

FCalendar ライブラリ

佐原伸

ss@shinsahara.com

2005 年 12 月 31 日

概要

暦（カレンダー）に関わる関数を提供する、いわゆる日付計算のためのモジュールである。本モジュールは、グレゴリオ暦切替日（1582 年 10 月 15 日）以後はグレゴリオ暦を、それより前（1582 年 10 月 4 日以前）はユリウス暦の日付を使用する。各国でのグレゴリオ暦切替日は異なる（日本の場合 1873 年）ので、歴史上の暦計算は注意しなければならない。

本モジュールは、暦日付を修正ユリウス日¹で表しているため、その有効桁の範囲内（紀元前 4294288353+1 年から、4295646239 年まで）で使うことができるはずである。もっとも、グレゴリオ暦は 4909 年に 1 日ずれる予定なので、実際にはこのあたりまでが有効であろう。少なくとも、グレゴリオ暦切替前日（1582 年 10 月 4 日）から、2099 年の秋分まではテストで確認している。3000 年の春分以後は、秋分・秋分の計算は現在の天文観測計算の精度と誤差のため特定できないので、秋分・秋分の計算は 3000 年以降使えない。

なお、本モジュールは、ユリウス暦の閏年の計算を紀元前と紀元後とで同一にすることができるよう、ユリウス暦の紀元前 1 年を 0 年または -0 年と表し、紀元前 4713 年は -4712 年と表す。すなわち、紀元前の年を表すときは、以下の式が成り立つ。

本モジュールのユリウス暦の年 = ユリウス暦の年 + 1

時刻に関わる計算はグリニッジ標準時を使用しているので、本クラスのスバークラスとして各国別・標準時別の暦クラスを作成し、**グリニッジ標準時との差**に相当する値を設定しなければならない。

0.1 用語の定義

ふなばただよし氏による用語の定義²を参考に記述する。

0.1.1 暦日付

暦日付は、通常の年月日による日付である。

¹後述。

²<http://www.funaba.org/>

0.1.2 ユリウス日

紀元前 4713 年 1 月 1 日（ユリウス暦）正午（グリニッジ平均時）を暦元とした通日（経過日数）である。

0.1.3 修正ユリウス日

1858 年 11 月 17 日（グレゴリオ暦）0 時（協定世界時）を暦元とした通日（経過日数）である。もともとはユリウス日を暦の計算に用いていたが、桁数があふれないように修正したユリウス日である。

0.1.4 日付

本モジュールで日付という場合、修正ユリウス日を表す Date 型（実体は実数型）の値を言う。

0.1.5 年日付（年間通算日）

年の中の序数によって指定される日付で、例えば、1 月 10 日の年日付は 10 で、2 月 1 日の年日付は 32 である。

0.1.6 実数年日付

整数部が年、小数点以下が年日付 / 年間総日数を表す形式の日付。日付計算の過程で使用するので、通常は気にする必要はない。例えば、暦日付 2001 年 7 月 1 日は、実数年日付では 2001.5 となる。

0.1.7 月日付（月間通算日）

月の中の序数によって指定される日付で、例えば、1 月 31 日の月日付は 31 で、2 月 1 日の月日付は 1 である。

0.2 謝辞

本モジュールの算法は、Nifty-Serve 天文計算フォーラムの方々の助言を参考に実装した。また、HowManyDayOfWeekWithin2Days 関数の算法は、山崎利治さんのアイデアによるものである。

0.3 歴史

本モジュールの最初の版は Digitalk Smalltalk で実装した。次に、VisualWorks (Smalltalk の本家) に移植し、その後関数型プログラミング言語 Concurrent Clean に移植し、最後に VDM++ に移植した。本ライブラリーは、(SSLib と称するライブラリーに含まれる) オブジェクト指向で作成した VDM++ 版を、さらに関数型指向に移植したものである。なお、数多くのディストラブルを経て、Digitalk Smalltalk 版と VisualWorks 版は現存しない。もともと、バックアップ用のディスクなどに残っている可能性はあるが、まだ確認していない。

0.4 FCalendar

グレゴリオ暦に関わる関数を定義する。

class *FCalendar*

使用する型は以下の通りである。Date は、修正ユリウス日を表すので実数型である。DayOfWeek は、曜日計算が便利のように 0,...,6 の値を各曜日にふっていて、日曜日が 0、土曜日が 6 である。

types

```
1.0 public Date = ℝ;
2.0 public DayOfWeekName = SUN | MON | TUE | WED | THU | FRI |
SAT;
3.0 public DayOfWeek = ℕ
.1 inv dayOfWeek  $\triangle$  dayOfWeek  $\leq 6$ 
```

使用する値は以下の通り。diffJDandMJD はユリウス日 (Julian Date) と修正ユリウス日 (Modified Julian Date) の日数差である。reviseMonths は、日付計算が便利のように使用する月数を意味する。

values

private

```
4.0 diffJDandMJD = 2400000.5;
```

private

```
5.0 DayOfWeekSequence = [SUN, MON, TUE, WED, THU, FRI, SAT];
```

private

```
6.0 daysInYear = 365.25;
```

private

```
7.0 monthsInYear = 12;
```

private

```
8.0 reviseMonths = 14;
```

private

```
9.0 daysInWeek = 7;
```

private

```
10.0 averageDaysInMonth = 30.6001;
```

private

```
11.0 yearsInCentury = 100;
```

private

```
12.0 constForDayCalc = 122.1;
```

private

```
13.0 constForYearCalc = 4800;
```

private

```
14.0 constForCenturyCalc = 32044.9;
```

private

```
15.0 theDayBeforeGregorianCalendar = 2299160;
```

private

```
16.0 theFirstDayOfGregorianCalendar = 1582.78;
```

public

```
17.0 max = FNumber‘Max[ℝ] (FNumber‘GT);
```

public

```
18.0 min = FNumber‘Min[ℝ] (FNumber‘GT)
```

DateFromInt は、(yyyy)(mm)(dd) で表した暦日付に対応する日付を返す。日 dd は、(1,...,31) でなくてよい。0 日は、前月末尾に補正され、12 月 32 日は翌年 1 月 1 日に補正される。

functions

public static

```
19.0 DateFromInt :  $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Q} \rightarrow \text{Date}$ 
.1 DateFromInt (yyyy) (mm) (dd)  $\triangleq$ 
.2   let [year, month] =
.3     if (mm > reviseMonths - monthsInYear)
.4     then [yyyy + constForYearCalc, mm + 1]
.5     else [yyyy + constForYearCalc - 1, mm + reviseMonths - 1],
.6   aCentury = year div yearsInCentury,
.7   constCentury =
.8     if (ConvToYear (yyyy, mm, dd) >
theFirstDayOfGregorianCalendar)
.9     then aCentury div 4 - aCentury - 32167
.10    else - 32205,
.11   halfDay = 0.5 in
.12   floor (daysInYear × year) +
.13   floor (averageDaysInMonth × month) +
.14   dd +
.15   constCentury - halfDay - diffJDandMJD;
```

GetLegalDate は、年月日を通常の値の範囲内に変換する。月 *tempM* は、(1,...12) でなく、13 以上や 0 や負数でもよい。例えば、13 は翌年 1 月に補正され、0 年は前年 12 月に補正される。日 *dd* も、(1,...,31) でなくてよい。0 日は、前月末尾に補正され、12 月 32 日は翌年 1 月 1 日に補正される。

public static

```
20.0 GetLegalDate :  $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \text{Date}$ 
.1 GetLegalDate (tmpY) (tempM) (dd)  $\triangleq$ 
.2   let mk - (yyyy, mm) = GetLegalMonth (tmpY) (tempM) in
.3   DateFromInt (yyyy) (mm) (dd);
```

GetLegalMonth は、年月を通常の値の範囲内に変換する。

public static

```
21.0 GetLegalMonth :  $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z} \times \mathbb{Z}$ 
.1 GetLegalMonth (tempY) (tempM)  $\triangleq$ 
.2   let y =
.3     if tempM ≤ 0
.4     then tempY + (tempM - 12) div monthsInYear
.5     else tempY + (tempM - 1) div monthsInYear,
.6   m = FInteger‘amod’ (tempM) (monthsInYear) in
.7   mk - (y, m);
```

Int3FromDate は、日付から三つ組み数 (*yyyy*, *mm*, *dd*) で表した暦日付を得る。

public static

```
22.0 Int3FromDate :  $\text{Date} \rightarrow \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ 
.1 Int3FromDate (aDate)  $\triangleq$ 
.2   mk - (Year (aDate), Month (aDate), Day (aDate));
```

Year は、日付から、その日付の属する年を得る。

public static

```

23.0  $Year : Date \rightarrow \mathbb{Z}$ 
    .1  $Year(aDate) \triangleq$ 
    .2   if  $MonthAux(aDate) < reviseMonths$ 
    .3   then  $YearAux(aDate) - constForYearCalc$ 
    .4   else  $YearAux(aDate) - constForYearCalc + 1$ ;
    Month は、日付から、その日付の属する月を得る。
public static
24.0  $Month : Date \rightarrow \mathbb{Z}$ 
    .1  $Month(aDate) \triangleq$ 
    .2   if  $MonthAux(aDate) < reviseMonths$ 
    .3   then  $MonthAux(aDate) - 1$ 
    .4   else  $MonthAux(aDate) - 13$ ;
    Day は、日付から、日を得る。
public static
25.0  $Day : Date \rightarrow \mathbb{Z}$ 
    .1  $Day(aDate) \triangleq$ 
    .2    $DayInMonth(aDate)$ ;
    DayInYear は、年日付を得る。
public static
26.0  $DayInYear : Date \rightarrow \mathbb{Z}$ 
    .1  $DayInYear(aDate) \triangleq$ 
    .2   let  $firstDay = DateFromInt(Year(aDate))(1)(0)$  in
    .3    $Diff(aDate)(firstDay)$ ;
    DayInMonth は、月日付けを得る。
public static
27.0  $DayInMonth : Date \rightarrow \mathbb{Z}$ 
    .1  $DayInMonth(aDate) \triangleq$ 
    .2    $\text{floor}(DayInMonthAsReal(aDate))$ ;
    DayInMonthAsReal は、実数の月日付けを得る。

28.0  $DayInMonthAsReal : Date \rightarrow \mathbb{R}$ 
    .1  $DayInMonthAsReal(aDate) \triangleq$ 
    .2    $YMDAUX(aDate) + constForDayCalc - \text{floor}(daysInYear \times$ 
    .3    $YearAux(aDate)) -$ 
    .4    $\text{floor}(averageDaysInMonth \times MonthAux(aDate))$ ;
    MonthAux は、日付計算上都合の良い月 (4..15) を返す補助関数。

29.0  $MonthAux : Date \rightarrow \mathbb{Z}$ 
    .1  $MonthAux(aDate) \triangleq$ 
    .2    $\text{floor}((YMDAUX(aDate) + constForDayCalc -$ 
    .3    $\text{floor}(daysInYear \times YearAux(aDate)))/$ 
    .4    $averageDaysInMonth)$ ;
    YMDAUX は、日付を年月日に変更するための補助関数。グレゴリオ暦切
    替前と、グレゴリオ暦切替後の考慮を行っている。

```

```

30.0  YMDAUX : Date → ℝ
.1    YMDAUX (aDate) △
.2    let JD = MJD2JD (aDate),
.3        aCentury = floor ((JD + constForCenturyCalc)/36524.25) in
.4    if JD > theDayBeforeGregorianCalendar
.5    then JD + constForCenturyCalc + aCentury − aCentury div 4 + 0.5
.6    else JD + 32082.9 + 0.5;

```

YearAux は、日付から日付計算に都合の良い補正をした年数を求めるための補助関数。

```

31.0  YearAux : Date → ℤ
.1    YearAux (aDate) △
.2    floor (YMDAUX (aDate)/daysInYear);

```

ConvToYear は、(整数三つ組の) 暦日付を年³に変換する。例えば、ConvToYear(2001,7,1) は 2001.5 を返す。

```

public static
32.0  ConvToYear : ℤ × ℤ × ℚ → ℝ
.1    ConvToYear (yyyy, mm, dd) △
.2    yyyy + (mm − 1)/monthsInYear + (floor (dd) − 1)/daysInYear;

```

MJD2JD は、修正ユリウス日をユリウス日に変換する。

```

public static
33.0  MJD2JD : Date → ℝ
.1    MJD2JD (aMJD) △
.2    aMJD + diffJDandMJD;

```

JD2MJD は、ユリウス日を修正ユリウス日すなわち日付に変換する。

```

public static
34.0  JD2MJD : ℝ → Date
.1    JD2MJD (aJD) △
.2    aJD − diffJDandMJD;

```

0.4.1 計算関数群

Diff は、2つの日付 d1, d2 の差を得る。

```

public static
35.0  Diff : Date → Date → ℤ
.1    Diff (d1)(d2) △
.2    floor (d1 − d2);

```

0.4.2 照会関数群

IsLeapYear は、閏年であれば true、平年であれば false を返す。

```

public static
36.0  IsLeapYear : ℤ → ℬ
.1    IsLeapYear (yyyy) △
.2    yyyy mod 400 = 0 ∨ (yyyy mod yearsInCentury ≠ 0 ∧
yyyy mod 4 = 0);

```

³整数部が年、小数点以下が年の中での日付を表す形式で、修正ユリウス日ではない。

GetDayOfWeek は、曜日数を得る。

public static

37.0 *GetDayOfWeek* : *Date* → *DayOfWeek*

.1 *GetDayOfWeek* (*d*) \triangleq

.2 $(\text{floor } (d) - 4) \bmod \text{daysInWeek};$

GetDayOfWeekName は、曜日名を得る。

public static

38.0 *GetDayOfWeekName* : *Date* → *DayOfWeekName*

.1 *GetDayOfWeekName* (*d*) \triangleq

.2 *DayOfWeekSequence* (*GetDayOfWeek* (*d*) + 1);

GetDayOfWeekFromName は、曜日名から曜日数を求める。

public static

39.0 *GetDayOfWeekFromName* : *DayOfWeekName* → *DayOfWeek*

.1 *GetDayOfWeekFromName* (*dn*) \triangleq

.2 *FSequence* 'Index [*DayOfWeekName*] (*dn*) (*DayOfWeekSequence*) -

1;

FirstDayOfWeekOfMonth は、指定した yyyy 年 m 月の最初の dayOfWeek-
Name 曜日名の日付を得る。

public static

40.0 *FirstDayOfWeekOfMonth* : *DayOfWeekName* → \mathbb{Z} → \mathbb{Z} → *Date*

.1 *FirstDayOfWeekOfMonth* (*dayOfWeekName*) (*m*) (*yyyy*) \triangleq

.2 let *dayOfWeek* = *GetDayOfWeekFromName* (*dayOfWeekName*),

.3 *firstDayOfMonth* = *GetFirstDayOfMonth* (*m*) (*yyyy*),

.4 *diff* = *dayOfWeek* - *GetDayOfWeek* (*firstDayOfMonth*) in

.5 cases true :

.6 (*diff* = 0) → *firstDayOfMonth*,

.7 (*diff* > 0) → *firstDayOfMonth* + *diff*,

.8 (*diff* < 0) → *firstDayOfMonth* + ((*daysInWeek* +
diff) mod *daysInWeek*)

.9 end;

GetLastDayOfWeekOfMonth は、指定した yyyy 年 m 月の最後の day-
OfWeekName 曜日名の日付を得る。

指定された月の翌月の最初の指定曜日から 7 日前を返す。月が 1 2 月の場
合でも本クラスの関数は yyyy 年 13 月を yyyy+1 年 1 月と解釈するので、問
題ない。

public static

41.0 *GetLastDayOfWeekOfMonth* : *DayOfWeekName* → \mathbb{Z} → \mathbb{Z} →
Date

.1 *GetLastDayOfWeekOfMonth* (*dayOfWeekName*) (*m*) (*yyyy*) \triangleq

.2 *FirstDayOfWeekOfMonth* (*dayOfWeekName*) (*m* + 1) (*yyyy*) -
daysInWeek;

GetNthDayOfWeekOfMonth は、指定された yyyy 年 m 月 dayOfWeek-
Name 曜日名の、第 n 曜日を求める。第 n 曜日が存在しなければ nil を返す。

月初指定曜日の (n - 1) * 7 日後を返す。

public static


```

42.0 GetNthDayOfWeekOfMonth : DayOfWeekName →  $\mathbb{Z}$  →  $\mathbb{Z}$  →  $\mathbb{Z}$ 
→ [Date]
.1 GetNthDayOfWeekOfMonth (dayOfWeekName)(n)(m)(yyyy)  $\triangleq$ 
.2   let firstDayOfWeekOfMonth = FirstDayOfWeekOfMonth (dayOfWeekName)(m)(yyyy),
.3     r = firstDayOfWeekOfMonth + (daysInWeek × (n − 1)) in
.4   cases Month (r) :
.5     (m) → r,
.6     others → nil
.7   end;

```

GetFirstDayOfMonth は、指定した *yyyy* 年 *m* 月の月初日を得る。

```

public static
43.0 GetFirstDayOfMonth :  $\mathbb{Z}$  →  $\mathbb{Z}$  → Date
.1 GetFirstDayOfMonth (m)(yyyy)  $\triangleq$ 
.2   GetLegalDate (yyyy)(m)(1);

```

GetLastDayOfMonth は、指定した *yyyy* 年 *m* 月の月末日を求める。

翌月の月初日の 1 日前を返す。

```

public static
44.0 GetLastDayOfMonth :  $\mathbb{Z}$  →  $\mathbb{Z}$  → Date
.1 GetLastDayOfMonth (m)(yyyy)  $\triangleq$ 
.2   GetLegalDate (yyyy)(m + 1)(1) − 1;

```

IsSunday は、指定日が日曜日か否かを返す。

```

public static
45.0 IsSunday : Date →  $\mathbb{B}$ 
.1 IsSunday (d)  $\triangleq$ 
.2   GetDayOfWeek (d) = 0;
IsSaturday は、指定日が土曜日か否かを返す。

```

```

public static
46.0 IsSaturday : Date →  $\mathbb{B}$ 
.1 IsSaturday (d)  $\triangleq$ 
.2   GetDayOfWeek (d) = 6;
IsWeekDay は、指定日がウィークデイか否かを返す。

```

```

public static
47.0 IsWeekDay : Date →  $\mathbb{B}$ 
.1 IsWeekDay (d)  $\triangleq$ 
.2   GetDayOfWeek (d) ∈ {1, ..., 5};
IsDayOfWeekNameWeekDay は、指定した dayOfWeekName 曜日名がウ
ィークデイか否かを返す。

```

```

public static
48.0 IsDayOfWeekNameWeekDay : DayOfWeekName →  $\mathbb{B}$ 
.1 IsDayOfWeekNameWeekDay (dayOfWeekName)  $\triangleq$ 
.2   dayOfWeekName ∉ {SAT, SUN};

```

0.4.3 指定された曜日が何日あるかを返す関数

HowManyDayOfWeekWithin2Days は、指定された *dayOfWeekName* 曜日名が、指定された日付間 (*d1* と *d2* の間) に何日あるかを返す。*d1* と *d2* が指定された曜日であれば勘定に入れる。

以下は、*HowManyDayOfWeekWithin2Days* 関数の山崎利治さんによる段階的洗練を佐原が「翻訳」した記述である。

前件は以下である。

$type\ R = \{ | \text{rng}[n \rightarrow n/7 \mid n \in Int] | \}$ (注) 7 で割った商の集合

$f, t \in Int, w \in R, 0 \leq f \leq t, h: Int \rightarrow R$ (注) 環準同型 (ring homomorphism)

後件は以下のようになる。

$$\exists S \cdot \bullet S = h^{-1}(w) \cap f..t \wedge \text{答え} \equiv \text{card}(S)$$

すなわち、整数系を環 (ring) と見て、その商環 (quotient ring) への準同型写像があり、その代数系上で後件 (事後条件) を満たすプログラムを作るといふ問題に抽象化された。HowManyDayOfWeekWithin2Days は、**7 で割る** 特殊な場合の実装であるということになる。

```
I = {f..t}
d = t - f + 1  -- = card(I)
q = d / 7
r = d \ 7  --7 で割った余り
```

とすると、答え A に対して $q \leq A \leq q + 1$ が成り立つ。なぜなら、

- 任意の連続する 7 日間には、必ず w 曜日がちょうど 1 日存在する。
- $\text{card}(I) = 7 \times q + r (0 \leq r < 7)$ であるから、I には少なくとも q 個の連続する 7 日間が存在するが、q+1 個は存在しない。
- 余りの r 日間に w 曜日が存在するかも知れない。

次に、

```
x ++ y = (x + y) \ 7
x ⊣ y = max(x - y, 0)
```

として、

```
T = {h(f)..h(f) ++ (r ⊣ 1)}
```

を考える。T は余り r 日間の曜日に対応する ($\text{card}(T) = r$)。すると、

```
A ≡ if w ∈ T then q + 1 else q end
```

ここで、

```
x minus y = if x ≥ y then x - y else x - y + 7 end
```

とすれば、

$$w \in T \Leftrightarrow (w \text{ minus } h(f)) + 1 \leq r$$

である。なぜならば

$$\begin{aligned} w \in T &\Leftrightarrow \{0..(r \text{ ⊣ } 1)\} \ni w = w \text{ minus } h(f) \\ &\Leftrightarrow r \text{ ⊣ } 1 \geq w \\ &\Leftrightarrow r \geq (w \text{ minus } h(f)) + 1 \end{aligned}$$

従って、プログラムは以下のようになる。

```

A(f, t w) ≡
let
d ≡ t - f + 1
q ≡ d / 7
r ≡ d \ 7
delta ≡ if (w minus h(f)) + 1 ≤ r then 1 els 0 end
x minus y ≡ if x ≥ y then x - y els x - y + 7 end
in
q + delta
end

```

あとは、上記プログラムを VDM++ に翻訳すればよい。

```

public static
49.0  HowManyDayOfWeekWithin2Days : DayOfWeekName → Date →
Date → ℤ
.1  HowManyDayOfWeekWithin2Days (dayOfWeekName)(d1)(d2) △
.2    let dayOfWeek = GetDayOfWeekFromName (dayOfWeekName),
.3      f = min (d1) (d2),
.4      t = max (d1) (d2),
.5      days = Diff (t) (f) + 1,
.6      q = days div daysInWeek,
.7      r = days mod daysInWeek,
.8      delta = if SubtractDayOfWeek (dayOfWeek) (GetDayOfWeek (f)) +
1 ≤ r
.9          then 1
.10         else 0 in
.11    q + delta;

```

SubtractDayOfWeek は、曜日数の減算を行う。

```

private static
50.0  SubtractDayOfWeek : ℤ → ℤ → ℤ
.1  SubtractDayOfWeek (x)(y) △
.2    if x ≥ y
.3    then x - y
.4    else x - y + daysInWeek;

```

GetVernalEquinoxInGMT は、yyyy 年のグリニッジ標準時の春分を得る。

```

public static
51.0  GetVernalEquinoxInGMT : ℤ → Date
.1  GetVernalEquinoxInGMT (yyyy) △
.2    let y = yyyy/1000 in
.3    JD2MJD (1721139.2855 + 365.242138 × yyyy + y × y × (0.067919 -
0.002788 × y));

```

GetSummerSolsticeInGMT は、yyyy 年のグリニッジ標準時の夏至を得る。

```

public static
52.0  GetSummerSolsticeInGMT : ℤ → Date
.1  GetSummerSolsticeInGMT (yyyy) △
.2    let y = yyyy/1000 in
.3    JD2MJD (1721233.2486 + 365.241728 × yyyy - y × y × (0.053018 -
0.009332 × y));

```

GetAutumnalEquinoxInGMT は、yyyy 年のグリニッジ標準時の秋分を得る。

```

public static
53.0 GetAutumnalEquinoxInGMT :  $\mathbb{Z} \rightarrow \text{Date}$ 
    .1 GetAutumnalEquinoxInGMT (yyyy)  $\triangleq$ 
    .2   let y = yyyy/1000 in
    .3   JD2MJD ( $1721325.6978 + 365.242505 \times \text{yyyy} - y \times y \times (0.126689 -$ 
0.00194  $\times y)$ );
    GetWinterSolsticeInGMT は、yyyy 年のグリニッジ標準時の冬至を得る。
public static
54.0 GetWinterSolsticeInGMT :  $\mathbb{Z} \rightarrow \text{Date}$ 
    .1 GetWinterSolsticeInGMT (yyyy)  $\triangleq$ 
    .2   let y = yyyy/1000 in
    .3   JD2MJD ( $1721414.392 + 365.24289 \times \text{yyyy} - y \times y \times (0.010965 -$ 
0.008485  $\times y)$ );
    GetDateInST は、GMT 基準の日付と、求めたい（日本標準時などの）標準時との差 diff（単位=時間）を与えて、標準時基準の日付を得る。日本の場合、日本標準時 = GMT + 9 時間。
public static
55.0 GetDateInST :  $\mathbb{Q} \rightarrow \text{Date} \rightarrow \text{Date}$ 
    .1 GetDateInST (diff)(d)  $\triangleq$ 
    .2   floor (d + diff/24);

```

0.4.4 休日に関わる照会関数群

以下は、休日の考慮をした機能である。

GetHolidaysWithinDates は、ある年の休日の集合を得る *getHolidays* 関数で決まる、2つの日付の間の休日の集合を返す。日曜日である休日も含むが、休日でない日曜日は含まない。

```

public static
56.0 GetHolidaysWithinDates : ( $\mathbb{Z} \rightarrow \text{Date-set}$ )  $\rightarrow \text{Date} \rightarrow \text{Date} \rightarrow$ 
Date-set
    .1 GetHolidaysWithinDates (getHolidays)(d1)(d2)  $\triangleq$ 
    .2   let date1 = min (d1) (d2),
    .3     date2 = max (d1) (d2),
    .4     setOfYear = { Year (date1), ..., Year (date2) },
    .5     holidays =  $\bigcup \{ \text{getHolidays} (y) \mid y \in \text{setOfYear} \}$  in
    .6   { h | h  $\in$  holidays  $\cdot d1 \leq h \wedge h \leq d2$  };

```

GetHolidaysWithinDatesNotSunday は、ある年の休日の集合を得る *getHolidays* 関数で決まる、2つの日付の間の日曜日を含まない休日の集合を返す。

```

public static
57.0 GetHolidaysWithinDatesNotSunday : ( $\mathbb{Z} \rightarrow \text{Date-set}$ )  $\rightarrow \text{Date} \rightarrow$ 
Date  $\rightarrow \text{Date-set}$ 
    .1 GetHolidaysWithinDatesNotSunday (getHolidays)(d1)(d2)  $\triangleq$ 
    .2   let holidays = GetHolidaysWithinDates (getHolidays) (d1) (d2) in
    .3   { h | h  $\in$  holidays  $\cdot \neg \text{IsSunday} (h)$  };

```

GetHolidaysWithinDatesAsSunday は、日曜日である休日の集合を返す。

```

public static

```

58.0 *GetHolidaysWithinDatesAsSunday* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date* \rightarrow *Date-set*

```
.1 GetHolidaysWithinDatesAsSunday (getHolidays) (d1) (d2)  $\triangleq$ 
.2   let holidays = GetHolidaysWithinDates (getHolidays) (d1) (d2) in
.3   {h | h  $\in$  holidays  $\cdot$  IsSunday (h)};
```

GetNumberOfHolidaysWithinDates は、2つの日付の間の休日数を返す。日曜日である休日も含むが、休日でない日曜日は含まない。

public static

59.0 *GetNumberOfHolidaysWithinDates* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date* \rightarrow \mathbb{Z}

```
.1 GetNumberOfHolidaysWithinDates (getHolidays) (d1) (d2)  $\triangleq$ 
.2   card GetHolidaysWithinDates (getHolidays) (d1) (d2);
```

GetNumberOfDayOff は、2つの日付の間の休日あるいは日曜日の数を返す（両端が該当日であれば含む）

public static

60.0 *GetNumberOfDayOff* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date* \rightarrow \mathbb{Z}

```
.1 GetNumberOfDayOff (getHolidays) (d1) (d2)  $\triangleq$ 
.2   let date1 = min (d1) (d2),
.3       date2 = max (d1) (d2),
.4       numOfSunday = HowManyDayOfWeekWithin2Days (SUN) (d1) (d2) in
.5       numOfSunday +
```

card *GetHolidaysWithinDatesNotSunday* (*getHolidays*) (*date1*) (*date2*);

GetNumberOfDayOff1 は、2つの日付の間の休日あるいは日曜日の数を返す（開始日を含まない）

public static

61.0 *GetNumberOfDayOff1* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date* \rightarrow \mathbb{Z}

```
.1 GetNumberOfDayOff1 (getHolidays) (d1) (d2)  $\triangleq$ 
.2   let date1 = min (d1) (d2),
.3       date2 = max (d1) (d2) in
.4   GetNumberOfDayOff (getHolidays) (date1 + 1) (date2);
```

0.4.5 休日に関わる計算関数群

BusinessDateToFuture は、休日でない日付を返す（未来へ向かって探索する）。

public static

62.0 *BusinessDateToFuture* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date*

```
.1 BusinessDateToFuture (getHolidays) (d)  $\triangleq$ 
.2   cases IsDayOff (getHolidays) (d)  $\vee$  IsSaturday (d) :
.3     true  $\rightarrow$  BusinessDateToFuture (getHolidays) (d + 1),
.4     others  $\rightarrow$  d
.5   end;
```

BusinessDateToPast は、休日でない日付を返す（過去へ向かって探索する）。

public static

63.0 *BusinessDateToPast* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow *Date*

```
.1 BusinessDateToPast (getHolidays)(d)  $\triangleq$ 
.2   cases IsDayOff (getHolidays) (d)  $\vee$  IsSaturday (d) :
.3     true  $\rightarrow$  BusinessDateToPast (getHolidays) (d - 1),
.4     others  $\rightarrow$  d
.5   end;
```

AddBusinessDays は、与えられた平日 *d* に、平日 *n* 日分を加算する。

public static

64.0 *AddBusinessDays* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow $\mathbb{Z} \rightarrow$ *Date*

```
.1 AddBusinessDays (getHolidays)(d)(n)  $\triangleq$ 
.2   AddBusinessDaysAux (getHolidays) (BusinessDateToFuture (getHolidays) (d)) (n);
```

public static

65.0 *AddBusinessDaysAux* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow $\mathbb{Z} \rightarrow$ *Date*

```
.1 AddBusinessDaysAux (getHolidays)(d)(n)  $\triangleq$ 
.2   cases IsDayOff (getHolidays) (d)  $\vee$  IsSaturday (d) :
.3     true  $\rightarrow$  AddBusinessDaysAux (getHolidays) (d + 1) (n),
.4     others  $\rightarrow$  if n  $\leq$  0
.5       then d
.6       else AddBusinessDaysAux (getHolidays) (d + 1) (n -
1)
.7   end;
```

SubtractBusinessDays は、与えられた平日に、平日 *n* 日分を減算する。

public static

66.0 *SubtractBusinessDays* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow $\mathbb{Z} \rightarrow$ *Date*

```
.1 SubtractBusinessDays (getHolidays)(d)(n)  $\triangleq$ 
.2   SubtractBusinessDaysAux (getHolidays) (BusinessDateToPast (getHolidays) (d)) (n);
```

public static

67.0 *SubtractBusinessDaysAux* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow $\mathbb{Z} \rightarrow$ *Date*

```
.1 SubtractBusinessDaysAux (getHolidays)(d)(n)  $\triangleq$ 
.2   cases IsDayOff (getHolidays) (d)  $\vee$  IsSaturday (d) :
.3     true  $\rightarrow$  SubtractBusinessDaysAux (getHolidays) (d - 1) (n),
.4     others  $\rightarrow$  if n  $\leq$  0
.5       then d
.6       else SubtractBusinessDaysAux (getHolidays) (d - 1) (n -
1)
.7   end;
```

0.4.6 休日に関わる検査関数群

IsHoliday は、指定した日 *d* が休日か否かを返す。

public static

68.0 *IsHoliday* : ($\mathbb{Z} \rightarrow \text{Date-set}$) \rightarrow *Date* \rightarrow \mathbb{B}

```
.1 IsHoliday (getHolidays)(d)  $\triangleq$ 
.2   d  $\in$  getHolidays (Year (d));
```

IsDayOff は、指定した日 *d* が休み（休日または日曜日）であるか否かを返す。

public static

```

69.0  IsDayOff : ( $\mathbb{Z} \rightarrow \text{Date-set}$ )  $\rightarrow$  Date  $\rightarrow$   $\mathbb{B}$ 
      .1 IsDayOff (getHolidays)(d)  $\triangleq$ 
      .2 IsSunday (d)  $\vee$  IsHoliday (getHolidays) (d)
end FCalendar
  Test Suite :      vdm.tc
  Class :          FCalendar

```

Name	#Calls	Coverage
FCalendar'Day	9	✓
FCalendar'Diff	43	✓
FCalendar'Year	207	✓
FCalendar'Month	593	✓
FCalendar'JD2MJD	320	✓
FCalendar'MJD2JD	3031	✓
FCalendar'YMDAUX	3029	✓
FCalendar'YearAux	1618	✓
FCalendar'IsDayOff	109	✓
FCalendar'IsSunday	2340	✓
FCalendar'MonthAux	1402	✓
FCalendar'DayInYear	12	✓
FCalendar'IsHoliday	100	✓
FCalendar'IsWeekDay	7	✓
FCalendar'ConvToYear	2392	✓
FCalendar'DayInMonth	9	✓
FCalendar'IsLeapYear	10	✓
FCalendar'IsSaturday	58	✓
FCalendar'DateFromInt	2392	✓
FCalendar'GetDateInST	306	✓
FCalendar'GetDayOfWeek	3072	✓
FCalendar'GetLegalDate	653	✓
FCalendar'Int3FromDate	8	✓
FCalendar'GetLegalMonth	675	✓
FCalendar'AddBusinessDays	4	✓
FCalendar'DayInMonthAsReal	9	✓
FCalendar'GetDayOfWeekName	22	✓
FCalendar'GetLastDayOfMonth	31	✓
FCalendar'GetNumberOfDayOff	18	✓
FCalendar'SubtractDayOfWeek	29	✓
FCalendar'AddBusinessDaysAux	19	✓

Name	#Calls	Coverage
FCalendar‘BusinessDateToPast	22	✓
FCalendar‘GetFirstDayOfMonth	604	✓
FCalendar‘GetNumberOfDayOff1	9	✓
FCalendar‘BusinessDateToFuture	31	✓
FCalendar‘GetDayOfWeekFromName	637	87%
FCalendar‘SubtractBusinessDays	5	✓
FCalendar‘FirstDayOfWeekOfMonth	600	✓
FCalendar‘GetVernalEquinoxInGMT	148	✓
FCalendar‘GetHolidaysWithinDates	43	✓
FCalendar‘GetNthDayOfWeekOfMonth	584	✓
FCalendar‘GetSummerSolsticeInGMT	5	✓
FCalendar‘GetWinterSolsticeInGMT	5	✓
FCalendar‘IsDayOfWeekNameWeekDay	7	✓
FCalendar‘GetAutumnalEquinoxInGMT	148	✓
FCalendar‘GetLastDayOfWeekOfMonth	9	✓
FCalendar‘SubtractBusinessDaysAux	31	✓
FCalendar‘HowManyDayOfWeekWithin2Days	29	✓
FCalendar‘GetHolidaysWithinDatesAsSunday	3	✓
FCalendar‘GetNumberOfHolidaysWithinDates	10	✓
FCalendar‘GetHolidaysWithinDatesNotSunday	21	✓
Total Coverage		99%

0.5 FCalendarT

FCalendar のテストを行う。

本テストは、以下の WWW ページの暦計算結果によってもチェックした。

<http://www.funaba.org/calendar-conversion.cgi>

また、テストデータのいくつかは、Edward M. Reingold, Nachum Dershowitz : Calendrical Calculations The Millennium Edition, Cambridge University Press, 2001, ISBN 0-521-77752-6 を参考にした。

class *FCalendarT* is subclass of *FCalendar*

functions

public static

70.0 *run* : () → \mathbb{B}

.1 *run* () \triangleq

.2 let *testcases* =

.3 [*t1* (), *t2* (), *t3* (), *t4* (), *t5* (), *t6* (), *t7* (), *t8* (), *t9* (), *t10* (),

.4 *t11* (), *t12* (), *t13* (), *t14* ()] in

.5 *FTestDriver*.*run* (*testcases*);

0.5.1 DateFromInt, Int3FromDate を検査する

修正ユリウス日開始日 (1858 年 11 月 17 日 0 時)、グレゴリオ暦切替前日正午 (1582 年 10 月 4 日 12 時)、グレゴリオ暦初日正午 (1582 年 10 月 15 日 12 時) を確認することで、DateFromInt 関数などをテストする。

```

71.0  t1 : () → FTestDriver' TestCase
.1    t1 ()  $\triangle$ 
.2      mk-FTestDriver' TestCase
.3      (
.4        "FCalendarT01:\t DateFromInt, Int3FromDate を検査す
る",
.5        DateFromInt (1858) (11) (17) = 0  $\wedge$ 
.6        Int3FromDate (0) = mk- (1858, 11, 17)  $\wedge$ 
.7        DateFromInt (1582) (10) (4.5) = JD2MJD (2299160)  $\wedge$ 
.8        Int3FromDate (DateFromInt (1582) (10) (4.5)) =
mk- (1582, 10, 4)  $\wedge$ 
.9        DateFromInt (1582) (10) (15.5) = JD2MJD (2299161)  $\wedge$ 
.10       Int3FromDate (DateFromInt (2004) (2) (28)) =
mk- (2004, 2, 28)  $\wedge$ 
.11       DateFromInt (2094) (7) (18) = 86076  $\wedge$ 
.12       DateFromInt (2038) (11) (10) = 65737  $\wedge$ 
.13       DateFromInt (1996) (2) (25) = 50138  $\wedge$ 
.14       DateFromInt (1648) (6) (10) = - 76860  $\wedge$ 
.15       DateFromInt (1560) (2) (24) = - 109099  $\wedge$ 
.16       DateFromInt (1436) (1) (25) = - 154420  $\wedge$ 
.17       DateFromInt (1391) (6) (4) = - 170726  $\wedge$ 
.18       DateFromInt (1096) (5) (18) = - 278491  $\wedge$ 
.19       DateFromInt (1013) (4) (19) = - 308836  $\wedge$ 
.20       DateFromInt (694) (11) (7) = - 425149  $\wedge$ 
.21       DateFromInt (70) (9) (26) = - 653107  $\wedge$ 
.22       DateFromInt (- 168) (12) (8) = - 739963  $\wedge$ 
.23       DateFromInt (- 586) (7) (30) = - 892769);

```

0.5.2 MJD2JD, JD2MJD を検査する

修正ユリウス日開始日 (1858 年 11 月 17 日 0 時)、グレゴリオ暦切替前日正午 (1582 年 10 月 4 日 12 時)、グレゴリオ暦初日正午 (1582 年 10 月 15 日 12 時)、ユリウス日起算開始日 (紀元前 4713 年 1 月 1 日正午) を確認することで、MJD2JD 関数などをテストする。ユリウス日は 12 時が起算時刻なので、DateFromInt(-4712, 1, 1.5) がユリウス日 0 になる。

```

72.0  t2 : () → FTestDriver' TestCase
.1    t2 ()  $\triangle$ 
.2      mk-FTestDriver' TestCase
.3      (
.4        "FCalendarT02:\t MJD2JD, JD2MJD を検査する",
.5        DateFromInt (1582) (10) (4.5) = JD2MJD (2299160)  $\wedge$ 
.6        DateFromInt (1582) (10) (15.5) = JD2MJD (2299161)  $\wedge$ 
.7        MJD2JD (DateFromInt (- 4712) (1) (1.5)) = 0  $\wedge$ 
.8        FCalendar' JD2MJD (0) =
FCalendar' DateFromInt (- 4712) (1) (1.5)  $\wedge$ 
.9        DateFromInt (- 586) (7) (30) = JD2MJD (1507231.5)  $\wedge$ 
.10       DateFromInt (2094) (7) (18) = JD2MJD (2486076.5));

```

0.5.3 DayInYear を検査する

```
73.0 t3 : () → FTestDriverc TestCase
.1 t3 ()  $\triangle$ 
.2 mk-FTestDriverc TestCase
.3 (
.4   "FCalendarT03 : \t DayInYear を検査する",
.5   DayInYear (DateFromInt (2000) (1) (1)) = 1  $\wedge$ 
.6   DayInYear (DateFromInt (2000) (2) (1)) = 32  $\wedge$ 
.7   DayInYear (DateFromInt (2000) (3) (1)) = 61  $\wedge$ 
.8   DayInYear (DateFromInt (2000) (12) (31)) = 366  $\wedge$ 
.9   DayInYear (DateFromInt (2001) (3) (1)) = 60  $\wedge$ 
.10  DayInYear (DateFromInt (2001) (12) (31)) = 365);
```

0.5.4 日付の計算を検査する

```
74.0 t4 : () → FTestDriverc TestCase
.1 t4 ()  $\triangle$ 
.2 mk-FTestDriverc TestCase
.3 (
.4   "FCalendarT04 : \t 日付の計算を検査する",
.5   let d1 = DateFromInt (2000) (1) (1),
.6       d2 = DateFromInt (2000) (3) (1) in
.7   Diff (d2) (d1) = 60  $\wedge$ 
.8   d1 + 60 = d2  $\wedge$ 
.9   Int3FromDate (d1 + 60) = mk- (2000, 3, 1)  $\wedge$ 
.10  d2 - 60 = d1  $\wedge$ 
.11  DateFromInt (2004) (2) (28)      +      2      =
DateFromInt (2004) (3) (1)  $\wedge$ 
.12  DateFromInt (2004) (3) (1)      -      2      =
DateFromInt (2004) (2) (28)  $\wedge$ 
.13  DateFromInt (2094) (7) (18)      =
DateFromInt (- 586) (7) (30) + 214193 + 764652);
```

0.5.5 閏年の判定を検査する

```
75.0 t5 : () → FTestDriverc TestCase
.1 t5 ()  $\triangle$ 
.2 mk-FTestDriverc TestCase
.3 (
.4   "FCalendarT05 : \t 閏年の判定を検査する",
.5   IsLeapYear (1996)  $\wedge$ 
.6   IsLeapYear (2000)  $\wedge$ 
.7   IsLeapYear (2004)  $\wedge$ 
.8   IsLeapYear (1900) = false  $\wedge$ 
.9   IsLeapYear (2003) = false);
```

0.5.6 曜日に関する関数を検査する

```

76.0   $t6 : () \rightarrow FTestDriver\text{'Test}Case$ 
.1     $t6 () \triangle$ 
.2     $mk\text{-}FTestDriver\text{'Test}Case$ 
.3    (
.4      "FCalendarT06 : \t 曜日に関する関数を検査する",
.5       $GetDayOfWeek (DateFromInt (2004) (3) (1)) = 1 \wedge$ 
.6       $GetDayOfWeek (DateFromInt (2004) (3) (2)) = 2 \wedge$ 
.7       $GetDayOfWeek (DateFromInt (2004) (3) (3)) = 3 \wedge$ 
.8       $GetDayOfWeek (DateFromInt (2004) (3) (4)) = 4 \wedge$ 
.9       $GetDayOfWeek (DateFromInt (2004) (3) (5)) = 5 \wedge$ 
.10      $GetDayOfWeek (DateFromInt (2004) (3) (6)) = 6 \wedge$ 
.11      $GetDayOfWeek (DateFromInt (2004) (3) (7)) = 0 \wedge$ 
.12      $GetDayOfWeek (DateFromInt (2004) (3) (8)) = 1 \wedge$ 
.13      $GetDayOfWeekName (DateFromInt (2004) (3) (1)) =$ 
MON  $\wedge$ 
.14      $GetDayOfWeekName (DateFromInt (2004) (3) (2)) =$ 
TUE  $\wedge$ 
.15      $GetDayOfWeekName (DateFromInt (2004) (3) (3)) =$ 
WED  $\wedge$ 
.16      $GetDayOfWeekName (DateFromInt (2004) (3) (4)) =$ 
THU  $\wedge$ 
.17      $GetDayOfWeekName (DateFromInt (2004) (3) (5)) = \text{FRI} \wedge$ 
.18      $GetDayOfWeekName (DateFromInt (2004) (3) (6)) = \text{SAT} \wedge$ 
.19      $GetDayOfWeekName (DateFromInt (2004) (3) (7)) = \text{SUN} \wedge$ 
.20      $GetDayOfWeekName (DateFromInt (2004) (3) (8)) =$ 
MON  $\wedge$ 
.21      $GetDayOfWeekName (DateFromInt (- 586) (7) (30)) =$ 
SUN  $\wedge$ 
.22      $GetDayOfWeekName (DateFromInt (70) (9) (26)) = \text{WED} \wedge$ 
.23      $GetDayOfWeekName (DateFromInt (2094) (7) (18)) =$ 
SUN  $\wedge$ 
.24      $GetDayOfWeekFromName (\text{SUN}) = 0 \wedge$ 
.25      $GetDayOfWeekFromName (\text{MON}) = 1 \wedge$ 
.26      $GetDayOfWeekFromName (\text{TUE}) = 2 \wedge$ 
.27      $GetDayOfWeekFromName (\text{WED}) = 3 \wedge$ 
.28      $GetDayOfWeekFromName (\text{THU}) = 4 \wedge$ 
.29      $GetDayOfWeekFromName (\text{FRI}) = 5 \wedge$ 
.30      $GetDayOfWeekFromName (\text{SAT}) = 6$ );

```

0.5.7 日付の大小を検査する

```

77.0   $t7 : () \rightarrow FTestDriver' TestCase$ 
.1     $t7 () \triangleq$ 
.2       $mk\_FTestDriver' TestCase$ 
.3      (
.4        "FCalendarT07 : \t 日付の大小を検査する",
.5         $DateFromInt (2004) (3) (1) = DateFromInt (2004) (3) (1) \wedge$ 
.6         $DateFromInt (2004) (3) (1) \leq DateFromInt (2004) (3) (1) \wedge$ 
.7         $DateFromInt (2004) (3) (1) \geq DateFromInt (2004) (3) (1) \wedge$ 
.8         $DateFromInt (2004) (3) (1) > DateFromInt (2004) (2) (29) \wedge$ 
.9         $DateFromInt (2003) (12) (31) <$ 
 $DateFromInt (2004) (1) (1));$ 

```

0.5.8 指定曜日の日付獲得を検査する

```

78.0  t8 : () → FTestDriver' TestCase
.1    t8 ()  $\triangleq$ 
.2      mk-FTestDriver' TestCase
.3      (
.4        "FCalendarT08 : \t 指定曜日の日付獲得を検査する",
.5        FirstDayOfWeekOfMonth (SUN) (3) (2004) =
DateFromInt (2004) (3) (7)  $\wedge$ 
.6        FirstDayOfWeekOfMonth (MON) (3) (2004) =
DateFromInt (2004) (3) (1)  $\wedge$ 
.7        FirstDayOfWeekOfMonth (TUE) (3) (2004) =
DateFromInt (2004) (3) (2)  $\wedge$ 
.8        FirstDayOfWeekOfMonth (WED) (3) (2004) =
DateFromInt (2004) (3) (3)  $\wedge$ 
.9        FirstDayOfWeekOfMonth (THU) (3) (2004) =
DateFromInt (2004) (3) (4)  $\wedge$ 
.10       FirstDayOfWeekOfMonth (FRI) (3) (2004) =
DateFromInt (2004) (3) (5)  $\wedge$ 
.11       FirstDayOfWeekOfMonth (SAT) (3) (2004) =
DateFromInt (2004) (3) (6)  $\wedge$ 
.12       GetLastDayOfWeekOfMonth (SUN) (2) (2004) =
DateFromInt (2004) (2) (29)  $\wedge$ 
.13       GetLastDayOfWeekOfMonth (SUN) (3) (2004) =
DateFromInt (2004) (3) (28)  $\wedge$ 
.14       GetLastDayOfWeekOfMonth (MON) (3) (2004) =
DateFromInt (2004) (3) (29)  $\wedge$ 
.15       GetLastDayOfWeekOfMonth (TUE) (3) (2004) = DateFromInt (2004) (3) (30)  $\wedge$ 

.16       GetLastDayOfWeekOfMonth (WED) (3) (2004) = DateFromInt (2004) (3) (31)  $\wedge$ 

.17       GetLastDayOfWeekOfMonth (THU) (3) (2004) = DateFromInt (2004) (3) (25)  $\wedge$ 

.18       GetLastDayOfWeekOfMonth (FRI) (3) (2004) = DateFromInt (2004) (3) (26)  $\wedge$ 

.19       GetLastDayOfWeekOfMonth (SAT) (3) (2004) = DateFromInt (2004) (3) (27)  $\wedge$ 

.20       GetNthDayOfWeekOfMonth (SUN) (5) (2) (2004) = DateFromInt (2004) (2) (29)  $\wedge$ 

.21       GetNthDayOfWeekOfMonth (SUN) (6) (2) (2004) = nil  $\wedge$ 
.22       GetNthDayOfWeekOfMonth (SAT) (5) (2) (2004) = nil  $\wedge$ 
.23       GetNthDayOfWeekOfMonth (SUN) (2) (3) (2004) = DateFromInt (2004) (3) (14)  $\wedge$ 

.24       GetNthDayOfWeekOfMonth (SUN) (3) (3) (2004) = DateFromInt (2004) (3) (21)  $\wedge$ 

.25       GetNthDayOfWeekOfMonth (SUN) (4) (3) (2004) = GetLastDayOfWeekOfMonth (SUN) (3) (2004)  $\wedge$ 

.26       GetNthDayOfWeekOfMonth (MON) (2) (3) (2004) = DateFromInt (2004) (3) (8)  $\wedge$ 

.27       GetNthDayOfWeekOfMonth (TUE) (3) (3) (2004) = DateFromInt (2004) (3) (16)  $\wedge$ 

.28       GetNthDayOfWeekOfMonth (WED) (4) (3) (2004) = DateFromInt (2004) (3) (24)  $\wedge$ 

.29       GetNthDayOfWeekOfMonth (THU) (5) (1) (2004) = DateFromInt (2004) (1) (29)  $\wedge$ 

.30       GetNthDayOfWeekOfMonth (FRI) (5) (7) (2004) = DateFromInt (2004) (7) (30)  $\wedge$ 

.31       GetNthDayOfWeekOfMonth (SAT) (5) (5) (2004) = DateFromInt (2004) (5) (29));

```

0.5.9 月初日、月末日、曜日の照会を検査する

```

79.0  t9 : () → FTestDriver' TestCase
.1    t9 () △
.2    mk-FTestDriver' TestCase
.3    (
.4      "FCalendarT09:\t 月初日、月末日、曜日の照会を検査する
",
.5      GetFirstDayOfMonth (3) (2004)                                =
DateFromInt (2004) (3) (1) ∧
.6      GetFirstDayOfMonth (4) (2004)                                =
DateFromInt (2004) (4) (1) ∧
.7      GetLastDayOfMonth (2) (2004)                                  =
DateFromInt (2004) (2) (29) ∧
.8      GetLastDayOfMonth (8 + 6) (2003)                             =
DateFromInt (2004) (2) (29) ∧
.9      GetLastDayOfMonth (2) (2004)      +      1                  =
GetFirstDayOfMonth (3) (2004) ∧
.10     GetLastDayOfMonth (2) (2004)                                  =
GetFirstDayOfMonth (3) (2004) - 1 ∧
.11     GetLastDayOfMonth (2) (2003)                                  =
DateFromInt (2003) (2) (28) ∧
.12     GetLastDayOfMonth (2) (1900)                                  =
DateFromInt (1900) (2) (28) ∧
.13     IsSunday (DateFromInt (2004) (3) (14)) ∧
.14     IsSunday (DateFromInt (2004) (3) (15)) = false ∧
.15     IsSaturday (DateFromInt (2004) (3) (13)) ∧
.16     IsSaturday (DateFromInt (2004) (3) (14)) = false ∧
.17     IsWeekDay (DateFromInt (2004) (3) (13)) = false ∧
.18     IsWeekDay (DateFromInt (2004) (3) (14)) = false ∧
.19     IsWeekDay (DateFromInt (2004) (3) (1)) ∧
.20     IsWeekDay (DateFromInt (2004) (3) (2)) ∧
.21     IsWeekDay (DateFromInt (2004) (3) (3)) ∧
.22     IsWeekDay (DateFromInt (2004) (3) (4)) ∧
.23     IsWeekDay (DateFromInt (2004) (3) (5)) ∧
.24     IsDayOFWeekName WeekDay (MON) ∧
.25     IsDayOFWeekName WeekDay (TUE) ∧
.26     IsDayOFWeekName WeekDay (WED) ∧
.27     IsDayOFWeekName WeekDay (THU) ∧
.28     IsDayOFWeekName WeekDay (FRI) ∧
.29     IsDayOFWeekName WeekDay (SAT) = false ∧
.30     IsDayOFWeekName WeekDay (SUN) = false);

```

0.5.10 指定された曜日が何日あるかを検査する

```

80.0  t10 : () → FTestDriver' TestCase
.1    t10 () ≜
.2      mk-FTestDriver' TestCase
.3      (
.4        "FCalendarT10:\t 指定された曜日が何日あるかを検査する
",
.5        let f = DateFromInt in
.6        HowManyDayOfWeekWithin2Days (SUN) (f (2001) (7) (11)) (f (2001) (3) (1)) =
19 ∧
.7        HowManyDayOfWeekWithin2Days (SUN) (f (2001) (7) (11)) (f (1001) (3) (1)) =
52196 ∧
.8        HowManyDayOfWeekWithin2Days (SUN) (f (2004) (1) (1)) (f (2004) (3) (1)) =
9 ∧
.9        HowManyDayOfWeekWithin2Days (MON) (f (2004) (1) (1)) (f (2004) (3) (1)) =
9 ∧
.10       HowManyDayOfWeekWithin2Days (TUE) (f (2004) (1) (1)) (f (2004) (3) (1)) =
8 ∧
.11       HowManyDayOfWeekWithin2Days (WED) (f (2004) (1) (1)) (f (2004) (3) (1)) =
8 ∧
.12       HowManyDayOfWeekWithin2Days (THU) (f (2004) (1) (1)) (f (2004) (3) (1)) =
9 ∧
.13       HowManyDayOfWeekWithin2Days (FRI) (f (2004) (1) (1)) (f (2004) (3) (1)) =
9 ∧
.14       HowManyDayOfWeekWithin2Days (SAT) (f (2004) (1) (1)) (f (2004) (3) (1)) =
9 ∧
.15       HowManyDayOfWeekWithin2Days (SAT) (f (- 586) (7) (30)) (f (2094) (7) (18)) =
(214193 + 764652) div 7 ∧
.16       HowManyDayOfWeekWithin2Days (SUN) (f (- 586) (7) (30)) (f (2094) (7) (18)) =
(214193 + 764652) div 7 + 1);

```

0.5.11 春分・夏至・秋分・冬至を検査する

グリニッジ標準時の春分などを求め、それを日本標準時 (GMT+9) に変換して比較している。


```

81.0  t11 : () → FTestDriver `TestCase
.1    t11 () ≜
.2      mk-FTestDriver `TestCase
.3      (
.4        "FCalendarT11 : \t 春分・夏至・秋分・冬至を検査する",
.5        let f = DateFromInt,
.6        diff = 9 in
.7        GetDateInST (diff) (GetVernalEquinoxInGMT (2001)) =
f (2001) (3) (20) ∧
.8        GetDateInST (diff) (GetVernalEquinoxInGMT (2999)) =
f (2999) (3) (20) ∧
.9        GetDateInST (diff) (GetSummerSolsticeInGMT (2001)) =
f (2001) (6) (21) ∧
.10       GetDateInST (diff) (GetSummerSolsticeInGMT (2999)) =
f (2999) (6) (20) ∧
.11       GetDateInST (diff) (GetAutumnalEquinoxInGMT (2001)) =
f (2001) (9) (23) ∧
.12       GetDateInST (diff) (GetAutumnalEquinoxInGMT (2999)) =
f (2999) (9) (22) ∧
.13       GetDateInST (diff) (GetWinterSolsticeInGMT (2001)) =
f (2001) (12) (22) ∧
.14       GetDateInST (diff) (GetWinterSolsticeInGMT (2999)) =
f (2999) (12) (22));

```