

das gleiche Datum des Ostersonntags. Nun ging ich davon aus, dass beide Funktionen korrekt sind.



Abbildung 11.1:

Test mithilfe der Osterregel nach dem »Immerwährenden Kalender« (Graßl, 1998).

*Das Modularisieren ist in den EPA Informatik bei den fachlichen und methodischen Kompetenzen aufgeführt. Beim Modularisieren erfolgt ein Strukturieren und Zerlegen in Teilprobleme. Eine interessante Aufgabe für den Informatikunterricht in der Sekundarstufe II ist das Entwerfen und Realisieren von Modulen und deren anschließende Nutzung.*

Ein Beispiel ist das Modul DatumOperations mit der folgenden Schnittstelle (in Oberon-2):

- Datentyp:  
TYPE Datum = RECORD  
  jahr, monat, tag: INTEGER;  
  END;

- Operationen:  
PROCEDURE Abstand (a, b: Datum): LONGINT;  
PROCEDURE Existenz (d: Datum): BOOLEAN;  
PROCEDURE IstGleich (a, b: Datum): BOOLEAN;  
PROCEDURE LiegtVor (a, b: Datum): BOOLEAN;  
PROCEDURE Nachfolger (VAR d: Datum);  
PROCEDURE Ostersonntag (jahr: INTEGER; VAR d: Datum);  
PROCEDURE Schaltjahr (jahr: INTEGER): BOOLEAN;  
PROCEDURE Summe (a: Datum; anzahl: LONGINT; VAR b: Datum);  
PROCEDURE TagelMJan (jahr: INTEGER): INTEGER;  
PROCEDURE TagelMMonat (jahr, monat: INTEGER): INTEGER;  
PROCEDURE Vorgaenger (VAR d: Datum);  
PROCEDURE Wochentag (d: Datum): INTEGER;

Das Realisieren der Operationen kann im Unterricht arbeitsteilig erfolgen. Einige Beispiele werden nachfolgend angegeben.

Die Funktionsprozedur LiegtVor stellt fest, ob das Datum a vor dem Datum b liegt. Sind die Jahre unterschiedlich, so erfolgt die Feststellung nur unter Heranziehung der beiden Jahreszahlen. Liegen beide Kalenderdaten im gleichen Jahr, so müssen die Monate betrachtet werden. Liegen beide Kalenderdaten sogar im gleichen Monat, so müssen die Tage betrachtet werden.

```

PROCEDURE LiegtVor*(a, b: Datum): BOOLEAN;
BEGIN
  IF a.jahr < b.jahr THEN
    RETURN TRUE
  ELSEIF a.jahr > b.jahr THEN
    RETURN FALSE
  ELSEIF a.monat < b.monat THEN
    RETURN TRUE
  ELSEIF a.monat > b.monat THEN
    RETURN FALSE
  ELSE RETURN a.tag < b.tag
  END
END LiegtVor;

```

Die Prozedur Nachfolger ermittelt das Datum des nächsten Tages. Dabei werden drei Fälle unterschieden:

- Es handelt sich nicht um einen Monatsletzten.
- Es handelt sich um einen Monatsletzten, nicht jedoch um den 31. Dezember.
- Es handelt sich um den 31. Dezember.

```

PROCEDURE Nachfolger*(VAR d: Datum);
VAR r: Datum;
BEGIN
  IF d.tag < TagelmMonat(d.jahr, d.monat) THEN
    r.jahr := d.jahr;
    r.monat := d.monat;
    r.tag := d.tag + 1
  ELSEIF d.monat < 12 THEN
    r.jahr := d.jahr;
    r.monat := d.monat + 1;
    r.tag := 1
  ELSE
    r.jahr := d.jahr + 1;
    r.monat := 1;
    r.tag := 1
  END;
  d := r
END Nachfolger;

```

Die Prozedur Summe ermittelt, ausgehend von einem bestimmten Datum, das Datum eines ganz bestimmten nachfolgenden Tages (z.B. des 1000. Tages). Bei der Realisierung wird die Prozedur Nachfolger (bzw. die Prozedur Vorgaenger) genutzt, was zu einem einfachen Algorithmus führt.

```

PROCEDURE Summe*(a: Datum; anzahl: LONGINT; VAR b: Datum);
VAR z: LONGINT;
BEGIN
  IF anzahl > 0 THEN
    FOR z := 1 TO anzahl DO Nachfolger(a) END
  ELSEIF anzahl < 0 THEN
    FOR z := 1 TO -anzahl DO Vorgaenger(a) END
  END;
  b := a
END Summe;

```

Die Funktionsprozedur Abstand ermittelt, wie viele Tage es vom Datum a zum Datum b sind. Möglich wäre auch hier die Verwendung der Prozeduren Nachfolger bzw. Vorgaenger, was eine einfach aufgebaute Funktionsprozedur erwarten ließe. Ein komplizierterer Algorithmus ist der Folgende, der gerade bei weit auseinanderliegenden Daten vergleichsweise schnell arbeitet. Es werden drei Fälle unterschieden:

- Die beiden Kalenderdaten liegen in unterschiedlichen Jahren.
- Die beiden Kalenderdaten liegen im gleichen Jahr, jedoch in unterschiedlichen Monaten.
- Die beiden Kalenderdaten liegen sogar im gleichen Monat.

Sehen wir uns z.B. den ersten Fall genauer an: Zuerst wird die Anzahl an Tagen vom Datum a bis zum Monatsletzten ermittelt. Dann werden die vollständigen Monate bis zum Jahresende addiert. Es werden die Jahre bis zum Vorgängerjahr vom Datum b addiert, dann die vollständigen Monate bis zum Vormonat vom Datum b und abschließend die Tage im Zielmonat.

```

PROCEDURE Abstand*(a, b: Datum): LONGINT;
(* Voraussetzung: a liegt vor b *)
VAR i: INTEGER;
    anzahl: LONGINT;
BEGIN
  IF a.jahr # b.jahr THEN
    anzahl := TagelmMonat(a.jahr, a.monat) - a.tag;
    FOR i := a.monat + 1 TO 12 DO
      anzahl := anzahl + TagelmMonat(a.jahr, i)
    END;
    FOR i := a.jahr + 1 TO b.jahr - 1 DO
      anzahl := anzahl + TagelmJahr(i)
    END;
    FOR i := 1 TO b.monat - 1 DO
      anzahl := anzahl + TagelmMonat(b.jahr, i)
    END;
  ELSEIF a.monat # b.monat THEN
    anzahl := TagelmMonat(a.jahr, a.monat) - a.tag;
    FOR i := a.monat + 1 TO b.monat - 1 DO
      anzahl := anzahl + TagelmMonat(b.jahr, i)
    END;
  ELSE
    anzahl := anzahl + b.tag
  END;
  ELSE anzahl := b.tag - a.tag
END;
RETURN anzahl
END Abstand;

```

Das Modul kann z.B. für das Planen von Baumaßnahmen eingesetzt werden. Die wichtigste Frage: Wann muss ich anfangen, um pünktlich fertig zu werden?

das gleiche Datum des Ostersonntags. Nun ging ich davon aus, dass beide Funktionen korrekt sind.

```

PythonWin
File Edit View Tools Window Help

def ostern(jahr):
    k = jahr / 100
    a = jahr % 10
    e = (11*a+8+(8*k+13)/25-(3*k+3)/4)%28
    if e<24:
        n=44-e
    elif e==24:
        n=49
    elif e==25:
        n=49-3/11
    else:
        n=74-e
    w=(n+5*jahr/4+2-(3*k+3)/4)%7
    s=n*7-w
    return s

j=1701
while j<2400:
    if ostern(j)!=ostern2(j):
        print j
        j=j+1
    print "ende"

```

Interpretieren Window

```

1701
1704
2011
2163
2326
ende

```

Script C:\Python\Python\Python\ostern.py / returned exit code 0

Abbildung 11.1:  
Test mithilfe der Osterregel nach dem »Immerwährenden Kalender« (Graßl, 1998).

Das Modularisieren ist in den EPA Informatik bei den fachlichen und methodischen Kompetenzen aufgeführt. Beim Modularisieren erfolgt ein Strukturieren und Zerlegen in Teilprobleme. Eine interessante Aufgabe für den Informatikunterricht in der Sekundarstufe II ist das Entwerfen und Realisieren von Modulen und deren anschließende Nutzung.

Ein Beispiel ist das Modul DatumOperations mit der folgenden Schnittstelle (in Oberon-2):

- Datentyp:  
TYPE Datum = RECORD  
    jahr, monat, tag: INTEGER;  
END;

- Operationen:

```

PROCEDURE Abstand (a, b: Datum): LONGINT;
PROCEDURE Existenz (d: Datum): BOOLEAN;
PROCEDURE IstGleich (a, b: Datum): BOOLEAN;
PROCEDURE LiegtVor (a, b: Datum): BOOLEAN;
PROCEDURE Nachfolger (VAR d: Datum);
PROCEDURE Ostersonntag (jahr: INTEGER; VAR d: Datum);
PROCEDURE Schaltjahr (jahr: INTEGER): BOOLEAN;
PROCEDURE Summe (a: Datum; anzahl: LONGINT; VAR b: Datum);
PROCEDURE TagelmJahr (jahr: INTEGER): INTEGER;
PROCEDURE TagelmMonat (jahr, monat: INTEGER): INTEGER;
PROCEDURE Vorgaenger (VAR d: Datum);
PROCEDURE Wochentag (d: Datum): INTEGER;

```

Das Realisieren der Operationen kann im Unterricht arbeitsteilig erfolgen. Einige Beispiele werden nachfolgend angegeben.

Die Funktionsprozedur LiegtVor stellt fest, ob das Datum a vor dem Datum b liegt. Sind die Jahre unterschiedlich, so erfolgt die Feststellung nur unter Heranziehung der beiden Jahreszahlen. Liegen beide Kalenderdaten im gleichen Jahr, so müssen die Monate betrachtet werden. Liegen beide Kalenderdaten sogar im gleichen Monat, so müssen die Tage betrachtet werden.

```

PROCEDURE LiegtVor*(a, b: Datum): BOOLEAN;
BEGIN
  IF a.jahr < b.jahr THEN
    RETURN TRUE
  ELSEIF a.jahr > b.jahr THEN
    RETURN FALSE
  ELSEIF a.monat < b.monat THEN
    RETURN TRUE
  ELSEIF a.monat > b.monat THEN
    RETURN FALSE
  ELSE RETURN a.tag < b.tag
  END
END LiegtVor;

```

Die Prozedur Nachfolger ermittelt das Datum des nächsten Tages. Dabei werden drei Fälle unterschieden:

- Es handelt sich nicht um einen Monatsletzten.
- Es handelt sich um einen Monatsletzten, nicht jedoch um den 31. Dezember.
- Es handelt sich um den 31. Dezember.