VDM++ ライブラリのドキュメント雛形

佐原　伸

目次

1　はじめに

本ドキュメントは、VDM++ ライブラリ・ドキュメントの雛形であり、まだ、すべての VDM モジュールを記述しているわけではない。

また、使用している回帰テストライブラリーは、最新の VDMUnit ではなく、古いものを使用している。

2　関数型ライブラリのドキュメント

2.1 AllT

2.1.1 責任

回帰テスト全テストケースのテストを行う。

```
class
AllT
operations
public static
 run : () ==> bool
 run () ==
   let テスト結果列 =
           [
            new SequenceT ().run (),
            new SetT ().run (),
            new TermT ().run (),
            new TimeT ().run (),
            new MapT ().run (),
            new HashtableT ().run (),
            new DoubleListQueueT ().run (),
            new QueueT ().run (),
            new 発番者 T ().run (),
            new RealT ().run (),
            new SetT ().run (),
            new NumberT ().run (),
            new IntegerT ().run (),
            new CalendarT ().run (),
            new SBCalendarT ().run (),
            new DateT ().run (),
            new FunctionT ().run (),
            new ProductT ().run (),
            new StringT ().run ()],
       メッセージ = "全テストの結果" in
   if  forall i in set inds テスト結果列 & テスト結果列 (i)
   then return new TestLogger ().全体が成功した ( メッセージ)
   else return new TestLogger ().全体が失敗した ( メッセージ)
end
AllT
```

2.2 CalendarDefinition

2.2.1 責任

暦に関する共通定義を行う。

```
class
CalendarDefinition
values
public
  homedir = "."
types
  public 曜日名 = <月> | <火> | <水> | <木> | <金> | <土> | <日>;
  public 曜日数 = nat
  inv 曜日 == 曜日 <= 6
end
CalendarDefinition
```

2.3 Calendar

2.3.1 責任

グレゴリオ暦を表す暦である。

2.3.2 概要

- グレゴリオ歴の処理を行い、2099 年までの春分・秋分を自動計算する。
- 日付を表す Date クラスと協調して、暦の処理を行う。
- 同一日付のオブジェクトは複数個あり得る
- 休日の集合はサブクラスで提供されることを期待している。
- 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、本クラスのサブクラスとして各国別・標準時別の暦クラスを作成し、インスタンス変数「グリニッジ標準時との差」を設定しなければならない。

```
class
Calendar is subclass of CalendarDefinition
values
private
  修正ユリウス日日数差 = 2400000.5;
private
  曜日名列 = [<日>, <月>, <火>, <水>, <木>, <金>, <土>];
```

```
private
  年の実日数 = 365.25;
protected
  年の月数 = 12;
private
  補正月数 = 14;
private
  週の日数 = 7;
private
  月平均日数 = 30.6001;
private
  一世紀の年数 = 100;
private
  日付計算係数 = 122.1;
private
  年計算係数 = 4800;
private
  世紀計算係数 = 32044.9;
private
  グレゴリオ暦切替前日 = 2299160;
private
  グレゴリオ暦初日 = 1582.78;
  io = new IO()
instance variables
  protected グリニッジ標準時との差 : real := 0;
  protected 今日という日 : [Date] := nil;
  protected 休日集合写像 : map int to set of Date := { |-> };

operations
public
  暦を得る : () ==> Calendar
  暦を得る () ==
    return self
functions
public
  小さい : Date * Date -> bool
  小さい ( 日付 1, 日付 2) ==
    日付 1.修正ユリウス日獲得 () < 日付 2.修正ユリウス日獲得 ();
public
```

```
  大きい : Date * Date -> bool
  大きい ( 日付 1, 日付 2) ==
    日付 1.修正ユリウス日獲得 () > 日付 2.修正ユリウス日獲得 ();
public
  以下 : Date * Date -> bool
  以下 ( 日付 1, 日付 2) ==
    not 大きい ( 日付 1, 日付 2);
public
  以上 : Date * Date -> bool
  以上 ( 日付 1, 日付 2) ==
    not 小さい ( 日付 1, 日付 2);
public
  等しい : Date * Date -> bool
  等しい ( 日付 1, 日付 2) ==
    日付 1.修正ユリウス日獲得 () = 日付 2.修正ユリウス日獲得 ();
public
  min : Date -> Date -> Date
  min ( 日付 1)( 日付 2) ==
    if 日付 1.小さい ( 日付 2)
    then 日付 1
    else 日付 2;
public
  max : Date -> Date -> Date
  max ( 日付 1)( 日付 2) ==
    if 日付 1.大きい ( 日付 2)
    then 日付 1
    else 日付 2
operations
public
  日付文字列か？ : seq of char ==>
                   bool
  日付文字列か？ ( 年月日) ==
    if 文字列から日付を得る ( 年月日) = false
    then return false
    else return true
functions
public
```

```
  閏年か？ : int ->
             bool
  閏年か？(年) ==
    年 mod 400 = 0 or (年 mod 一世紀の年数 <> 0 and 年 mod 4 = 0);
public
  曜日数を得る : Date -> 曜日数
  曜日数を得る(日付) ==
    let 修正ユリウス日 = floor (日付.修正ユリウス日獲得()) in
    (修正ユリウス日 - 4) mod 週の日数;
public
  年月日の整数組を得る : Date -> int * int * int
  年月日の整数組を得る(日付) ==
    mk_(年(日付),月(日付),日(日付));
public
  曜日名を得る : Date -> 曜日名
  曜日名を得る(日付) ==
    曜日名列(曜日数を得る(日付) + 1);
public
  曜日名から曜日数を求める : 曜日名 -> 曜日数
  曜日名から曜日数を求める(a 曜日名) ==
    Sequence`Index[曜日名](a 曜日名)(曜日名列) - 1
operations
public
  月初指定曜日を得る : int * int * 曜日名 ==> Date
  月初指定曜日を得る(年,月,a 曜日名) ==
    let w 曜日数 = 曜日名から曜日数を求める(a 曜日名),
        月初日 = 月初日を求める(年,月),
        差 = w 曜日数 - 曜日数を得る(月初日) in
     cases true:
      (差 > 0) +> return 月初日.+(差),
      (差 < 0) +> return 月初日.+((週の日数 + 差) mod 週の日数),
      others +> return 月初日
    end;
public
  月末指定曜日を得る : int * int * 曜日名 ==> Date
  月末指定曜日を得る(年,月,a 曜日名) ==
    return 月初指定曜日を得る(年,(月 + 1),a 曜日名).-(週の日数);
public
```

```
  第n指定曜日を得る : int * int * int * 曜日名 ==>
                      Date | bool
  第n指定曜日を得る(指定年,指定月,第n曜日,a曜日名) ==
    let 月初日 = 月初指定曜日を得る(指定年,指定月,a曜日名),
        結果 = 月初日.+(週の日数 * (第n曜日 - 1)) in
     cases 月(結果):
      (指定月) +> return 結果,
      others +> return false
    end;
public
  月初日を求める : int * int ==> Date
  月初日を求める(年,月) ==
    return 正則日付を得る(年,月,1);
public
  月末日を求める : int * int ==> Date
  月末日を求める(年,月) ==
    return 正則日付を得る(年,月 + 1,1).-(1)
functions
public
  日曜日か? : Date -> bool
  日曜日か?(日付) ==
    曜日数を得る(日付) = 0;
public
  土曜日か? : Date -> bool
  土曜日か?(日付) ==
    曜日数を得る(日付) = 6;
public
  ウィークデイか? : Date -> bool
  ウィークデイか?(日付) ==
    曜日数を得る(日付) in set {1,...,5}
operations
public
  平日か? : Date ==> bool
  平日か?(日付) ==
    return not お休みか?(日付)
functions
public
  曜日は平日か? : 曜日名 -> bool
  曜日は平日か?(a曜日名) ==
    a曜日名 not in set {<土>,<日>};
```

2.3.3 指定曜日が何日あるか得る

「指定曜日が何日あるか得る」は、指定された曜日名が、指定された日付間（日付 1 と日付 2 の間）に何日あるかを返す。日付 1 と日付 2 が指定された曜日であれば勘定に入れる。

以下は、指定曜日が何日あるか得る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。

VDM で後件を表すと、以下のようになる。

```
public
  指定曜日が何日あるか得る_陰仕様 : Date * Date * 曜日名 -> int
  指定曜日が何日あるか得る_陰仕様 ( 日付 1, 日付 2, a 曜日名 ) ==
    is not yet specified
  post let 開始日 = min ( 日付 1) ( 日付 2),
           終了日 = max ( 日付 1) ( 日付 2) in
       let 指定曜日の集合 = { 日 | 日 : Date & a 曜日名 = 曜日名を得る ( 日) and 開始日.以上 ( 日
) and 日.以下 ( 終了日)} in
       RESULT = card 指定曜日の集合 and
        forall 日 f, 日 t in set 指定曜日の集合 &
            日 f.以下 ( 日 t) => 日付の差を求める ( 日 t, 日 f) mod 7 = 0 and
             exists1 日 i in set 指定曜日の集合 &
                 日付の差を求める ( 日 i, 開始日) < 7 and
                 exists1 日 j in set 指定曜日の集合 &
                     日付の差を求める ( 終了日, 日 j) < 7 and
                     日付の差を求める ( 日 j, 日 i) = 7 * (RESULT - 1) ;
```

前件は以下である。ここで、VDM はサブタイプが記述できないので、VDM の親戚と言える形式仕様記述言語 RAISE-SL の記法を拝借した。

$$\texttt{type R} = \{|\texttt{rng}[\texttt{n} \to \texttt{n}/7|\texttt{n} \in \texttt{Int}]|\}^{*1}$$

$$\texttt{f}, \texttt{t} \in \texttt{Int}, \texttt{w} \in \texttt{R}, 0 \leq \texttt{f} \leq \texttt{t}, \texttt{h} : \texttt{Int} \to \texttt{R}^{*2}$$

より抽象化した後件は以下のようになる。ここでは、日付 1 と日付 2 を、抽象化のため f, t として記述した。VDM 記法風に記述しているが、VDM で逆関数は記述できないので、VDM 風数学記述である。

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

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う問題に抽象化された。

「指定曜日が何日あるか得る」は、**7 で割る**特殊な場合の実装であるということになる。ただし、以下の説明では、分りやすさのため曜日問題の用語を使っている。

*1 7 で割った商の集合

*2 （環準同型（ring homomorphism）

以下に、上記の後件から、段階的にプログラム（この場合仕様でもある）に洗練していく。

```
I = {f,...,t}
d = card I --集合 I の要素数
q = d div 7 --整数除算
r = d mod 7 --7 で割った余り
```

とすると、答え A に対して q ≤ A ≤ 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。
- card(I) = 7 * q + r(0 ≤ 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) mod 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（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 ∈ T ⇔ (w minus h(f)) + 1 以下 r
```

である。なぜならば

```
w ∈ T ⇔ {0,...,(r ⊥ 1)}∋ w = w minus h(f)
⇔ r ⊥ 1 ≧ w
⇔ r ≧ (w minus h(f)) + 1
```

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 div 7
    r ≡ d mod 7
```

```
    delta ≡ if (w minus h(f)) + 1 以下 r then 1 else 0 end
    x minus y ≡ if x ≧ y then x - y else 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
  指定曜日が何日あるか得る : Date * Date * 曜日名 -> int
  指定曜日が何日あるか得る (日付 1, 日付 2, a 曜日名) ==
    let w 曜日数 = 曜日名から曜日数を求める (a 曜日名),
        開始日 = min (日付 1) (日付 2),
        終了日 = max (日付 1) (日付 2),
        日数 = 日付の差を求める (終了日, 開始日) + 1,
        商 = 日数 div 週の日数,
        剰余 = 日数 mod 週の日数,
        delta = if 曜日数の減算 (w 曜日数, 曜日数を得る (開始日)) + 1 <= 剰余
                then 1
                else 0 in
    商 + delta;
private
  曜日数の減算 : int * int -> int
  曜日数の減算 (x, y) ==
    if x >= y
    then x - y
    else x - y + 週の日数;
public
  年 : Date -> int
  年 (日付) ==
    if 月補助関数 (日付) < 補正月数
    then 年補助関数 (日付) - 年計算係数
    else 年補助関数 (日付) - 年計算係数 + 1;
public
  月 : Date -> int
  月 (日付) ==
    if 月補助関数 (日付) < 補正月数
    then 月補助関数 (日付) - 1
    else 月補助関数 (日付) - 13;
public
```

```
  日 : Date -> int
  日 ( 日付 ) ==
    月始からの日数 ( 日付)
operations
public
  年始からの日数 : Date ==> int
  年始からの日数 ( 日付 ) ==
    let 年初日 = 整数三つ組から日付を得る (年 ( 日付), 1, 0) in
    return 日付の差を求める ( 日付, 年初日)
functions
  月始からの日数 : Date -> int
  月始からの日数 ( 日付 ) ==
    floor (月始からの日数を実数で求める ( 日付));
  月始からの日数を実数で求める : Date -> real
  月始からの日数を実数で求める ( 日付 ) ==
    年月日変更の補助関数 ( 日付 ) + 日付計算係数 -
    floor ( 年の実日数 * 年補助関数 ( 日付)) - floor ( 月平均日数 * 月補助関数 ( 日付));
  月補助関数 : Date ->
               int
  月補助関数 ( 日付 ) ==
    floor ((年月日変更の補助関数 ( 日付 ) + 日付計算係数 - floor ( 年の実日数 * 年補助関数 ( 日
付))) / 月平均日数);
  年月日変更の補助関数 : Date -> real
  年月日変更の補助関数 ( 日付 ) ==
    let ユリウス日 = ユリウス日に変換する (日付.修正ユリウス日獲得 ()),
        世紀 = floor (( ユリウス日 + 世紀計算係数 ) / 36524.25) in
    if ユリウス日 > グレゴリオ暦切替前日
    then ユリウス日 + 世紀計算係数 + 世紀 - 世紀 div 4 + 0.5
    else ユリウス日 + 32082.9 + 0.5;
  年補助関数 : Date ->
               int
  年補助関数 ( 日付 ) ==
    floor (年月日変更の補助関数 ( 日付 ) / 年の実日数)
operations
public
```

```
  グリニッジ標準時の春分を得る : int ==> Date
  グリニッジ標準時の春分を得る ( 年) ==
    let y = 年 / 1000 in
    return 修正ユリウス日から日付を得る
                (
                 修正ユリウス日に変換する (1721139.2855 + 365.242138 * 年 + y * y * (0.067919 - 0.002788 * y
public
  グリニッジ標準時の夏至を得る : int ==> Date
  グリニッジ標準時の夏至を得る ( 年) ==
    let y = 年 / 1000 in
    return 修正ユリウス日から日付を得る
                (
                 修正ユリウス日に変換する (1721233.2486 + 365.241728 * 年 - y * y * (0.053018 - 0.009332 * y
public
  グリニッジ標準時の秋分を得る : int ==> Date
  グリニッジ標準時の秋分を得る ( 年) ==
    let y = 年 / 1000 in
    return 修正ユリウス日から日付を得る
                (
                 修正ユリウス日に変換する (1721325.6978 + 365.242505 * 年 - y * y * (0.126689 - 0.00194 * y)
public
  グリニッジ標準時の冬至を得る : int ==> Date
  グリニッジ標準時の冬至を得る ( 年) ==
    let y = 年 / 1000 in
    return 修正ユリウス日から日付を得る
                (
                 修正ユリウス日に変換する (1721414.392 + 365.24289 * 年 - y * y * (0.010965 - 0.008485 * y))
public
  春分を得る : int ==> Date
  春分を得る ( 年) ==
    return 標準時基準の日付を得る (グリニッジ標準時の春分を得る ( 年));
public
  夏至を得る : int ==> Date
  夏至を得る ( 年) ==
    return 標準時基準の日付を得る (グリニッジ標準時の夏至を得る ( 年));
public
  秋分を得る : int ==> Date
  秋分を得る ( 年) ==
    return 標準時基準の日付を得る (グリニッジ標準時の秋分を得る ( 年));
public
```

```
  冬至を得る : int ==> Date
  冬至を得る ( 年) ==
    return 標準時基準の日付を得る (グリニッジ標準時の冬至を得る ( 年))
functions
public
  日付を加算する : Date * int -> Date
  日付を加算する ( 日付, 加算日数) ==
    日付.+ ( 加算日数);
public
  日付の差を求める : Date * Date -> int
  日付の差を求める ( 日付 1, 日付 2) ==
    floor (日付 1.修正ユリウス日獲得 () - 日付 2.修正ユリウス日獲得 ());
public
  日付を減算する : Date * int -> Date
  日付を減算する ( 日付, 減算日数) ==
    日付.− ( 減算日数);
public
  ユリウス日に変換する : real -> real
  ユリウス日に変換する ( 修正ユリウス日) ==
    修正ユリウス日 + 修正ユリウス日日数差;
public
  修正ユリウス日に変換する : real -> real
  修正ユリウス日に変換する ( ユリウス日) ==
    ユリウス日 - 修正ユリウス日日数差
operations
public
  正則日付を得る : int * int * int ==> Date
  正則日付を得る ( 年候補, 月候補, 日候補) ==
    let mk_ ( 年, 月) = 正則月を得る ( 年候補, 月候補) in
    return 整数三つ組から日付を得る ( 年, 月, 日候補)
functions
public
```

```
  正則月を得る : int * int -> int * int
  正則月を得る ( 年候補, 月候補 ) ==
    let 年 =
            if 月候補 <= 0
            then 年候補 + ( 月候補 - 12) div 年の月数
            else 年候補 + ( 月候補 - 1) div 年の月数,
        月候補 2 = 月候補 mod 年の月数,
        月 =
            if 月候補 2 = 0
            then 12
            else 月候補 2 in
    mk_( 年, 月);
public
  日付を年に変換する : int * int * int ->
                       real
  日付を年に変換する ( 年, 月, 日 ) ==
    年 + ( 月 - 1) / 年の月数 + ( 日 - 1) / 年の実日数;
public
  日付を文字列に変換する : Date +> seq of char
  日付を文字列に変換する ( 日付 ) ==
    日付.日付文字列を得る ()
operations
public
  日付文字列から日付を得る : seq of char ==> [Date]
  日付文字列から日付を得る ( 日付文字列 ) ==
    let 日付 = 文字列から日付を得る ( 日付文字列 ) in
    if 日付 = false
    then return nil
    else return 日付;
public
  日付間の休日集合を得る : Date * Date ==> set of Date
  日付間の休日集合を得る ( 日付 1, 日付 2 ) ==
    let 日 1 = min ( 日付 1) ( 日付 2),
        日 2 = max ( 日付 1) ( 日付 2),
        年集合 = {年 ( 日 1), . . . , 年 ( 日 2)},
        休日集合 = dunion {休日集合を得る ( 年 ) | 年 in set 年集合 } in
    return { 休日 | 休日 in set 休日集合 & 日付 1.以下 ( 休日 ) and 休日.以下 ( 日付 2)};
public
```

```
  日曜日を除く休日数を得る : Date * Date ==> int
  日曜日を除く休日数を得る ( 日付 1, 日付 2) ==
    return card (日付間の休日集合を得る ( 日付 1, 日付 2));
public
  休日数を得る : Date * Date ==> int
  休日数を得る ( 日付 1, 日付 2) ==
    let 日 1 = min ( 日付 1) ( 日付 2),
        日 2 = max ( 日付 1) ( 日付 2),
        日曜日数 = 指定曜日が何日あるか得る ( 日 1, 日 2, <日>) in
    return 日曜日数 + card 日曜日を除く休日集合を得る ( 日 1, 日 2);
public
  開始日を含まない休日数を得る : Date * Date ==> int
  開始日を含まない休日数を得る ( 日付 1, 日付 2) ==
    let 日 1 = min ( 日付 1) ( 日付 2),
        日 2 = max ( 日付 1) ( 日付 2) in
    return 休日数を得る (日 1.+ (1), 日 2);
private
  日曜日を除く休日集合を得る : Date * Date ==> set of Date
  日曜日を除く休日集合を得る ( 日付 1, 日付 2) ==
    let 休日集合 = 日付間の休日集合を得る ( 日付 1, 日付 2) in
    return { 休日 | 休日 in set 休日集合 & not 日曜日か？ ( 休日)};
public
  日曜日である休日の集合を得る : Date * Date ==> set of Date
  日曜日である休日の集合を得る ( 日付 1, 日付 2) ==
    let 休日集合 = 日付間の休日集合を得る ( 日付 1, 日付 2) in
    return { 休日 | 休日 in set 休日集合 & 日曜日か？ ( 休日)};
public
  未来の平日を得る : Date ==> Date
  未来の平日を得る ( 日付) ==
     cases お休みか？ ( 日付) or 土曜日か？ ( 日付):
      true +> return 未来の平日を得る (日付.+ (1)),
      others +> return 日付
    end;
public
  過去の平日を得る : Date ==> Date
  過去の平日を得る ( 日付) ==
     cases お休みか？ ( 日付) or 土曜日か？ ( 日付):
      true +> return 過去の平日を得る (日付.- (1)),
      others +> return 日付
    end;
```

```
public
  平日を加算する : Date * int ==> Date
  平日を加算する ( 日付, 加算日数) ==
    return 平日を加算する補助関数 (未来の平日を得る ( 日付), 加算日数);
public
  平日を加算する補助関数 : Date * int ==> Date
  平日を加算する補助関数 ( 日付, 加算日数) ==
     cases お休みか？ ( 日付) or 土曜日か？ ( 日付):
      true +> return 平日を加算する補助関数 (日付.＋ (1), 加算日数),
      others +> if 加算日数 <= 0
                then return 日付
                else return 平日を加算する補助関数 (日付.＋ (1), 加算日数 - 1)
    end;
public
  平日を減算する : Date * int ==> Date
  平日を減算する ( 日付, 減算日数) ==
    return 平日を減算する補助関数 (過去の平日を得る ( 日付), 減算日数);
public
  平日を減算する補助関数 : Date * int ==> Date
  平日を減算する補助関数 ( 日付, 減算日数) ==
     cases お休みか？ ( 日付) or 土曜日か？ ( 日付):
      true +> return 平日を減算する補助関数 (日付.－ (1), 減算日数),
      others +> if 減算日数 <= 0
                then return 日付
                else return 平日を減算する補助関数 (日付.－ (1), 減算日数 - 1)
    end;
public
  休日か？ : Date ==> bool
  休日か？ ( 日付) ==
    let 休日集合 = {日.修正ユリウス日獲得 () | 日 in set 休日集合を得る (日付.年 ())} in
    return 日付.修正ユリウス日獲得 () in set 休日集合;
public
  お休みか？ : Date ==> bool
  お休みか？ ( 日付) ==
    return 日曜日か？ ( 日付) or 休日か？ ( 日付)
functions
public
```

```
  日付集合に含まれる : Date * set of Date -> bool
  日付集合に含まれる ( 日, a 国民の休日) ==
    (let 国民の休日の修正ユリウス日集合 = {floor d.修正ユリウス日獲得 () | d in set a 国民の休
日 } in
     日.修正ユリウス日獲得 () in set 国民の休日の修正ユリウス日集合)
operations
public
  修正ユリウス日から日付を得る : real ==> Date
  修正ユリウス日から日付を得る ( 修正ユリウス日) ==
    return new Date (self, 修正ユリウス日);
public
  整数三つ組から日付を得る : int * int * int ==> Date
  整数三つ組から日付を得る ( 年, 月, 日) ==
    let [year, month] = if ( 月 > 補正月数 - 年の月数)
                        then [年 + 年計算係数, 月 + 1]
                        else [年 + 年計算係数 - 1, 月 + 補正月数 - 1],
        世紀 = year div 一世紀の年数,
        世紀係数 = if (日付を年に変換する ( 年, 月, 日) > グレゴリオ暦初日)
                   then 世紀 div 4 - 世紀 - 32167
                   else -32205,
        半日 = 0.5 in
    return 修正ユリウス日から日付を得る
                (
                 floor ( 年の実日数 * year) +
                 floor ( 月平均日数 * month) +
                 日 + 世紀係数 - 半日 - 修正ユリウス日日数差);
public
  文字列から日付を得る : seq of char ==>
                         Date | bool
  文字列から日付を得る ( 年月日) ==
    (   if not String`isDigits ( 年月日)
        then return false ;
        let 整数年月日 = String`asInteger ( 年月日),
            年 = 整数年月日 div 10000,
            整数月日 = 整数年月日 mod 10000,
            月 = 整数月日 div 100,
            日 = 整数月日 mod 100 in
        if 整数三つ組から日付を得る ( 年, 月, 日).日付文字列を得る () = 年月日
        then return 整数三つ組から日付を得る ( 年, 月, 日)
```

```
        else return false
    );
public
  標準時基準の日付を得る : Date ==> Date
  標準時基準の日付を得る ( 日付) ==
    return 修正ユリウス日から日付を得る (日付.修正ユリウス日獲得 () + 日付.暦 ().グリニッジ標準時との差を得る ());
public
  ある年の指定曜日集合を得る : int * 曜日名 ==> set of Date
  ある年の指定曜日集合を得る ( 年, 曜日) ==
    (   dcl ある曜日の集合 : set of Date := {},
            日 : Date := self.第 n 指定曜日を得る ( 年, 1, 1, 曜日);
        while 日.以下 (self.月末指定曜日を得る ( 年, 12, 曜日))
        do (   ある曜日の集合 := ある曜日の集合 union { 日 };
               日 := 日.+ (7)
           ) ;
        return ある曜日の集合
    );
public
  グリニッジ標準時との差を得る : () ==> real
  グリニッジ標準時との差を得る () ==
    return グリニッジ標準時との差;
public
  グリニッジ標準時との差を設定する : (real) ==> ()
  グリニッジ標準時との差を設定する ( 差) ==
    グリニッジ標準時との差 := 差;
public
  休日集合を設定する : int ==> ()
  休日集合を設定する (-) ==
    is subclass responsibility;
public
  休日集合を得る : int ==> set of Date
  休日集合を得る ( 年) ==
    (   if not 年 in set dom 休日集合写像
        then self.休日集合を設定する ( 年) ;
        return self.休日集合写像 ( 年)
    );
public
```

```
  今日を読み込む : seq of char ==> [Date]
  今日を読み込む ( ファイル名 ) ==
    let mk_ ( 結果,mk_ (y,m,d)) = io.freadval[int * int * int] ( ファイル名 ) in
    if 結果
    then return 整数三つ組から日付を得る (y,m,d)
    else let - = io.echo ("Can't read today's data file.") in
         return nil;
public
  今日 : () ==> Date
  今日 () ==
    if 今日という日 = nil
    then return 今日を読み込む (homedir^"/temp/Today.txt")
    else return 今日という日;
public
  ファイルから読み込む今日 : seq of char ==> Date
  ファイルから読み込む今日 ( ファイル名 ) ==
    if 今日という日 = nil
    then return 今日を読み込む ( ファイル名 )
    else return 今日という日;
public
  今日を設定する : Date ==> ()
  今日を設定する ( 日付 ) ==
    今日という日 := 日付
end
Calendar
```

Test Suite : vdm.tc

Class : Calendar

Name	#Calls	Coverage
Calendar‘年	948	√
Calendar‘日	479	√
Calendar‘月	528	√
Calendar‘今日	8	√
Calendar‘以上	4	√
Calendar‘以下	4	√
Calendar‘max	21	√
Calendar‘min	21	√
Calendar‘大きい	6	√
Calendar‘小さい	6	√

Name	#Calls	Coverage
Calendar‘等しい	2	√
Calendar‘休日か？	202	√
Calendar‘平日か？	1	√
Calendar‘暦を得る	0	0%
Calendar‘閏年か？	5	√
Calendar‘お休みか？	220	√
Calendar‘冬至を得る	5	√
Calendar‘土曜日か？	94	√
Calendar‘夏至を得る	2	√
Calendar‘年補助関数	3910	√
Calendar‘日曜日か？	504	√
Calendar‘春分を得る	18	√
Calendar‘月補助関数	2483	√
Calendar‘秋分を得る	19	√
Calendar‘休日数を得る	3	√
Calendar‘曜日名を得る	3	√
Calendar‘曜日数の減算	8	√
Calendar‘曜日数を得る	946	√
Calendar‘正則月を得る	534	√
Calendar‘今日を設定する	4	√
Calendar‘今日を読み込む	7	75%
Calendar‘休日集合を得る	222	√
Calendar‘平日を加算する	2	√
Calendar‘平日を減算する	3	√
Calendar‘年始からの日数	2	√
Calendar‘日付を加算する	1	√
Calendar‘日付を減算する	1	√
Calendar‘日付文字列か？	3	√
Calendar‘曜日は平日か？	7	√
Calendar‘月初日を求める	335	√
Calendar‘月始からの日数	479	√
Calendar‘月末日を求める	91	√
Calendar‘正則日付を得る	444	√
Calendar‘ウィークデイか？	1	√
Calendar‘日付の差を求める	11	√
Calendar‘未来の平日を得る	8	94%
Calendar‘過去の平日を得る	134	94%

Name	#Calls	Coverage
Calendar‘休日集合を設定する	0	0%
Calendar‘日付を年に変換する	1008	√
Calendar‘日付集合に含まれる	223	√
Calendar‘月初指定曜日を得る	335	√
Calendar‘月末指定曜日を得る	270	√
Calendar‘第 n 指定曜日を得る	61	√
Calendar‘ユリウス日に変換する	6872	√
Calendar‘年月日の整数組を得る	205	√
Calendar‘年月日変更の補助関数	6872	√
Calendar‘文字列から日付を得る	36	√
Calendar‘平日を加算する補助関数	13	96%
Calendar‘平日を減算する補助関数	13	96%
Calendar‘日付を文字列に変換する	3	√
Calendar‘日付間の休日集合を得る	9	√
Calendar‘標準時基準の日付を得る	44	√
Calendar‘ファイルから読み込む今日	3	80%
Calendar‘修正ユリウス日に変換する	46	√
Calendar‘指定曜日が何日あるか得る	8	√
Calendar‘整数三つ組から日付を得る	1008	√
Calendar‘日付文字列から日付を得る	2	√
Calendar‘日曜日を除く休日数を得る	2	√
Calendar‘曜日名から曜日数を求める	348	√
Calendar‘ある年の指定曜日集合を得る	5	√
Calendar‘日曜日を除く休日集合を得る	3	√
Calendar‘グリニッジ標準時との差を得る	44	√
Calendar‘グリニッジ標準時の冬至を得る	5	√
Calendar‘グリニッジ標準時の夏至を得る	2	√
Calendar‘グリニッジ標準時の春分を得る	18	√
Calendar‘グリニッジ標準時の秋分を得る	19	√
Calendar‘修正ユリウス日から日付を得る	2751	√
Calendar‘日曜日である休日の集合を得る	4	√
Calendar‘月始からの日数を実数で求める	479	√
Calendar‘開始日を含まない休日数を得る	1	√
Calendar‘グリニッジ標準時との差を設定する	46	√
Calendar‘指定曜日が何日あるか得る_陰仕様	0	0%
Total Coverage		**92%**

2.4 Character

2.4.1 責任

文字共通の振る舞いを表す。

2.4.2 概要

すべての文字に共通な機能を定義する。

- 数字を表す文字の整数への変換を行う。
- 文字の大小を判定する。

```
.....................................................................................
class
Character
values
  v 数字順序写像 = {'0' |-> 1,'1' |-> 2,'2' |-> 3,'3' |-> 4,'4' |-> 5,'5' |-> 6,'6' |-> 7,'7' |-> 8,'8' |-> 9,'9'
  v 英大文字順序写像 = {'A' |-> 12,'B' |-> 14,'C' |-> 16,'D' |-> 18,'E' |-> 20,'F' |-> 22,'G' |-> 24,'H' |-> 26,'
                        'J' |-> 30,'K' |-> 32,'L' |-> 34,'M' |-> 36,'N' |-> 38,'O' |-> 40,'P' |-> 42,'Q' |-> 44,'
                        'S' |-> 48,'T' |-> 50,'U' |-> 52,'V' |-> 54,'W' |-> 56,'X' |-> 58,'Y' |-> 60,'Z' |-> 62};
  v 小大文字順序写像 = {'a' |-> 11,'b' |-> 13,'c' |-> 15,'d' |-> 17,'e' |-> 19,'f' |-> 21,'g' |-> 23,'h' |-> 25,'
                        'j' |-> 29,'k' |-> 31,'l' |-> 33,'m' |-> 35,'n' |-> 37,'o' |-> 39,'p' |-> 41,'q' |-> 43,'
                        's' |-> 47,'t' |-> 49,'u' |-> 51,'v' |-> 53,'w' |-> 55,'x' |-> 57,'y' |-> 59,'z' |-> 61};
  v ひらがな順序写像 = {'あ' |-> 63,'い' |-> 64,'う' |-> 65,'え' |-> 66,'お' |-> 67,
                        'か' |-> 68,'き' |-> 69,'く' |-> 70,'け' |-> 71,'こ' |-> 72,
                        'さ' |-> 73,'し' |-> 74,'す' |-> 75,'せ' |-> 76,'そ' |-> 77,
                        'た' |-> 78,'ち' |-> 79,'つ' |-> 80,'て' |-> 81,'と' |-> 82,
                        'な' |-> 83,'に' |-> 84,'ぬ' |-> 85,'ね' |-> 86,'の' |-> 87,
                        'は' |-> 88,'ひ' |-> 89,'ふ' |-> 90,'へ' |-> 91,'ほ' |-> 92,
                        'ま' |-> 93,'み' |-> 94,'む' |-> 95,'め' |-> 96,'も' |-> 97,
                        'や' |-> 98,'ゆ' |-> 99,'よ' |-> 100,
                        'ら' |-> 101,'り' |-> 102,'る' |-> 103,'れ' |-> 104,'ろ' |-> 105,
                        'わ' |-> 106,'ん' |-> 107};
  v 文字順序写像 = v 数字順序写像 munion v 英大文字順序写像 munion v 小大文字順序写像 munion v ひ
らがな順序写像
functions
public static
```

```
  asDigit : char -> int | bool
  asDigit(c) ==
    if isDigit(c)
    then let s = [c] in
         let mk_(-,i) = VDMUtil`seq_of_char2val[int](s) in
         i
    else false;
public static
  as 辞書順序 : char -> int
  as 辞書順序(c) ==
    if c in set dom v 文字順序写像
    then v 文字順序写像(c)
    else 999999;
public static
  isDigit : char -> bool
  isDigit(c) ==
    c in set dom v 数字順序写像;
public static
  isLetter : char -> bool
  isLetter(c) ==
    c in set dom (v 英大文字順序写像 munion v 小大文字順序写像);
public static
  isLetterOrDigit : char -> bool
  isLetterOrDigit(c) ==
    isDigit(c) or isLetter(c);
public static
  isCapitalLetter : char -> bool
  isCapitalLetter(c) ==
    c in set dom v 英大文字順序写像;
public static
  isLowercaseLetter : char -> bool
  isLowercaseLetter(c) ==
    c in set dom v 小大文字順序写像;
public static
  isHiragana : char -> bool
  isHiragana(c) ==
    c in set dom v ひらがな順序写像;
public static
```

```
  小さい : char -> char -> bool
  小さい(c1)(c2) ==
    Character`as 辞書順序(c1) < Character`as 辞書順序(c2);
public static
  以下 : char -> char -> bool
  以下(c1)(c2) ==
    Character`小さい(c1)(c2) or c1 = c2;
public static
  大きい : char -> char -> bool
  大きい(c1)(c2) ==
    Character`小さい(c2)(c1);
public static
  以上 : char -> char -> bool
  以上(c1)(c2) ==
    not Character`小さい(c1)(c2)
end
Character
```

Test Suite : vdm.tc

Class : Character

Name	#Calls	Coverage
Character`以上	1	√
Character`以下	1	√
Character`大きい	3	√
Character`小さい	180	√
Character`as 辞書順序	368	√
Character`asDigit	11	√
Character`isDigit	409	√
Character`isLetter	178	√
Character`isHiragana	0	0%
Character`isCapitalLetter	7	√
Character`isLetterOrDigit	86	√
Character`isLowercaseLetter	12	√
Total Coverage		**94%**

2.5 Control

2.5.1 責任

制御量を計算する。

```
class
Control
functions
public static
  PID 制御する : real * real * real * real * real * real * real * real * real -> real
  PID 制御する (a 比例動作定数 Kp, a 目標値, a 現在値,
                a 積分動作定数 Ki, a 前回制御量, a 前々回までの制御量合計,
                a 微分動作定数 Kd, a 前回現在値, a 前々回現在値) ==
    比例動作制御量を得る (a 比例動作定数 Kp, a 目標値, a 現在値) +
    積分動作制御量を得る (a 積分動作定数 Ki, a 前回制御量, a 前々回までの制御量合計) +
    微分動作制御量を得る (a 微分動作定数 Kd, a 現在値, a 前回現在値, a 前々回現在値);

  比例動作制御量を得る : real * real * real -> real
  比例動作制御量を得る (a 比例動作定数 Kp, a 目標値, a 現在値) ==
    a 比例動作定数 Kp * (a 目標値 - a 現在値);

  積分動作制御量を得る : real * real * real -> real
  積分動作制御量を得る (a 積分動作定数 Ki, a 前回制御量, a 前々回までの制御量合計) ==
    a 積分動作定数 Ki * (a 前回制御量 + a 前々回までの制御量合計);

  微分動作制御量を得る : real * real * real * real -> real
  微分動作制御量を得る (a 微分動作定数 Kd, a 現在値, a 前回現在値, a 前々回現在値) ==
    a 微分動作定数 Kd * ((a 現在値 - a 前回現在値) - (a 前回現在値 - a 前々回現在値))
end
Control
```

Test Suite : vdm.tc

Class : Control

Name	#Calls	Coverage
Control‘PID 制御する	0	0%
Control‘微分動作制御量を得る	0	0%
Control‘比例動作制御量を得る	0	0%
Control‘積分動作制御量を得る	0	0%

Name	#Calls	Coverage
Total Coverage		**0%**

2.6 ControlT

Control クラスの回帰テスト を行う。

```
class
ControlT is subclass of TestDriver
functions
public
  tests : () -> seq of TestCase
  tests () ==
    [
     new ControlT01 ()]
end
ControlT
```

Test Suite : vdm.tc

Class : ControlT

Name	#Calls	Coverage
ControlT`tests	0	0%
Total Coverage		**0%**

2.7 ControlT01

ET ロボコンの PID 制御のテストを行う。

```
class
ControlT01 is subclass of TestCase
operations
public
  test : () ==> bool
  test () ==
    let w 比例動作定数 Kp = 500,
        w 積分動作定数 Ki = 100,
        w 微分動作定数 Kd = 2000,
        w 目標値 = 10,
        w 現在値 = 6,
        w 前回制御量 = 1,
        w 前々回までの制御量合計 = 3,
```

```
        w 前回現在値 = 5,
        w 前々回現在値 = 2 in
    return Control‘PID 制御する
                (
                 w 比例動作定数 Kp, w 目標値, w 現在値,
                 w 積分動作定数 Ki, w 前回制御量, w 前々回までの制御量合計,
                 w 微分動作定数 Kd, w 前回現在値, w 前々回現在値) =
            -1600;
protected
  準備する : () ==> ()
  準備する () ==
    テスト名 := "ControlT01 : \t ET ロボコンの PID 制御のテストを行う";
protected
  後始末する : () ==> ()
  後始末する () ==
    return
end
ControlT01
```

Test Suite : vdm.tc

Class : ControlT01

Name	#Calls	Coverage
ControlT01‘test	0	0%
ControlT01‘準備する	0	0%
ControlT01‘後始末する	0	0%
Total Coverage		**0%**

2.8 DoubleListQueue

待ち行列に関わる関数を定義する。

2.8.1 実装について

先頭列と後続列の 2 つの列の組を使って待ち行列を実装している。

2.8.2 使用方法

最初に

```
let Q0 = DoubleListQueue`empty[int]() in ...
```

などとして空の待ち行列を定義する。

待ち行列への追加は

```
DoubleListQueue`enQueue(1)(Q0)
```

などとすればよい。

2.8.3 仕様

```
.....................................................................................
class
DoubleListQueue
.....................................................................................
```

empty は、空の待ち行列を定義する。

```
.....................................................................................
functions
public static
  empty[@型] : () -> seq of @型 * seq of @型
  empty() ==
    mk_([],[]);
.....................................................................................
```

IsEmpty は、待ち行列が空か否かを返す。

```
.....................................................................................
public static
  isEmpty[@型] : (seq of @型 * seq of @型) -> bool
  isEmpty(s) ==
    s = mk_([],[]);
.....................................................................................
```

enQueue は、待ち行列 mk_(ある先頭列, ある後続列) にある要素を追加した待ち行列を返す。実装としては、ある後続列の前にある要素を追加する。

```
public static
  enQueue[@型] : @型 * (seq of @型 * seq of @型) -> seq of @型 * seq of @型
  enQueue(ある要素,mk_(ある先頭列,ある後続列)) ==
    mk_(ある先頭列,[ある要素]^ある後続列);
```

deQueue は、待ち行列 mk_(ある先頭列, ある後続列) の先頭を削除した待ち行列を返す。待ち行列がすでに空の場合は nil を返す。ある先頭列が空の場合は、ある後続列を反転して先頭列とし、後続列を空とした待ち行列を返す。

```
public static
  deQueue[@型] : (seq of @型 * seq of @型) -> [seq of @型 * seq of @型]
  deQueue(mk_(ある先頭列,ある後続列)) ==
    cases ある先頭列 :
      [-]^ある先頭列の残り -> mk_(ある先頭列の残り,ある後続列),
      [] ->
          cases ある後続列 :
            [] -> nil,
            others -> mk_(tl Sequence`freverse[@型](ある後続列),[])
          end
    end;
```

top は、待ち行列の先頭要素を返す。待ち行列がすでに空なら、nil を返す。ある先頭列が空の場合は、ある後続列を反転して、その先頭要素を返す。

```
public static
  top[@型] : (seq of @型 * seq of @型) -> [@型]
  top(mk_(ある先頭列,ある後続列)) ==
    cases ある先頭列 :
      [先頭]^- -> 先頭,
      [] ->
          cases ある後続列 :
            [] -> nil,
            others -> hd Sequence`freverse[@型](ある後続列)
          end
    end;
```

fromList は、ある列を待ち行列に変換して返す。

```
public static
```

```
  fromList[@型] : seq of @型 * (seq of @型 * seq of @型) -> seq of @型 * seq of @型
  fromList(ある列, 待ち行列) ==
    cases ある列 :
      [] +> 待ち行列,
      [先頭]^列の残り +> fromList[@型](列の残り, enQueue[@型](先頭, 待ち行列))
    end;
```

toList は、ある待ち行列を列に変換して返す。

```
public static
  toList[@型] : (seq of @型 * seq of @型) -> seq of @型
  toList(ある待ち行列) ==
    cases ある待ち行列 :
      (mk_([], [])) +> [],
      待ち行列 +> [top[@型](待ち行列)]^toList[@型](deQueue[@型](待ち行列))
    end
end
DoubleListQueue
```

Test Suite : vdm.tc

Class : DoubleListQueue

Name	#Calls	Coverage
DoubleListQueue‘top	34	√
DoubleListQueue‘empty	2	√
DoubleListQueue‘toList	43	√
DoubleListQueue‘deQueue	38	√
DoubleListQueue‘enQueue	16	√
DoubleListQueue‘isEmpty	2	√
DoubleListQueue‘fromList	12	√
Total Coverage		**100%**

3 SBCalendar

3.1 責任

証券会社システムの暦クラス。

3.2 概要

国・会社・休日ベース区分を考慮した、トレードワンシステムの暦処理のスタブである。土曜日や年末年始を休日として扱う。

```
.....................................................................................
class
SBCalendar is subclass of JapaneseCalendar
values
  io = new IO();
  暦 = new SBCalendar()
instance variables
  public 基準日という日 : [Date] := nil;
  public 会社基準日写像 : [map seq of char to Date] := { |-> };
  public システムの時刻 : [Time] := nil;

operations
public
  暦を得る : () ==> Calendar
  暦を得る() ==
    return self
functions
public static
  限月が正当 : seq of char -> bool
  限月が正当(限月) ==
    暦.文字列から日付を得る(限月^"01") <> false;
public static
  権利行使日を得る : seq of char -> Date
  権利行使日を得る(限月) ==
    let 限月の月初日 = 暦.文字列から日付を得る(限月^"01"),
        指定年 = 限月の月初日.年(),
        指定月 = 限月の月初日.月() in
    暦.第n指定曜日を得る(指定年, 指定月, 2, <金>).過去の平日を得る()
  pre  限月が正当(限月);
.....................................................................................
```

3.2.1 信用期日を得る

信用取引の決済日（期日）を得る。弁済期限とは、信用建玉に対して当社がお客様に信用を供与する期限をいいます。弁済期限は、現在のところ 6 ヶ月のみを取扱っています。弁済期限が 6 ヶ月であるということは、信用建玉の建日（信用建玉が約定した日）の 6 ヶ月目応当日が信用期日となり、この日を超えて建玉を保有することは法律で禁じられています。信用期日が休日の場合には、直近の前営業日が信用期日となります。

この日本語仕様だと、以下の問題があります。

1. 応答日の定義がない。
2. たとえば信用建玉の建日が８月３１日だと、応当日は翌年２月３１日になってしまうのだが、そのとき２月２９日にするのか３月１日にするのかの記述がない。
3. 応当日から月初日まですべて休日の場合、前月の営業日にしてよいのかどうかの記述がない。
4. 信用取引の決済日（信用期日）と弁済期限が同じものを指しているが、明確な定義がない。

```
public static
  信用期日を得る : Date -> Date
  信用期日を得る (aDate) ==
    let mk_(年,月) = 暦.六ヶ月後を求める (aDate.年 (), aDate.月 ()),
        日 = aDate.日 (),
        期日候補 = 期日候補を求める (年,月,日) in
    期日候補.過去の平日を得る ()
  post let mk_(年,月) = 暦.六ヶ月後を求める (aDate.年 (), aDate.月 ()),
           日 = aDate.日 (),
           期日候補 = 期日候補を求める (年,月,日) in
       RESULT.等しい (期日候補.過去の平日を得る ()) and
       if 月初から期日候補日まで休み (期日候補)
       then RESULT.月 () = 前月を得る (年,月)
       else RESULT.月 () = 月 ;
public static
  六ヶ月後を求める : int * int -> int * int
  六ヶ月後を求める (年,月) ==
    暦.正則月を得る (年,月 + 6);
public static
  期日候補を求める : int * int * int -> Date
  期日候補を求める (年,月,日) ==
    let 月末日 = 暦.月末日を求める (年,月) in
    if 月末日.日 () < 日
    then 月末日
    else 暦.整数三つ組から日付を得る (年,月,日);
public static
```

```
  月初から期日候補日まで休み : Date -> bool
  月初から期日候補日まで休み ( 期日候補 ) ==
     forall day in set {1,...,期日候補.日 ()} &
         暦.お休みか？ (暦.整数三つ組から日付を得る (期日候補.年 (),期日候補.月 (),day));
public static
  前月を得る : int * int -> int
  前月を得る ( 年,月 ) ==
    let mk_(-,前月) = 暦.正則月を得る ( 年,月 - 1) in
    前月;
public static
  日付が空か？ : [Date] -> bool
  日付が空か？ ( 日付 ) ==
    日付 = nil;
public static
  システム日付 : () -> Date
  システム日付 () ==
    暦.今日 ()
operations
public
  休日集合を設定する : int ==> ()
  休日集合を設定する ( 年 ) ==
    let 日本の暦 = new JapaneseCalendar (),
        日本の休日集合 = 日本の暦.休日集合を得る ( 年),
        TR1 の休日集合 = {
                          日本の暦.整数三つ組から日付を得る ( 年,1,2),
                          日本の暦.整数三つ組から日付を得る ( 年,1,3),
                          日本の暦.整数三つ組から日付を得る ( 年,12,29),
                          日本の暦.整数三つ組から日付を得る ( 年,12,30),
                          日本の暦.整数三つ組から日付を得る ( 年,12,31)},
        土曜日集合 = 日本の暦.ある年の指定曜日集合を得る ( 年,<土>) in
    休日集合写像 := 休日集合写像 munion { 年 |-> 日本の休日集合 union TR1 の休日集合 union 土曜
日集合 }
  pre 年 >= 2000 ;
public
  基準日を読み込む : seq of char ==> [Date]
  基準日を読み込む ( ファイル名 ) ==
    let mk_( 結果,mk_(y,m,d)) = io.freadval[int * int * int] ( ファイル名) in
    if 結果
    then return 整数三つ組から日付を得る (y,m,d)
```

```
    else let - = io.echo ("Can't read BaseDay's data file.") in
         return nil;
public
  基準日 : () ==> Date
  基準日 () ==
    if 基準日という日 = nil
    then return 基準日を読み込む (homedir^"/temp/BaseDay.txt")
    else return 基準日という日;
public
  ファイルから読み込む基準日 : seq of char ==> Date
  ファイルから読み込む基準日 ( ファイル名) ==
    if 基準日という日 = nil
    then return 基準日を読み込む ( ファイル名)
    else return 基準日という日;
public
  基準日を設定する : Date ==> ()
  基準日を設定する ( 日付) ==
    基準日という日 := 日付;
public
  会社基準日 : seq of char ==> Date
  会社基準日 ( 会社コード) ==
    (   if 会社基準日写像 = nil
        then 会社基準日を設定する ( 会社コード, 基準日 ()) ;
        return 会社基準日写像 ( 会社コード)
    ) ;
public
  会社基準日を設定する : seq of char * Date ==> ()
  会社基準日を設定する ( 会社コード, 日付) ==
    会社基準日写像 := 会社基準日写像 ++ { 会社コード |-> 日付 };
public
  システム時刻を読み込む : () ==> [Time]
  システム時刻を読み込む () ==
    let mk_ ( 結果, now) = io.freadval[Time] (homedir^"/temp/SystemTime.txt") in
    if 結果
    then return now
    else let - = io.echo ("Can't read System Time data file.") in
         return nil ;
public
```

```
  システム時刻 : () ==> Time
  システム時刻 () ==
    if システムの時刻 = nil
    then システム時刻を読み込む ()
    else return システムの時刻;
public
  システム時刻を設定する : Time ==> ()
  システム時刻を設定する ( 時刻) ==
    システムの時刻 := 時刻;
public
  SBCalendar : () ==> SBCalendar
  SBCalendar () ==
    (   グリニッジ標準時との差を設定する ( 日本標準時とグリニッジ標準時との差) ;
        return self
    )
end
SBCalendar
```

Test Suite : vdm.tc

Class : SBCalendar

Name	#Calls	Coverage
SBCalendar‘基準日	3	√
SBCalendar‘暦を得る	0	0%
SBCalendar‘会社基準日	2	69%
SBCalendar‘前月を得る	4	√
SBCalendar‘限月が正当	5	√
SBCalendar‘システム日付	1	√
SBCalendar‘システム時刻	1	85%
SBCalendar‘日付が空か？	2	√
SBCalendar‘信用期日を得る	32	√
SBCalendar‘六ヶ月後を求める	64	√
SBCalendar‘基準日を設定する	2	√
SBCalendar‘基準日を読み込む	2	75%
SBCalendar‘期日候補を求める	64	√
SBCalendar‘権利行使日を得る	2	√
SBCalendar‘休日集合を設定する	5	√
SBCalendar‘SBCalendar	8	√
SBCalendar‘会社基準日を設定する	2	√

Name	#Calls	Coverage
SBCalendar‘システム時刻を設定する	1	√
SBCalendar‘システム時刻を読み込む	0	0%
SBCalendar‘ファイルから読み込む基準日	1	80%
SBCalendar‘月初から期日候補日まで休み	32	√
Total Coverage		**89%**

4 Sequence

4.1 責任

列を表す。

4.2 概要

Sequence 型で定義された機能以外の機能を。

4.3 注意事項

歴史的経過のため、より関数型プログラミングに適した関数と、そうでないものがある。関数型プログラミングに適した関数は、英字名の場合大文字で始まる。大文字で始まる同一名の関数がある場合、小文字で始まる関数・操作は、古い定義で、互換性のために存在する。大文字で始まる同一名の関数が無い場合は、小文字で始まる関数・操作も関数型プログラミングに適している。

4.4 参照

多くの関数は、関数型プログラミング言語 Concurrent Clean のライブラリーから移植した。

```
class
Sequence
functions
public static
  Sum : seq of real -> real
  Sum ( 列) ==
    Foldl[real,real] (Plus) (0) ( 列);
public static
  Prod : seq of real -> real
  Prod ( 列) ==
    Foldl[real,real] (Product) (1) ( 列);
public static
  Plus : real -> real -> real
  Plus (a)(b) ==
    a + b;
public static
  Product : real -> real -> real
  Product (a)(b) ==
    a * b;
public static
  平均を得る : seq of real -> [real]
  平均を得る ( 列) ==
    if 列 = []
    then nil
    else let sum : real = Sum ( 列) in
         sum / len 列;
public static
  全順序昇順か？ [@型] : (@型 * @型 -> bool) -> seq of @型 -> bool
  全順序昇順か？ ( 順序決定関数)( 列) ==
     forall i,j in set inds 列 & i < j => 順序決定関数 (列 (i),列 (j)) or 列 (i) = 列 (j);
public static
  全順序降順か？ [@型] : (@型 * @型 -> bool) -> seq of @型 -> bool
  全順序降順か？ ( 順序決定関数)( 列) ==
     forall i,j in set inds 列 & i < j => 順序決定関数 (列 (j),列 (i)) or 列 (i) = 列 (j);
public static
  昇順か？ : seq of real -> bool
  昇順か？ ( 列) ==
    全順序昇順か？ [real] (lambda x : real,y : real & x < y) ( 列);
public static
```

```
  降順か？ : seq of real -> bool
  降順か？ ( 列) ==
    全順序降順か？ [real] (lambda x : real, y : real & x < y) ( 列);
public static
  sort[@型] : (@型 * @型 -> bool) -> seq of @型 -> seq of @型
  sort ( 順序決定関数)( 列) ==
    cases 列 :
      [] +> [],
      [要素]^部分列 +>
          sort[@型] ( 順序決定関数) ([部分列 (i) |  & 順序決定関数 (部分列 (i), 要素)])^
          [要素]^
          sort[@型] ( 順序決定関数) ([部分列 (i) |  & not 順序決定関数 (部分列 (i), 要素)])
    end;
public static
  昇順 Sort : seq of real -> seq of real
  昇順 Sort ( 列) ==
    sort[real] (lambda x : real, y : real & x < y) ( 列);
public static
  降順 Sort : seq of real -> seq of real
  降順 Sort ( 列) ==
    sort[real] (lambda x : real, y : real & x > y) ( 列);
public static
  順序通りか？ [@型] : seq of (@型 * @型 -> bool) -> seq of @型 -> seq of @型 -> bool
  順序通りか？ ( 順序決定関数列)( 列 1)( 列 2) ==
    cases mk_( 列 1, 列 2) :
      mk_([], []) +> false,
      mk_([], -) +> true,
      mk_(-, []) +> false,
      mk_([先頭 1]^後続 1, [先頭 2]^後続 2) +>
          if (hd 順序決定関数列) ( 先頭 1, 先頭 2)
          then true
          elseif (hd 順序決定関数列) ( 先頭 2, 先頭 1)
          then false
          else Sequence‘順序通りか？ [@型] (tl 順序決定関数列) ( 後続 1) ( 後続 2)
    end;
public static
```

```
  マージする[@型] : (@型 * @型 -> bool) -> seq of @型 -> seq of @型 -> seq of @型
  マージする(順序決定関数)(s1)(s2) ==
    cases mk_(s1,s2) :
      mk_([],y) +> y,
      mk_(x,[]) +> x,
      mk_([先頭 1]^後続 1,[先頭 2]^後続 2) +>
          if 順序決定関数(先頭 1,先頭 2)
          then [先頭 1]^Sequence`マージする[@型](順序決定関数)(後続 1)(s2)
          else [先頭 2]^Sequence`マージする[@型](順序決定関数)(s1)(後続 2)
    end;
public static
  InsertAt[@型] : nat1 -> @型 -> seq of @型 -> seq of @型
  InsertAt(位置)(要素)(列) ==
    cases mk_(位置,列) :
      mk_((1),a 列) +> [要素]^a 列,
      mk_(-,[]) +> [要素],
      mk_(a 位置,[先頭]^後続列) +> [先頭]^InsertAt[@型](a 位置 - 1)(要素)(後続列)
    end;
public static
  RemoveAt[@型] : nat1 -> seq of @型 -> seq of @型
  RemoveAt(位置)(列) ==
    cases mk_(位置,列) :
      mk_((1),[-]^後続列) +> 後続列,
      mk_(a 位置,[先頭]^後続列) +> [先頭]^RemoveAt[@型](a 位置 - 1)(後続列),
      mk_(-,[]) +> []
    end;
public static
  RemoveDup[@型] : seq of @型 -> seq of @型
  RemoveDup(列) ==
    cases 列 :
      [先頭]^後続 +> [先 頭]^RemoveDup[@型](filter[@型](lambda x : @型 & x <> 先頭)( 後 続
)),
      [] +> []
    end
  measure length_measure ;

  length_measure[@型] : seq of @型 -> nat
  length_measure(s) ==
    len s;
public static
```

```
  RemoveMember[@型] : @型 -> seq of @型 -> seq of @型
  RemoveMember(要素)(列) ==
    cases 列 :
      [先頭]^後続 +> if 要素 = 先頭
                     then 後続
                     else [先頭]^RemoveMember[@型](要素)(後続),
      [] +> []
    end;
public static
  RemoveMembers[@型] : seq of @型 -> seq of @型 -> seq of @型
  RemoveMembers(要素列)(列) ==
    cases 要素列 :
      [] +> 列,
      [先頭]^後続 +> RemoveMembers[@型](後続)(RemoveMember[@型](先頭)(列))
    end;
public static
  UpdateAt[@型] : nat1 -> @型 -> seq of @型 -> seq of @型
  UpdateAt(位置)(要素)(列) ==
    cases mk_(位置,列) :
      mk_(-,[]) +> [],
      mk_((1),[-]^後続列) +> [要素]^後続列,
      mk_(a位置,[先頭]^後続列) +> [先頭]^UpdateAt[@型](a位置 - 1)(要素)(後続列)
    end;
public static
  take[@型] : int -> seq of @型 -> seq of @型
  take(i)(列) ==
    列(1,...,i);
public static
  TakeWhile[@型] : (@型 -> bool) -> seq of @型 -> seq of @型
  TakeWhile(関数)(列) ==
    cases 列 :
      [先頭]^後続列 +>
          if 関数(先頭)
          then [先頭]^TakeWhile[@型](関数)(後続列)
          else [],
      [] +> []
    end;
public static
```

```
  drop[@型] : int -> seq of @型 -> seq of @型
  drop(i)(列) ==
    列(i + 1,...,len 列);
public static
  DropWhile[@型] : (@型 -> bool) -> seq of @型 -> seq of @型
  DropWhile(関数)(列) ==
    cases 列 :
      [先頭]^後続列 +>
          if 関数(先頭)
          then DropWhile[@型](関数)(後続列)
          else 列,
      [] +> []
    end;
public static
  Span[@型] : (@型 -> bool) -> seq of @型 -> seq of @型 * seq of @型
  Span(関数)(列) ==
    cases 列 :
      [先頭]^後続列 +>
          if 関数(先頭)
          then let mk_(条件を満たす列,条件を満たさない列) = Span[@型](関数)(後続列) in
               mk_([先頭]^条件を満たす列,条件を満たさない列)
          else mk_([],列),
      [] +> mk_([],[])
    end;
public static
  SubSeq[@型] : nat -> nat -> seq of @型 -> seq of @型
  SubSeq(開始位置)(要素数)(列) ==
    列(開始位置,...,開始位置 + 要素数 - 1);
public static
  last[@型] : seq of @型 -> @型
  last(列) ==
    列(len 列);
public static
  fmap[@型1,@型2] : (@型1 -> @型2) -> seq of @型1 -> seq of @型2
  fmap(関数)(列) ==
    [関数(列(i)) | ];
public static
  filter[@型] : (@型 -> bool) -> seq of @型 -> seq of @型
  filter(関数)(列) ==
    [列(i) | & 関数(列(i))];
```

```
public static
  Foldl[@型1,@型2] : (@型1 -> @型2 -> @型1) -> @型1 -> seq of @型2 -> @型1
  Foldl(関数)(引数)(列) ==
    cases 列 :
      [] +> 引数,
      [先頭]^後続列 +> Foldl[@型1,@型2](関数)(関数(引数)(先頭))(後続列)
    end;
public static
  Foldr[@型1,@型2] : (@型1 -> @型2 -> @型2) -> @型2 -> seq of @型1 -> @型2
  Foldr(関数)(引数)(列) ==
    cases 列 :
      [] +> 引数,
      [先頭]^後続列 +> 関数(先頭)(Foldr[@型1,@型2](関数)(引数)(後続列))
    end;
public static
  isMember[@型] : @型 -> seq of @型 -> bool
  isMember(要素)(列) ==
    cases 列 :
      [先頭]^後続 +> 要素 = 先頭 or isMember[@型](要素)(後続),
      [] +> false
    end;
public static
  isAnyMember[@型] : seq of @型 -> seq of @型 -> bool
  isAnyMember(要素列)(列) ==
    cases 要素列 :
      [先頭]^後続 +> isMember[@型](先頭)(列) or isAnyMember[@型](後続)(列),
      [] +> false
    end;
public static
  Index[@型] : @型 -> seq of @型 -> int
  Index(要素)(列) ==
    let i = 0 in
    Index 補助関数 [@型](要素)(列)(i);
```

```
  Index 補助関数 [@型] : @型 -> seq of @型 -> int -> int
  Index 補助関数 ( 要素)( 列)( 索引) ==
    cases 列 :
      [] +> 0,
      [先頭]^後続 +>
          if 先頭 = 要素
          then 索引 + 1
          else Index 補助関数 [@型] ( 要素) ( 後続) ( 索引 + 1)
    end;
public static
  IndexAll[@型] : @型 -> seq of @型 -> set of int
  IndexAll ( 要素)( 列) ==
    {i | i in set inds 列 & 列 (i) = 要素 };
public static
  flatten[@型] : seq of seq of @型 -> seq of @型
  flatten ( 列) ==
    conc 列;
public static
  compact[@型] : seq of [@型] -> seq of @型
  compact ( 列) ==
    [列 (i) |  & 列 (i) <> nil];
public static
  freverse[@型] : seq of @型 -> seq of @型
  freverse ( 列) ==
    [列 (len 列 + 1 - i) | ];
public static
  Permutations[@型] : seq of @型 -> set of seq of @型
  Permutations ( 列) ==
    cases 列 :
      [],[-] +> { 列 },
      others +> dunion  {{[列 (i)]^j | j  in set  Permutations[@型] (RestSeq[@型] (  列
,i))} | i in set inds 列 }
    end
  measure  length_measure ;
public static
  IsPermutation[@型] : seq of @型 -> seq of @型 -> bool
  IsPermutation ( 列 1)( 列 2) ==
    asBag[@型] ( 列 1) = asBag[@型] ( 列 2);
public static
```

```
  asBag[@型] : seq of @型 -> map @型 to @型
  asBag(列) ==
    {e |-> card {列(i) = e} | i in set inds 列, e in set elems 列};
public static
  RestSeq[@型] : seq of @型 * nat -> seq of @型
  RestSeq(列, i) ==
    [列(j) | ];
public static
  Unzip[@型1, @型2] : seq of (@型1 * @型2) -> seq of @型1 * seq of @型2
  Unzip(列) ==
    cases 列 :
      [] +> mk_([], []),
      [mk_(x, y)]^後続列 +>
          let mk_(xs, ys) = Unzip[@型1, @型2](後続列) in
          mk_([x]^xs, [y]^ys)
    end
  measure  Unzip_measure ;

  Unzip_measure[@型1, @型2] : seq of (@型1 * @型2) -> nat
  Unzip_measure(s) ==
    len s;
public static
  Zip[@型1, @型2] : seq of @型1 * seq of @型2 -> seq of (@型1 * @型2)
  Zip(列1, 列2) ==
    Zip2[@型1, @型2](列1)(列2);
.....
  列の組を、組の列に変換する（より関数型プログラミングに適した形式）
.....
public static
  Zip2[@型1, @型2] : seq of @型1 -> seq of @型2 -> seq of (@型1 * @型2)
  Zip2(列1)(列2) ==
    cases mk_(列1, 列2) :
      mk_([先頭1]^後続列1, [先頭2]^後続列2) +> [mk_(先頭1, 先頭2)]^Zip2[@型1, @型2](後続列1)(後
続列2),
      mk_(-, -) +> []
    end
end
Sequence
.....
```

Test Suite : vdm.tc

Class : Sequence

Name	#Calls	Coverage
Sequence‘Sum	16	√
Sequence‘Zip	4	√
Sequence‘Plus	3162	√
Sequence‘Prod	5	√
Sequence‘Span	14	√
Sequence‘Zip2	16	√
Sequence‘drop	2	√
Sequence‘fmap	21	√
Sequence‘last	1	√
Sequence‘sort	35	√
Sequence‘take	13	√
Sequence‘昇順か？	2	√
Sequence‘降順か？	1	√
Sequence‘Foldl	3204	√
Sequence‘Foldr	12	√
Sequence‘Index	358	√
Sequence‘Unzip	5	√
Sequence‘asBag	2	√
Sequence‘マージする	16	√
Sequence‘平均を得る	5	√
Sequence‘SubSeq	1	√
Sequence‘filter	10	√
Sequence‘昇順 Sort	1	√
Sequence‘降順 Sort	1	√
Sequence‘順序通りか？	15	√
Sequence‘Product	9	√
Sequence‘RestSeq	49	√
Sequence‘compact	3	√
Sequence‘flatten	1	√
Sequence‘全順序昇順か？	2	√
Sequence‘全順序降順か？	3	√
Sequence‘IndexAll	21	√
Sequence‘InsertAt	20	95%
Sequence‘RemoveAt	29	94%
Sequence‘UpdateAt	22	95%

Name	#Calls	Coverage
Sequence‘freverse	21	√
Sequence‘isMember	39	√
Sequence‘DropWhile	177	√
Sequence‘Index 補助関数	2111	√
Sequence‘RemoveDup	11	√
Sequence‘TakeWhile	170	√
Sequence‘isAnyMember	6	√
Sequence‘Permutations	52	√
Sequence‘RemoveMember	19	√
Sequence‘IsPermutation	1	√
Sequence‘RemoveMembers	14	√
Sequence‘Unzip_measure	0	0%
Sequence‘length_measure	0	0%
Total Coverage		**98%**

5　参考文献等

VDM++[2] は、1970 年代中頃に IBM ウィーン研究所で開発された VDM-SL[1] を拡張し、さらにオブジェクト指向拡張したオープンソース[*3]の形式仕様記述言語である。

参考文献

[1] CSK システムズ. VDM-SL 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[2] CSK システムズ. VDM++ 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[*3] 使用に際しては、SCSK（株）との契約締結が必要になる。

索引