35. Bundeswettbewerb Informatik

Runde 1

01.09.2016 - 28.11.2016

Aufgabe 1

Sprichwort

23. November 2016

Eingereicht von: *Wouldn't IT be nice...* (Team-ID: 00007)

Tim Hollmann (6753)
ich@tim-hollmann.de

Anike Heikrodt (6841)
anikeheikrodt@online.de

Wir versichern hiermit, die vorliegende Arbeit ohne unerlaubte fremde Hilfe entsprechend der Wettbewerbsregeln des Bundeswettbewerb Informatik angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Tim Hollmann & *Anike Heikrodt*, den 23. November 2016

1 Die Kalendersysteme

Der julianische Kalender ging aus einer Reform des ehemaligen römischen Kalenders im Jahr 45 v. Chr. durch Gaius Julius Cäsars hervor. Der gregorianische Kalender entstand durch eine Reform des julianischen Kalenders im Jahr 1582 durch Papst Gregor.

1.1 Römischer Kalender

Der römische Kalender war ein zwölfmonatiger, an Mondzyklen orientier Kalender und besaß eine reguläre Jahreslänge von 355 Tagen. Die daraus resultierende Verschiebung gegenüber dem tatsächlichen Jahreszyklus (*tropischem Jahr*) von $365,24219$ Tagen wurde durch je nach Bedarf in unregelmäßigen Abständen eingeschobenen Schalttagen und sogar regelmäßigen Schaltmonaten korrigiert.
(Frei nach [1])

1.2 Julianischer Kalender

Durch die julianische Kalenderreform entfiel der Schaltmonat, eine regelmäßige Schaltjahresregel wurde eingeführt (jedes vierte Jahr einen Schalttag am Ende des Februars[1]) sowie die Längen der Monate angepasst. Die durchschnittliche Jahreslänge betrug nun $365 + \frac{1}{4}$ Tage, was der Länge des tropischen Jahres schon recht nahe kommt. Dieser Unterschied von $\Delta t = (\frac{1}{4} - 0,24219) = 0,00781$ Tagen (≈ 11 Minuten) pro Jahr verursacht nach $\frac{1}{\Delta t} \approx 128$ Jahren einen Unterschied von 1 Tag. Der julianische Kalender benötigt länger für einen Jahreszyklus, sodass er natürlichen Phänomenen wie z.B. dem Sommeranfang oder bestimmten Mondphasen (u.A. der für Ostern wichtige Frühlingsvollmond) nachhinkt und sich diese aus seiner Sicht immer früher ereignen. Im Jahre 1582 (also 1627 Jahre nach der julianischen Kalenderreform) hatte sich dieser Unterschied bereits auf $\frac{1627}{\Delta t} \approx 10$ Tage aufsummiert. Eine Kalenderreform war überfällig.
Der julianische Kalender findet in heutiger Zeit meist lediglich im kirchlichen Bereich Anwendung.
(Frei nach [2] und [3, *Mängel im julianischen Kalender*])

1.3 Gregorianischer Kalender

Im Jahr 1582 verfügte Papst Gregor XIII Kraft seines Amtes einen neuen Kalender: den gregorianischen Kalender. Dieser erweiterte den julianischen Kalender um eine weitere Schaltregel: Alle Jahre, die ganzzahlig durch 100 teilbar sind, sind keine Schaltjahre (obwohl sie durch 4 teilbar sind). Ausnahme hierzu sind die Jahre, die durch 400 teilbar sind; diese sind wieder Schaltjahre. Weiterhin verfügte er, um die Verspätung des julianischen Kalenders gegenüber dem tropischen Jahr wieder auszugleichen, dass auf den 4.Oktober 1582_{jul} der 15.Oktober 1582_{greg} folgte. Da nach diesen Schaltregeln auf 400 Jahre 97 Schalttage kommen, ergibt sich eine durchschnittliche Jahreslänge von

[1] Diese Regel wurde zuerst nach Art eines Zaunpfahlfehlers missverstanden und erst nach ca. 40 Jahren durch Kaiser Augustus korrigiert.

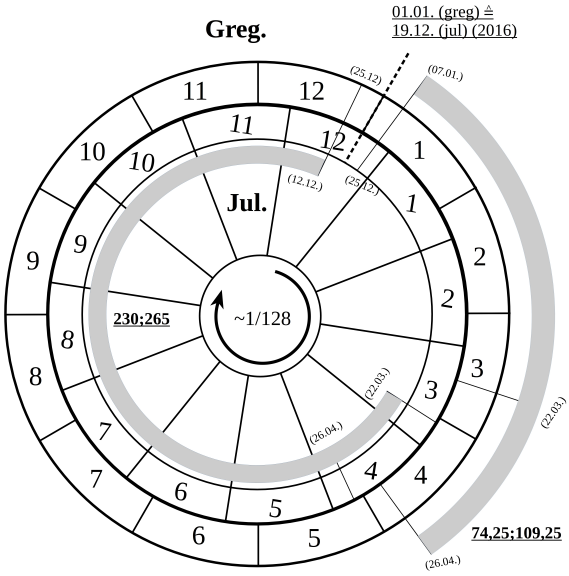

Abbildung 1: Der julianische Kalender (innen) verschiebt sich durchschnittlich alle 128 Jahre gegenüber dem gregorianischen Kalender (außen) um einen Tag (im Uhrzeigersinn); eine Verschiebung von 13 Tagen liegt bereits vor. Von $\text{Weihnachten}_{\text{jul}}$ bis $\text{Ostern}_{\text{greg}}$ ist noch eine Verschiebung von 74,25 bis 109,25 Tagen notwendig; von $\text{Ostern}_{\text{jul}}$ bis $\text{Weihnachten}_{\text{greg}}$ 230 bis 265 Tage.

$365 + (\frac{97}{400}) = 365,2425$, was dem tropischen Jahr sehr nahe kommt ($\Delta t = 11$ Sekunden pro Jahr); eine Verschiebung um einen Tag gegenüber dem tropischen Jahr findet erst nach ca. 8000 Jahren statt.
Dieser Kalender wird heute weltweit verwendet.
(Mit Informationen aus [3] und [4, *Wie unterscheiden sich julianischer und gregorianischer Kalender?*])

2 Christians Hoffnung

Durch die unterschiedlichen durchschnittlichen Jahreslängen von 365,25 und 365,242 Tagen (siehe **1.2** und **1.3**) verschieben sich der julianische und gregorianische Kalender gegeneinander; der julianische Kalender benötigt länger für einen Jahreszyklus und fällt daher im Schnitt alle 128 Jahre gegenüber dem gregorianischen Kalender um einen Tag zurück[2].

[2] 128 Jahre, da der gregorianische Kalender als hinreichend genaue Approximation des tropischen Jahres angenommen wird und sich das julianische Jahr diesem gegenüber derartig verschiebt, siehe **1.2**.

Daher ist es auch möglich, dass die beiden Kalender soweit auseinander driften, dass Ostern und Weihnachten auf den selben Tag fallen können, was Christians Hoffnungen berechtigt. Damit $\text{Weihnachten}_{\text{jul}}$ und $\text{Ostern}_{\text{greg}}$ aufeinander fallen, ist eine Verschiebung von $74,25$ bis maximal $109,25$ Tagen notwendig.[3](vgl. Abbildung **1**) Rechnet man nun mit einer durchschnittlichen Verschiebung von einem Tag pro 128 Jahre, kann mit diesem Ereignis zwischen $128 * 74,25 + 2000 = 11.504$ und 15.984 gerechnet werden.[4]
Für den umgekehrten Fall, dass $\text{Ostern}_{\text{jul}}$ auf $\text{Weihnachten}_{\text{greg}}$ fällt, sind 230 bis 265 Tage Verschiebung notwendig (vgl. Abbildung **1**). Diese werden zwischen den Jahren $230 * 128 + 2000 = 31.440$ und 35.920 erreicht.
Es ist zu erwarten, dass sich das tatsächliche Ergebnis *relativ* nahe an der jeweils früheren Jahreszahl befindet, da zwischen jeder Verschiebung um einen Tag jeweils 128 Jahre und damit 128 gregorianische/julianische Osterfeste liegen, von denen mit zunehmender Verschiebung immer unwahrscheinlicher wird, dass nicht mindestens eines auf den jeweiligen Tag fallen wird.
Zusammenfassend sind Christians Hoffnungen zwar begründet und berechtigt, aber im Kontext nicht realistisch, da er diesen Tag schlicht nicht erleben wird.

3 Lösungsidee

Die erste Teilaufgabe wurde dadurch gelöst, dass solange das Datum des nächsten Osterns im gregorianischen Kalender simuliert wird, bis dieses Datum in den julianischen Kalender übertragen auf Weihnachten fällt.
Der Algorithmus dazu sieht folgendermaßen aus: (Algorithmus **1**).

Algorithmus 1 : Algorithmus zu Teilaufgabe 1

```
1 for Jahr_greg ← 2016 bis ∞ do
2 |  Osterdatum_greg ← OsterdatumInJahr(Jahr_greg);
3 |  wenn greg2jul(Osterdatum_greg).Tag == 25 und
  |   greg2jul(Osterdatum_greg).Monat == 12 dann
4 |  |  Erfolg
```

Der Algorithmus zur zweiten Teilaufgabe ($\text{Ostern}_{\text{jul}}$ fällt auf $\text{Weihnachten}_{\text{greg}}$) ist analog; man ermittelt solange das Osterdatum im julianischen Kalender, bis es in den gregorianischen Kalender übertragen auf Weihnachten fällt. (Algorithmus **2**)
(Dabei ist anzumerken, dass man theoretisch nicht unendlich lange in die Zukunft gehen bräuchte; bis zum Jahr $9 * 10^8$ wird sich die Sonne kontinuierlich zu einem roten Riesen entwickeln und jegliches höhere Leben auf der Erde zerstören. Daher ist der Suchraum (glücklicherweise??) endlich.)

[3]Ostern findet laut klerikaler Definition am ersten Sonntag nach dem Frühlingsvollmond (Paschafest) statt, der frühestens am 21. März stattfinden kann, sodass Ostersonntag frühestens am 22.März und spätestens am 26.April liegt. [5, Einleitung]
[4]Es wird +2000 gerechnet, da die aktuelle Verschiebung von 13 Tagen seit dem Jahr 2000 besteht (bis zum Jahr 2099).

Algorithmus 2 : Algorithmus zu Teilaufgabe 2

```
1 for Jahr_jul ← 2016 bis ∞ do
2     Osterdatum_jul ← OsterdatumInJahr(Jahr_jul);
3     wenn jul2greg(Osterdatum_jul).Tag == 25 und
        jul2greg(Osterdatum_jul).Monat == 12 dann
4         Erfolg
```

Zur Realisierung der zwei Algorithmen ist notwendig:

1. Das Ermitteln des Osterdatums zu einem beliebigen gregorianischen oder julianischen Jahr.

2. Das Umrechnen zwischen gregorianischem und julianischem Kalender in beide Richtungen.

3.1 Gauß'sche Osterformel

Mit Hilfe der nach ihrem Erfinder Carl Friedrich Gauß bennanten *Gauß'schen Osterformel* (ursprünglich aufgestellt im Jahre 1800 und später mehrfach überarbeitet) kann der Monat und der Tag eines jeden Ostersonntags zu einem beliebigen Jahr im wahlweise gregorianischen oder julianischen Datum sehr komfortabel ausgerechnet werden. Die Osterformel ist als Satz von Gleichungen notiert, die nacheinander zu berechnen sind; im Grunde ein Algorithmus.
Im Folgenden wird die im 19. Jahrhundert von Hermann Kinkelin aufgestellte ergänzte Osterformel betrachtet, die zusätzlich einige spezielle Ausnahmen der Osterberechnung berücksichtigt (siehe [6, *Eine ergänzte Osterformel*]).

Algorithmus 3 : Ergänzte Osterformel nach Kinkelin

Eingabe : Jahr (gregorianisch oder julianisch)
Ausgabe : Datum des Ostersonntags als Märzdatum (OS)[a]

```
1  K ← ⌊Jahr/100⌋;
2  wenn Jahr gregorianisch dann
3  |  M ← ⌊15 + (3 ∗ K + 3)/4 − (8 ∗ K + 13)/25⌋;
4  |  S ← ⌊2 − (3 ∗ K + 3)/4⌋;
5  sonst
6  |  M ← 15;
7  |  S ← 0;
8  A ← Jahr  mod 19;
9  D ← (19 ∗ A + M)  mod 30;
10 R ← ⌊(D + A/11)/29⌋;
11 OG ← 21 + D − R;
12 SZ ← 7 − (Jahr + Jahr/4 + S)  mod 7;
13 OE ← 7 − (OG − SZ)  mod 7;
14 OS ← OG + OE;
15 return OS;
```

[a] Der Algorithmus gibt den Ostersonntag als Märzdatum zurück; z.B. wäre der 40.März der 9.April.

Die genaue Bedeutung der einzelnen variablen Größen (K-Säkularzahl, M-säkuläre Mondschaltung usw.) können Sie in [6, *Eine ergänzte Osterformel*] nachlesen.
Die Implementierung dieses Algorithmus ist nunmehr trivial.

3.2 Umrechnung zwischen den Kalendersystemen

Ein Algorithmus, der direkt die Umrechnung des julianischen zum gregorianischen Kalender (oder umgekehrt) vornimmt, ist uns nicht bekannt. Die Umrechnung zwischen den Kalendersystemen ist allerdings möglich, wenn man deren Tagesdifferenz zu einem beliebigen gregorianischen und julianischen Datum ermitteln kann und das Datum entsprechend korrigiert;
Der folgende Algorithmus ist angelehnt an [7, *Mathematische Lösung*]. Berechnung der Tagesdifferenz nach [7, *Berechnung der Tagesdifferenz*]. (Algorithmus **4**)
Um von einem beliebigen julianischen oder gregorianischen Datum auf das korrespondierende Datum im anderen System umrechnen zu können, ermittelt man zunächst die Tagesdifferenz zwischen den Systemen zu diesem Zeitpunkt. (Funktion `Tegesdifferenz`, Zeile 1). Man in-/dekrementiert das Datum nun um die Tagesdifferenz. (vgl. Zeilen 10+11 / 16+17). Geht man von einem julianischen Datum aus, muss die Tagesdifferenz auf das aktuelle Datum addiert werden (der julianische Kalender „hinkt“ dem gregorianischen nach). Umgekehrt entsprechend andersherum. Es muss allerdings beachtet werden, dass man beim „In-Die-Vergangenheit/Zukunft-Gehen“ die Schaltregeln des Zielsystems beachtet (hierzu siehe Zeilen 9 und 15; durch das Verändern des Kalendertyps (sprich: der

Membervariable `isGregorian`) verändern die Funktionen `previousDay` und `nextDay` ihr Verhalten - Details siehe Umsetzung).

Algorithmus 4 : Umrechnung zwischen greg. und jul. Kalendersystem

Funktion `Tagesdifferenz(`Jahr, Monat`)`

$$h \leftarrow \left\{ \begin{array}{ll} \lfloor \frac{\text{Jahr}-1}{100} \rfloor, & \text{wenn Monat} \leq 2 \\ \lfloor \frac{\text{Jahr}}{100} \rfloor, & \text{wenn Monat} > 2 \end{array} \right\};$$

$a \leftarrow \lfloor \frac{h}{4} \rfloor$;
$b \leftarrow h \mod 4$;
$\Delta t = 3 * a + b - 2$;
return Δt;

Funktion `jul2greg(`$Datum_{jul}$ `d)`
$\Delta t \leftarrow$ `Tagesdifferenz(d.Jahr, d.Monat)`;
`d`.isGregorian = true;
for $x \leftarrow 1$ **bis** Δt **do**
  `d.nextDay();`
return `d`;

Funktion `greg2jul(`$Datum_{greg}$ `d)`
$\Delta t \leftarrow$ `Tagesdifferenz(d.Jahr, d.Monat)`;
`d`.isGregorian = false;
for $x \leftarrow 1$ **bis** Δt **do**
  `d.previousDay();`
return `d`;

4 Umsetzung

Die Umsetzung unserer Lösungsidee bestand im Wesentlichen aus drei Teilen;

1. dem Entwurf eines Datentypen, der ein Datum im julianischen oder gregorianischen Kalender repräsentiert und in der Lage ist, sich selbst - unter Berücksichtigung der jeweils gültigen Schaltregeln - um einen Tag zu in- und dekementieren (letzter Tag, nächster Tag),
2. die Implementierung der Osterformel (Algorithmus **3**) und des Algorithmus zur Datumsumwandlung (Algorithmus **4**) sowie
3. der Implementierung der "finalen" (da problemlösenden) Algorithmen **1** und **2**.

4.1 Der Datentyp `date`

Der Datentyp `date` besitzt zunächst vier Eigenschaften - Tag, Monat, Jahr und isGregorian; letztere bestimmt die Art des Kalendersystems. Diese Eigenschaften werden mit

dem Konstruktor übergeben.

```
7  struct Date
8  {
9      int day, month;
10     long long year;
11     bool isGregorian;
12
13     Date( const int& _d, const int& _m, const long long& _y, const ↩
           bool& _g = true) : day(_d), month(_m), year(_y), ↩
           isGregorian(_g) {};
```

Beim de- und inkrementieren des Datums ist es bei Monatsübergängen notwendig, die Länge des vorherigen oder aktuellen Monates zu kennen. Diese kann je nach Schaltjahr abweichen. Und die Schaltjahre werden durch das Kalendersystem bestimmt; Ob das jeweilige Jahr ein Schaltjahr ist, ermittelt die Memberfunktion `date::isLeapYear()`.

```
15     bool isLeapYear() {
16         // Erweiterte Schaltregeln des gregorianischen Kalendars
17         if ( year % 400 == 0 && isGregorian ) return true;
18         if ( year % 100 == 0 && isGregorian ) return false;
19
20         // "default" julian leap year rule
21         if ( year % 4 == 0 ) return true;
22
23         // sonst: kein Schaltjahr
24         return false;
25     }
```

Die Länge der Monate wird durch die Memberfunktion `date::getMonthLength` bereitgestellt;

```
27     int getMonthLength( const int _m ) {
28         switch (_m) {
29             case 1: return 31;
30             case 2: return isLeapYear() ? 29 : 28;
31             case 3: return 31;
32             case 4: return 30;
33             case 5: return 31;
34             case 6: return 30;
35             case 7: return 31;
36             case 8: return 31;
37             case 9: return 30;
38             case 10: return 31;
39             case 11: return 30;
40             case 12: return 31;
41             default: return 0;
42         }
43     }
```

Der nächste bzw. vorherige Tag wird nun folgendermaßen bestimmt:

```
45      void nextDay(void)
46      { // Inkrementieren des Datums
47          if ( day == getMonthLength(month) ) { // Monatsgrenze ü↩
                berschritten
48              day = 1;
49              if ( month == 12 ) { // Jahresgrenze überschritten
50                  month = 1;
51                  year++;
52              }else{ month++; }
53          }else{ day++; }
54      }
55      Date& operator++() { nextDay(); return *this; }
56
57      void previousDay(void)
58      { // Dekrementieren des Datums
59          if ( day == 1 ) { // Monatsgrenze überschritten
60              if ( month == 1 ) { // Jahresgrenze überschritten
61                  day = 31;
62                  month = 12;
63                  year--;
64              } else{ day = getMonthLength(month - 1); month--; }
65          }else{ day--; }
66      }
67      Date& operator--() { previousDay(); return *this; }
68  }
```

Damit ist ein Datentyp geschaffen, der im Folgenden dazu benutzt werden kann, ein bestimmtes Datum um einen Tag in die Vergangenheit oder Zukunft zu bewegen. Mehrfach hintereinander angewandt kann damit um eine bestimmte Tagesdifferenz verschoben werden - die Tagesdifferenz der Kalendersysteme.

4.2 Osterformel und Umrechnung der Kalendersysteme

Die Implementierungen der Gauß'schen Osterformel und des Algorithmus zur Umrechnung der Kalendersysteme sind unspektakulär, da sie direkt den Algorithmen **3** und **4** entsprechen.

4.3 Algorithmen zur Problemlösung

Jetzt können die Algorithmen **1** und **2** umgesetzt werden, die ermitteln wann das greogorianische Weihnachten und das julianische Ostern auf einen Tag fallen und umgekehrt.

```
131     // Gregorianisches Ostern fällt auf julianisches Weihnachten
132     for( long long int y = 2000; y <= TOTAL_APOCALYPSE; y++ )
133     {
134         Date easterDate_greg = getEasterDate(y, true); // Osterdatum ↩
                im Jahr 'y'
135         Date easterDateInJul = greg2jul(easterDate_greg); // Umrechnen ↩
                in den julianischen Kalender
136
```

```
137         if ( easterDateInJul.day == 25 && easterDateInJul.month == 12 ↩
                ) // fällt dieses im julianischen Kalender auf den 25.12., ↩
                dann Erfolg
138         {
139             cout << endl << "ERFOLG 1" << endl << "Ostern, " << ↩
                    easterDate_greg.to_str() << " [greg] = Weihnachten, " ↩
                    << easterDateInJul.to_str() << " [jul]";
140             cout << endl << "Zeitunterschied: " << ↩
                    getTimeDiff(easterDate_greg) << "Tage.";
141             break;
142         }
143     }
144
145     // Julianisches Ostern fällt auf gregorianisches Weihnachten
146     for( long long int y = 2000; y <= TOTAL_APOCALYPSE; y++ )
147     {
148         Date easterDate_jul = getEasterDate( y, false );
149         Date easterDateInGreg = jul2greg(easterDate_jul);
150
151         if ( easterDateInGreg.day == 25 && easterDateInGreg.month == ↩
                12 )
152         {
153             cout << endl << "ERFOLG 2" << endl << "Ostern, " << ↩
                    easterDate_jul.to_str() << " [jul] = Weihnachten, " << ↩
                    easterDateInGreg.to_str() << " [greg]";
154             cout << endl << "Zeitunterschied: " << getTimeDiff( ↩
                    easterDateInGreg) << "Tage.";
155             break;
156         }
157     }
```

5 Ausgabe

Die Programmausgabe lautet:

```
ERFOLG 1:
Ostern, 23.3.11919 [greg] = Weihnachten, 25.12.11918 [jul]
Zeitunterschied: 88 Tage

ERFOLG 2:
Ostern, 25.4.32839 [jul] = Weihnachten, 25.12.32839 [greg]
Zeitunterschied: 244 Tage
```

Laut Programm fallen $\text{Ostern}_{\text{greg}}$ und $\text{Weihnachten}_{\text{jul}}$ zuerst im Jahr 11.919_{greg} (11.918_{jul}) zusammen (Teilaufgabe 1). $\text{Weihnachten}_{\text{greg}}$ und $\text{Ostern}_{\text{jul}}$ (Teilaufgabe 2) dagegen erst im Jahr 32.839_{greg}.
Zur Kontrolle: Beide Daten liegen sowohl im vorausgesagtem Intervall (siehe **2**) als auch relativ Nahe an deren unteren Grenzen.

6 Quelltext

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  #define TOTAL_APOCALYPSE 900000000 // Datum der totalen Apokalypse
6
7  struct Date
8  {
9      int day, month;
10     long long year;
11     bool isGregorian;
12
13     Date( const int& _d, const int& _m, const long long& _y, const ↩
           bool& _g = true) : day(_d), month(_m), year(_y), ↩
           isGregorian(_g) {};
14
15     bool isLeapYear() {
16         // Erweiterte Schaltregeln des gregorianischen Kalendars
17         if ( year % 400 == 0 && isGregorian ) return true;
18         if ( year % 100 == 0 && isGregorian ) return false;
19
20         // "default" julian leap year rule
21         if ( year % 4 == 0 ) return true;
22
23         // sonst: kein Schaltjahr
24         return false;
25     }
26
27     int getMonthLength( const int _m ) {
28         switch (_m) {
29             case 1: return 31;
30             case 2: return isLeapYear() ? 29 : 28;
31             case 3: return 31;
32             case 4: return 30;
33             case 5: return 31;
34             case 6: return 30;
35             case 7: return 31;
36             case 8: return 31;
37             case 9: return 30;
38             case 10: return 31;
39             case 11: return 30;
40             case 12: return 31;
41             default: return 0;
42         }
43     }
44
45     void nextDay(void)
46     { // Inkrementieren des Datums
47         if ( day == getMonthLength(month) ) { // Monatsgrenze ü↩
               berschritten
```

```
48              day = 1;
49              if ( month == 12 ) { // Jahresgrenze überschritten
50                  month = 1;
51                  year++;
52              }else{ month++; }
53          }else{ day++; }
54      }
55      Date& operator++() { nextDay(); return *this; }
56
57      void previousDay(void)
58      { // Dekrementieren des Datums
59          if ( day == 1 ) { // Monatsgrenze überschritten
60              if ( month == 1 ) { // Jahresgrenze überschritten
61                  day = 31;
62                  month = 12;
63                  year--;
64              } else{ day = getMonthLength(month - 1); month--; }
65          }else{ day--; }
66      }
67      Date& operator--() { previousDay(); return *this; }
68
69      string to_str(void) { return to_string(day) + "." + ↩
            to_string(month) + "." + to_string(year); }
70
71      bool operator==(const Date& b) {
72          return ( year == b.year && month == b.month && day == b.day && ↩
                isGregorian == b.isGregorian );
73      }
74
75  };
76
77  int getTimeDiff( const Date& d )
78  { // Zeitdifferenz des julianischen und gregorianischen Datums zum ↩
        Zeitpunkt 'd' zur gegenseitigen Umrechnung
79      // Formel: siehe ↩
            https://de.wikipedia.org/wiki/Umrechnung_zwischen_julianischem_und_gregorianischem
80      int h = int( ( d.month <= 2 ? d.year - 1 : d.year ) / 100 );
81      int a = int( h / 4 );
82      int b = h % 4;
83      int dT = 3*a+b-2;
84
85      return dT;
86  }
87
88  Date greg2jul( const Date& d )
89  {
90      Date tempDate = d;
91      tempDate.isGregorian = false;
92      int dT = getTimeDiff(tempDate);
93      for( int i = 1; i <= dT; i++ ) --tempDate;
94      return tempDate;
95  }
96
```

```
97  Date jul2greg( const Date& d )
98  {
99      Date tempDate = d;
100     tempDate.isGregorian = true;
101     int dT = getTimeDiff( tempDate);
102     for( int i = 1; i <= dT; i++ ) ++tempDate;
103     return tempDate;
104 }
105
106 Date getEasterDate ( const long long year, const bool isGregorian )
107 { // Errechnet das Datum des Ostersonntag im Jahr 'year' mit der ↩
        Gauß'schen Osterformel
108     // Siehe ↩
            https://de.wikipedia.org/wiki/Gau\%C3\%9Fsche_Osterformel#Eine_erg.C3.A4nzte_Oster
109     int K = year / 100;
110
111     int M = !isGregorian ? 15    : (15 + (3*K + 3) / 4 - (8*K + 13) / 25);
112     int S = !isGregorian ? 0     : (2 - (3*K + 3) / 4);
113
114     int A = year % 19;
115     int D = (19*A + M) % 30;
116     int R = (D + A / 11) / 29;
117     int OG = 21 + D - R;
118     int SZ = 7 - (year + year / 4 + S) % 7;
119     int OE = 7 - (OG - SZ) % 7;
120     int OS = OG + OE;
121
122     int month    = (OS <= 31) ? 3: 4;
123     int day      = (OS <= 31) ? OS : (OS - 31);
124
125     return Date( day, month, year );
126 }
127
128 int main(void)
129 {
130
131     // Gregorianisches Ostern fällt auf julianisches Weihnachten
132     for( long long int y = 2000; y <= TOTAL_APOCALYPSE; y++ )
133     {
134         Date easterDate_greg = getEasterDate(y, true); // Osterdatum ↩
                im Jahr 'y'
135         Date easterDateInJul = greg2jul(easterDate_greg); // Umrechnen ↩
                in den julianischen Kalender
136
137         if ( easterDateInJul.day == 25 && easterDateInJul.month == 12 ↩
                ) // fällt dieses im julianischen Kalender auf den 25.12., ↩
                dann Erfolg
138         {
139             cout << endl << "ERFOLG 1" << endl << "Ostern, " << ↩
                    easterDate_greg.to_str() << " [greg] = Weihnachten, " ↩
                    << easterDateInJul.to_str() << " [jul]";
140             cout << endl << "Zeitunterschied: " << ↩
                    getTimeDiff(easterDate_greg);
```

```
141             break;
142         }
143     }
144
145     // Julianisches Ostern fällt auf gregorianisches Weihnachten
146     for( long long int y = 2000; y <= TOTAL_APOCALYPSE; y++ )
147     {
148         Date easterDate_jul = getEasterDate( y, false );
149         Date easterDateInGreg = jul2greg(easterDate_jul);
150
151         if ( easterDateInGreg.day == 25 && easterDateInGreg.month == ↩
                12 )
152         {
153             cout << endl << "ERFOLG 2" << endl << "Ostern, " << ↩
                    easterDate_jul.to_str() << " [jul] = Weihnachten, " << ↩
                    easterDateInGreg.to_str() << " [greg]";
154             cout << endl << "Zeitunterschied: " << getTimeDiff( ↩
                    easterDateInGreg);
155             break;
156         }
157     }
158
159     cout << endl;
160     return 1;
161 }
```

Listing 1: 1 ▸ src ▸ main.cpp - Quelltext der main.cpp

Literatur

[1] Wikipedia. Römischer kalender. *Wikipedia - Die freie Enzyklopädie*, Zugriff am 02.10.2016. `https://de.wikipedia.org/wiki/Römischer_Kalender`.

[2] Wikipedia. Julianischer kalender. *Wikipedia - Die freie Enzyklopädie*, Zugriff am 29.09.2016. `https://de.wikipedia.org/wiki/Julianischer_Kalender`.

[3] Wikipedia. Gregorianischer kalender. *Wikipedia - Die freie Enzyklopädie*, Zugriff am 05.10.2016. `https://de.wikipedia.org/wiki/Gregorianischer_Kalender`.

[4] Claus Seyfried. Kleine kalendergeschichte. *www.csey.de*, 2001, Zugriff am 06.10.2016. `http://www.csey.de/kalender.htm`.

[5] Wikipedia. Osterdatum. *Wikipedia - Die freie Enzyklopädie*, Zugrif am 04.10.2016. `https://de.wikipedia.org/wiki/Osterdatum`.

[6] Wikipedia. Gauß'sche osterformel. *Wikipedia - Die freie Enzyklopädie*, Zugrif am 20.09.2016. `https://de.wikipedia.org/wiki/Gau%C3%9Fsche_Osterformel`.

[7] Wikipedia. Umrechnung zwischen julianischem und gregorianischem datum. *Wikipedia - Die freie Enzyklopädie*, Zugrif am 25.09.2016. `https://de.wikipedia.org/wiki/Umrechnung_zwischen_julianischem_und_gregorianischem_Kalender`.