

# **24. Bundeswettbewerb-Informatik**

**Lösungen der ersten Runde**

Karsten Bohlen

7. November 2005

## Inhaltsverzeichnis

<b>Allgemeines</b>	<b>2</b>
<b>1 Favorites First</b>	<b>4</b>
1.1 Lösungsidee . . . . .	4
1.2 Programm-Dokumentation . . . . .	5
1.3 Programm-Ablaufprotokoll . . . . .	6
1.4 Programm-Text . . . . .	8
<b>2 Formel-Up</b>	<b>11</b>
2.1 Lösungsidee . . . . .	11
2.2 Programm-Dokumentation . . . . .	12
2.3 Programm-Ablaufprotokoll . . . . .	14
2.4 Programm-Text . . . . .	19
<b>3 Zaras Zauberfolie</b>	<b>23</b>
3.1 Lösungsidee . . . . .	23
3.2 Programm-Dokumentation . . . . .	25
3.3 Programm-Ablaufprotokoll . . . . .	27
3.4 Programm-Text . . . . .	30
<b>5 Kleingeld</b>	<b>36</b>
5.1 Lösungsidee . . . . .	36
5.2 Programm-Dokumenation . . . . .	37
5.3 Programm-Ablaufprotokoll . . . . .	38
5.4 Programm-Text . . . . .	48

## Allgemeines

### Dokumentation

Diese Dokumentation wurde mit dem Texteditor *Kate* geschrieben und durch das Textsatzsystem  $\text{\LaTeX}$  kompiliert. Alle Seiten sind nummeriert und mit den Namen der Aufgaben versehen.

### Computersystem

Es wurde ein AMD64 3400+ mit 2.4 GHz und Mandriva Linux (Kernel 2.6.11) verwendet.

### Programmiersprache

Für alle Aufgaben wurde C++ verwendet. Die Programme wurden mit g++ (gcc) 3.4.3 kompiliert.

### Aufgaben

Es wurden die Aufgaben 1, 2, 3 und 5 gelöst.

### Kontakt

Fragen und Anmerkungen an [b.karsten\\_@freenet.de](mailto:b.karsten_@freenet.de)

## 1 Favorites First

### 1.1 Lösungsidee

#### *Favorites First Spezifikation*

Bei jedem Einschalten des Players werden vorhandene  $N$  Stücke in eine zufällige, gleichverteilte Reihenfolge gebracht. Aus der Teilmenge von 10 meistgehörten werden einer möglichen Auswahl von 5 jeweils eine neue, zufällige Position aus  $[0, 20)$  zugeordnet. Insgesamt ändert sich also die Position von max. 10 Stücken in der gesamten Abspielfolge.

#### *Simulation*

Es soll ein spezielles Abspielverhalten simuliert werden. In jedem Simulationsschritt (bei jeder Benutzung des Players) wird zunächst die *Favorites-First*-Spezifikation ausgeführt. Die ersten  $L \in [10, 20]$  Stücke werden abgespielt oder angespielt (=übersprungen),  $\lfloor \frac{L}{4} \rfloor$  davon auf jedenfall übersprungen.

#### *Unerwünschtes Verhalten*

Wenn ein bestimmtes Stück beim ersten Abspielen zufällig schon unter den ersten  $L$  war und abgespielt wird, so ist die Wahrscheinlichkeit hoch, dass es wieder unter die ersten 20 gerät usw. Einige Stücke werden also so sehr häufig abgespielt nur weil sie zu Beginn zufällig unter den ersten Stücken waren, andere dafür wesentlich seltener.

#### *Die Algorithmen*

Mehrmals im Laufe der Simulation müssen Stichproben genommen werden. Dabei interessiert man sich für eine zufällige Auswahl von  $k$  Elementen aus dem Intervall  $[0, n)$ . Der Algorithmus wählt in jedem Schritt  $i$  von  $k$  Schritten eine Zufallszahl aus  $[0, (n - i))$ . Der gewählte Wert wird inkrementiert für jede Ziffer in der bisherigen Liste, die kleiner oder gleich besagtem Wert ist. Mehrfachauswahlen werden so vermieden.

## 1.2 Programm-Dokumentation

### Die Struktur `struct Titel`

Hier wird jedes Stück, das sich auf dem Player befindet beschrieben. In `int id` wird eine feste Bezeichnung / Nummer für das Stück gespeichert, `int count` zählt die Abspielungen dieses Musikstücks.

### Konstanten, globale Variablen, Arrays

`const int M = 20` und `const int N = 250` erste und totale Anzahl der Titel. Der `titel` (Vector von Titel) enthält alle Stücke (derzeitige Abspielfolge), in `TopTen` wird die jeweilige Top 10 gespeichert, in `Favorit` 5 Stücke aus 10, in `bool used[M]` werden zufällig selektierte Elemente markiert.

### Die Funktion `void select(int, int)`

Dies ist der Selektionsalgorithmus für `int k` Elemente zufällig aus `int n` zu wählen. Ein dynamisches Array `int *a` speichert die `k` Elemente. Die Auswahl wird nach Ende der Schleife an `bool used[M]` übergeben.

### Die Funktion `void fav()`

Hier wird zunächst der `vector` mit allen Titeln kopiert um ihn dann teilweise zu sortieren. Es werden die 10 meistgespielten Stücke an den Anfang des `vector` verschoben. Anschließend werden die ersten 10 Elemente in `TopTen` kopiert.

### Die Funktion `inline int pos(Title t)`

Hier wird die Position eines Titels in der Abspielfolge zurückgegeben.

### Die Funktion `int main(int argc, char *argv[])`

Mit `argv[1]` wird die Anzahl der Simulationen entgegengenommen. Anschließend wird der (Pseudo-)Zufallszahlengenerator initialisiert. Die Simulation wird dann in einer `for`-Schleife ausgeführt.

### 1.3 Programm-Ablaufprotokoll

#### Ausgabeformat

<Verwendung Nr.> <jeweils gespielter Titel(<Anzahl der Abspielungen>)>

Am Ende wird die Top-Ten nach **S** Simulationen ausgegeben.

#### 300 Benutzungen (Ausgabe gekürzt)

```

0: Spiele Titel...
185(1) 245(1) 160(1) 152(1) 7(1) 75(1) 200(1) 36(1) 3(1)

1: Spiele Titel...
115(1) 61(1) 200(2) 207(1) 152(2) 172(1) 160(2) 25(1) 26(1)

2: Spiele Titel...
200(3) 204(1) 25(2) 18(1) 182(1) 84(1) 246(1) 185(2) 115(2) 36(2) 223(1) 238(1)

3: Spiele Titel...
200(4) 52(1) 243(1) 160(3) 147(1) 73(1) 65(1) 92(1) 104(1) 188(1) 175(1) 53(1)

4: Spiele Titel...
163(1) 141(1) 219(1) 84(2) 160(4) 122(1) 212(1) 96(1) 230(1) 25(3) 182(2) 80(1) 19(1)

5: Spiele Titel...
183(1) 196(1) 90(1) 127(1) 78(1) 11(1) 79(1) 231(1) 18(2) 25(4) 186(1) 17(1) 36(3)

6: Spiele Titel...
152(3) 33(1) 84(3) 232(1) 25(5) 200(5) 182(3) 220(1) 185(3) 171(1) 156(1)

7: Spiele Titel...
66(1) 161(1) 48(1) 135(1) 131(1) 150(1) 25(6) 21(1) 152(4) 185(4) 167(1) 191(1)
[...]
280: Spiele Titel...
84(12) 73(13) 183(13) 25(108) 9(11) 76(8) 214(15) 192(10)

281: Spiele Titel...
36(92) 26(7) 72(12) 200(111) 223(9) 168(6) 130(8) 239(10) 249(11) 152(90) 185(120) 120(4)

282: Spiele Titel...
241(4) 71(12) 21(6) 104(9) 191(14) 200(112) 25(109) 182(84) 160(108) 126(11) 18(75) 63(53)

283: Spiele Titel...
25(110) 48(13) 189(10) 185(121) 160(109) 2(8) 46(8) 81(12) 142(13) 36(93) 31(10) 187(11)

284: Spiele Titel...
25(111) 160(110) 152(91) 189(11) 31(11) 182(85) 205(35) 28(8) 54(12)

285: Spiele Titel...
18(76) 177(12) 36(94) 123(11) 37(7) 152(92) 90(14) 156(13) 107(8) 139(14) 53(9) 193(11)

286: Spiele Titel...
38(9) 160(111) 61(11) 90(15) 54(13) 187(12) 152(93) 33(10) 8(10) 10(9) 14(10) 209(11) 135(12)
18(77) 64(14)

287: Spiele Titel...
25(112) 162(5) 55(13) 206(14) 200(113) 152(94) 142(14) 52(13) 178(8) 18(78)

288: Spiele Titel...
41(12) 36(95) 86(11) 141(14) 131(9) 18(79) 130(9) 162(6) 39(13) 3(13) 90(16) 63(54) 238(13)

289: Spiele Titel...
108(9) 135(13) 41(13) 160(112) 204(8) 36(96) 23(15) 194(7) 63(55) 123(12) 50(10) 184(13)

290: Spiele Titel...
58(10) 34(10) 147(11) 80(9) 47(13) 250(10) 200(114) 2(9) 25(113)

291: Spiele Titel...
193(12) 25(114) 98(9) 110(5) 135(14) 182(86) 18(80) 63(56) 166(12) 45(10) 205(36) 248(11)
20(13) 93(10)

292: Spiele Titel...
43(9) 199(11) 211(11) 25(115) 75(13) 169(13) 126(12) 65(17) 152(95) 194(8) 114(24) 51(11)
18(81)

```

293: Spiele Titel...  
200(115) 215(13) 22(11) 178(9) 212(14) 187(13) 145(10) 209(12) 160(113) 120(5) 182(87) 95(9)  
12(11)

294: Spiele Titel...  
200(116) 191(15) 85(10) 152(96) 18(82) 11(13) 31(12) 165(6) 196(17) 216(11) 156(14) 104(10)

295: Spiele Titel...  
54(14) 15(10) 36(97) 156(15) 161(16) 217(10) 94(10) 184(14) 245(9) 182(88) 208(9) 141(15)  
20(14) 96(12) 18(83)

296: Spiele Titel...  
125(13) 218(8) 176(14) 25(116) 36(98) 152(97) 219(13) 120(6) 182(89) 114(25) 132(11) 205(37)  
91(8) 234(7)

297: Spiele Titel...  
25(117) 243(8) 108(10) 69(11) 250(11) 58(11) 159(13) 182(90) 194(9)

298: Spiele Titel...  
222(12) 120(7) 14(11) 8(11) 29(9) 185(122) 25(118) 88(16) 152(98) 78(17) 129(8) 233(8)  
171(11) 229(9) 18(84)

299: Spiele Titel...  
54(15) 174(8) 57(11) 217(11) 116(11) 138(12) 25(119) 173(6) 152(99) 11(14) 18(85) 8(12)  
205(38)

Top 10 nach 300 Simulationen  
Titel 185 122 mal abgespielt  
Titel 25 119 mal abgespielt  
Titel 200 116 mal abgespielt  
Titel 160 113 mal abgespielt  
Titel 152 99 mal abgespielt  
Titel 36 98 mal abgespielt  
Titel 182 90 mal abgespielt  
Titel 18 85 mal abgespielt  
Titel 63 56 mal abgespielt  
Titel 205 38 mal abgespielt

Einige Stücke werden sehr häufig abgespielt, da diese schon zu Beginn zufällig in der Abspielliste enthalten waren.

## 1.4 Programm-Text

```
1 // BWNF 24.1
2 // Lösung Aufgabe 2
3 // favorites.cpp
4 // geschrieben 2005 by Karsten Bohlen
5 #include <iostream>
6 #include <vector>
7 #include <algorithm>
8 #include <cstdlib>
9 #include <ctime>
10 using namespace std;
11
12 const int M = 20;
13 const int N = 250; // Anzahl der Stücke auf dem Player
14
15 struct Titel {
16     Titel() { count = 0; }
17     int id, count;
18 };
19
20 vector<Titel> titel(N); // Liste der Titel auf dem Player
21 vector<Titel> TopTen(10); // Liste der 10 beliebtesten Titel
22 vector<Titel> Favorit(5); // 5 Stücke aus TopTen
23 bool used[M]; // Für den Auswahl-Algorithmus
24
25 void select(int n, int k); // k Elemente zufällig aus [0,n)
26 // markieren
27 void fav(); // stelle aktuelle Top 10 zusammen
28
29 inline int pos(Titel); // Position in der Liste abfragen
30
31 int main(int argc, char *argv[])
32 {
33     if (argc != 2) {
34         cerr << "Anzahl der Simulationen nicht angegeben." << endl;
35         return 1;
36     }
37
38     int S = atoi(argv[1]);
39
40     for (int i = 0; i < N; i++)
41         titel[i].id = i+1; // eindeutige ID festlegen
42
43     srand(time(NULL)); // seed initialisieren
44
45     // simuliere Abspielung
46     for (int i = 0; i < S; i++)
47     {
48         random_shuffle(titel.begin(), titel.end()); // (re)shuffle
49
50         int L = (rand()%10)+10; // L \in [10,20]
51
52         fav(); // Top 10 zusammenstellen
```



```
52
53     select(10, 5);           // 5 aus 10 markieren
54
55     for (int j = 0, k = 0; j < 10; j++)
56     {
57         if (used[j])
58         {
59             Favorit[k] = TopTen[j];
60             ++k;
61         }
62     }
63
64     select(M, 5);           // Favoriten 5 Positionen zuordnen
65
66     for (int j = 0, k = 0; j < M; j++)
67     {
68         if (used[j])
69         {
70             int p = pos(Favorit[k]);    // alte Position merken
71             Titel tmp = titel[j];
72             titel[j] = Favorit[k];
73             titel[p] = tmp;
74             ++k;
75         }
76     }
77
78     cout << endl;
79
80     select(L, int(L/4));    // zu überspringende Stücke
                             // markieren
81
82     cout << i << ": Spiele Titel... " << endl;
83
84     for (int j = 0; j < L; j++)
85     {
86         if (!used[j])
87         {
88             ++titel[j].count; // Titel abspielen
89             cout << titel[j].id << "(" << titel[j].count << ") ";
90         }
91     }
92
93     cout << endl;
94 }
95
96 fav();           // neue Top 10
97
98 cout << "\nTop 10 nach " << S << " Simulationen " << endl;
99
100 for (int i = 0; i < 10; i++)
101 {
102     cout << "Titel " << TopTen[i].id << " "
103         << TopTen[i].count << " mal abgespielt" << endl;
104 }
```

```
105
106     return 0;
107 }
108
109 void select(int n, int k)
110 {
111     int *a = new int[k];
112
113     for (int i = 0; i < n; i++) used[i] = false;
114     for (int i = 0; i < k; i++) a[i] = 0;
115
116     for (int i = 0; i < k; i++)
117     {
118         a[i] = rand()%(n-i);
119         for (int j = 0; j < i; j++) if (a[j] <= a[i]) a[i]++;
120         for (int j = 0; j < i; j++) if (a[j] == a[i]) a[i]++;
121     }
122
123     for (int i = 0; i < k; i++)
124         used[a[i]] = true;
125
126     delete [] a;
127 }
128
129 void fav()
130 {
131     vector<Titel> Tmp(titel.begin(), titel.end());
132
133     // Liste teil-sortieren
134     for (int i = 0; i < 10; i++)
135     {
136         int max = i;
137         for (int j = i+1; j <= N; j++)
138             if (Tmp[j].count > Tmp[max].count) max = j;
139
140         swap(Tmp[i], Tmp[max]);
141     }
142
143     // 10 Stücke übernehmen
144     for (int i = 0; i < 10; i++)
145         TopTen[i] = Tmp[i];
146 }
147
148 inline int pos(Titel t) {
149     for (int i = 0; i < N; i++)
150         if (titel[i].id == t.id)
151             return i;
152     return -1;
153 }
```

## 2 Formel-Up

### 2.1 Lösungsidee

Im folgenden bezeichne  $N$  die Anzahl der Experimente in der Tabelle,  $(a, b, c)$  bezeichnet ein Lösungstriplett aus ganzzahligen Faktoren,  $(A, B, C)$  seien die Werte ( $H$  – Wert,  $A$  – Wert,  $M$  – Wert). Der M-Idealwert ist determiniert durch  $\frac{aA+bB}{c}$ , die Abweichung also  $|C - \frac{aA+bB}{c}|$ . Im Programm verwende ich die äquivalente Definition:  $abw(A, B, C, a, b, c) := |aA + Bb - cC|$ , da diese aufgrund ganzzahliger Ergebnisse schneller ist und besser vergleichbar.

Ziel des Anwenders kann immer nur sein die Abweichung zu minimieren. Wollte man möglichst kleine Faktoren könnte man gleich  $(1, 1, 1)$  wählen. Hinzu kommt, dass die in der Aufgabenstellung gegebenen Bedingungen zu unpräzise bzw. subjektiv sind für eine mathematisch exakte Lösung. Der Lösungsansatz besteht also darin alle Lösungstriplett  $(a, b, c)$  durchzuprobieren. Dies wird bis zu einer bestimmten Obergrenze getan, die ich mit  $K$  bezeichne. Wir stellen also eine Menge zusammen, welche die Form:  $M = \{(a, b, c) | 1 \leq a, b, c \leq K\}$  hat. Dabei gilt  $|M| = K^3$ . Jedes Lösungstriplett wird auf die Tabelle angewendet. Dabei ist zu beachten, dass für jedes Triplett  $(a, b, c)$  zwei Formeln erzeugt werden, einmal mit positivem und einmal mit negativem Wert  $b$ . Mit gegebenem  $(a, b, c)$ , für zwei Formeln mit  $(a, b, c)$  und  $(a, -b, c)$  für jedes der  $N$  Triplett  $(A, B, C)$  wird also:

- Die absolute M-Wert Abweichung bestimmt
- Maximal 3 fehlerhafte Experimente festgehalten ODER:
- Die Abweichung zur bisherigen Summe hinzuaddiert

Es werden immer nur die 3 Experimente mit den größten Abweichungen in die Lösungsliste aufgenommen. Nach dem Ende der Schleife wird diese Formel zu einer der drei besten / schönsten hinzugefügt, wenn die Abweichung kleiner ist bzw. die Anzahl fehlerhafter Experimente kleiner ist als in einer der bisher besten. Die Laufzeitkomplexität dieses Algorithmus ist also  $O(NK^3)$ . Der Parameter  $K$  spielt hier eine wichtige Rolle, er ist standardmäßig auf  $K = 100$  gesetzt, kann aber vom Benutzer frei gewählt werden. So hat dieser die freie Wahl bei der Minimierung der Abweichung. Es können nämlich entweder sehr kleine Faktoren gewählt werden oder die Abweichung mit steigenden Faktoren minimiert werden. Etwas dazwischen ist mathematisch undefiniert.

## 2.2 Programm-Dokumentation

### Die Struktur `struct Experiment`

Hier sind für jedes Experiment H-Wert, A-Wert, M-Wert in **unsigned HWert**, **AWert**, **MWert** festgehalten. Außerdem die Bezeichnung des Experiments in **string name** und im Konstrukt **Experiment(const string& name\_)** initialisiert.

### Die Struktur `struct Formel`

Für jede Formel sind hier die 3 Faktoren in **int f1**, **f2**, **f3** gespeichert, die Abweichung **unsigned Abw**, die Anzahl der zu löschenden Experimente (maximal 3) **int l**, die Indizes der zu löschenden Experimente in **int del[3]**. In **map<int, unsigned> A** wird die Abweichung zu einem bestimmten **del[i]** gespeichert. Im Konstrukt **Formel(int f1\_, int f2\_, int f3\_)** werden diese Werte initialisiert.

### globale Variablen etc

In **int N** wird die Anzahl der Experimente erfasst, **int K = 100** ist der erwähnte Parameter mit Defaultwert 100, **counter** merkt sich wieviele schönste Formeln (maximal 3) schon gefunden wurden. Die schönsten Formeln werden im Array **Formel Beste[3]** gespeichert. Ein Vektor **Tabelle** beinhaltet die Tabelle mit allen Experimenten.

### Funktionen

#### Die Funktion `int main(int argc, char *argv[])`

- Nimmt Dateinamen und Parameter **K** (falls gewünscht) mit **argv[1]**, **argv[2]** entgegen
- Ruft die Funktion **bool initialisiere(char \*)** mit dem Dateinamen auf
- Gibt alle möglichen Tripel natürlicher Zahlen an **formuliere(int, int, int)** weiter
- Gibt die drei schönsten Formeln in geforderter Weise aus

#### Die Funktion `bool initialisiere(char *datei)`

Hier wird der Dateiname mit den Experimenten entgegen genommen und in **Tabelle** überführt.

#### Die Funktion `void formuliere(int a, int b, int c)`

Zunächst werden zwei Formeln erzeugt mit  $(a, b, c)$  und  $(a, -b, c)$ . Für alle Experimente wird nun in der Schleife die Abweichung berechnet etc. Dazu

wird die Hilfsfunktion **void evalAbw(Formel&, int, int)** für jede der beiden Formeln aufgerufen mit der Formel, der errechneten Abweichung und der Tabellenspalte. Am Ende der Schleife wird für beide Formeln die Hilfsfunktion **void neueBeste(const Formel&)** aufgerufen.

**Die Hilfsfunktion void evalAbw(Formel& F, int Abw, int m)**

Für eine Formel wird die Tabellenspalte **int m** entweder als zu löschendes Experiment markiert, falls noch keine drei gespeichert wurden oder die Abweichung **int Abw** zu der Summe der Abweichungen hinzuaddiert. Dabei wird vorher noch getestet ob die jetzige Abweichung größer ist als die eines Löschkandidaten und gegebenenfalls ersetzt. So werden nur die größten Abweichungen gelöscht.

**Die Hilfsfunktion void neueBeste(const Formel&)**

Nach jeder Anwendung einer Formel auf die Tabelle wird hier ermittelt ob die Formel eine neue schönste ist. Standardmäßig hinzugefügt wird sie falls noch keine drei besten gespeichert wurden. Ansonsten wird festgestellt ob die Abweichung geringer ist oder weniger fehlerhafte Experimente (falls weniger als drei vorhanden) als bei einer der bisher gespeicherten.

## 2.3 Programm-Ablaufprotokoll

### Eingabezeile

```
formulierer Dateiname <K>
```

### Beispiele

#### Beispiel 0: Messtabelle aus der Aufgabenstellung

##### *Input*

```
Experiment:  H-Wert  A-Wert  M-Wert
1: 22 5 13
2: 57 31 29
3: 44 31 19
4: 42 21 21
5: 128 1 85
```

##### *Output*

```
./formulierer bsp.txt
```

```
2 * H-Wert -1 * A-Wert = 3 * M-Wert ist eine schöne Formel
Habe fehlerhaftes Experiment 2: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
4 * H-Wert -2 * A-Wert = 6 * M-Wert ist eine schöne Formel
Habe fehlerhaftes Experiment 2: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
6 * H-Wert -3 * A-Wert = 9 * M-Wert ist eine schöne Formel
Habe fehlerhaftes Experiment 2: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

#### Beispiel 1: test.txt

##### *Input*

```
Experiment:  H-Wert  A-Wert  M-Wert
1: 5 4 6
2: 6 8 7
3: 11 11 11
4: 5 4 3
5: 7 7 7
6: 12 10 11
7: 4 5 6
8: 255 257 256
9: 8 6 7
10: 14 8 11
```

*Output*

```
./formulierer test.txt

2 * H-Wert + 2 * A-Wert = 4 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 4: 7: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0

3 * H-Wert + 3 * A-Wert = 6 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 4: 7: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0

1 * H-Wert + 1 * A-Wert = 2 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 4: 7: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

**Beispiel 2: bsp2.txt***Input*

```
Experiment: H-Wert A-Wert M-Wert
1: 5 4 6
2: 5 10 10
3: 50 200 1
4: 70 1660 6
5: 2 296 1
6: 9 7 19
```

*Output*

```
./formulierer bsp2.txt 900

2 * H-Wert + 1 * A-Wert = 300 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 2: 6: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0

4 * H-Wert + 2 * A-Wert = 600 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 2: 6: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0

6 * H-Wert + 3 * A-Wert = 900 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 2: 6: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

**Beispiel 3: bsp3.txt***Input*

```
Experiment H-Wert A-Wert M-Wert
1: 1 2 51
2: 3 6 153
3: 6 7 204
```

```
4: 90 40 1734
5: 10 10 306
6 2 4 102
```

### *Output*

```
./formulierer bsp3.txt 400
```

```
51 * H-Wert + 102 * A-Wert = 5 * M-Wert ist eine schöne Formel
Keinerlei M-Wert Abweichungen!
```

```
102 * H-Wert + 204 * A-Wert = 10 * M-Wert ist eine schöne Formel
Keinerlei M-Wert Abweichungen!
```

```
153 * H-Wert + 306 * A-Wert = 15 * M-Wert ist eine schöne Formel
Keinerlei M-Wert Abweichungen!
```

### **Beispiel 4: bsp4.txt**

#### *Input*

```
Experiment:  H-Wert  A-Wert  M-Wert
1: 5 10 135
2: 11 45 410
3: 3 1 16
4: 99 13 358
5: 10 8 93
6: 7 5 60
7: 22 12 157
8: 9 9 99
9: 12 25 5
10: 78 24 399
11: 713 23 1978
12: 4 8 78
13: 1 7 62
14: 88 12 322
15: 1 3 28
```

### *Output*

```
./formulierer bsp4.txt
```

```
5 * H-Wert + 17 * A-Wert = 2 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 9: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
10 * H-Wert + 34 * A-Wert = 4 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 9: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
15 * H-Wert + 51 * A-Wert = 6 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 1: 9: gelöscht.
```



Damit ist die Summe der absoluten M-Wert Abweichungen 0

### Beispiel 5: bsp5.txt

#### *Input*

```
Experiment:  H-Wert  A-Wert  M-Wert
1: 1 1 1
2: 3 5 10
3: 4 7 14
```

#### *Output*

```
./formulierer bsp5.txt
```

```
1 * H-Wert + 2 * A-Wert = 3 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 2: 3: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
1 * H-Wert + 3 * A-Wert = 4 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 2: 3: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

```
1 * H-Wert + 1 * A-Wert = 2 * M-Wert ist eine schöne Formel
Habe fehlerhafte Experimente 2: 3: gelöscht.
Damit ist die Summe der absoluten M-Wert Abweichungen 0
```

### Zufallszahlen

#### *Input*

```
Experiment  H-Wert  A-Wert  M-Wert
1: 126   4   38
2: 143  128   71
3: 6   187  198
4: 82  164   46
5: 143   30  191
6: 67  122   10
7: 136  169  175
8: 79  155  195
9: 137  178   14
10:   57  18  152
11:   13   8   88
12:  142   80  166
13:  129   78  110
14:   45  76   68
15:   75  67  102
16:  150   77  46
17:   71   5   65
18:  178  152  154
19:  108  118   33
20:  126   71  122
```

*Output*

```
./formulierer rand.txt 1000
```

```
2 * H-Wert -1 * A-Wert = 1 * M-Wert ist eine schöne Formel  
Habe fehlerhafte Experimente 1: 2: 3: gelöscht.  
Damit ist die Summe der absoluten M-Wert Abweichungen 1084
```

```
1 * H-Wert -1 * A-Wert = 1 * M-Wert ist eine schöne Formel  
Habe fehlerhafte Experimente 1: 2: 3: gelöscht.  
Damit ist die Summe der absoluten M-Wert Abweichungen 1364
```

```
2 * H-Wert -2 * A-Wert = 1 * M-Wert ist eine schöne Formel  
Habe fehlerhafte Experimente 1: 2: 3: gelöscht.  
Damit ist die Summe der absoluten M-Wert Abweichungen 1571
```

Die Abweichung lässt sich auch mit großem **K** kaum minimieren.

## 2.4 Programm-Text

**formulierer.cpp**, außerdem **RandTable.cpp** zum Erzeugen der Zufallszahlen (hier nicht abgedruckt)

```

1 // BWNF 24.1
2 // Lösung – Aufgabe 2
3 // Formulierer v0.1
4 // geschrieben 2005 von Karsten Bohlen
5 #include <iostream>
6 #include <fstream>
7 #include <vector>
8 #include <map>
9 #include <cstdlib>
10 #include <cmath>
11 using namespace std;
12
13 // repräsentiert ein Experiment
14 struct Experiment {
15     unsigned HWert, AWert, MWert;
16     string name;
17
18     Experiment(const string& name_) { name = name_; }
19 };
20
21 struct Formel {
22     int f1, f2, f3;
23     unsigned Abw; // Summe absolute M-Wert Abweichung
24     int l; // Anzahl der zu löschenden Experimente
25     int del[3]; // zu löschende Experimente (Indizes)
26     map<int, unsigned> A; // del[i] nach Abweichung
27
28     Formel(int f1_, int f2_, int f3_) {
29         f1 = f1_; f2 = f2_; f3 = f3_;
30         for (int i = 0; i < 3; i++)
31             del[i] = -1;
32         Abw = 0; l = 0;
33     }
34     Formel() {}
35 };
36
37 int N; // Anzahl der Experimente
38 int K = 100; // K Durchläufe, Defaultwert 100
39 int counter = 0; // zählt schönste Formeln
40 vector<Experiment> Tabelle; // Tabelle mit Experimenten
41 Formel Beste[3]; // 3 schönste Formeln speichern
42
43 bool initialisiere(char *); // Tabelle einlesen etc
44 void formuliere(int, int, int); // Hauptalgorithmus
45
46 // Hilfsfunktionen:
47 void evalAbw(Formel&, int, int); // summiert Abweichung /
    speichert zu löschende Formeln

```

Karsten Bohlen

```

48 void neueBeste(const Formel&);           // ermittelt schönste Formeln
49
50 int main(int argc, char *argv[])
51 {
52     if (argc == 1) {
53         cerr << "Dateiname fehlt!" << endl;
54         return 1;
55     }
56     else if (!initialisiere(argv[1]))
57         return 2;
58     else if (argc == 3)
59         K = atoi(argv[2]);
60
61     // Algorithmus...
62     for (int a = 1; a <= K; a++)
63         for (int b = 1; b <= K; b++)
64             for (int c = 1; c <= K; c++)
65                 formuliere(a, b, c); // berechne Abweichungen etc
66
67     // Ausgabe...
68     for (int i = 0; i < 3; i++)
69     {
70         cout << endl;
71
72         if (Beste[i].f2 > 0)
73         {
74             cout << Beste[i].f1 << " * H-Wert + " << Beste[i].f2 << " *
75                 * A-Wert = "
76                 << Beste[i].f3 << " * M-Wert ist eine schöne Formel"
77                 << endl;
78         }
79         else {
80             cout << Beste[i].f1 << " * H-Wert " << Beste[i].f2 << " *
81                 A-Wert = "
82                 << Beste[i].f3 << " * M-Wert ist eine schöne Formel"
83                 << endl;
84         }
85
86         if (Beste[i].l == 1)
87             cout << "Habe fehlerhaftes Experiment ";
88         else if (Beste[i].l != 0)
89             cout << "Habe fehlerhafte Experimente ";
90
91         for (int j = 0; j < Beste[i].l; j++)
92             cout << Tabelle[Beste[i].del[j]].name << " ";
93
94         if (Beste[i].l != 0) {
95             cout << " gelöscht." << endl;
96             cout << "Damit ist die Summe der absoluten M-Wert
97                 Abweichungen "
98                 << Beste[i].Abw << endl;
99         }
100        else
101            cout << "Keinerlei M-Wert Abweichungen!" << endl;

```

```
97     }
98
99     return 0;
100 }
101
102 bool initialisiere(char *datei)
103 {
104     ifstream ifs(datei);
105     string tmp;
106
107     if (!ifs.is_open()) {
108         cerr << "Konnte Datei nicht öffnen!" << endl;
109         return false;
110     }
111
112     ifs >> tmp; ifs >> tmp; ifs >> tmp; ifs >> tmp;
113
114     while (!ifs.eof())
115     {
116         ifs >> tmp;
117
118         if (tmp == "Experiment:")
119             break;
120
121         Experiment experiment(tmp);
122
123         ifs >> experiment.HWert;
124         ifs >> experiment.AWert;
125         ifs >> experiment.MWert;
126
127         Tabelle.push_back(experiment);
128     }
129
130     N = Tabelle.size()-1;
131
132     return true;
133 }
134
135 void formuliere(int a, int b, int c)
136 {
137     Formel F1(a, b, c);
138     Formel F2(a, -b, c);
139
140     for (int m = 0; m < N; m++)
141     {
142         unsigned A = Tabelle[m].HWert;
143         unsigned B = Tabelle[m].AWert;
144         unsigned C = Tabelle[m].MWert;
145
146         int Abw1 = a*A + b*B - c*C;
147         int Abw2 = a*A - b*B - c*C;
148
149         evalAbw(F1, Abw1, m); evalAbw(F2, Abw2, m);
150     }
```

```
151
152     neueBeste(F1);
153     neueBeste(F2);
154 }
155
156 void evalAbw(Formel& F, int Abw, int m)
157 {
158     if (Abw != 0)
159     {
160         if (F.l != 3)
161         {
162             F.del[F.l] = m;
163             F.A[m] = abs(Abw);
164             ++F.l;
165             return;
166         }
167
168         for (int i = 0; i < 3; i++)
169         {
170             int p = F.del[i];
171
172             if (abs(Abw) > F.A[p])
173             {
174                 unsigned tmp = F.A[p];
175                 F.A[p] = abs(Abw);
176                 Abw = tmp;
177                 break;
178             }
179         }
180     }
181
182     F.Abw += abs(Abw);
183 }
184
185 void neueBeste(const Formel& F)
186 {
187     if (counter != 3)
188     {
189         Beste[counter] = F;
190         ++counter;
191         return;
192     }
193
194     for (int i = 0; i < 3; i++)
195     {
196         if (Beste[i].Abw > F.Abw || Beste[i].l > F.l)
197         {
198             Beste[i] = F;
199             return;
200         }
201     }
202 }
```

## 3 Zaras Zauberfolie

### 3.1 Lösungsidee

#### Verschlüsselung / Entschlüsselung

Die Verschlüsselung ist eine XOR-Verknüpfung:

Folie XOR Original = Fax

Die Entschlüsselung ist eine OR-Verknüpfung:

Folie OR Fax = Nachricht

Schlüssel OR Code = Klartext

Die Unterschiede auf Folie und Fax werden also überdeckt. Jeder Code beinhaltet also den Wert des Schlüssels an den Stellen, die nicht überdeckt werden. Die sich unterscheidenden Stellen auf den Codes sind für die Entschlüsselung relevant. Je mehr Codes man zur Verfügung hat, desto besser lässt sich die Nachricht rekonstruieren. Die Lösung basiert somit auf dem Vergleich eines Codes mit jeweils allen anderen. Für jeden Vergleich wird eine eigene Xor-Matrix erstellt. Mit einer AND Verknüpfung auf alle diese Matrizen wird der Bereich hervorgehoben an dem die Folie sich von allen anderen gleichsam unterscheidet und die Nachricht erkennbar wird. Mit  $N$  als Anzahl der Codes sieht der Algorithmus in kürze und verständlicher also so aus:

- Für alle Codes  $k=1..N$ :
- Für alle Codes  $i=1..N$ :  $\text{Matrix}(k, i) := \text{Code}(k) \text{ XOR } \text{Code}(i)$
- $\text{Maske}(k) := \text{AND}(i=1..N, i \neq k) \text{ Matrix}(k, i)$

**Verbesserung der Lesbarkeit**

Je akkurater die per AND Verknüpfung erzeugte Maske jeweils ist, desto besser ist die Lesbarkeit der einzelnen Klartexte. Diese Maske approximiert das jeweilige Original der Nachricht. Da sie nur aus den zwei Formen A und B besteht ist es eine gute Verbesserung diese auf den Originalen zu ergänzen. Von jedem gesetzten Pixel ausgehend wird Form A, beim nächsten Pixel Form B angewendet, viele Fehler somit ausgebessert.



## 3.2 Programm-Dokumentation

### Darstellung des Rasters (Klasse Raster)

In dieser Klasse wird das  $m \times n$  Raster verwaltet für jedes eingelesene Fax. Innerhalb der Klasse sind die statischen Variablen **M**, **N** und **MN** definiert, welche jeweils Breite, Höhe und das Produkt von beiden speichern. Das Raster wird in einem eindimensionalen Array **int \*map** verwaltet. Die 2d Koordinaten auf dem Raster werden mit der Funktion **int C2N(int, int)** umgewandelt:

```
inline int C2N(int i, int j) {  
    return i*M + j;  
}
```

### Operatoren in Raster

Alle für den Algorithmus notwendigen bitweisen Operatoren wurden überladen (AND, OR, XOR, ...). Auch der Standardausgabe Operator wurde redefiniert für die Ausgabe der Faxe. Dabei wird die pbm-Konvention beachtet alle 70 Zeilen einen Zeilenumbruch vorzunehmen (wie es bei allen Eingabedateien der Fall ist).

### Funktionen in Raster

#### Der Konstruktor Raster(istream& ifs)

Diese Funktion wird aufgerufen um ein Fax einzulesen. Die Eingabe wird hier also geparkt. Die Variablen **M**, **N**, **MN** werden gesetzt und die **map** wird gefüllt.

#### Der Konstruktor Raster()

Für ein alleinstehendes Objekt (temporäre Variablen etc..) im Programm.

#### Die Funktion void init()

Allokation der **map**.

#### Die Funktion void improve()

Die Funktion ist für die Aufbesserung der Originale zuständig. Form A und B wird hier jeweils von gesetzten Pixeln ausgehend ergänzt.

### Das Hauptprogramm (zara.cpp)

#### Die Funktion `int main(int argc, char *argv[])`

- Nimmt mit `argv[i]` den Namen der Eingabedateien entgegen
- Übergibt diese an die Funktion `initialisiere(char *datei)`
- Führt den in der Lösungsidee besprochenen Algorithmus aus

#### Die Funktion `bool initialisiere(char *datei)`

Hier wird für jeden eingelesenen Dateinamen der Konstruktor `Raster(ifstream& ifs)` aufgerufen, ein neues Raster Objekt erzeugt und im globalen `vector` als Fax gespeichert.

#### Die Templatefunktion `string toString(const T&)`

Dies ist nur eine Hilfsfunktion zum umwandeln der Nummer des Faxes in einen `std::string` für den Dateinamen der ausgegebenen Nachricht.

#### Der Hauptalgorithmus

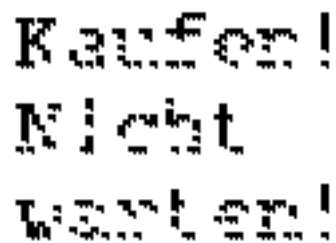
Dieser ist bis auf die Ausgabe identisch mit dem in der Lösungsidee besprochenen (nur halt in C++ Code)...

### 3.3 Programm-Ablaufprotokoll

#### Entschlüsselung der Faxe

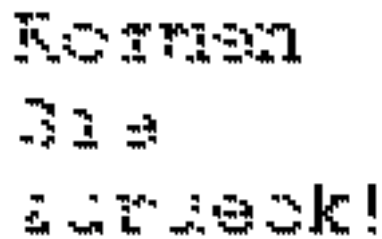
```
./zara crypt1_fax.pbm crypt2_fax.pbm crypt3_fax.pbm crypt4_fax.pbm  
      crypt5_fax.pbm crypt6_fax.pbm  
6 Faxe eingelesen...  
Entschlüssele Fax 1  
Entschlüssele Fax 2  
Entschlüssele Fax 3  
Entschlüssele Fax 4  
Entschlüssele Fax 5  
Entschlüssele Fax 6
```

#### Nachrichten



Kaufen!  
Nicht  
warten!

Abbildung 1: Nachricht 1 - Kaufen! Nicht warten!



Kommen  
Sie  
zurueck!

Abbildung 2: Nachricht 2 - Kommen Sie zurueck!

Viel  
Glueck!

Abbildung 3: Nachricht 3 - Viel Glueck!

Sie sind  
gefeuert

Abbildung 4: Nachricht 4 - Sie sind gefeuert

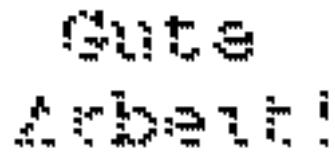
The text "Gute Arbeit!" is rendered in a pixelated, monospaced font. The letters are black on a white background, with a slightly irregular, hand-drawn appearance. The text is centered horizontally and vertically within the image area.

Abbildung 5: Nachricht 5 - Gute Arbeit!

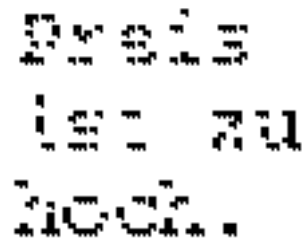
The text "Preis ist zu hoch." is rendered in a pixelated, monospaced font. The letters are black on a white background, with a slightly irregular, hand-drawn appearance. The text is centered horizontally and vertically within the image area.

Abbildung 6: Nachricht 6 - Preis ist zu hoch.

### 3.4 Programm-Text

#### Raster.h, Raster.cpp und zara.cpp

```
1 // BWINF 24.1
2 // Lösung Aufgabe 3
3 // Raster.h
4 // geschrieben 2005 von Karsten Bohlen
5
6 #ifndef RASTER_H
7 #define RASTER_H
8
9 #include <iostream>
10 #include <fstream>
11 using namespace std;
12
13 // Raster Datenstruktur
14 class Raster
15 {
16     public:
17         Raster(ifstream&);
18         Raster();
19         ~Raster();
20
21         void init(); // Speicher Allokation
22         void improve(); // Pixel anpassen
23
24         inline int C2N(int i, int j) {
25             return i*M + j;
26         }
27
28         Raster& operator=(const Raster&);
29         Raster& operator&=(const Raster&);
30         Raster& operator|=(const Raster&);
31         Raster& operator^=(const Raster&);
32
33         friend ostream& operator<<(ostream& t, const Raster& raster)
34         {
35             t << "P1\n";
36             t << M << " " << N << "\n";
37
38             for (int i = 0; i < N; i++) {
39                 for (int j = 0, k = 0; j < M; j++, k++)
40                 {
41                     if (k == 70) {
42                         t << "\n";
43                         k = 0;
44                     }
45
46                     int pos = i*M + j;
47
48                     t << raster.map[pos];
49                 }
50             }
51         }
52     };
53 }
```

```
49         t << "\n";
50     }
51
52     return t;
53 }
54
55 private:
56     int *map;
57     static int M, N; // N = Höhe, M = Breite
58     static int MN;
59 };
60
61 #endif
```

```
1 // BWINF 24.1
2 // Lösung Aufgabe 3
3 // Raster.cpp
4 // geschrieben 2005 von Karsten Bohlen
5
6 #ifndef RASTER_CPP
7 #define RASTER_CPP
8
9 #include "Raster.h"
10
11 int Raster::M, Raster::N;
12 int Raster::MN;
13
14 Raster::Raster(ifstream& ifs)
15 {
16     char tmp;
17
18     ifs >> tmp; ifs >> tmp; // P1
19     ifs >> M; ifs >> N; // Breite und Höhe
20
21     init(); // Speicher alloziieren
22
23     for (int i = 0; i < N; i++)
24         for (int j = 0; j < M; j++)
25         {
26             ifs >> tmp;
27
28             if (tmp == '1') map[C2N(i, j)] = 1;
29             else
30                 map[C2N(i, j)] = 0;
31         }
32 }
33
34 Raster::Raster() {
35     init();
36
37     for (int i = 0; i < MN; i++)
38         map[i] = 0;
39 }
40
41 Raster::~Raster() {
42     delete [] map;
43 }
44
45 void Raster::init() {
46     MN = M*N;
47     map = new int[MN];
48 }
49
50 void Raster::improve()
51 {
52     for (int i = 1; i < N; i+=2) // zwei Zeilen springen
53         for (int j = 1; j < M; j++)
54             {
```



```
55         if (map[C2N(i,j)])
56         {
57             map[C2N(i-1,j-1)] = 1; // Form A
58             map[C2N(i+1,j+1)] = 1; // ...
59             map[C2N(i-1,j)] = 1; // Form B
60             map[C2N(i,j-1)] = 1; // dito
61             ++j;
62         }
63     }
64 }
65
66 Raster& Raster::operator=(const Raster& raster) {
67     memcpy(map, raster.map, MN*sizeof(int));
68     return *this;
69 }
70
71 Raster& Raster::operator&=(const Raster& raster) {
72     for (int i = 0; i < MN; i++)
73         map[i] &= raster.map[i];
74     return *this;
75 }
76
77 Raster& Raster::operator|=(const Raster& raster) {
78     for (int i = 0; i < MN; i++)
79         map[i] |= raster.map[i];
80     return *this;
81 }
82
83 Raster& Raster::operator^=(const Raster& raster) {
84     for (int i = 0; i < MN; i++)
85         map[i] ^= raster.map[i];
86     return *this;
87 }
88
89 #endif
```

```
1 // BWNF 24.1
2 // Lösung Aufgabe 3
3 // zara.cpp
4 // geschrieben 2005 von Karsten Bohlen
5
6 // g++ zara.cpp Raster.cpp -o zara -pedantic -Wall -O3
7
8 #include <vector>
9 #include <string>
10 #include <sstream>
11 #include <cstdlib>
12 #include "Raster.h"
13
14
15 vector<Raster*> Fax; // alle Faxe
16
17 bool initialisiere(char *);
18
19 template <typename T>
20 string toString(const T&); // Nummerierung der Entschlüsselungen
21
22 int main(int argc, char *argv[])
23 {
24     if (argc == 1) {
25         cerr << "Dateinamen fehlen!" << endl;
26         return 1;
27     }
28
29     for (int i = 1; i <= argc-1; i++)
30         if (!initialisiere(argv[i]))
31             return 2;
32
33     cout << Fax.size() << " Faxe eingelesen..." << endl;
34
35     Raster *Matrix; // dynamische Xor-Matrix
36
37     for (unsigned k = 0; k < Fax.size(); k++) // Für jedes Fax
38     {
39         cout << "Entschlüssele Fax " << k+1 << endl;
40
41         Matrix = new Raster[Fax.size()];
42
43         for (unsigned i = 0; i < Fax.size(); i++)
44         {
45             Matrix[i] = *Fax[k]; // Matrix initialisieren
46             Matrix[i] ^= *Fax[i]; // speichern
47         }
48
49         // Xor-Matrizen auswerten:
50         Raster Maske;
51
52         if (k < Fax.size()-1) // wichtig: Maske != Matrix[k]
53             Maske = Matrix[k+1];
54         else
```

```
55     Maske = Matrix[k-1];
56
57     for (unsigned j = 0; j < Fax.size(); j++)
58         if (j != k) // Fax.size()-1 Matrizen (wie in Lösungsidee)
59             Maske &= Matrix[j];
60
61     Maske.improve(); // Lesbarkeit des "Originals" verbessern
62
63     string name = "Nachricht";
64     name += toString(k+1); name += ".pbm";
65     ofstream ofs(name.c_str());
66     ofs << Maske;
67     ofs.close();
68
69     delete [] Matrix; // Matrix löschen
70 }
71
72 for (vector<Raster*>::iterator i = Fax.begin(); i != Fax.end();
73      i++)
74     delete *i;
75 return 0;
76 }
77
78 bool initialisiere(char *datei)
79 {
80     ifstream ifs;
81
82     ifs.open(datei);
83
84     if (!ifs.is_open()) {
85         cerr << "Konnte Datei nicht öffnen!" << endl;
86         return false;
87     }
88
89     Raster *fax = new Raster(ifs);
90
91     Fax.push_back(fax);
92
93     ifs.close();
94
95     return true;
96 }
97
98 template <typename T>
99 string toString(const T& t) {
100     ostringstream s;
101     s << t;
102     return s.str();
103 }
```

## 5 Kleingeld

### 5.1 Lösungsidee

Eine Möglichkeit dieses Problem zu lösen ist es einfach alle Münzanzahlen durchzugehen. Wenn  $N$  die Anzahl der Münzen im Portemonnaie ist gilt stets  $N \leq 6$ , ein Durchprobieren wäre also vertretbar. Eine einfachere Lösung ist es aber einfach stets alle Münzen auf den Tisch zu legen. Egal wieviele Münzen man auf den Tisch legt (wobei es reichen muss um den Betrag zu decken/auszugleichen), die Summe der Werte der Münzen nach einer Zahlung ist gleich. Die Summe der Werte der Münzen im Portemonnaie ist also unabhängig davon wieviele der eigenen Münzen man auf den Tisch legt. Diese Summe verfügt über eine minimale Darstellung bezüglich der Anzahl der Cent-Stücke. Der Verkäufer ist ein Minimierer. Zahlt man mit allem was man hat wird die Summe der Werte der Münzen im Portemonnaie bezüglich der Anzahl der Stücke minimiert. Noch zu erwähnen ist vielleicht die Berechnung der Rückzahlungs-Münzen. Da der Verkäufer unendlich viele Münzen zur Verfügung hat wird hier immer die größte Münze, die noch in den Betrag passt genommen, wiederholt usw.

#### *Durchschnittlicher Münzbestand*

Eine gute Annäherung an den durchschnittlichen Münzbestand liefert der Durchschnitt der optimalen Zerlegung  $f(n)$  von  $n = 0$  bis  $n = 99$  Cent:

$$\begin{aligned}f(0) &= 0 \\f(1) &= 1 \\f(2) &= 1 \\f(3) &= 2 \\&\dots \\f(50) &= 1 \\&\dots \\f(99) &= 6\end{aligned}$$

Man kommt auf genau 3.4 als Durchschnitt der Funktionswerte. Lässt man eine bestimmte Münze (außer 1 Cent natürlich) bei der Berechnung weg steigert sich der durchschnittliche Bestand.

## 5.2 Programm-Dokumenation

Die Funktion `int main(int argc, char *argv[])`

- Nimmt mit `argv[1]` die Anzahl der Einkäufe entgegen
- Setzt den Wert und Anfangs-Anzahl der Münzen
- Simuliert **E** Einkäufe

Die Funktion `int f(int n)`

Hier wird das Rückgeld für einen gegebenen Betrag **n** berechnet. Und zwar immer die nächstgrößte Münze abgezogen und wiederholt. Rückgabewert ist die Anzahl der Münzen.

Die Struktur `struct Cent`

Hier wird eine Cent Münze verwaltet. `int a` determiniert die Anzahl dieser Münze, `int r` die Rückgeld-Anzahl dieser Münze. `int Wert` ist der Cent-Wert der Münze und `int Stats` hält die Verwendungsanzahl dieser Münze fest.

Globale Variablen, Arrays

Alle Münzen werden in dem Array `Cent Muenze[6]` verwaltet. Außerdem sind da noch `int N` die Gesamt-Münzanzahl und `int Betrag` der aktuelle Rechnungsbetrag.

### 5.3 Programm-Ablaufprotokoll

#### Format der Ausgabe

Für jeden Einkauf wird eine Zeile verwendet, die das folgende Format hat:

```
<Nr.> <Anzahl der vorhandenen Münzen> <Zahlbetrag (zufällig)> <
gezahlte Cent-Münzen> <erhaltene Cent-Münzen>
```

Es werden immer nur die ersten 100 Einkäufe hier abgedruckt. Der Anfangsbestand an Münzen ist wie gefordert 0.

#### Simulation: 1000 Einkäufe

```
1: 0 Münzen. Betrag: 513 Cent. Zahle: Erhalte: 1 mal 2 Cent + 1
    mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +
-----
```

```
2: 5 Münzen. Betrag: 292 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent
    + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1
    mal 5 Cent + 2 mal 20 Cent + 1 mal 50 Cent +
-----
```

```
3: 4 Münzen. Betrag: 995 Cent. Zahle: 1 mal 5 Cent + 2 mal 20 Cent
    + 1 mal 50 Cent + Erhalte:
-----
```

```
4: 0 Münzen. Betrag: 608 Cent. Zahle: Erhalte: 1 mal 2 Cent + 2
    mal 20 Cent + 1 mal 50 Cent +
-----
```

```
5: 4 Münzen. Betrag: 8 Cent. Zahle: 1 mal 2 Cent + 2 mal 20 Cent +
    1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1 mal 10 Cent + 1 mal
    20 Cent + 1 mal 50 Cent +
-----
```

```
6: 5 Münzen. Betrag: 859 Cent. Zahle: 2 mal 2 Cent + 1 mal 10 Cent
    + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 5 Cent + 1
    mal 20 Cent +
-----
```

```
7: 2 Münzen. Betrag: 597 Cent. Zahle: 1 mal 5 Cent + 1 mal 20 Cent
    + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal
    20 Cent +
-----
```

```
8: 4 Münzen. Betrag: 944 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent
    + 1 mal 5 Cent + 1 mal 20 Cent + Erhalte: 2 mal 2 Cent + 1 mal
    10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +
-----
```

9: 5 Münzen. Betrag: 265 Cent. Zahle: 2 mal 2 Cent + 1 mal 10 Cent  
+ 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1  
mal 5 Cent + 1 mal 10 Cent +

-----

10: 4 Münzen. Betrag: 465 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + Erhalte: 2 mal 2 Cent + 1 mal 50 Cent +

-----

11: 3 Münzen. Betrag: 307 Cent. Zahle: 2 mal 2 Cent + 1 mal 50  
Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent +

-----

12: 4 Münzen. Betrag: 296 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1 mal 50 Cent +

-----

13: 2 Münzen. Betrag: 102 Cent. Zahle: 1 mal 1 Cent + 1 mal 50  
Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent +

-----

14: 5 Münzen. Betrag: 886 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1  
mal 10 Cent + 1 mal 50 Cent +

-----

15: 4 Münzen. Betrag: 655 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 10 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 2 Cent + 1 mal 5 Cent +

-----

16: 3 Münzen. Betrag: 971 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 1 mal  
10 Cent + 1 mal 20 Cent +

-----

17: 4 Münzen. Betrag: 351 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

18: 5 Münzen. Betrag: 762 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2  
mal 2 Cent + 1 mal 20 Cent +

-----

19: 3 Münzen. Betrag: 86 Cent. Zahle: 2 mal 2 Cent + 1 mal 20 Cent  
+ Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal  
10 Cent + 1 mal 20 Cent +

-----

20: 5 Münzen. Betrag: 255 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 2 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1

mal 50 Cent +

-----

21: 5 Münzen. Betrag: 109 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2  
mal 2 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

22: 4 Münzen. Betrag: 810 Cent. Zahle: 2 mal 2 Cent + 1 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1 mal 10 Cent +  
1 mal 50 Cent +

-----

23: 4 Münzen. Betrag: 27 Cent. Zahle: 2 mal 2 Cent + 1 mal 10 Cent  
+ 1 mal 50 Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 1  
mal 10 Cent + 1 mal 20 Cent +

-----

24: 4 Münzen. Betrag: 607 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 10 Cent + 1  
mal 20 Cent +

-----

25: 2 Münzen. Betrag: 954 Cent. Zahle: 1 mal 10 Cent + 1 mal 20  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 5 Cent + 1 mal 20 Cent +  
1 mal 50 Cent +

-----

26: 4 Münzen. Betrag: 66 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 10 Cent +

-----

27: 1 Münzen. Betrag: 605 Cent. Zahle: 1 mal 10 Cent + Erhalte: 1  
mal 5 Cent +

-----

28: 1 Münzen. Betrag: 891 Cent. Zahle: 1 mal 5 Cent + Erhalte: 2  
mal 2 Cent + 1 mal 10 Cent +

-----

29: 3 Münzen. Betrag: 367 Cent. Zahle: 2 mal 2 Cent + 1 mal 10  
Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent +

-----

30: 4 Münzen. Betrag: 17 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + Erhalte: 1 mal 10 Cent + 1 mal 20 Cent +

-----

31: 2 Münzen. Betrag: 688 Cent. Zahle: 1 mal 10 Cent + 1 mal 20  
Cent + Erhalte: 1 mal 2 Cent + 2 mal 20 Cent +

-----

32: 3 Münzen. Betrag: 231 Cent. Zahle: 1 mal 2 Cent + 2 mal 20  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 10 Cent +



-----  
33: 2 Münzen. Betrag: 308 Cent. Zahle: 1 mal 1 Cent + 1 mal 10  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent +  
-----

34: 2 Münzen. Betrag: 683 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ Erhalte: 1 mal 20 Cent +  
-----

35: 1 Münzen. Betrag: 839 Cent. Zahle: 1 mal 20 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +  
-----

36: 4 Münzen. Betrag: 667 Cent. Zahle: 1 mal 1 Cent + 1 mal 10  
Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent +  
1 mal 10 Cent +  
-----

37: 3 Münzen. Betrag: 541 Cent. Zahle: 2 mal 2 Cent + 1 mal 10  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 20 Cent +  
1 mal 50 Cent +  
-----

38: 4 Münzen. Betrag: 435 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent +  
-----

39: 5 Münzen. Betrag: 610 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 20 Cent +  
-----

40: 4 Münzen. Betrag: 158 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 20 Cent + Erhalte: 1 mal 20 Cent + 1  
mal 50 Cent +  
-----

41: 2 Münzen. Betrag: 251 Cent. Zahle: 1 mal 20 Cent + 1 mal 50  
Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent +  
-----

42: 4 Münzen. Betrag: 917 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + Erhalte: 1 mal 2 Cent +  
-----

43: 1 Münzen. Betrag: 805 Cent. Zahle: 1 mal 2 Cent + Erhalte: 1  
mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent + 1 mal 50 Cent +  
-----

44: 5 Münzen. Betrag: 705 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 2 Cent + 2  
mal 20 Cent + 1 mal 50 Cent +

-----  
45: 4 Münzen. Betrag: 802 Cent. Zahle: 1 mal 2 Cent + 2 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 2 mal 20 Cent + 1 mal 50 Cent  
+  
-----

46: 3 Münzen. Betrag: 811 Cent. Zahle: 2 mal 20 Cent + 1 mal 50  
Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent + 1 mal 20 Cent +  
1 mal 50 Cent +  
-----

47: 5 Münzen. Betrag: 27 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 2 Cent + 1  
mal 50 Cent +  
-----

48: 2 Münzen. Betrag: 505 Cent. Zahle: 1 mal 2 Cent + 1 mal 50  
Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent +  
-----

49: 4 Münzen. Betrag: 572 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + Erhalte: 1 mal 5 Cent + 1 mal 20 Cent + 1  
mal 50 Cent +  
-----

50: 3 Münzen. Betrag: 113 Cent. Zahle: 1 mal 5 Cent + 1 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 1 mal 2 Cent + 1 mal 10 Cent +  
1 mal 50 Cent +  
-----

51: 3 Münzen. Betrag: 111 Cent. Zahle: 1 mal 2 Cent + 1 mal 10  
Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1 mal 50 Cent +  
-----

52: 2 Münzen. Betrag: 33 Cent. Zahle: 1 mal 1 Cent + 1 mal 50 Cent  
+ Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal  
10 Cent +  
-----

53: 4 Münzen. Betrag: 274 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 10 Cent + Erhalte: 2 mal 2 Cent + 2  
mal 20 Cent +  
-----

54: 4 Münzen. Betrag: 489 Cent. Zahle: 2 mal 2 Cent + 2 mal 20  
Cent + Erhalte: 1 mal 5 Cent + 1 mal 50 Cent +  
-----

55: 2 Münzen. Betrag: 639 Cent. Zahle: 1 mal 5 Cent + 1 mal 50  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 5 Cent + 1 mal 10 Cent +  
-----

56: 3 Münzen. Betrag: 227 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent + 1  
mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

57: 6 Münzen. Betrag: 555 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2  
mal 2 Cent + 1 mal 10 Cent + 1 mal 20 Cent +

-----

58: 4 Münzen. Betrag: 596 Cent. Zahle: 2 mal 2 Cent + 1 mal 10  
Cent + 1 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent +  
1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent +

-----

59: 5 Münzen. Betrag: 469 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 2  
mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 50 Cent +

-----

60: 5 Münzen. Betrag: 921 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent +

-----

61: 5 Münzen. Betrag: 612 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 2 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent +

-----

62: 4 Münzen. Betrag: 157 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 2 mal 2 Cent + 1  
mal 5 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

63: 5 Münzen. Betrag: 504 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 5 Cent + 1  
mal 20 Cent + 1 mal 50 Cent +

-----

64: 3 Münzen. Betrag: 271 Cent. Zahle: 1 mal 5 Cent + 1 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent +

-----

65: 2 Münzen. Betrag: 191 Cent. Zahle: 2 mal 2 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 2 Cent + 1 mal 10 Cent +

-----

66: 3 Münzen. Betrag: 342 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 10 Cent + Erhalte: 1 mal 1 Cent + 1 mal 20 Cent + 1  
mal 50 Cent +

-----

67: 3 Münzen. Betrag: 937 Cent. Zahle: 1 mal 1 Cent + 1 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1 mal 10 Cent +  
1 mal 20 Cent +

-----

68: 4 Münzen. Betrag: 83 Cent. Zahle: 2 mal 2 Cent + 1 mal 10 Cent  
+ 1 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1 mal 50 Cent +

-----

69: 2 Münzen. Betrag: 128 Cent. Zahle: 1 mal 1 Cent + 1 mal 50  
Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 20 Cent +

-----

70: 3 Münzen. Betrag: 899 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 20 Cent + Erhalte: 2 mal 2 Cent + 1 mal 20 Cent +

-----

71: 3 Münzen. Betrag: 240 Cent. Zahle: 2 mal 2 Cent + 1 mal 20  
Cent + Erhalte: 2 mal 2 Cent + 1 mal 10 Cent + 1 mal 20 Cent +  
1 mal 50 Cent +

-----

72: 5 Münzen. Betrag: 379 Cent. Zahle: 2 mal 2 Cent + 1 mal 10  
Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 5 Cent +

-----

73: 1 Münzen. Betrag: 167 Cent. Zahle: 1 mal 5 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1  
mal 20 Cent +

-----

74: 5 Münzen. Betrag: 396 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1  
mal 2 Cent + 2 mal 20 Cent +

-----

75: 3 Münzen. Betrag: 435 Cent. Zahle: 1 mal 2 Cent + 2 mal 20  
Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent +

-----

76: 2 Münzen. Betrag: 320 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1  
mal 20 Cent + 1 mal 50 Cent +

-----

77: 5 Münzen. Betrag: 559 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 20 Cent +

-----

78: 4 Münzen. Betrag: 461 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent  
+ 1 mal 5 Cent + 1 mal 20 Cent + Erhalte: 1 mal 2 Cent + 1  
mal 5 Cent + 1 mal 10 Cent + 1 mal 50 Cent +

-----  
79: 4 Münzen. Betrag: 176 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 2  
mal 20 Cent + 1 mal 50 Cent +  
-----

80: 4 Münzen. Betrag: 130 Cent. Zahle: 1 mal 1 Cent + 2 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1 mal 10 Cent +  
1 mal 50 Cent +  
-----

81: 3 Münzen. Betrag: 925 Cent. Zahle: 1 mal 1 Cent + 1 mal 10  
Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1 mal 5 Cent +  
1 mal 10 Cent + 1 mal 20 Cent +  
-----

82: 4 Münzen. Betrag: 286 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 50 Cent +  
-----

83: 1 Münzen. Betrag: 514 Cent. Zahle: 1 mal 50 Cent + Erhalte: 1  
mal 1 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent +  
-----

84: 4 Münzen. Betrag: 199 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 2 Cent + 1  
mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent +  
-----

85: 4 Münzen. Betrag: 127 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 10 Cent +  
-----

86: 1 Münzen. Betrag: 153 Cent. Zahle: 1 mal 10 Cent + Erhalte: 1  
mal 2 Cent + 1 mal 5 Cent + 1 mal 50 Cent +  
-----

87: 3 Münzen. Betrag: 777 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent  
+ 1 mal 50 Cent + Erhalte: 1 mal 10 Cent + 1 mal 20 Cent + 1  
mal 50 Cent +  
-----

88: 3 Münzen. Betrag: 681 Cent. Zahle: 1 mal 10 Cent + 1 mal 20  
Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent +  
2 mal 20 Cent + 1 mal 50 Cent +  
-----

89: 6 Münzen. Betrag: 748 Cent. Zahle: 2 mal 2 Cent + 1 mal 5 Cent  
+ 2 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1  
mal 50 Cent +  
-----

90: 2 Münzen. Betrag: 246 Cent. Zahle: 1 mal 1 Cent + 1 mal 50 Cent + Erhalte: 1 mal 5 Cent +

-----

91: 1 Münzen. Betrag: 953 Cent. Zahle: 1 mal 5 Cent + Erhalte: 1 mal 2 Cent + 1 mal 50 Cent +

-----

92: 2 Münzen. Betrag: 711 Cent. Zahle: 1 mal 2 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 2 mal 20 Cent +

-----

93: 3 Münzen. Betrag: 754 Cent. Zahle: 1 mal 1 Cent + 2 mal 20 Cent + Erhalte: 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

94: 5 Münzen. Betrag: 456 Cent. Zahle: 1 mal 2 Cent + 1 mal 5 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1 mal 10 Cent + 1 mal 20 Cent +

-----

95: 3 Münzen. Betrag: 333 Cent. Zahle: 1 mal 1 Cent + 1 mal 10 Cent + 1 mal 20 Cent + Erhalte: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent + 1 mal 50 Cent +

-----

96: 6 Münzen. Betrag: 944 Cent. Zahle: 1 mal 1 Cent + 1 mal 2 Cent + 1 mal 5 Cent + 2 mal 20 Cent + 1 mal 50 Cent + Erhalte: 2 mal 2 Cent + 1 mal 50 Cent +

-----

97: 3 Münzen. Betrag: 149 Cent. Zahle: 2 mal 2 Cent + 1 mal 50 Cent + Erhalte: 1 mal 5 Cent +

-----

98: 1 Münzen. Betrag: 621 Cent. Zahle: 1 mal 5 Cent + Erhalte: 2 mal 2 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent +

-----

99: 5 Münzen. Betrag: 378 Cent. Zahle: 2 mal 2 Cent + 1 mal 10 Cent + 1 mal 20 Cent + 1 mal 50 Cent + Erhalte: 1 mal 1 Cent + 1 mal 5 Cent +

-----

100: 2 Münzen. Betrag: 277 Cent. Zahle: 1 mal 1 Cent + 1 mal 5 Cent + Erhalte: 2 mal 2 Cent + 1 mal 5 Cent + 1 mal 20 Cent +

....

Im Durchschnitt waren 3.342 Münzen im Portemonnaie

Statistik:

1 Cent-Stück wurde 839 mal verwendet.

2 Cent-Stück wurde 1540 mal verwendet.

5 Cent-Stück wurde 976 mal verwendet.

10 Cent-Stück wurde 795 mal verwendet.  
20 Cent-Stück wurde 1566 mal verwendet.  
50 Cent-Stück wurde 965 mal verwendet.

Bei dieser Simulation wurde das 10-Cent Stück am seltensten verwendet. Während man das 1-Cent Stück sicherlich nicht weglassen kann sind alle 6 Münzen notwendig um ein zu volles Portemonnaie zu verhindern. Der durchschnittliche Bestand steigt sonst (siehe auch Lösungsidee).

## 5.4 Programm-Text

```
1 // BWNF 24.1
2 // Lösung Aufgabe 5
3 // kleingeld v0.1
4 // geschrieben 2005 von Karsten Bohlen
5 #include <iostream>
6 #include <cstdlib>
7 #include <ctime>
8 using namespace std;
9
10 struct Cent {
11     int a,          // Anzahl
12     r,             // Rückgeld-Anzahl
13     Wert,          // Wert der Münze
14     Stats;         // Statistik (Münz-Nutzung)
15
16     Cent() { a = 0; }
17 };
18
19 Cent Muenze[6];    // 6 Cent Münzen
20 int N,              // Gesamt Münzanzahl
21 Betrag;            // Rechnungsbetrag
22
23 int f(int n);       // Errechne Rückgeld und vergleiche
24
25 int main(int argc, char *argv[])
26 {
27     if (argc != 2) return 1;
28
29     int E = atoi(argv[1]);    // Anzahl der Einkäufe
30     int Sum = 0;              // Münzsumme nach E Durchläufen
31
32     // Münzen definieren
33     Muenze[0].Wert = 1; Muenze[1].Wert = 2; Muenze[2].Wert = 5;
34     Muenze[3].Wert = 10; Muenze[4].Wert = 20; Muenze[5].Wert = 50;
35
36     srand(time(NULL));    // Zufallszahlen
37
38     for (int e = 0; e < E; e++)
39     {
40         Betrag = rand()%1000+1; // zufälliger Rechnungsbetrag
41
42         cout << e+1 << ": " << N << " Münzen. Betrag: " << Betrag <<
43             " Cent. ";
44
45         // Lösung ausgeben
46         // Werte aktualisieren
47
48         cout << "Zahle: ";
49
50         int B = 0; // Zahlbetrag
51
52         for (int i = 0; i < 6; i++)
```



```

52     {
53         if (Muenze[i].a != 0) {
54             cout << Muenze[i].a << " mal "
55                 << Muenze[i].Wert << " Cent + ";
56         }
57
58         B += (Muenze[i].a)*(Muenze[i].Wert);
59
60         Muenze[i].Stats += Muenze[i].a;
61     }
62
63     int R = f(B);  // bezahlen
64
65     cout << "Erhalte: ";
66
67     for (int i = 0; i < 6; i++)
68     {
69         if (Muenze[i].r != 0) {
70             cout << Muenze[i].r << " mal "
71                 << Muenze[i].Wert << " Cent + ";
72         }
73
74         // Werte aktualisieren
75         Muenze[i].a = Muenze[i].r;
76         Muenze[i].Stats += Muenze[i].r;
77         Muenze[i].r = 0;
78     }
79
80     cout << endl;
81     cout << "
82         -----
83         " << endl;
84
85     N = R;  // neue optimale Münzanzahl
86     Sum += N;
87
88     cout << "Im Durchschnitt waren " << double(Sum)/double(E) << "
89         Münzen im Portemonnaie " << endl;
90
91     cout << "\nStatistik: " << endl;
92
93     for (int i = 0; i < 6; i++)
94     {
95         cout << Muenze[i].Wert << " Cent-Stück wurde " << Muenze[i].
96             Stats << " mal verwendet."
97             << endl;
98     }
99
100    return 0;
101 }
102
103 int f(int n)
104 {

```

```
102     while (n - Betrag < 0) {
103         n += 100;
104     }
105
106     int Re = n - Betrag, R = 0;
107
108     while (Re - 100 >= 0) {
109         Re -= 100;
110     }
111     if (Re - 50 >= 0) {
112         Re -= 50;
113         ++Muenze[5].r;
114         ++R;
115     }
116     while (Re - 20 >= 0) {
117         Re -= 20;
118         ++Muenze[4].r;
119         ++R;
120     }
121     if (Re - 10 >= 0) {
122         Re -= 10;
123         ++Muenze[3].r;
124         ++R;
125     }
126     if (Re - 5 >= 0) {
127         Re -= 5;
128         ++Muenze[2].r;
129         ++R;
130     }
131     while (Re - 2 >= 0) {
132         Re -= 2;
133         ++Muenze[1].r;
134         ++R;
135     }
136     if (Re - 1 == 0) {
137         Re -= 1;
138         ++Muenze[0].r;
139         ++R;
140     }
141
142     return R;
143 }
```