichzeitig in einen Buffer zwischengespeichert werden müssen, so lange bis der HTML-Tag „<*img*“ durch ein „>“ abgeschlossen wurde.
Wurde das Quellcodefragment der Grafik erfolgreich in den Buffer verschoben, so wird nochmals der gesamte Quellcode durchlaufen um die Zeilennummern zu ermitteln in denen der HTML-Tag „<*body*>“ und „</*body*>“ vorkommt, um anhand dieser Information mittels einem Zufallsgenerators eine zufällige Position zu ermitteln, an welcher die Grafik wieder eingefügt werden soll. Dieser Vorgang ist nötig, um sicherzustellen, dass die Grafik innerhalb des Body-Tags eingefügt wird im Quellcode. Nach erfolgreichem einfügen der Grafik an einer neuen, zufälligen Position im Quellcode, wird dieser wiederum abspeichern und auch diese Änderungsart ist damit abgeschlossen.

Die Änderung der Position einer Tabelle wird aufgrund der Prozedur „*struckt_change_table()*“realisiert. In dieser Prozedur wird der HTML Quellcode in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „<*table*“ vorkommt. Sollte dies der Fall sein, so wird eine Variable um den Wert 1 erhöht, damit das Programm weiß, dass alle Zeichen ausgeschnitten und gleichzeitig in einen Buffer zwischengespeichert werden müssen, so lange bis der HTML-Tag „<*table*“ durch den HTML-Tag „</*table*>“ abgeschlossen wurde.
Wurde das Quellcodefragment der Tabelle erfolgreich in den Buffer verschoben, so wird nochmals der gesamte Quellcode durchlaufen um die Zeilennummern zu ermitteln in denen der HTML-Tag „<*body*>“ und „</*body*>“ vorkommt, um anhand dieser Information mittels einem Zufallsgenerators eine zufällige Position zu ermitteln, an welcher die Tabelle wieder eingefügt werden soll. Dieser Vorgang ist nötig, um sicherzustellen, dass die Tabelle innerhalb des Body-Tags eingefügt wird im Quellcode. Nach erfolgreichem einfügen der Tabelle an einer neuen, zufälligen Position im Quellcode, wird dieser wiederum abspeichern und auch diese Änderungsart ist damit abgeschlossen.

Die Änderung der Position eines Links wird aufgrund der Prozedur „*struckt_change_link()*“realisiert. In dieser Prozedur wird der HTML Quellcode in ein Memofeld importiert und dann in einer Schleife Zeilenweise durchlaufen. Hierbei wird dann pro Schleifendurchlauf darauf geprüft ob der HTML-Tag „*<a href*“ vorkommt. Sollte dies der Fall sein, so wird eine Variable um den Wert 1 erhöht, damit das Programm weiß, dass alle Zeichen ausgeschnitten und gleichzeitig in einen Buffer zwischengespeichert werden müssen, so lange bis der HTML-Tag „*<a href*“ durch den HTML-Tag „**“ abgeschlossen wurde.
Wurde das Quellcodefragment des Links erfolgreich in den Buffer verschoben, so wird nochmals der gesamte Quellcode durchlaufen um die Zeilennummern zu ermitteln in denen der HTML-Tag „*<body>*“ und „*</body>*“ vorkommt, um anhand dieser Information mittels einem Zufallsgenerators eine zufällige Position zu ermitteln, an welcher das Link wieder eingefügt werden soll. Dieser Vorgang ist nötig, um sicherzustellen, dass das Link innerhalb des Body-Tags eingefügt wird im Quellcode. Nach erfolgreichem einfügen der Tabelle an einer neuen, zufälligen Position im Quellcode, wird dieser wiederum abspeichern und auch diese Änderungsart ist damit abgeschlossen.

4.3. Programm-Ablaufprotokoll

Aufgrund der Aufgabenstellung werde ich hier beim Ablaufprotokoll zunächst anhand einer selbst geschriebenen Webseite die 6 Änderungsarten an einem Bildbeispiel verdeutlichen mit einem Bildpaar bestehenden aus „vorher“ und „nachher“.

Veränderung der Tabellenhintergrundfarbe

(vorher)

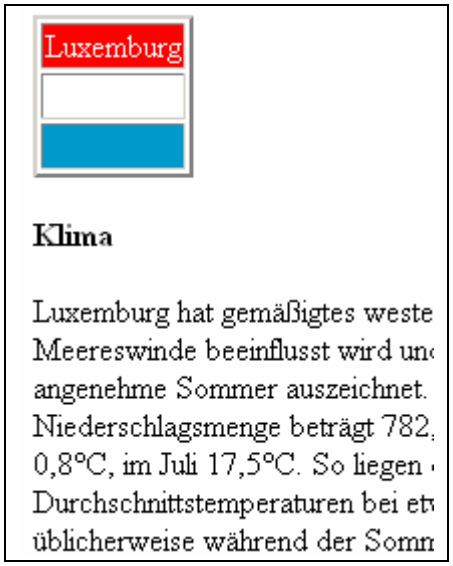
(nachher)

Veränderung der Tabellenstärke

(vorher)

(nachher)

Veränderung des Schrifttyps bei Zeilenabschnitten

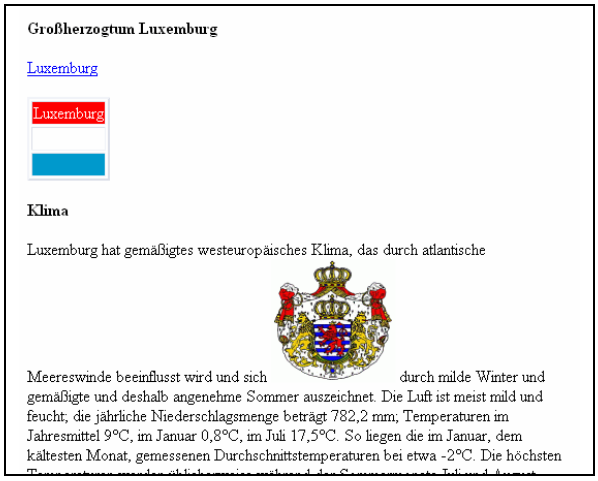


(vorher)

(nachher)

Veränderung der Position einer Grafik

(vorher)



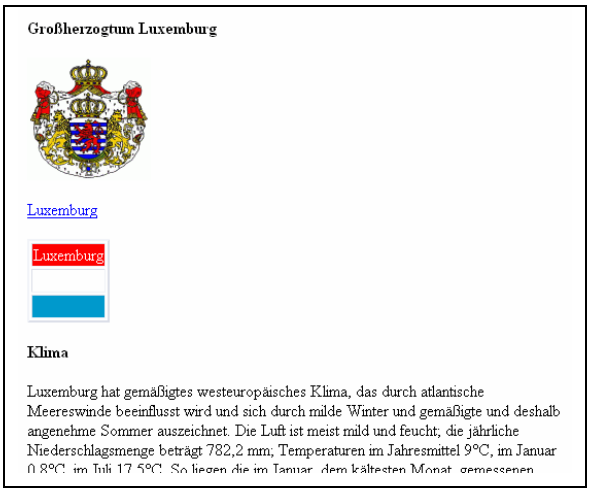
(nachher)

Veränderung der Position einer Tabelle

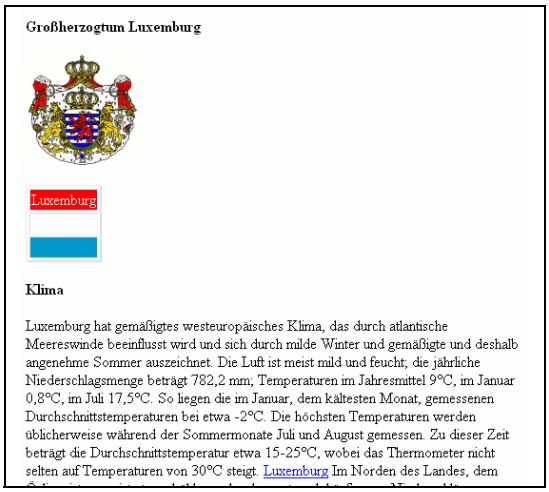
(vorher)

(nachher)

Veränderung der Position eines Links



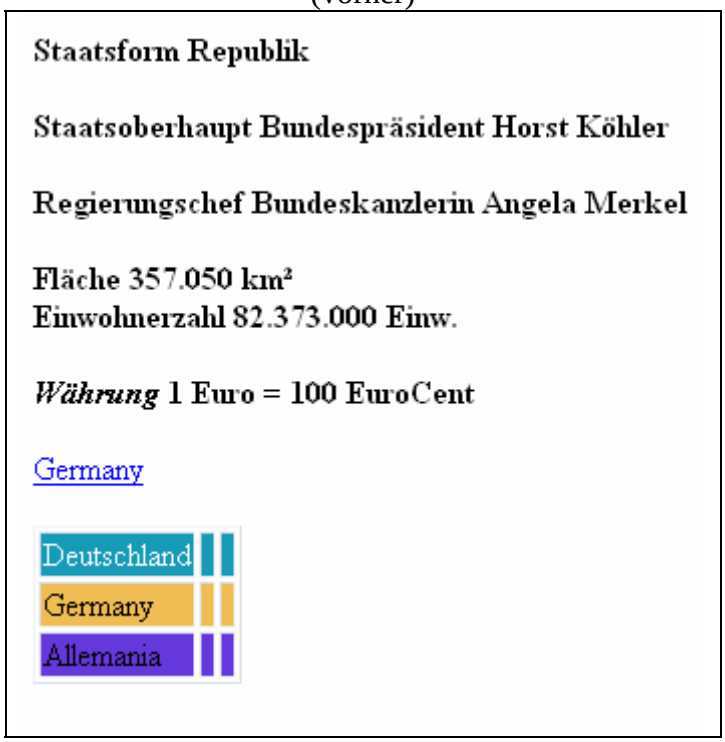
(vorher)



(nachher)

Komplette Veränderung der Website 1

(vorher)



(nachher)

Komplette Veränderung der Website 2

1.FC Köln Website

Gegründet	13. Februar 1948
Vereinsfarben	Rot-Weiß
Präsident	Wolfgang Overath

Stadion:

Das Heimstadion des 1. FC Köln ist derzeit das RheinEnergieStadion. Sein Vorläufer ist das 1923 an gleicher Stelle erbaute Müngersdorfer Stadion, welches 1975 neu errichtet wurde. Die heutige Fußballarena wurde am 31. Januar 2004 fertiggestellt. Im Gegensatz zu seinen Vorläuferbauten weist es keine Leichtathletikanlagen mehr auf und ist damit ein reines Fußballstadion. Das 50.997 Zuschauer fassende Stadion war im Jahr 2006 einer der zwölf Austragungsorte der Fußball-Weltmeisterschaft in Deutschland, hieß während des Turniers jedoch FIFA WM Stadion Köln, da die FIFA Sponsorennamen bei Stadien während einer WM verbietet. International wurde die

(vorher)

Stadion:

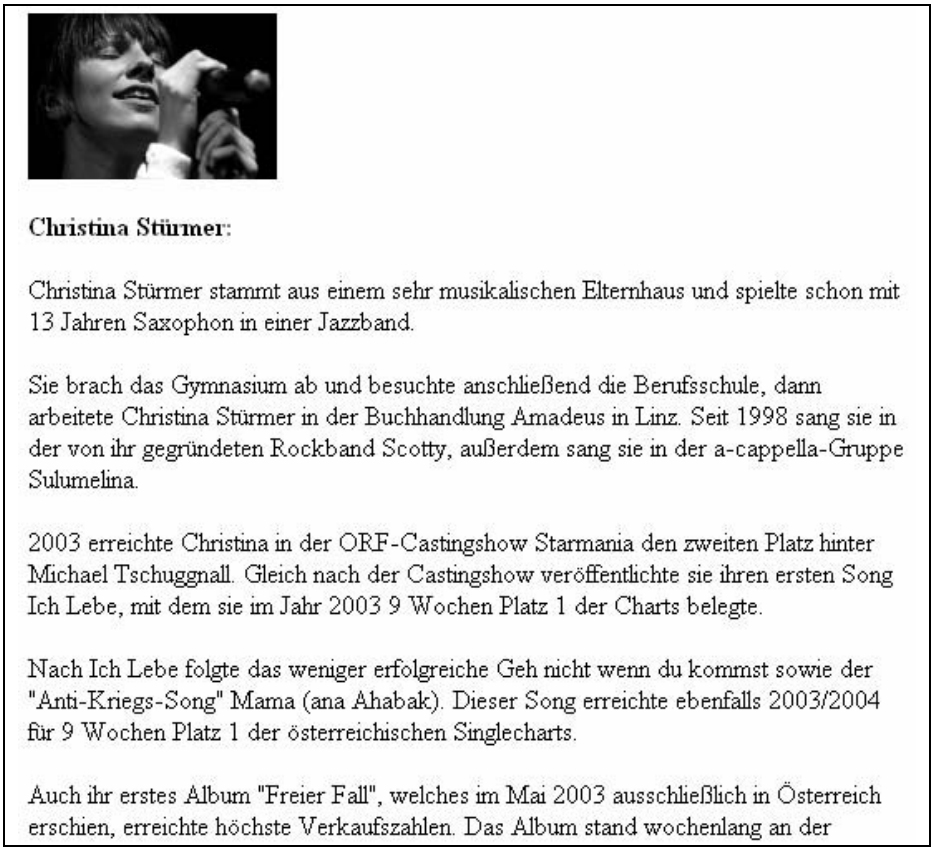
Das Heimstadion des 1. FC Köln ist derzeit das RheinEnergieStadion. Sein Vorläufer ist das 1923 an gleicher Stelle erbaute Müngersdorfer Stadion, welches 1975 neu errichtet wurde. Die heutige Fußballarena wurde am 31. Januar 2004 fertiggestellt. Im Gegensatz zu seinen Vorläuferbauten weist es keine Leichtathletikanlagen mehr auf und ist damit ein reines Fußballstadion. Das 50.997 Zuschauer fassende Stadion war im Jahr 2006 einer der zwölf Austragungsorte der Fußball-Weltmeisterschaft in Deutschland, hieß während des Turniers jedoch FIFA WM Stadion Köln, da die FIFA Sponsorennamen bei Stadien während einer WM verbietet. International wurde die Sportstätte in der Saison 2004/2005 übrigens auch für die Heimspiele im UEFA-Pokal durch Alemannia Aachen genutzt.

In der Nordtribüne des Stadions befindet sich das FC-Museum, in dem die Geschichte des 1. FC Köln vorgestellt wird.

Gegründet	13. Februar 1948

(nachher)

Komplette Veränderung der Website 3



Christina Stürmer:

Christina Stürmer stammt aus einem sehr musikalischen Elternhaus und spielte schon mit 13 Jahren Saxophon in einer Jazzband.

Sie brach das Gymnasium ab und besuchte anschließend die Berufsschule, dann arbeitete Christina Stürmer in der Buchhandlung Amadeus in Linz. Seit 1998 sang sie in der von ihr gegründeten Rockband Scotty, außerdem sang sie in der a-cappella-Gruppe Sulumelina.

2003 erreichte Christina in der ORF-Castingshow Starmania den zweiten Platz hinter Michael Tschuggnall. Gleich nach der Castingshow veröffentlichte sie ihren ersten Song Ich Lebe, mit dem sie im Jahr 2003 9 Wochen Platz 1 der Charts belegte.

Nach Ich Lebe folgte das weniger erfolgreiche Geh nicht wenn du kommst sowie der "Anti-Kriegs-Song" Mama (ana Ahabak). Dieser Song erreichte ebenfalls 2003/2004 für 9 Wochen Platz 1 der österreichischen Singlecharts.

Auch ihr erstes Album "Freier Fall", welches im Mai 2003 ausschließlich in Österreich erschien, erreichte höchste Verkaufszahlen. Das Album stand wochenlang an der

(vorher)

Christina Stürmer:

Christina Stürmer stammt aus einem sehr musikalischen Elternhaus und spielte schon mit 13 Jahren Saxophon in einer Jazzband.

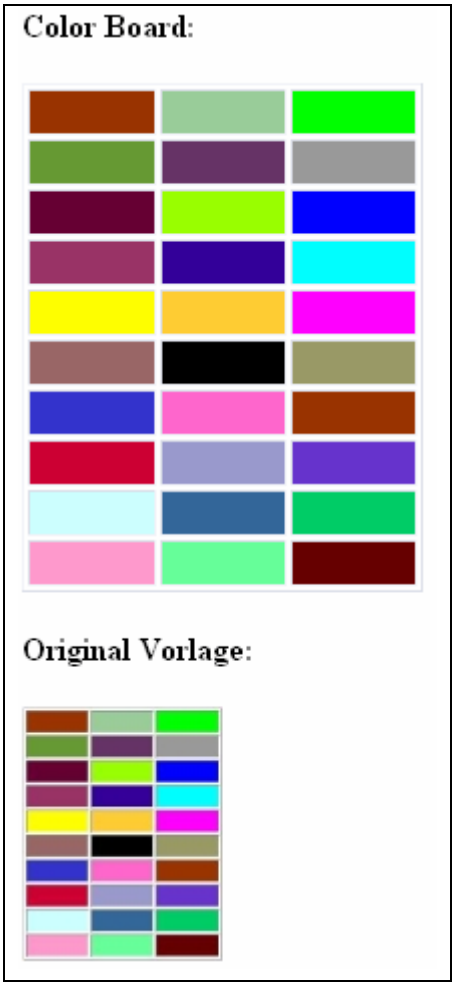
Sie brach das Gymnasium ab und besuchte anschließend die Berufsschule, dann arbeitete Christina Stürmer in der Buchhandlung Amadeus in Linz. Seit 1998 sang sie in der von ihr gegründeten Rockband Scotty, außerdem sang sie in der a-cappella-Gruppe Sulumelina.

2003 erreichte Christina in der ORF-Castingshow Starmania den zweiten Platz hinter

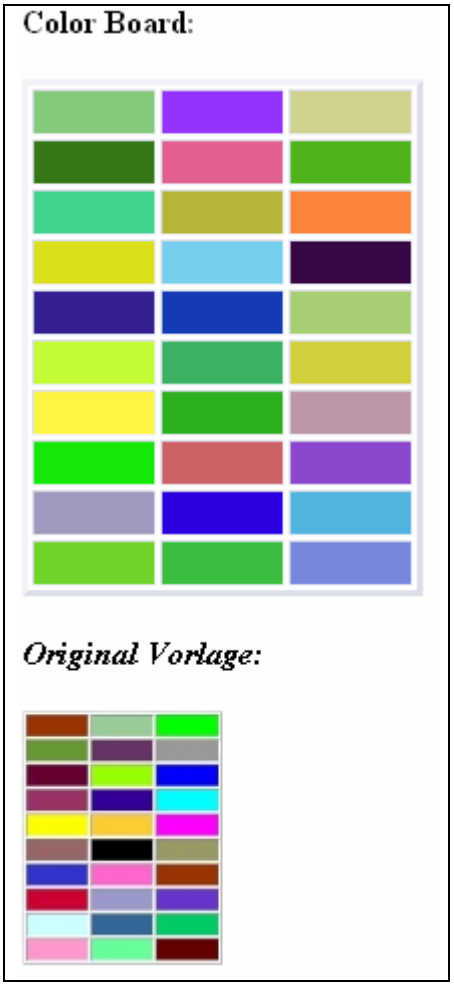
Michael Tschuggnall. Gleich nach der Castingshow veröffentlichte sie

(nachher)

Komplette Veränderung der Website 4



(vorher)



(nachher)

Komplette Veränderung der Website 5

(vorher)

(nachher)

Komplette Veränderung der Website 6

(vorher)

(nachher)

4.4. Programm-Text

```
// 25. Bundeswettbewerb Informatik 2006/2007
// Aufgabe 3: HTML Móbiles
//
// (c) Programmierung Andreas Albert

unit unit_main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, OleCtrls, SHDocVw;

type
  Thtml_mobiles = class(TForm)
    mmbuffer1: TMemo;
    btn_load: TButton;
    btn_change: TButton;
    OpenDialog1: TOpenDialog;
    mmbuffer2: TMemo;
    WebBrowser: TWebBrowser;
    btn_refresh: TButton;
    procedure btn_changeClick(Sender: TObject);
    procedure btn_loadClick(Sender: TObject);
    procedure btn_refreshClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  html_mobiles: Thtml_mobiles;

implementation

{$R *.dfm}

// Funktion um Dezimalzahlen in Hexzahlen umzuwandeln
function DecToHex(n:integer):string;
var s:string;
    c:char;
begin
  // Variableninhalt leeren
  s:='';
  // So lange noch eine Ziffer vorhanden ist die Schleife ausfuehren
  while n<>0 do
  begin
   if n mod 16 <10 then
```

```
   begin
     c:=chr(n mod 16+ord('0'))
   end
   else
   begin
     c:=chr(n mod 16+ord('A')-10);
   end;
   // Hexzahl um den neuen Zahlenwert erweitern
   s:=c+s;
   // Restwertigkeit der Dezimalzahl berechnen
   n:=n div 16;
  end;
  // Umgewandelte Hexzahl zurueckgeben
  result:=s;
end;

// Prozedur um die Hintergrundfarbe einer Tabelle in einen zufalls Farbwert zu
// aendern
procedure inhalt_change_bgcolor();
Var i : Integer;
    temp : string;
    position : Integer;
    buffer : string[255];
begin
  // Initialisierung des Zufallsgenerators
  Randomize;
  // Memofeld loeschen
  html_mobiles.mmbuffer2.Lines.Clear;
  // Schleife, welche alle Zeilen des aktuellen HTML Dokumentes
  // durchlaeuft
  for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
  begin
    // In der Variable buffer wird die HTML Zeile des Dokumentes
    // zwischengespeichert
    buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
    // Suche ob der HTML Tag bgcolor="# in der HTML Zeile vorkommt
    position := pos('bgcolor="#',buffer);
    // Ueberpruefung ob in der HTML Zeile der Tag bgcolor="# vorkam
    if(position>0) then
    begin
      // Loeschen des Hex-Farbwertes hinter dem Tag bgcolor="#
      delete(buffer,position+10,6);
      // Schleife die so lange durchlaufen wird, bis ein 6 stelliger Hexwert
      // durch den Zufallsgenerator ermittelt wurde
      repeat
        // Ermittlung eines Zufallswertes, der anschließend in einen Hexwert
        // umgerechnet wird
        // Der Wert 16777217 wurde gewaehlt, da dies der maximalste Dezimalwert ist,
        // der bei der Umwandlung in eine Hexzahl mit 6 Ziffern darstellbar ist
        temp := DecToHex(random(16777217));
      until length(temp) = 6;
      // Einfuegen des neuen Hex-Farbwertes hinter dem Tag bgcolor="#
      insert(temp,buffer,position+10);
    end;
```

```
    // Bearbeitete Zeile in des 2.Memofeld einfuegen
    html_mobiles.mmbuffer2.lines.add(buffer);
  end;
  // Inhalt des 1.Memofeldes loeschen
  html_mobiles.mmbuffer1.Lines.Clear;
end;

// Prozedur um die Rahmenstaerke einer Tabelle in einen Zufallswert zu
// aendern
procedure inhalt_change_border();
Var i : Integer;
    position : Integer;
    buffer : string[255];
begin
  // Memofeld loeschen
  html_mobiles.mmbuffer2.Lines.Clear;
  // Alle Quellcodezeilen durchlaufen
  for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
  begin
    // Aktuelle Quellcodezeile in einen Buffer uebertragen
    buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
    // Ueberpruefung ob der HTML-Tag "border="" in der aktuellen Quellcode-
    // zeile vorkommt
    position := pos('border="',buffer);
    // Falls der HTML-Tag vorkam...
    if(position>0) then
    begin
      //...wird nun der Zahlwert hinter border=" geloescht...
      delete(buffer,position+8,1);
      // Initialisierung des Zufallsgenerators
      Randomize;
      //...und ein Zufallszahlenwert an der selben Stelle eingefuegt
      insert(inttostr(random(6)),buffer,position+8);
    end;
    // Uebertragen der aktuellen Quellcodezeile vom Buffer ins Memofeld
    html_mobiles.mmbuffer2.lines.add(buffer);
  end;
  // Memofeld loeschen
  html_mobiles.mmbuffer1.Lines.Clear;
end;

// Prozedur um den Schrifttyp eines Abschnittes der durch die HTML-Tags <p> und </p>
// eingeschlossen wird zu aendern in fett, kursiv ...
procedure inhalt_change_schrifttype();
Var i : Integer;
    j : Integer;
    k : Integer;
    l : Integer;
    position1 : Integer;
    position2 : Integer;
    buffer : string[255];
begin
  k:=0;
  j:=0;
```

```
// Memofeld loeschen
html_mobiles.mmbuffer2.Lines.Clear;
// Alle Quellcodezeilen durchlaufen
for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
begin
  // Aktuelle Quellcodezeile in einen Buffer uebertragen
  buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
  // Ueberpruefung ob der HTML-Tag "<p>" in der aktuellen Quellcode-
  // zeile vorkommt
  position1 := pos('<p>',buffer);
  // Ueberpruefung ob der HTML-Tag "</p>" in der aktuellen Quellcode-
  // zeile vorkommt
  position2 := pos('</p>',buffer);

  // Pruefung ob Variable k den Wert 0 enthaelt
  if(k=0) then
  begin
    // Initialisierung des Zufallsgenerators
    randomize;
    // Zufallswert ermitteln
    j := random(2)+1;
  end;

  // Pruefung ob HTML-Tag "<p>" gefunden wurder und die Variable k den Wert 0 enthaelt
  if(position1>0) AND (k=0) then
  begin
    // Ueberpruefung ob der HTML-Tag "<b>" in der aktuellen Quellcode-
    // zeile vorkommt
    l := pos('<b>',buffer);
    // Wenn HTML-Tag vorkam...
    if(l>0) then
    begin
      //...dann HTML-Tag entfernen
      delete(buffer,l,3);
    end;
    // Ueberpruefung ob der HTML-Tag "<i>" in der aktuellen Quellcode-
    // zeile vorkommt
    l := pos('<i>',buffer);
    // Wenn HTML-Tag vorkam...
    if(l>0) then
    begin
      // ...dann HTML-Tag entfernen
      delete(buffer,l,3);
    end;

    // Wenn oben erzeugter Zufallswert 1 ist, folgende Aenderung vornehmen
    if(j=1) then
    begin
      // Position der HTML-Tag "<p>" in der aktuellen Quellcode-
      // zeile ermitteln...
      position1 := pos('<p>',buffer);
      // ...und an der Stelle danach den HTML-Tag "<b>" einfuegen
      insert('<b>',buffer,position1+3);
    end;
```

```
    // Wenn oben erzeugter Zufallswert 2 ist, folgende Aenderung vornehmen
    if(j=2) then
    begin
      // Position der HTML-Tag "<p>" in der aktuellen Quellcode-
      // zeile ermitteln...
      position1 := pos('<p>',buffer);
      // ...und an der Stelle danach den HTML-Tag "<i>" einfuegen
      insert('<i>',buffer,position1+3);
    end;
    // Variable k um den Wert 1 erhoehen
    k:=k+1;
  end;

  // Pruefung ob HTML-Tag "</p>" gefunden wurder und die Variable k den Wert 1 enthaelt
  if(position2>0) AND (k=1) then
  begin
    // Ueberpruefung ob der HTML-Tag "</b>" in der aktuellen Quellcode-
    // zeile vorkommt
    l := pos('</b>',buffer);
    // Wenn HTML-Tag vorkam...
    if(l>0) then
    begin
      // ...dann HTML-Tag entfernen
      delete(buffer,l,4);
    end;
    // Ueberpruefung ob der HTML-Tag "</i>" in der aktuellen Quellcode-
    // zeile vorkommt
    l := pos('</i>',buffer);
    // Wenn HTML-Tag vorkam...
    if(l>0) then
    begin
      // ...dann HTML-Tag entfernen
      delete(buffer,l,4);
    end;

    // Wenn oben erzeugter Zufallswert 1 ist, folgende Aenderung vornehmen
    if(j=1) then
    begin
      // Position der HTML-Tag "</p>" in der aktuellen Quellcode-
      // zeile ermitteln...
      position2 := pos('</p>',buffer);
      // ...und an der Stelle danach den HTML-Tag "</b>" einfuegen
      insert('</b>',buffer,position2);
    end;

    // Wenn oben erzeugter Zufallswert 2 ist, folgende Aenderung vornehmen
    if(j=2) then
    begin
      // Position der HTML-Tag "</p>" in der aktuellen Quellcode-
      // zeile ermitteln...
      position2 := pos('</p>',buffer);
      // ...und an der Stelle danach den HTML-Tag "</i>" einfuegen
      insert('</i>',buffer,position2);
    end;
```

```
      // Variable k um den Wert 1 verringern
      k:=k-1;
    end;
    // Uebertragen der aktuellen Quellcodezeile vom Buffer ins Memofeld
    html_mobiles.mmbuffer2.lines.add(buffer);
  end;
  // Memofeld loeschen
  html_mobiles.mmbuffer1.Lines.Clear;
end;

// Prozedur um die Position einer Tabelle zu aendern
procedure struckt_change_table();
type buffertyp = array [0..100] of string[255];
Var i : Integer;
    k : Integer;
    l : Integer;
    m : Integer;
    position1 : Integer;
    position2 : Integer;
    str_buffer : string[255];
    buffer : buffertyp;

begin
  k:=0;
  l:=0;
  // Memofeld loeschen
  html_mobiles.mmbuffer2.Lines.Clear;
  // Alle Quellcodezeilen durchlaufen
  for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
  begin
    // Aktuelle Quellcodezeile in einen Buffer uebertragen
    str_buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
    // Ueberpruefung ob der HTML-Tag "<table" in der aktuellen Quellcode-
    // zeile vorkommt
    position1 := pos('<table',str_buffer);

    // Pruefung ob HTML-Tag "<table" gefunden wurder und die Variable k den Wert 0
    // enthaelt
    if((position1>0) AND (k=0)) then
    begin
      // Variable k um den Wert 1 erhoehen
      k:=k+1;
    end;

    // Pruefung ob die Variable k den Wert 0 oder 2 enthaelt
    if((k=0) OR (k=2)) then
    begin
      // Uebertragen der aktuellen Quellcodezeile vom Buffer ins Memofeld
      html_mobiles.mmbuffer2.lines.add(str_buffer);
    end;

    // Ueberpruefung ob der HTML-Tag "</table>" in der aktuellen Quellcode-
    // zeile vorkommt
    position2 := pos('</table>',str_buffer);
```

```
  // Pruefung ob HTML-Tag "</table>" gefunden wurder und die Variable k den Wert 1
  // enthaelt
  if((position2>0) AND (k=1) AND (position1=0)) then
  begin
    // Uebertragen des Teils der Quellcodezeile vom Buffer ins Memofeld, der nicht
    // vom Table-Tag umschlossen wird
    html_mobiles.mmbuffer2.lines.add(copy(  str_buffer,
                                            position2+8,
                                            length(str_buffer)-position2+8));
    // Teil der Quellcodezeile loeschen im Buffer, der nicht
    // vom Table-Tag umschlossen wird
    delete(str_buffer,position2+8,length(str_buffer)-position2+8);
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable k um den Wert 1 erhoehen
    k:=k+1;
  end;

  // Pruefung ob HTML-Tag "</table>" nicht gefunden wurder und die Variable k den Wert 1
  // enthaelt
  if((position2=0) AND (k=1)) then
  begin
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable l um den Wert 1 erhoehen um mitzuzaehlen, wie viele Quellcode-
    // zeilen ausgeschnitten werden
    l:=l+1;
  end;
end;

// Alle Quellcodezeilen durchlaufen
for i := 0 to html_mobiles.mmbuffer2.Lines.Count do
begin
  // Aktuelle Quellcodezeile in einen Buffer uebertragen
  str_buffer := html_mobiles.mmbuffer2.Lines.Strings[i];
  // Ueberpruefung ob der HTML-Tag "<body" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('<body',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    position1 := i;
  end;
  // Ueberpruefung ob der HTML-Tag "</body>" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('</body>',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    position2 := i;
  end;
end;
```

```
// Initialisierung des Zufallsgenerators
Randomize;
// 1. zwischenspeichern Zeilennummer von der 2. zwischenspeichern Zeilennummer
// subtrahieren und in die Variable k uebertragen
k:=position2-position1;
// Zufallszahl ermitteln im berechneten Bereich wo spaeter die Tabelle innerhalb
// des Body-Tags eingefuegt werden soll
k:=random(k)+1;
// Zu der Variable k die 1. zwischenspeichern Zeilennummer addieren
k:=k+position1;
// Variable m auf den Wert 1 setzen
m:=1;
// Schleife so lange durchlaufen wie der Wert m nicht 0 ist
// (notwendig, damit HTML-Tag nicht innerhalb eines Tabellen-Tags eingefuegt wird)
while(m<>0) do
begin
  // Variable k um den Wert 1 erhoehen
  k:=k+1;
  // Variable m auf den Wert 0 setzen
  m:=0;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('</table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<td',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('</tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
end;
```

```
  // Pruefung ob die Variable l einen Wert groesser 0 enthaelt
  if(l>0) then
  begin
    // In dieser Schleife werden alle ausgeschnittenen Quellcodezeilen durchlaufen,
    // um sie so nacheinander an einer bestimmten Stelle wieder einfuegen zu koennen
    for i := 0 to l do
    begin
      // Alle vorhin ausgeschnittenen Quellcodezeilen einfuergen an der zuvor ermittelten
      // Zufallsposition innerhalb des Body-Tags
      html_mobiles.mmbuffer2.Lines.Insert(k,buffer[i]);
      // Variable k um den Wert 1 erhoehen
      k:=k+1;
    end;
    // Memofeld loeschen
    html_mobiles.mmbuffer1.Lines.Clear;
  end;
end;

// Prozedur um die Position eines Bildes /einer Grafik zu aendern
procedure struckt_change_image();
type buffertyp = array [0..100] of string[255];
Var i : Integer;
    k : Integer;
    l : Integer;
    m : Integer;
    pos1 : Integer;
    pos2 : Integer;
    str_buffer : string[255];
    buffer : buffertyp;

begin
  k:=0;
  l:=0;
  // Memofeld loeschen
  html_mobiles.mmbuffer2.Lines.Clear;
  // Alle Quellcodezeilen durchlaufen
  for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
  begin
    // Aktuelle Quellcodezeile in einen Buffer uebertragen
    str_buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
    // Ueberpruefung ob der HTML-Tag "<img" in der aktuellen Quellcode-
    // zeile vorkommt
    pos1 := pos('<img',str_buffer);

    // Pruefung ob HTML-Tag "<img" gefunden wurde und die Variable k den Wert 0 enthaelt
    if(pos1>0) AND (k=0) then
    begin
      html_mobiles.mmbuffer2.lines.add(copy(str_buffer,1,pos1-1));
      // Teil der Quellcodezeile loeschen im Buffer, der nicht
      // vom img-Tag umschlossen wird
      delete(str_buffer,1,pos1-1);
      // Variable k um den Wert 1 erhoehen
      k:=k+1;
    end;
```

```
  // Pruefung ob die Variable k den Wert 0 oder 2 enthaelt
  if((k=0) OR (k=2)) then
  begin
    // Uebertragen der aktuellen Quellcodezeile vom Buffer ins Memofeld
    html_mobiles.mmbuffer2.lines.add(str_buffer);
  end;

  // Ueberpruefung ob der HTML-Tag ">" in der aktuellen Quellcode-
  // zeile vorkommt
  pos2 := pos('>',str_buffer);

  // Pruefung ob HTML-Tag ">" gefunden wurde und die Variable k den Wert 1 enthaelt
  if(pos2>0) AND (k=1) then
  begin
    html_mobiles.mmbuffer2.lines.add(copy(str_buffer,pos2+1,length(str_buffer)-pos2+1));
    // Teil der Quellcodezeile loeschen im Buffer, der nicht
    // vom img-Tag umschlossen wird
    delete(str_buffer,pos2+1,length(str_buffer)-pos2+1);
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable k um den Wert 1 erhoehen
    k:=k+1;
    // Variable l um den Wert 1 erhoehen
    l:=l+1;
  end;

  // Pruefung ob HTML-Tag ">" nicht gefunden wurde und die Variable k den Wert 1 enthaelt
  if((pos2=0) AND (k=1)) then
  begin
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable l um den Wert 1 erhoehen um mitzuzaehlen, wie viele Quellcode-
    // zeilen ausgeschnitten werden
    l:=l+1;
  end;
end;

// Alle Quellcodezeilen durchlaufen
for i := 0 to html_mobiles.mmbuffer2.Lines.Count do
begin
  // Aktuelle Quellcodezeile in einen Buffer uebertragen
  str_buffer := html_mobiles.mmbuffer2.Lines.Strings[i];
  // Ueberpruefung ob der HTML-Tag "<body" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('<body',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    pos1 := i;
  end;
```

```
  // Ueberpruefung ob der HTML-Tag "</body>" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('</body>',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    pos2 := i;
  end;
end;

// Initialisierung des Zufallsgenerators
Randomize;
// 1. zwischenspeichern Zeilennummer von der 2. zwischenspeichern Zeilennummer
// subtrahieren und in die Variable k uebertragen
k:=pos2-pos1;
// Zufallszahl ermitteln im berechneten Bereich wo spaeter die Tabelle innerhalb
// des Body-Tags eingefuegt werden soll
k:=random(k)+1;
// Zu der Variable k die 1. zwischenspeichern Zeilennummer addieren
k:=k+pos1;
// Variable m auf den Wert 1 setzen
m:=1;
// Schleife so lange durchlaufen wie der Wert m nicht 0 ist
// (notwendig, damit HTML-Tag nicht innerhalb eines Tabellen-Tags eingefuegt wird)
while(m<>0) do
begin
  // Variable k um den Wert 1 erhoehern
  k:=k+1;
  // Variable m auf den Wert 0 setzen
  m:=0;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('</table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<td',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
```

```
    // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
    if(pos('</tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
    begin
      // Variable m auf den Wert 1 setzen
      m:=1
    end;
  end;
  // Pruefung ob die Variable l einen Wert groesser 0 enthaelt
  if(l>0) then
  begin
    for i := 1 to l do
    begin
      // Alle vorhin ausgeschnittenen Quellcodezeilen einfuergen an der zuvor ermittelten
      // Zufallsposition innerhalb des Body-Tags
      html_mobiles.mmbuffer2.Lines.Insert(k,buffer[i-1]);
      // Variable k um den Wert 1 erhoehen
      k:=k+1;
    end;
    // Memofeld loeschen
    html_mobiles.mmbuffer1.Lines.Clear;
  end;
end;

// Prozedur um die Position einer Hyperlinks zu aendern
procedure struckt_change_link();
type buffertyp = array [0..100] of string[255];
Var i : Integer;
    k : Integer;
    l : Integer;
    m : Integer;
    pos1 : Integer;
    pos2 : Integer;
    str_buffer : string[255];
    buffer : buffertyp;

begin
  k:=0;
  l:=0;
  // Memofeld loeschen
  html_mobiles.mmbuffer2.Lines.Clear;
  // Alle Quellcodezeilen durchlaufen
  for i := 0 to html_mobiles.mmbuffer1.Lines.Count do
  begin
    // Aktuelle Quellcodezeile in einen Buffer uebertragen
    str_buffer := html_mobiles.mmbuffer1.Lines.Strings[i];
    // Ueberpruefung ob der HTML-Tag "<a href" in der aktuellen Quellcode-
    // zeile vorkommt
    pos1 := pos('<a href',str_buffer);
```

```
  // Pruefung ob HTML-Tag "<a href" gefunden wurde und die Variable k den Wert 0
  // enthaelt
  if(pos1>0) AND (k=0) then
  begin
    html_mobiles.mmbuffer2.lines.add(copy(str_buffer,1,pos1-1));
    // Teil der Quellcodezeile loeschen im Buffer, der nicht
    // vom a href-Tag umschlossen wird
    delete(str_buffer,1,pos1-1);
    // Variable k um den Wert 1 erhoehen
    k:=k+1;
  end;

  // Pruefung ob die Variable k den Wert 0 oder 2 enthaelt
  if((k=0) OR (k=2)) then
  begin
    // Uebertragen der aktuellen Quellcodezeile vom Buffer ins Memofeld
    html_mobiles.mmbuffer2.lines.add(str_buffer);
  end;

  // Ueberpruefung ob der HTML-Tag "</a>" in der aktuellen Quellcode-
  // zeile vorkommt
  pos2 := pos('</a>',str_buffer);

  // Pruefung ob HTML-Tag "</a>" gefunden wurde und die Variable k den Wert 1 enthaelt
  if(pos2>0) AND (k=1) then
  begin
    html_mobiles.mmbuffer2.lines.add(copy(str_buffer,pos2+4,length(str_buffer)-pos2+4));
    // Teil der Quellcodezeile loeschen im Buffer, der nicht
    // vom a href-Tag umschlossen wird
    delete(str_buffer,pos2+4,length(str_buffer)-pos2+4);
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable k um den Wert 1 erhoehen
    k:=k+1;
    // Variable l um den Wert 1 erhoehen
    l:=l+1;
  end;

  // Pruefung ob HTML-Tag "</a>" nicht gefunden wurde und die Variable k den Wert 1
  // enthaelt
  if((pos2=0) AND (k=1)) then
  begin
    // Ausgeschnittene Quellcodezeile in den Buffer verschieben
    buffer[l] := str_buffer;
    // Variable l um den Wert 1 erhoehen um mitzuzaehlen, wie viele Quellcode-
    // zeilen ausgeschnitten werden
    l:=l+1;
  end;
 end;
```

```
// Alle Quellcodezeilen durchlaufen
for i := 0 to html_mobiles.mmbuffer2.Lines.Count do
begin
  // Aktuelle Quellcodezeile in einen Buffer uebertragen
  str_buffer := html_mobiles.mmbuffer2.Lines.Strings[i];
  // Ueberpruefung ob der HTML-Tag "<body" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('<body',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    pos1 := i;
  end;
  // Ueberpruefung ob der HTML-Tag "</body>" in der aktuellen Quellcode-
  // zeile vorkommt
  if(pos('</body>',str_buffer) > 0) then
  begin
    // Zeilennummer zwischenspeichern
    pos2 := i;
  end;
end;

// Initialisierung des Zufallsgenerators
Randomize;
// 1. zwischenspeichern Zeilennummer von der 2. zwischenspeichern Zeilennummer
// subtrahieren und in die Variable k uebertragen
k:=pos2-pos1;
// Zufallszahl ermitteln im berechneten Bereich wo spaeter die Tabelle innerhalb
// des Body-Tags eingefuegt werden soll
k:=random(k)+1;
// Zu der Variable k die 1. zwischenspeichern Zeilennummer addieren
k:=k+pos1-1;
// Variable m auf den Wert 1 setzen
m:=1;
// Schleife so lange durchlaufen wie der Wert m nicht 0 ist
// (notwendig, damit HTML-Tag nicht innerhalb eines Tabellen-Tags eingefuegt wird)
while(m<>0) do
begin
  // Variable k um den Wert 1 erhoehen
  k:=k+1;
  // Variable m auf den Wert 0 setzen
  m:=0;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('<table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
  // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
  if(pos('</table',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
  begin
    // Variable m auf den Wert 1 setzen
    m:=1
  end;
```

```
    // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
    if(pos('<tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
    begin
      // Variable m auf den Wert 1 setzen
      m:=1
    end;
    // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
    if(pos('<td',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
    begin
      // Variable m auf den Wert 1 setzen
      m:=1
    end;
    // Pruefung ob Quellcodezeile den angegeben Wert enthaelt
    if(pos('</tr',html_mobiles.mmbuffer2.Lines.Strings[k]) > 0) then
    begin
      // Variable m auf den Wert 1 setzen
      m:=1
    end;
  end;
  // Pruefung ob die Variable l einen Wert groesser 0 enthaelt
  if(l>0) then
  begin
    for i := 1 to l do
    begin
      // Alle vorhin ausgeschnittenen Quellcodezeilen einfuergen an der zuvor ermittelten
      // Zufallsposition innerhalb des Body-Tags
      html_mobiles.mmbuffer2.Lines.Insert(k,buffer[i-1]);
      // Variable k um den Wert 1 erhoehen
      k:=k+1;
    end;
    // Memofeld loeschen
    html_mobiles.mmbuffer1.Lines.Clear;
  end;
end;

// HTML-Tags mit Zufallswerten und in zufaelliger Reihenfolge anwenden
procedure Thtml_mobiles.btn_changeClick(Sender: TObject);
Var i,j : Integer;
    change_1 : bool;
    change_2 : bool;
    change_3 : bool;
    change_4 : bool;
    change_5 : bool;
    change_6 : bool;

begin
  // Variable mit dem default-Wert true belegen
  change_1 := true;
  // Variable mit dem default-Wert true belegen
  change_2 := true;
  // Variable mit dem default-Wert true belegen
  change_3 := true;
  // Variable mit dem default-Wert true belegen
  change_4 := true;
```

```
// Variable mit dem default-Wert true belegen
change_5 := true;
// Variable mit dem default-Wert true belegen
change_6 := true;
// Variable mit dem default-Wert 0 belegen
i:=0;
// Schleife so lange ausfuehren, bis der Wert 6 erreicht wurde und somit
// alle 6 Aenderungsarten durchgefuehrt wurden
while(i<6) do
begin
  // Zufallsgenerator neu initialisieren
  Randomize;
  // Zufallszahl zwischen 1 und 6 ermittlen
  j := random(6)+1;

  if((j=1) AND (change_1)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Inhaltliche Aenderung der Tabellen Hintergrundfarbe durchfuehren
    inhalt_change_bgcolor();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_1:=false
  end;

  if((j=2) AND (change_2)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Inhaltliche Aenderung der Tabellenstaerke durchfuehren
    inhalt_change_border();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_2:=false
  end;

  if((j=3) AND (change_3)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Inhaltliche Aenderung des Schrifttyps durchfuehren
    inhalt_change_schrifttype();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
```

```
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_3:=false
  end;

  if((j=4) AND (change_4)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Struckturelle Aenderung durch Umpositionierung des Bildes durchfuehren
    struckt_change_image();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_4:=false
  end;

  if((j=5) AND (change_5)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Struckturelle Aenderung durch Umpositionierung des Links durchfuehren
    struckt_change_link();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_5:=false
  end;

  if((j=6) AND (change_6)) then
  begin
    // HTML Quellcode in Buffer-Memofeld1 laden
    mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
    // Struckturelle Aenderung durch Umpositionierung der Tabelle durchfuehren
    struckt_change_table();
    // Geaenderten HTML Quellcode aus Buffer-Memofeld2 speichern
    mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
    // Zaehlvariable um den Wert 1 erhoehen
    i:=i+1;
    // Wert auf false setzen, um zu kennzeichnen, dass diese Aenderungsart
    // bereits ausgefuehrt wurde
    change_6:=false
  end;
end;
```

```
  // <---
  // Ab hier finden eine zum Abschluss noch eine eventuelle Fehlerkorrektur statt,
  // um entstandene Leerzeilen waehrend der Bearbeitung zu entfernen
  // <---

  // Bufferinhalt von mmbuffer1 entfernen
  mmbuffer1.Lines.Clear;
  // Bufferinhalt von mmbuffer2 entfernen
  mmbuffer2.Lines.Clear;
  // HTML Quellcode in den Buffer mmbuffer1 kopieren
  mmbuffer1.Lines.LoadFromFile(opendialog1.FileName);
  // Alle HTML Quellcode-Zeilen durchlaufen
  for i := 0 to mmbuffer1.Lines.Count do
  begin
    // Pruefung ob die Zeile Zeichen enthaelt
    if (mmbuffer1.Lines.Strings[i] <> '') then
    begin
      // Da Zeile Zeichen enthaelt, wird sie in den Buffer mmbuffer2 kopiert
      mmbuffer2.Lines.Add(mmbuffer1.Lines.Strings[i]);
    end;
  end;
  // HTML Quellcode nach entfernen der Leerzeilen wieder speichern
  mmbuffer2.Lines.SavetoFile(opendialog1.FileName );
end;

// HTML Seite auswaehlen
procedure Thtml_mobiles.btn_loadClick(Sender: TObject);
begin
  // Datei-Oeffnen Dialig von Windows anzeigen um eine HTML Datei auswaehlen
  // zu koennen
  opendialog1.Execute;
end;

// Browserfenster aktualisieren
procedure Thtml_mobiles.btn_refreshClick(Sender: TObject);
begin
  // Browserfenster aktualisieren mit der aktuell ausgewaehlten HTML Seite
  WebBrowser.Navigate(opendialog1.FileName);
end;

end.
```

5. Aufgabe 4: Supermarkt

5.1. Lösungsidee

Zuerst sollte es wichtig sein, sich einmal genau zu überlegen, welche Daten aufgrund der Aufgabenstellung überhaupt vorhanden sind bzw. welche Daten sinnvoll strukturiert werden müssen.
Dabei denke ich, dass es sinnvoll wäre folgende Daten aufzunehmen und diese dann in einem 2. Schritt zu einer sinnvollen Struktur zusammenzufassen.
Jeder Artikel sollte einen eindeutigen Artikelcode besitzen, welche Identisch sein muss mit dem Barcode auf dem entsprechenden Artikel, damit die Scanner der Kasse später auch den gescannten Barcode mit dem entsprechenden Artikel verknüpfen können.
Im Falle von Obst und Gemüse würde der Kassierer halt der Artikelcode manuell einzugeben.
Da sich aber der Preis für Obst und Gemüse im Gegensatz zu allen andren Artikel darin unterscheidet, dass er vom Gewicht der gewogenen Ware abhängt, müsste man von vorneherein schon 2 verschieden Artikelarten in Form einer Struktur realisieren, oder aber, so wie ich es machen möchte eine Variable einbauen, anhand deren man erkenne kann, ob es sich um Obst und Gemüse oder um einen anderen Artikel handelt. Dies ist sehr wichtig, da es einmal auf das Verhältnis des Gewichtes ankommt bei Obst bzw. Gemüse und bei den übrigen Artikel ein fester Preis je Artikeleinheit vorliegt.
Die Artikelstruktur müsste nach meiner Auffassung noch folgende Daten enthalten. Zunächst eine Zählvorrichtung, in welcher der Lagerbestand des Artikels erfasst wird, zusätzlich eine eindeutige Kennziffer, welche den Artikel einer bestimmten Produktgruppe zuweist. Natürlich sollte auch der Name des Produktes in der Artikelstruktur abgebildet werden. Um zu einem bestimmten Zeitpunkt abrufen zu können, wie oft der Artikel in einem Monat verkauft wurde, wäre es sicherlich sinnvoll innerhalb der Artikelstruktur eine Zähleinrichtung zu integrieren, mit Hilfe deren man für jeden Monat genau mitzählen kann, wie oft der Artikel verkauft wurde.
In einer 2.Struktur müsste man nun noch den Kunden mit seinen Eigenschaften abbilden. Da es in dieser Aufgabenstellung nicht weiter gefordert ist, irgendwelche Eigenschaften abzubilden, die vielleicht sinnvoll wären, aber zu diesem Zeitpunkt nicht gefordert werden, habe ich mich dazu entschlossen lediglich die für diese Aufgabenstellung notwendigen Eigenschaften abzubilden. In der Kundenstruktur sollen der Vorname, der Name, die Strasse, die Hausnummer, die Postleitzahl und der Ort zunächst abgebildet werden. Hinzu kommt noch eine Vorrichtung, mit deren man feststellen kann, ob der Kunde eine Kundenkarte besitzt, die er beim Einkauf verwendet und der gekaufte Weinbestand sollte für den Geschäftsfall Nr.4 der Aufgabenstellung noch abgedeckt werden, indem man der Kundenstruktur noch eine Zähleinheit zufügt, um die gekaufte Weinmenge abzudecken.
Mit Hilfe der eben beschrieben und so strukturierten Daten, kann man nun auf einfachste Art und Weise die in der Aufgabenstellung geforderten Ausdrucke erstellen.

5.2. Programm–Dokumentation

Die Lösungsidee wurde mit der Programmiersprache C++ aufgrund des oben genannten Lösungsansatzes realisiert. Da in der Aufgabenstellung nur Programmteile gefordert sind und kein komplettes Programm, wurden von mir auch nur die benötigten Programmteile realisiert.

Zunächst wurden von mir die Artikelstrukturen realisiert. Dabei habe ich für den besonderen Fall, dass es sich bei dem Artikel um Obst oder Gemüse handelt ja wie oben im Lösungsansatz bereits beschrieben die Variable „*per_kg*“ integriert, damit im späteren Programm bei der Berechnung des Artikelgesamtpreises festgestellt werden kann, ob es sich um einen Artikel handelt, bei dem der Preis pro gekaufte Einheit gilt, oder ob der Preis anhand des gewogenen Artikel bei Obst und Gemüse noch berechnet werden muss. Daher wurden auch die Variablen „*verkauft[12]*“ und „*bestand*“ vom Typ als Gleitkommazahl verwendet.

```
struct artikel{
  string name;
  string code;
  int gruppe;
  float preis;
  float bestand;
  float verkauft[12];
  bool per_kg;
};
```

Bei dieser Struktur wird der Artikelname in der Variable „*name“* in Form des Typs „String“ gespeichert.
Der Artikelcode wird in der Variable „*code“* in Form einer Zeichenkette gespeichert. Um den geläufige EAN-13 Code, der heutzutage bei Scannerkassen in der Regel benutzt wird, zu unterstützen wurde hier der Datentyp „*string*“ verwendet um die Anzahl von 13 gültigen Stellen bzw. Zeichen zu gewährleisten. Die Variable „*gruppe*“ enthält eine eindeutige Identifikationsnummer mit deren Hilfe man später über eine Prozedur die Hitliste verkaufter Artikel innerhalb eines Monats, geordnet nach Produktgruppen erstellen kann.
Für den Preis eines Artikels wurde die Variable „*preis*“ integriert, welche den Artikelpreis enthält. Im Falle eines gewöhnlichen Artikels, wäre der Wert der Variable „*per_kg*“ „*false*“, so dass man erkennt, dass es sich um einen Preis handelt, der sich pro Artikeleinheit bezieht. Sollte es sich bei dem Artikel um Obst oder Gemüse handeln, so ist der Preis abhängig von dem Gewicht der gewogenen Ware. In diesem Falle wäre der Wert der Variable „*per_kg*“ „*true*“ und der Preis der Variable „*preis*“ würde sich pro Kg des Artikels beziehen. Damit wäre auch an dieser Stelle klar, aus welchem Grund die Variable „*per_kg*“ notwendigerweise in die Artikelstruktur integriert wurde.
Die Variable „*bestand*“ wurde vor allem für den Geschäftsfall Nr.2 der Aufgabenstellung angelegt, um jederzeit zu wissen, wie groß der Restbestand des Artikels auf dem Lager ist bzw. um eine Liste aller Artikel über eine später noch beschriebene Prozedur ausdrucken zu können, deren Bestand einen bestimmten Wert unterschreitet.
Als letztes wurde dann noch ein Array in die Struktur integriert mit dem Namen „*verkauf*“, welches insgesamt 12 float-Werte beinhalten kann. So kann später für jeden Monat des aktuellen Jahres eine Verkaufsstatistik erstellt werden, wie viele Artikel verkauft wurden. Dies ist auch wichtig, um später den Geschäftsfall Nr.3 abbildenden bzw. abdecken zu können.

Die zweite Struktur mit dem Strukturnamen „*kunde*“, wurde von mir entwickelt, um die Daten eines Kunden abbilden zu können, um den Geschäftsfall Nr.4 aus der Aufgabenstellung später abbilden zu können.

```
struct kunde{
  string name;
  string vorname;
  string strasse;
  char hausnummer[6];
  unsigned long plz;
  string ort;
  bool karte;
  int wein;
};
```

Zunächst wurden in dieser Struktur die Variablen „*name*“, „*vorname*“, „*strasse*“ und „*ort*“ integriert, um den Namen, den Vorname, den Straßennamen und den Ortsnamen des Kunden erfassen zu können. Um die zu einer Adresse dazugehörigen Hausnummer zu speichern wurde die Variable „*hausnummer[6]*“ angelegt. Sicherlich wird sich nun die Frage stellen, warum es sich hierbei um eine Array handelt vom Typ „*char*“ und nicht vom Typ „*int*“. Dies ist ganz einfach so gewählt worden von mir, da es ja auch vorkommen könnte, dass eine Hausnummer z.B. „4A“ lautet, und diese in einer Variable vom Typ „*int*“ nicht richtig abgespeichert werden könnte.
Um die Postleitzahl einer Adresse abspeichern zu können wurde die Variable „plz“ angelegt vom Typ „*unsigned long*“. Der Datentyp wurde von mir so gewählt, da eine Postleitzahl keine negative Zahl ist, meist 4 oder 5 Ziffern umfasst und der Wertebereich des Typs „*long*“ im Gegensatz zum Typ „*int*“ nicht vom Betriebssystem abhängig ist.
Um späteren den Geschäftsfalles Nr.4 aus der Aufgabenstellung betrachten zu können, wurde die Variable „*karte*“ mit in die Kundenstruktur aufgenommen, mit deren Hilfe man feststellen kann, ob ein Kunde eine Kundenkarte besitzt, oder nicht. Gleichzeitig war es nötig die Variable „*wein*“ zu integrieren, da es durch den Geschäftsfall Nr.4 erforderlich ist.

Um die geforderten Ausdrucke der 4 Geschäftsfälle aus der Aufgabenstellung erstellen zu können wurden nachfolgende Prozeduren erstellt.

Geschäftsfall Nr.1

Der Kassenbon wird mithilfe der folgenden Prozedur erstellt:

```
// Geschaeftsfall Nr.1 aus der Aufgabenstellung
// Ausdruck eines Kassenbons
void kassenbon(artikel lager[MAX_ARTIKEL])
{
  // Daten-Struktur, die erstellt wurde, um einen Datenbuffer fuer den Kassenbon zu
  // erstellen
  struct bon{
    string name;
    float preis;
  };

  // Liste die als Datenbuffer dient um spaeter den Kassbon ausgeben zu koennen
  bon kassenbon[MAX_BON];

  // Buffer-Variable die dazu dient mitzuzaehlen, wie viele Artikel gescannt wurden
  int k=0;

  // Hier wird der Gesamtpreis zusammen addiert
  float gesamt=0;

  // Buffer in dem der von der Scannerkasse eingelesene Barcode zwischnengespeichert wird
  string scannercode;

  // Barcode der ersten Ware einlesen
  cin >> scannercode;
  while(scannercode != "ENTER")
  {
    // Da ein Artikel gescannt wurde, muss die Zaehlvariable um den Wert 1 erhoeht werden
    k++;
    // Alle Artikel des Lagers durchlaufen
    for(int i=0;i<MAX_ARTIKEL;i++)
    {
      // Pruefung ob der gescannte Barcode mit dem gerade zu pruefenden Artikel
      // uebereinstimmt
      if(scannercode==lager[i].code)
      {
        // Artikelname uebernehmen um ihn spaeter auf dem Kassenbon auszugeben
        kassenbon[k-1].name = lager[i].name;

        // Pruefung ob Artikelpreis pro Kg oder pro Einheit gilt
        // wegen dem Fall von Obst und Gemuese
        if(lager[i].per_kg)
        {
          // Variable in die das gewogene Gewicht eingelesen wird, da es sich
          // bei dem Artikel um Obst oder Gemuese handelt
          float gewicht;
          // Gewicht einlesen vom Obst oder dem Gemuese
          cin >> gewicht;
          // Berechnung des Preises fuer das Obst oder das Gemuese anhand des
          // Preise por Kg und dem gewogenen Gewicht
```

```
        kassenbon[k-1].preis = lager[i].preis / 1 * gewicht;
        // Artikelpreis zur Gesamtsumme dazuaddieren
        gesamt = gesamt + kassenbon[k-1].preis;
      }
      // Es handelt sich um einen gewoehnlichen Artikel, bei dem der Preis
      // pro gescannte Einheit gilt, also kein Obst oder Gemuese
      else
      {
        // Uebernehmen des Preises fuer den gescannten Artikel fuer die spaetere
        // Ausgabe des Kassenbons
        kassenbon[k-1].preis = lager[i].preis;
        // Artikelpreis zur Gesamtsumme dazuaddieren
        gesamt = gesamt + kassenbon[k-1].preis;
      }

    }
   }

   // Barcode der naechsten Ware einscannen oder "ENTER" fuer den Gesamtbetrag
   cin >> scannercode;
 }

 // Schleife, die nun den Kassenbon ausgiebt
 for(int i=0;i<k;i++)
 {
   // Ausgabe des Artikelnamen und des Artikelpreises auf dem Kassenbon
   cout << kassenbon[i].name << "\n" << kassenbon[i].preis << endl;
 }
 cout << endl << endl;
 // Ausgabe des Gesamtpreises
 cout << "Gesamt: " << gesamt << endl;
}
```

Erklärung:

Die Prozedur benötigt als Übergabeparameter die Daten des gesamten Lagerbestandes, in dem die einzelnen Artikel aufgeführt sind.
Um die Ausgabe des Kassenbons zu vereinfachen, wurde eine eigene Struktur angelegt mit dem Namen „*bon*“, welche die Variable „*name*“ enthält, in der später der Artikelname oder auch die Artikelbezeichnung der gekauften Artikel gespeichert wird. Des Weiteren enthält die Struktur die Variable „*preis*“, in der die einzelnen Preispositionen der gekauften Artikel zwischengespeichert werden.
Nun wird über den Scanner der Barcode der einzelnen Artikel eingescannt. Anhand des gescannten Barcodes wird nun im Lager nachgesehen, wie der Name und der Preis des gescannten Artikels ist. Wurde der Artikel im Lager gefunden wird in den Datenbuffer „*kassenbon*“ dieser Artikel zwischengespeichert um am Ende des Einlaufes, wenn der Kassierer „*ENTER*“ eingibt, den gesamten Kassenbon auszugeben. Gleichzeitig wird in der Variable „gesamt“ alle Artikel zusammenaddiert um später direkt den Gesamtpreis aller gekauften Artikel ausgeben zu können.
Beim Scanner der Artikel gibt es zwei Unterschiedliche Verfahren. Da man ja unterschieden muss, ob es sich um Obst/Gemüse handelt bei dem der Preis erst durch das Gewicht berechnet werden muss und den übrigen Artikel, bei denen es einen Festpreis pro Einheit gibt.

Unterschieden werden diese beiden Fälle dadurch, dass eine Prüfung stattfindet beim Scannen, ob in der Variable *„per_kg“* der Wert *„true“* gesetzt wurde. Sollte dies der Fall sein, so handelt es sich bei dem gescannten Artikel um Obst / Gemüse und der Preis muss anhand des gewogenen Gewichtes errechnet werden.
Im anderen Fall kann der Artikelpreis direkt übernommen werden, da sich der Preis pr Einheit bezieht.
Wurden alle Artikel eines Kunden eingescannt über das oben beschriebene Verfahren und der Kassierer hat dem Programm dies durch Eingabe von *„ENTER“* signalisiert, so wird nun über eine Programmschleife der Kassenbon in folgendem Format ausgegeben:

Artikelname *Artikelpreis*

Am Ende des Kassenbon wird zum Schluss dann noch der Gesamtbetrag des Einkaufes ausgegeben. Somit wurde der Kassenbon erfolgreich erstellt wie in der Aufgabenstellung verlangt.

Geschäftsfall Nr.2

Der Kassenbon wird mithilfe der folgenden Prozedur erstellt:

```
// Geschaeftsfall Nr.2 aus der Aufgabenstellung
// Ausdruck einer Liste aller Artikel, deren Bestand einen Wert unterschreitet
void artikel_unter_bestand(int wert, artikel lager[MAX_ARTIKEL])
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// wert  -> Wert, welcher unterschritten werden soll
// lager -> Uebergabe des gesamten Lagerbestandes
{

  cout << "Artikel, deren Lagerbestand weniger als " << wert << "betraegt" << endl << endl;
  for(int i=0;i<MAX_ARTIKEL;i++)
  {
    if(lager[i].bestand < wert)
    {
      cout << lager[i].name << "\n" << lager[i].bestand << endl;
    }
  }
}
```

Erklärung:

Die Prozedur benötigt als Übergabeparameter die Daten des gesamten Lagerbestandes, in dem die einzelnen Artikel aufgeführt sind und den Wert, der unterschritten werden soll.
In einer Programmschleife werden nun alle Lagerartikel durchlaufen und bei jedem Artikel findet eine Prüfung statt, ob der Lagerbestand geringer ist als der Suchwert. Sollte dies der Fall sein, so muss der entsprechende Artikel ausgegeben werden.

Geschäftsfall Nr.3

Der Ausdruck einer Hitliste verkaufter Artikel innerhalb eines Monats, geordnet nach Produktgruppen wird mit folgender Prozedur erstellt:

```
// Geschaeftsfall Nr.3 aus der Aufgabenstellung
// Ausdruck einer Hitliste verkaufter Artikel innerhalb eines Monats, geordnet
// nach Produktgruppen
void hitliste_artikel_pro_monat(int monat, artikel lager[MAX_ARTIKEL])
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// monat  -> Monat, welcher ausgewertet werden soll
// lager -> Uebergabe des gesamten Lagerbestandes
{
  // Daten-Struktur, die erstellt wurde, um einen Datenbuffer fuer die Hitliste zu
  // erstellen
  struct liste{
    string name;
    float verkauft;
  };

  // Liste die als Datenbuffer dient um die Hitliste der Artikel nach Produktgruppe
  // sortiert in absteigender Reihenfolge zu ermitteln
  liste hitliste[MAX_ARTIKEL];
  // Buffer-Variable die dazu dient mitzuzaehlen, wie viele Artikel in der gerade
  // auszuwertenden Artikelgruppe vorliegen
  int k;

  for(int i=0;i<MAX_ARTIKEL;i++)
  {
    k=0;

    // Alle Lagerartikel durchlaufen
    for(int j=0;j<MAX_ARTIKEL;j++)
    {
      // Pruefung ob Lagerartikel der gerade auszuwertenden Produktgruppe entspricht
      if(lager[j].gruppe == i)
      {
        hitliste[k].name = lager[j].name;
        hitliste[k].verkauft = lager[j].verkauft[monat-1];
        k++;
      }
    }
```

```
    // Soriteren der Artikel der Produktgruppe in abstiegender Reihenfolge
    for(int l=0;l<k;l++)
    {
      for(int m=0;m<k-1;m++)
      {
        if(hitliste[m].verkauft<hitliste[m+1].verkauft)
        {
          // Tauschen der Reihenfolge des Artikels, damit eine abstiegende Reihenfolge
          // erziehlt wird
          string buf_1 = hitliste[m+1].name;
          float buf_2 = hitliste[m+1].verkauft;
          hitliste[m+1].name = hitliste[m].name;
          hitliste[m+1].verkauft =  hitliste[m].verkauft;
          hitliste[m].name = buf_1;
          hitliste[m].verkauft = buf_2;
        }
      }
    }

    // Ausgabe der Hitliste verkaufter Artikel der Produktgruppe innerhalb des Monats,
    // in absteigender Reihenfolge
    cout << "PRODUKTGRUPPE " << i << endl << endl;
    for(int l=0;l<k;l++)
    {
      cout << hitliste[l].name << "\n" << hitliste[l].verkauft << endl;
    }
  }
}
```

Erklärung:

Die Prozedur benötigt als Übergabeparameter die Daten des gesamten Lagerbestandes, in dem die einzelnen Artikel liegen und den Namen des auszuwertenden Monats.
Um die Ausgabe der Hitliste zu vereinfachen, wurde eine eigene Struktur angelegt mit dem Namen „*liste*“, welche die Variable „*name*“ enthält, in der später der Artikelname oder auch die Artikelbezeichnung des Artikels gespeichert wird. Des Weiteren enthält die Struktur die Variable „*verkauft*“, in der die Menge des verkauften Artikels für den auszuwertenden Monat gespeichert wird.
In einer äußeren Schleife werden nun alle Produktgruppen nacheinander durchlaufen. Innerhalb dieser Programmschleife werden wiederum für jeden Produktgruppendurchlauf der gesamte Lagerbestand durchlaufen, um die Artikel der gerade auszuwertenden Produktgruppe in den Datenbuffer „*hitliste*“ zu übernehmen. Wurde der Lagerbestand einmal komplett durchlaufen, werden die gefunden Artikel in einer Sortierroutine in absteigender Reihenfolge sortiert, so dass sie anschließend auf ausgegeben werden können. Nachdem dies geschehen ist, kehrt das Programm wieder zur äußeren Schleife zurück und es wird die nächste Produktgruppe ausgewertet.
Wurden letztendlich alle Produktgruppen durchlaufen und abgearbeitet, so wird die Prozedur beendet, da sie abgeschlossen ist.

Geschäftsfall Nr.4

Der Ausdruck von Adressetiketten für den Versand eines Werbebriefes an Kunden, die mit Kundenkarte bezahlt haben und mindestens eine bestimmte Menge an Wein gekauft haben, wird mit folgender Prozedur erstellt:

```
// Geschaeftsfall Nr.4 aus der Aufgabenstellung
// Ausdruck von Adressetiketten fuer den Versand eines Werbebriefes an Kunden, die
// mit Kundenkarte bezahlt und viel Wein gekauft haben
void adressetikette(int wein_menge, kunde kundenstamm[MAX_ARTIKEL])
// ++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++
// wein_menge  -> Wert, den die gekaufte Weinmenge ueberschreiten muss,
//                damit ein Adressetikett mit der Kundenadresse fuer einen
//                Werbebrief gedruckt wird
// kundenstamm -> Uebergabe des gesamten Kundenstammes
{
  // Alle moeglichen Kundendaten in einer Schleife durchlaufen
  for(int i=0;i<MAX_KUNDEN;i++)
  {
    // Pruefung ob Kundensatz gueltig ist, indem ein Kundenname vorhanden ist
    if(kundenstamm[i].name.length() == 0)
    {
      // Pruefung ob Kunde eine Kundenkarte besitzt mit der er bezahlt
      if(kundenstamm[i].karte)
      {

        if(kundenstamm[i].wein > wein_menge)
        {
          // Hier kann nun das Adressetikett gedruckt werden. Ich werde an dieser
          // Stelle daher die Adressetiketten an dieser Stelle auf der Konsole
          // ausgeben
          cout << kundenstamm[i].name << kundenstamm[i].vorname << endl;
          cout << kundenstamm[i].strasse << kundenstamm[i].hausnummer << endl;
          cout << kundenstamm[i].plz << kundenstamm[i].ort << endl << endl;
        }
      }
    }
  }
}
```

Erklärung:

Die Prozedur benötigt als Übergabeparameter die Daten des Kundenstammes und einen Prüfwert, welcher angibt, welche Weinmenge mindestens gekauft werden musste um in diesem Falle ein Adressetikett für diesen Kunden auszudrucken.
In einer äußeren Schleife werden der gesamte Kundenstamm durchlaufen. Gleichzeitig wird geprüft ob es sich bei dem Kundensatz um einen leeren Datensatz handelt oder nicht. Sollte es sich um einen Kunden handeln, so wird nun überprüft ob der Kunde mit Kundenkarte bezahlt, indem der Wert der Variable „karte“ abgefragt wird. Sollte es sich um einen Kunden handeln, der mit Kundenkarte bezahlt, wird nun noch überprüft ob die gekaufte Weinmenge gleichgroß

oder größer ist als die Mindest-Weinmenge. Sollte dies der Fall sein, so handelt es sich um einen Kunde, der mit Kundenkarte bezahlt und viel Wein gekauft hat. Also wird nun die Adresse des Kunden ausgegeben für das Adressetikett.

Bewertung des Geschäftsfalles Nr.4

Dieser Geschäftsfall ist recht sinnvoll, da auf diese Weise z.B. nur Kunden den Wein-Werbebrief erhalten würden, die auch viel Wein kaufen und dem Unternehmen verbunden sind durch den Besitz einer Kundenkarte, welche sie nutzen.
Daher kann vereinfacht gesagt werden, dass durch diesen Fall eine gezielte Werbung stattfinden kann und gleichzeitig nicht der Fehler begangen wird, alle Kunden regelmäßig mit nicht interessanter Werbung zu belästigen. Es handelt sich also um eine gezielte Werbung.

5.3. Programm-Ablaufsprotokoll

Ein Programm Ablaufprotokoll war in dieser Aufgabenstellung nicht gefordert bzw. nicht erforderlich, da nur Programmteile realisiert werden mussten. Aus diesem Grunde wurde auch an dieser Stelle darauf verzichtet.

5.4. Programm-Text

Da kein komplettes Programm gefordert war in der Aufgabenstellung,aber man Programmteile realisieren sollte, habe ich mich dazu entschlossen hier den gesamten Quellcode der zur Lösung der Aufgabenstellung nötig war, noch einmal im ganzen aufzuführen.

```
// 25. Bundeswettbewerb Informatik 2006/2007
// Aufgabe 4: Supermarkt
//
// (c) Programmierung Andreas Albert

// Einbinden der benoetigten Programmbibliotheken
#include <conio.h>
#include <string>
#include <iostream>
using namespace std;

// Definierte Variable, anhand deren man die maximale Anzahl der verschiedenen Artikel
// leicht und unkompliziert aendern kann
#define MAX_ARTIKEL 200

// Definierte Variable, anhand deren man die maximale Anzahl der Kunden leicht
// und unkompliziert aendern kann
#define MAX_KUNDEN 200

// Definierte Variable, anhand deren man die maximale Anzahl der auf einem Kassenbon
// ausgebbaren Artikel leicht und unkompliziert aendern kann
#define MAX_BON 200

// Definition einer eigenen Struktur, die sich eignet zur Speicherung der
// einzelnen Artikel (außer Obst und Gemuese)
struct artikel{
  // Name des Artikels
  string name;

  // Einmalige Artikel-Code durch die der Artikel eindeutig identifizierbar ist
  // bei der Scanner-Kasse (Barcode des Artikels = Artikelcode)
  // Aufgrund des genormten EAN-13 Code auf 13 Ziffern wurde float verwendet
  string code;

  // Einmalige Nummer durch die der Artikel eindeutig einer Artikelgruppe zugewiesen wird
  int gruppe;

  // Preis der Artikels
  float preis;

  // die Menge des  Artikels auf Lager bzw. bei Obst & Gemuese waere die Menge
  // dann das Gesamt-Gewicht
  float bestand;
```

```
  // Array, in dem spaeter pro Monat die Menge des verkauften Artikels mitgezaehlt werden
  // bzw. bei Obst & Gemuese waere die Menge dann das Gesamt-Gewicht
  float verkauft[12];

  // Hier wird sich gemerkt, ob der Artikelpreis pro Einheit oder pro Kg gueltig ist
  // -> Das ist vor allem bei Obst & Gemüse wichtig, um den Preis zu berechnen
  bool per_kg;
};

// Definition einer eigenen Struktur, die sich nur zur Speicherung der Kunden eignet
struct kunde{
  // Name des Kunden
  string name;

  // Vorame des Kunden
  string vorname;

  // Strasse in welcher der Kunden wohnt
  string strasse;

  // Hausnummer (hier: Char-Werte, wegen z.B. der moeglichen Hausnummer 4A)
  char hausnummer[6];

  // PLZ des Kunden
  unsigned long plz;

  // Ort in welchem der Kunden wohnt
  string ort;

  // Hat der Kunde eine Kundenkarte ?
  bool karte;

  // Anzahl des gekauften Weines
  int wein;
};

// Geschaeftsfall Nr.1 aus der Aufgabenstellung
// Ausdruck eines Kassenbons
void kassenbon(artikel lager[MAX_ARTIKEL])
{
  // Daten-Struktur, die erstellt wurde, um einen Datenbuffer fuer den Kassenbon zu
  // erstellen
  struct bon{
    string name;
    float preis;
  };

  // Liste die als Datenbuffer dient um spaeter den Kassbon ausgeben zu koennen
  bon kassenbon[MAX_BON];

  // Buffer-Variable die dazu dient mitzuzaehlen, wie viele Artikel gescannt wurden
  int k=0;
```

```
// Hier wird der Gesamtpreis zusammen addiert
float gesamt=0;

// Buffer in dem der von der Scannerkasse eingelesene Barcode zwischnengespeichert wird
string scannercode;

// Barcode der ersten Ware einlesen
cin >> scannercode;
while(scannercode != "ENTER")
{
  // Da ein Artikel gescannt wurde, muss die Zaehlvariable um den Wert 1 erhoeht werden
  k++;
  // Alle Artikel des Lagers durchlaufen
  for(int i=0;i<MAX_ARTIKEL;i++)
  {
    // Pruefung ob der gescannte Barcode mit dem gerade zu pruefenden Artikel
    // uebereinstimmt
    if(scannercode==lager[i].code)
    {
      // Artikelname uebernehmen um ihn spaeter auf dem Kassenbon auszugeben
      kassenbon[k-1].name = lager[i].name;

      // Pruefung ob Artikelpreis pro Kg oder pro Einheit gilt
      // wegen dem Fall von Obst und Gemuese
      if(lager[i].per_kg)
      {
        // Variable in die das gewogene Gewicht eingelesen wird, da es sich
        // bei dem Artikel um Obst oder Gemuese handelt
        float gewicht;
        // Gewicht einlesen vom Obst oder dem Gemuese
        cin >> gewicht;
        // Berechnung des Preises fuer das Obst oder das Gemuese anhand des
        // Preise por Kg und dem gewogenen Gewicht
        kassenbon[k-1].preis = lager[i].preis / 1 * gewicht;
        // Artikelpreis zur Gesamtsumme dazuaddieren
        gesamt = gesamt + kassenbon[k-1].preis;
      }
      // Es handelt sich um einen gewoehnlichen Artikel, bei dem der Preis
      // pro gescannte Einheit gilt, also kein Obst oder Gemuese
      else
      {
        // Uebernehmen des Preises fuer den gescannten Artikel fuer die spaetere
        // Ausgabe des Kassenbons
        kassenbon[k-1].preis = lager[i].preis;
        // Artikelpreis zur Gesamtsumme dazuaddieren
        gesamt = gesamt + kassenbon[k-1].preis;
      }

    }
  }

  // Barcode der naechsten Ware einscannen oder "ENTER" fuer den Gesamtbetrag
  cin >> scannercode;
}
```

```
  // Schleife, die nun den Kassenbon ausgiebt
  for(int i=0;i<k;i++)
  {
    // Ausgabe des Artikelnamen und des Artikelpreises auf dem Kassenbon
    cout << kassenbon[i].name << "\n" << kassenbon[i].preis << endl;
  }
  cout << endl << endl;
  // Ausgabe des Gesamtpreises
  cout << "Gesamt: " << gesamt << endl;
}

// Geschaeftsfall Nr.2 aus der Aufgabenstellung
// Ausdruck einer Liste aller Artikel, deren Bestand einen Wert unterschreitet
void artikel_unter_bestand(int wert, artikel lager[MAX_ARTIKEL])
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// wert  -> Wert, welcher unterschritten werden soll
// lager -> Uebergabe des gesamten Lagerbestandes
{

  cout << "Artikel, deren Lagerbestand weniger als " << wert << "betraegt" << endl << endl;
  for(int i=0;i<MAX_ARTIKEL;i++)
  {
    if(lager[i].bestand < wert)
    {
      cout << lager[i].name << "\n" << lager[i].bestand << endl;
    }
  }
}

// Geschaeftsfall Nr.3 aus der Aufgabenstellung
// Ausdruck einer Hitliste verkaufter Artikel innerhalb eines Monats, geordnet
// nach Produktgruppen
void hitliste_artikel_pro_monat(int monat, artikel lager[MAX_ARTIKEL])
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// monat  -> Monat, welcher ausgewertet werden soll
// lager -> Uebergabe des gesamten Lagerbestandes
{
  // Daten-Struktur, die erstellt wurde, um einen Datenbuffer fuer die Hitliste zu
  // erstellen
  struct liste{
    string name;
    float verkauft;
  };

  // Liste die als Datenbuffer dient um die Hitliste der Artikel nach Produktgruppe
  // sortiert in absteigender Reihenfolge zu ermitteln
  liste hitliste[MAX_ARTIKEL];
```

```
  // Buffer-Variable die dazu dient mitzuzaehlen, wie viele Artikel in der gerade
  // auszuwertenden Artikelgruppe vorliegen
  int k;

  for(int i=0;i<MAX_ARTIKEL;i++)
  {
    k=0;

    // Alle Lagerartikel durchlaufen
    for(int j=0;j<MAX_ARTIKEL;j++)
    {
      // Pruefung ob Lagerartikel der gerade auszuwertenden Produktgruppe entspricht
      if(lager[j].gruppe == i)
      {
        hitliste[k].name = lager[j].name;
        hitliste[k].verkauft = lager[j].verkauft[monat-1];
        k++;
      }
    }

    // Soriteren der Artikel der Produktgruppe in abstiegender Reihenfolge
    for(int l=0;l<k;l++)
    {
      for(int m=0;m<k-1;m++)
      {
        if(hitliste[m].verkauft<hitliste[m+1].verkauft)
        {
          // Tauschen der Reihenfolge des Artikels, damit eine abstiegende Reihenfolge
          // erziehlt wird
          string buf_1 = hitliste[m+1].name;
          float buf_2 = hitliste[m+1].verkauft;
          hitliste[m+1].name = hitliste[m].name;
          hitliste[m+1].verkauft =  hitliste[m].verkauft;
          hitliste[m].name = buf_1;
          hitliste[m].verkauft = buf_2;
        }
      }
    }

    // Ausgabe der Hitliste verkaufter Artikel der Produktgruppe innerhalb des Monats,
    // in absteigender Reihenfolge
    cout << "PRODUKTGRUPPE " << i << endl << endl;
    for(int l=0;l<k;l++)
    {
      cout << hitliste[l].name << "\n" << hitliste[l].verkauft << endl;
    }
  }
}
```

```
// Geschaeftsfall Nr.4 aus der Aufgabenstellung
// Ausdruck von Adressetiketten fuer den Versand eines Werbebriefes an Kunden, die
// mit Kundenkarte bezahlt und viel Wein gekauft haben
void adressetikette(int wein_menge, kunde kundenstamm[MAX_ARTIKEL])
// ++++++++++++++++++++
// Uebergabeparameter:
// ++++++++++++++++++++
// wein_menge  -> Wert, den die gekaufte Weinmenge ueberschreiten muss,
//                damit ein Adressetikett mit der Kundenadresse fuer einen
//                Werbebrief gedruckt wird
// kundenstamm -> Uebergabe des gesamten Kundenstammes
{
  // Alle moeglichen Kundendaten in einer Schleife durchlaufen
  for(int i=0;i<MAX_KUNDEN;i++)
  {
    // Pruefung ob Kundensatz gueltig ist, indem ein Kundenname vorhanden ist
    if(kundenstamm[i].name.length() == 0)
    {
      // Pruefung ob Kunde eine Kundenkarte besitzt mit der er bezahlt
      if(kundenstamm[i].karte)
      {

        if(kundenstamm[i].wein > wein_menge)
        {
          // Hier kann nun das Adressetikett gedruckt werden. Ich werde an dieser
          // Stelle daher die Adressetiketten an dieser Stelle auf der Konsole
          // ausgeben
          cout << kundenstamm[i].name << kundenstamm[i].vorname << endl;
          cout << kundenstamm[i].strasse << kundenstamm[i].hausnummer << endl;
          cout << kundenstamm[i].plz << kundenstamm[i].ort << endl << endl;
        }
      }
    }
  }
}

int main()
{
  // Array, das spaeter alle Artikel des Lagers enthalten wird
  artikel lager[MAX_ARTIKEL];

  // Array, das spaeter alle Kunden des Kundenstammes enthalten wird
  kunde kundenstamm[MAX_KUNDEN];

  cout << "INFO:" << endl;
  cout << "Da es in der Aufgabenstellung nicht verlangt wurde, wurde an dieser" << endl;
  cout << "Stelle auch darauf verzichtet, ein Programm zu entwickeln bzw. die" << endl;
  cout << "einzelnen Prozeduren einzufuegen, da dies keinen Sinn macht," << endl;
  cout << "ohne ein entsprechendes Menue etc." << endl << endl << endl;

  cout << "Bitte eine beliebige Taste druecken um das Programm zu beenden." << endl;
  // Beenden des Programmes nach druecken einer beliebigen Taste
  getch();
  return 0;
}
```

6. Aufgabe 5: Die Paderbox

6.1. Lösungsidee

Ziel ist es eine Anwendung zu realisieren die vereinfacht beschrieben aus zwei Teilen besteht. Zum einen aus einer Klasse, welche alle für die Paderbox relevanten Berechnungsfunktionen implementiert und zum anderen aus einer Bedienoberfläche, die alle benötigten Bedienungselemente umfasst.
Im nachfolgenden Text wird die Klasse als „Paderbox-Klasse“ bezeichnet und die Benienoberfläche als „Formular“

Die Paderbox-Klasse muss folgendes können:

- Merken oder auch Speichern aller benötigten Informationen
- Bearbeitung von Informationen, die von außen kommen durch Benutzeraktionen
- Zufällige Berechnung der „geheimen Programmierung“ wie es die Aufgabenstellung erfordert
- Berechnung des entsprechenden Statuswechsels

Das Formular muss folgendes können:

- Darstellen einer Benutzeroberfläche
- Weitergabe von Benutzertransaktionen an die Paderbox-Klasse
- Durchführen des Statuswechsel, d.h. z.B. umschalten der Lampen

Bei der Zuordung des Status zu den 3 Lampen kann man sich darauf beschränken, 3 Zustände abzubilden, da alle weiteren Kombinationen die selbe Lampe zweimal beinhalten würden, was zu keiner Statusänderung führen würde.

Die drei Kombinationsmöglichkeiten wären:

- Lampe 1 + Lampe 2
- Lampe 2 + Lampe 3
- Lampe 1 + Lampe 3

Diese drei Kombinationen werden dann zufällig auf die einzelnen Stati verteilt, wobei es wiederum sinnlos wäre, eine Kombination auf mehrere Stati zuzuweisen, da dies Aufgabenstellung (jede Lampe muss zweimal vorkommen) wiedersprechen würde. Aus diesem Grunde wird das Zufalls- oder auch Lottoprinzip verwendet.
Für die geheime Programmierung wird einfach eine Zuordnungstabelle erstellt, die Status und Schalter zufälligen neuen Statuswerten zuordnet.

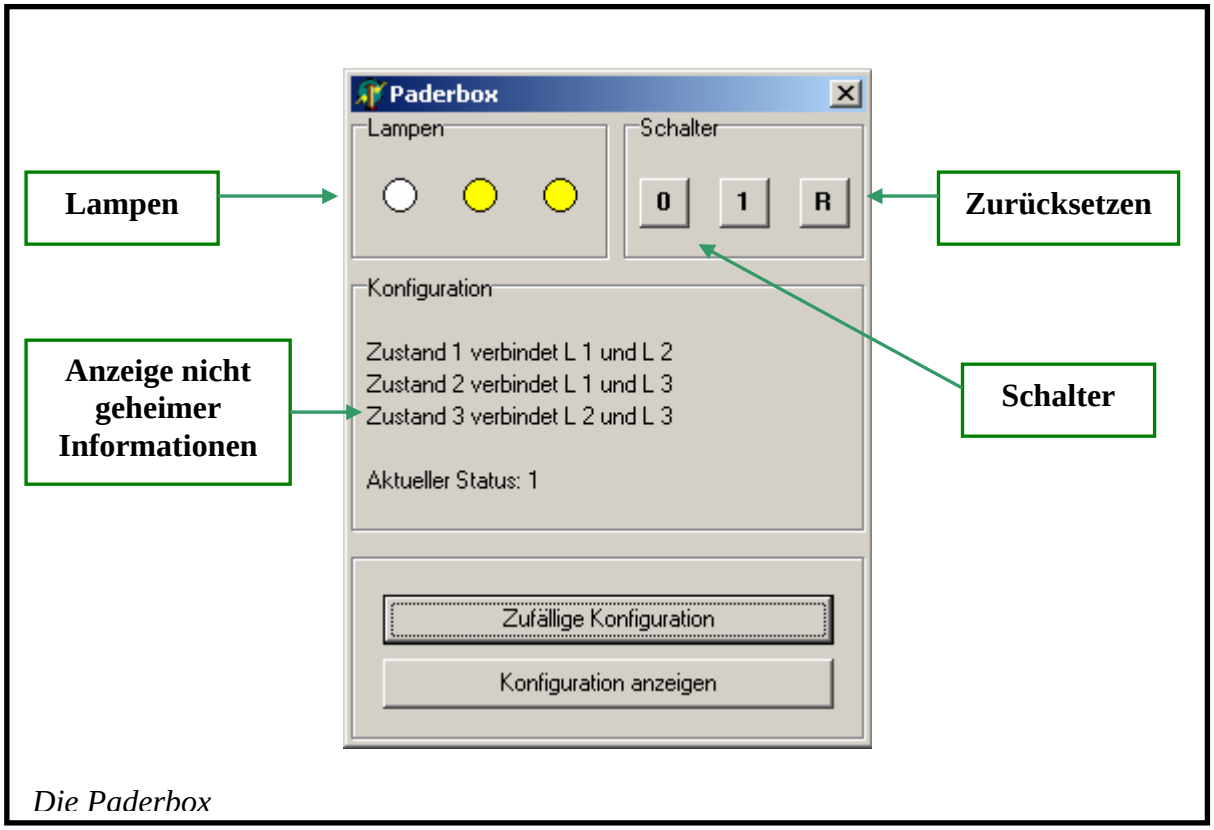
Als optimal kann man auch ansehen, dass die GUI in zwei sperate Formulare (=Fenster) getrennt wird, so dass ein Formular die Paderbox selbst und das zweite Formular die Konfiguration, also die geheime Programmierung beinhaltet. Damit ist gewährleistet, dass die „geheime Programmierung“ jederzeit „verraten“ werden kann.

Benötigte Bedienelemente:

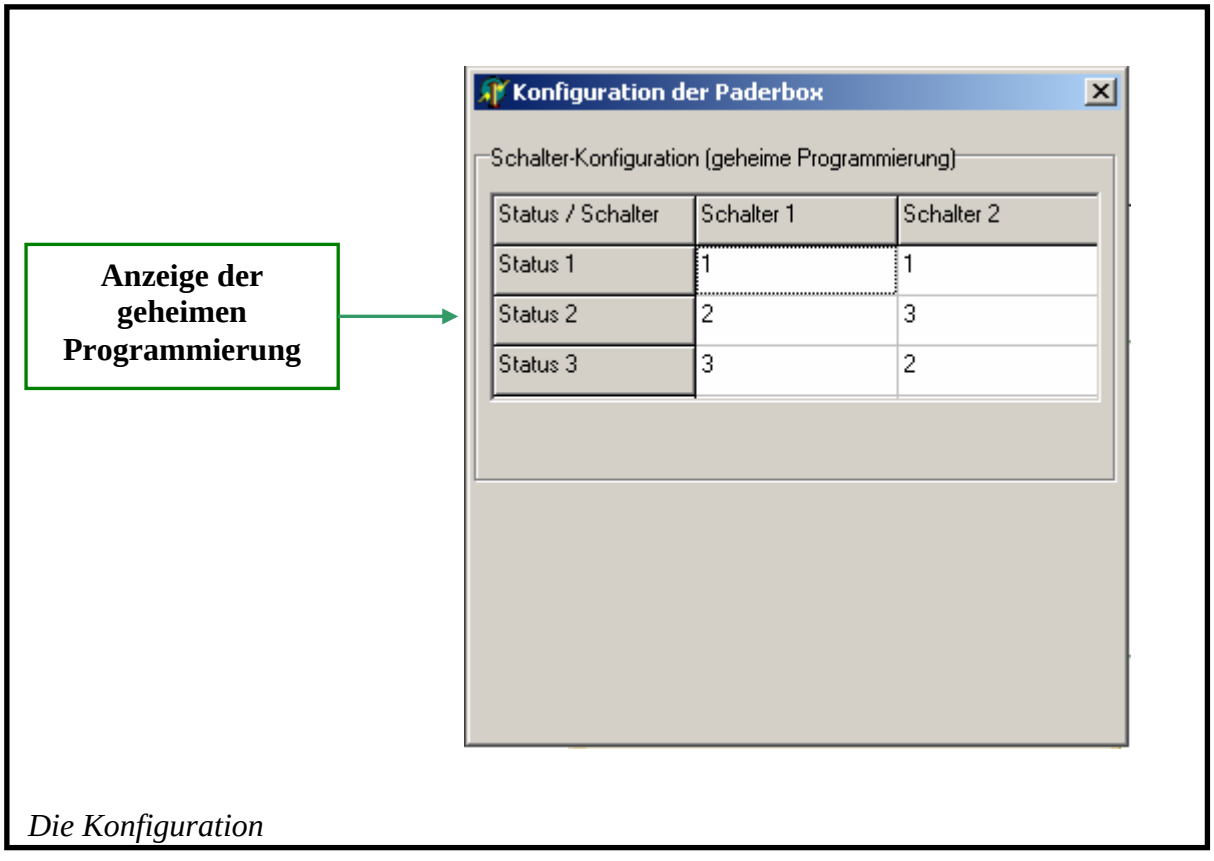
- Paderbox
  - Lampen
  - Schalter
  - Anzeige nicht geheimer Informationen

- Konfiguration
  - Anzeige der „geheimen Programmierung“ anhand einer Tabelle

6.2. Programm-Dokumentation



Die Paderbox



Die Konfiguration

Das komplette Programm wurde von mir in der Programmiersprache Delphi umgesetzt. Das Programm benutzt die von mir eigens selbgeschriebene Klasse „TPaderbox“, die alle Berechnungen und Transaktionen der Paderbox übernimmt und deren Methoden den einzelnen Buttons hinterlegt wurden.

In der Klasse „TPaderbox“ selbst gibt es nachfolgende Methoden:

- procedure TPaderbox.change_zustand(taste:Integer);
  - Diese Methode übernimt nach dem drücken einer Taste, also entweder der Taste „0“ oder „1“ die Berechnung des neuen Zustandes der Paderbox
- procedure TPaderbox.random_anfang();
  - Diese Methode generiert den zufälligen Anfangszustand der Paderbox
- procedure TPaderbox.random_zustand();
  - Diese Methode initialiert in zufälliger Weise die Anfangs-Zustände
- procedure TPaderbox.reset_paderbox();
  - Diese Methode setzt die Paderbox in ihren Anfangszustand zurück (=reset)
- procedure TPaderbox.set_kombi();
  - Diese Methode initialsiert die 3 möglichen Lampenkombinationen
- procedure TPaderbox.set_schalter();
  - Diese Methode erzeugt die zufällige „geheime Programmierung“
- procedure TPaderbox.set_zustaende();
  - Diese Methode initialisiert die Zustände in zufälliger Weise
- procedure TPaderbox.show_geheim(var b_config_schalter:type_schalter);
  - Diese Methode gibt die „geheime Programmierung“ dem Benutzer aus
- procedure TPaderbox.show_paderbox(var b_actual_lamps:type_lamp;
  var b_kombinationen:type_kombi;
  var b_actual_anfangzustand:Integer;
  var b_zustaende:type_zustand);
  - Diese Methode übergibt die Daten aus der Klasse „TPaderbox“ in das Formular, damit die Daten dort angezeigt werden können

6.3. Programm-Ablaufprotokoll

Beom Programmstart ist für den Benutzer nur die uninitialisierte Paderbox sichtbar. Erst nach einem Klick auf den Button „Zufällige Konfiguration“ wird die Paderbox initialisiert, indem die Lampen geschaltet werden und die Verbindung der Lampen mit den Zuständen angezeigt wird.
Ein Klick auf die erste Eingabetaste („0“) ändert den Status der Lampen. Genau das gleiche gilt auch für die zweite Eingabetaste („1“).
Durch drücken der Reset- oder auch Rücksetztaste („R“) wird die Paderbox wieder auf ihren Anfangszustand zurückgesetzt.
Der Button „Konfiguration anzeigen“ öffnet ein zweites Fenster, in dem sich der Benutzer die „geheime Programmierung“ anzeigen lassen kann.

6.4. Programm-Text

- **Formular „Paderbox“**

```
// 25. Bundeswettbewerb Informatik 2006/2007
// Aufgabe 5: Paderbox
//
// (c) Programmierung Andreas Albert

unit paderbox_1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls,paderbox_class, ComCtrls;

type
  Tfrm_paderbox_1 = class(TForm)
    GroupBox1: TGroupBox;
    lamp3: TShape;
    lamp2: TShape;
    lamp1: TShape;
    gb_schalter: TGroupBox;
    btn_0: TButton;
    btn_1: TButton;
    btn_R: TButton;
    gb_show_config: TGroupBox;
    lbl_config_1: TLabel;
    lbl_config_2: TLabel;
    lbl_config_3: TLabel;
    lbl_status: TLabel;
    gb_config: TGroupBox;
    btn_random_config: TButton;
    btn_show_config: TButton;
    procedure btn_random_configClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btn_RClick(Sender: TObject);
    procedure btn_0Click(Sender: TObject);
    procedure btn_1Click(Sender: TObject);
    procedure ausgabe();
    procedure btn_show_configClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;
```

```
var
  frm_paderbox_1: Tfrm_paderbox_1;
  Paderbox : TPaderbox;
  actual_lamps : type_lamp;
  kombinationen : type_kombi;
  actual_anfangzustand : Integer;
  zustaende : type_zustand;
  config_schalter : type_schalter;

implementation

uses config_1;

{$R *.dfm}

// Prozedur um die Paderbox-Daten aus der Klasse ins Formular zu importieren
procedure Tfrm_paderbox_1.ausgabe();
begin
  // Pruefung ob die Lampe den Wert 1 besitzt...
  if actual_lamps[1] = 1 then
    begin
      // ...wenn ja, Lampenfarbe in Gelb umaendern
      lamp1.Brush.Color := clYellow;
    end
  else
    begin
      // ...wenn nein, Lampenfarbe in Weiß umaendern
      lamp1.Brush.Color := clWhite;
    end;
  // Pruefung ob die Lampe den Wert 1 besitzt...
  if actual_lamps[2] = 1 then
    begin
      // ...wenn ja, Lampenfarbe in Gelb umaendern
      lamp2.Brush.Color := clYellow;
    end
  else
    begin
      // ...wenn nein, Lampenfarbe in Weiß umaendern
      lamp2.Brush.Color := clWhite;
    end;
  // Pruefung ob die Lampe den Wert 1 besitzt...
  if actual_lamps[3] = 1 then
    begin
      // ...wenn ja, Lampenfarbe in Gelb umaendern
      lamp3.Brush.Color := clYellow;
    end
  else
    begin
      // ...wenn nein, Lampenfarbe in Weiß umaendern
      lamp3.Brush.Color := clWhite;
    end;
```

```
  // Ausgabe der einzelnen Zustaende
  lbl_config_1.Caption := 'Zustand 1 verbindet L ' + inttostr(kombinationen[zustaende[1],0]) + '
  und L ' + inttostr(kombinationen[zustaende[1],1]);
  lbl_config_2.Caption := 'Zustand 2 verbindet L ' + inttostr(kombinationen[zustaende[2],0]) + '
  und L ' + inttostr(kombinationen[zustaende[2],1]);
  lbl_config_3.Caption := 'Zustand 3 verbindet L ' + inttostr(kombinationen[zustaende[3],0]) + '
  und L ' + inttostr(kombinationen[zustaende[3],1]);
  // Ausgabe des Aktuellen Status
  lbl_status.Caption := 'Aktueller Status: ' + inttostr(actual_anfangzustand);
end;

procedure Tfrm_paderbox_1.btn_random_configClick(Sender: TObject);
begin
  // Paderbox zufaellig initialisieren durch Aufruf der Prozedur random_anfang
  Paderbox.random_anfang;
  // Paderboxdaten aus der Klasse importieren ins Formular
  Paderbox.show_paderbox(actual_lamps, kombinationen, actual_anfangzustand,
                         zustaende);
  // Paderbox-Daten durch Aufruf der Prozedur ausgabe() ausgeben
  ausgabe();
end;

procedure Tfrm_paderbox_1.FormCreate(Sender: TObject);
begin
  // Objekt Paderbox erstellen aus der Klasse TPaderbox
  Paderbox := TPaderbox.Create;
end;

procedure Tfrm_paderbox_1.btn_RClick(Sender: TObject);
begin
  // Zuruecksetzen der Paderbox in den Anfangszustand durch Aufruf der Prozedur
  // reset_paderbox
  Paderbox.reset_paderbox;
  // Paderboxdaten aus der Klasse importieren ins Formular
  Paderbox.show_paderbox(actual_lamps, kombinationen, actual_anfangzustand,
                         zustaende);
  // Paderbox-Daten durch Aufruf der Prozedur ausgabe() ausgeben
  ausgabe();
end;

procedure Tfrm_paderbox_1.btn_0Click(Sender: TObject);
begin
  // Zustand der Paderbox aendern durch aufruf der entsprechenden Prozedur und
  // uebergabe der Wertes 1
  Paderbox.change_zustand(1);
  // Paderboxdaten aus der Klasse importieren ins Formular
  Paderbox.show_paderbox(actual_lamps, kombinationen, actual_anfangzustand,
                         zustaende);
  // Paderbox-Daten durch Aufruf der Prozedur ausgabe() ausgeben
  ausgabe();
end;
```

```
procedure Tfrm_paderbox_1.btn_1Click(Sender: TObject);
begin
  // Zustand der Paderbox aendern durch aufruf der entsprechenden Prozedur und
  // uebergabe der Wertes 2
  Paderbox.change_zustand(2);
  // Paderboxdaten aus der Klasse importieren ins Formular
  Paderbox.show_paderbox(actual_lamps, kombinationen, actual_anfangzustand,
                                    zustaende);
  // Paderbox-Daten durch Aufruf der Prozedur ausgabe() ausgeben
  ausgabe();
end;

procedure Tfrm_paderbox_1.btn_show_configClick(Sender: TObject);
begin
  // Formular mit der Konfiguration aufrufen
  frm_config_1.show;
end;

end.
```

- **Formular „Konfiguration“**

```
// Zweites Formular-Element um die Konfiguration der Paderbox anzuzeigen
// (= Wichtig fuer die Anzeige der geheimen Programmierung)
unit config_1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, Grids, StdCtrls, paderbox_1;

type
  Tfrm_config_1 = class(TForm)
    GroupBox1: TGroupBox;
    StringGrid1: TStringGrid;
    procedure FormActivate(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  frm_config_1: Tfrm_config_1;

implementation

{$R *.dfm}

procedure Tfrm_config_1.FormActivate(Sender: TObject);
Var
  i, j : Integer;
begin
  // Paderboxdaten aus der Klasse importieren ins Formular
  Paderbox.show_geheim(config_schalter);
  // Konfiguration des Stringgrids
  StringGrid1.Cells[0,0] := 'Status / Schalter';
  StringGrid1.Cells[1,0] := 'Schalter 1';
  StringGrid1.Cells[2,0] := 'Schalter 2';
  StringGrid1.Cells[0,1] := 'Status 1';
  StringGrid1.Cells[0,2] := 'Status 2';
  StringGrid1.Cells[0,3] := 'Status 3';
  // Alle Schalterstellungen nacheinander durchlaufen und dann im Stringgrid
  // ausgeben
  for i := 1 to 2 do
  begin
    for j := 1 to 3 do
    begin
      // Ubertragen des Wertes in das Stringgrid (= Tabellenzelle)
      StringGrid1.Cells[i,j] := IntToStr(config_schalter[i,j]);
    end;
  end;
end;
end.
```

- **Klasse „TPaderbox“**

```
// Klasse TPaderbox
unit paderbox_class;

interface

type type_kombi = array[1..3,0..1] of integer;
type type_lamp = array[1..3] of integer;
type type_zustand = array[1..3] of integer;
type type_schalter = array[1..2,1..3] of integer;

type
  TPaderbox=class
      private
        // Alle drei Kombinationsmoeglichkeiten werden hier gespeichert
        kombinationen : type_kombi;
        // Anfangsstatus der 3 Lampen (AN=1; AUS=0) fuer Ruecksetzen
        save_lamps : type_lamp;
        // Aktueller Status der 3 Lampen (AN=1; AUS=0)
        actual_lamps : type_lamp;
        // Anfangs-Zustand fuer Ruecksetzen
        save_anfangzustand : Integer;
        // Aktueller Zustand
        actual_anfangzustand : Integer;
        // Zuordnung der Lempen zu den einzelnen Zustaenden
        zustaende : type_zustand;
        // Zuzweisung von Schalter/Zustand --> Folgezustand
        config_schalter : type_schalter;
      public
        // Prozedur um eine zufaelligen Anfangszustand zu gerieren
        procedure random_anfang();
        // Prozedur um den Anfangs-Zustaende in zufaelliger Weise zu initialisieren
        procedure random_zustand();
        // Prozedur um eine zufaellige geheime Programmierung der Konfiguration
        // der Schlalter zu ermitteln
        procedure set_schalter();
        // Prozedur um die Zustaende in zufaelliger Weise zu initialisieren
        procedure set_zustaende();
        // Prozedur um die moeglichen Lampen-Kombinationen zu initialisieren
        procedure set_kombi();
        // Prozedur um die Paderbox auf ihren Anfangszustand zurueckzusetzen
        procedure reset_paderbox();
        // Ubermittlung der Daten aus der Klasse ins Formular zum anzeigen
        procedure show_paderbox(var b_actual_lamps:type_lamp;
                                var b_kombinationen:type_kombi;
                                var b_actual_anfangzustand:Integer;
                                var b_zustaende:type_zustand);
        // Prozedur um nach dem druecken einer Taste den neue Zustand zu berechnen
        procedure change_zustand(taste:Integer);
        // Prozedur um die geheime Programierung auszugeben
        procedure show_geheim(var b_config_schalter:type_schalter);
  end;
```

```
implementation

// Prozedur um die geheime Programierung auszugeben
procedure TPaderbox.show_geheim(var b_config_schalter:type_schalter);
begin
  // Geheime Programmierung an Formular uebergeben
  b_config_schalter := config_schalter;
end;

// Prozedur um die moeglichen Lampen-Kombinationen zu initialisieren
procedure TPaderbox.set_kombi();
begin
  // Setzen der 3 moeglichen Lampenverbindungen zu beginn
  // Kombination Lampe1 zu Lampe2
  kombinationen[1,0] := 1;
  kombinationen[1,1] := 2;
  // Kombination Lampe1 zu Lampe3
  kombinationen[2,0] := 1;
  kombinationen[2,1] := 3;
  // Kombination Lampe2 zu Lampe3
  kombinationen[3,0] := 2;
  kombinationen[3,1] := 3;
end;

// Prozedur um nach dem druecken einer Taste den neue Zustand zu berechnen
procedure TPaderbox.change_zustand(taste:Integer);
Var
  buf1, buf2 : Integer;
begin
  // Ermittlung des neuen Zustandes anhand der Tabelle
  // Schalter/Zustand --> Folgezustand
  actual_anfangzustand := config_schalter[taste,actual_anfangzustand];
  // Lampenwert 1 in buffer speichern
  buf1 := kombinationen[actual_anfangzustand,0];
  // Lampenwert 2 in buffer speichern
  buf2 := kombinationen[actual_anfangzustand,1];
  // Pruefung ob die Lampe an ist (=Wert 1)...
  if actual_lamps[buf1] = 1 then
  begin
    // ...wenn ja, dann Lampe ausschalten (=Wert 0)
    actual_lamps[buf1] := 0;
  end
  else
  begin
    // ...wenn nein, dann Lampe einschalten (=Wert 1)
    actual_lamps[buf1] := 1;
  end;
  // Pruefung ob die Lampe an ist (=Wert 1)...
  if actual_lamps[buf2] = 1 then
  begin
    // ...wenn ja, dann Lampe ausschalten (=Wert 0)
    actual_lamps[buf2] := 0;
  end
```

```
else
  begin
    // ...wenn nein, dann Lampe einschalten (=Wert 1)
    actual_lamps[buf2] := 1;
  end;
end;

// Ubermittlung der Daten aus der Klasse ins Formular zum anzeigen
procedure TPaderbox.show_paderbox(var b_actual_lamps:type_lamp;
                                  var b_kombinationen:type_kombi;
                                  var b_actual_anfangzustand:Integer;
                                  var b_zustaende:type_zustand);
begin
  // Aktuelle Statuse der Lampen uebergeben
  b_actual_lamps := actual_lamps;
  // Aktuelle Kombinationen der Zustaende uebergeben
  b_kombinationen := kombinationen;
  // Anfangszustand uebergeben
  b_actual_anfangzustand := actual_anfangzustand;
  // Zustaende uebergeben
  b_zustaende := zustaende;
end;

// Prozedur um die Paderbox auf ihren Anfangszustand zurueckzusetzen
procedure TPaderbox.reset_paderbox();
Var
  i : Integer;
begin
  // Alle drei Lampen durchlaufen
  for i := 1 to 3 do
  begin
    // Gespeicherten Anfangswert der Lampe in aktuellen Lampenwert uerbertragen
    actual_lamps[i] := save_lamps[i];
  end;
  // Gespeicherten Anfangszustand in den aktuellen Anfangszustand uebertragen
  actual_anfangzustand := save_anfangzustand;
end;

// Prozedur um die Zustaende in zufaelliger Weise zu initialisieren
procedure TPaderbox.set_zustaende();
Var
  i : integer;
begin
  // Initialisiert des Zufallszahlengenerator mit einem zufaelligen Wert
  Randomize;
  // Zufallszahl ermitteln von 1 bis 6
  i := random(6)+1;
  // Pruefung welcher Zufallswert ermittelt wurde...
  case i of
  1 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 1;
       zustaende[2] := 2;
       zustaende[3] := 3;
      end;
```

```
  2 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 1;
       zustaende[2] := 3;
       zustaende[3] := 2;
      end;
  3 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 2;
       zustaende[2] := 1;
       zustaende[3] := 3;
      end;
  4 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 2;
       zustaende[2] := 3;
       zustaende[3] := 1;
      end;
  5 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 3;
       zustaende[2] := 1;
       zustaende[3] := 2;
      end;
  6 : begin
       // ...die 3 Zustaende entsprechend zuweisen
       zustaende[1] := 3;
       zustaende[2] := 2;
       zustaende[3] := 1;
      end;
  end;
end;

// Prozedur um eine zufaellige geheime Programmierung der Konfiguration
// der Schlalter zu ermitteln
procedure TPaderbox.set_schalter();
Var
  i, j, buffer : Integer;
begin
  // Initialisiert des Zufallszahlengenerator mit einem zufaelligen Wert
  Randomize;
  // Alle Elemente der Zuweisungstabelle Schalter/Zustand --> Folgezustand durchlaufen
  for i := 1 to 2 do
  begin
    for j := 1 to 3 do
    begin
      // Zufallszahl ermitteln von 0 bis 3
      buffer := random(3);
      buffer := buffer + 1;
      // Werte den einzelnen Elementen der Zuweisungstabelle
      // Schalter/Zustand --> Folgezustand zuweisen
      config_schalter[i,j] := buffer;
    end;
  end;
end;
```

```
// Prozedur um die Anfangs-Zustaende in zufaelliger Weise zu initialisieren
procedure TPaderbox.random_zustand();
Var
  buffer : Integer;
begin
  // Initialisiert des Zufallszahlengenerator mit einem zufaelligen Wert
  Randomize;
  // Zufallszahl ermitteln von 0 bis 3
  buffer := random(3);
  // Wert der Variable um den Wert 1 erhoehen
  buffer := buffer + 1;
  // Wert fuer Anfangszustand sichern, zum spaeteren zuruecksetzen der Paderbox
  save_anfangzustand := buffer;
  // Wert fuer Anfangszustand uebertragen
  actual_anfangzustand := buffer;
end;

// Prozedur um eine zufaelligen Anfangszustand zu gerieren
procedure TPaderbox.random_anfang();
Var
  i, buffer : Integer;
begin
  // Aufruf der Prozedur set_kombi() um die moeglichen Lampen-Kombinationen zu
  // initialisieren
  set_kombi();
  // Initialisiert des Zufallszahlengenerator mit einem zufaelligen Wert
  Randomize;
  // Setzen der 3 Lampen in einen zufaelligen Anfangszustand
  for i := 1 to 3 do
  begin
    // Zufallszahl ermitteln von 0 bis 1
    buffer := random(2);
    // Anfangs-Lampenzustand speichern, um spaeter die paderbox zuruecksetzen zu
    // koennen
    save_lamps[i] := buffer;
    // Anfangs-Lampenzustand uebertragen
    actual_lamps[i] := buffer;
  end;
  // Aufruf der Prozedur set_zustaende() um die Zustaende in zufaelliger Weise zu
  // initialisieren
  set_zustaende();
  // Aufruf der Prozedur set_schalter() um eine zufaellige geheime Programmierung
  // der Konfiguration der Schlalter zu ermitteln
  set_schalter();
  // Aufruf der Prozedur random_zustand() um den Anfangs-Zustaende in
  // zufaelliger Weise zu initialisieren
  random_zustand();
end;

end.
```