メカニズムの解明や統計手法の精度評価など、既存の統計分析に止まらない多くの応用と可能性を持っている。パネル調査で捉えられた標本をシミュレーションモデルとして再現すれば、さまざまな仮想的条件や仮定の下での標本の変化を観察することが可能であり、それらを実際の変化と比較すれば、仮定やモデルの妥当性を評価することができる。

縦断型マイクロシミュレーションは、21 世紀縦断調査についても、その主要なテーマである結婚・出生・子育てなどの発生メカニズムと決定要因の解明や、制度・施策効果の評価を行う有力な手法となるほか、脱落をはじめとするパネル調査特有の統計分析上の困難に対して、さまざまな条件下におけるそれら統計手法の妥当性や精度を検証する有効な手段を与えると考えられる。

本研究では、21 世紀縦断調査データを活用して今後継続的なマイクロシミュレーション分析が行えるよう、その基礎としてエージェント型(agent-base)のマイクロシミュレーションモデルに必要な標本を生成するシステムを開発した。これはパネル調査データの管理情報を活用して、シミュレーション分析に必要となる標本モデルを半自動的に生成するシステムであり、現行では C++によるシミュレーションモデルを作成することができる。システムは、本事業で構築を行ったデータマネジメントシステムの一環として開発されており、統合的に扱うことができるものである。

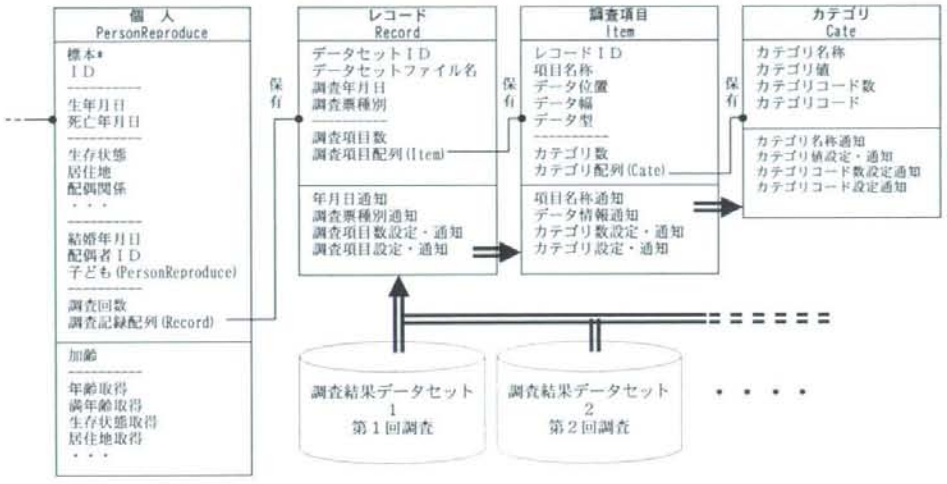
(1) マイクロシミュレーションにおける標本モデル

ここで想定する縦断型マイクロシミュレーションは、エージェント型(agent-base)のマイクロシミュレーションモデルを基礎とするものである。そこでは個人のモデルは、自律性を備えたオブジェクト、すなわちエージェントして実装される。図 1 には、本シミュレーションのベースモデルとなるプロトタイプモデルのクラス図を示した。これは観察単位（エージェント）の時間的変化・行動を継続的に発生するタイプのクラスの定義である（クラスとは、エージェントまたはオブジェクトのシミュレーション言語上の定義のことである）。21 縦断調査の対象者に対応するエージェント・クラスを中心として、その属性や家族などの関係者、さらには標本集団とその統計的特性を集合的に計測、記録、出力する統計制度のエージェント・クラスを配置している。これらを基本とし、出生児調査、成年者調査など各調査ごとに、また分析テーマごとに、必要なエージェント・クラスを追加して分析モデルを構築することとなる。

図1　標本モデルのクラス図

このエージェントモデルと、縦断調査結果のデータセットを具体的に結びつけるデータクラスを図２に示した。図１における個人（Person)（あるいはその継承クラスの「再生産する個人」PersonReproduce など）は、実はほとんどの属性を各調査回に対応する調査票イメージのデータセットの調査レコード(Record)として保有している。調査レコード(Record)は、たとえば、最終学歴、職業などの多数の調査項目（Item)の集合によって構成されている。さらに、各調査項目（Item)は、その中身としてカテゴリ(Cate）によって構成されており、各カテゴリ(Cate）は基本的に名前と値を持つ。これらは調査結果データを個人別に格納するが、調査レコード（Record）のメソッドを介して、調査結果データセットのファイルから実際のデータを読み込むことになる。これらのメソッドによって、縦断調査結果データとシミュレーションの標本モデルがユーザの見かけ上単純な操作によって、直接に結び付けられる。あとは、個人の振る舞いに関する加齢モジュールに結婚、出生、あるいは就業などの行動モデルを記述することによってライフコース事象が属性や環境に依存しながら発生する様子をシミュレートすることができる。

図２　調査結果データのクラス図



エージェント型シミュレーションモデルの優れた点は、各種のライフコース事象をそのエージェントが置かれた環境やエージェントどうしの相互作用に依存するモデルを構成し、検証することができる点である。またその依存関係についても境界値による事象の制御など、非線形の関係を記述することができる。これらは通常の統計モデルでは、ほぼ不可能である。実際の個人のライフコース選択においては、環境からの影響やオールオアナッシングの判断などが重要な役割を果たしている可能性があることから、こうした機構を持つモデルについて検証することは、少子化などの現象のメカニズム解明に対して大きな貢献が期待されるところである。また、こうした非線形現象は複雑系現象として知られるが、人間行動が複雑系現象であるとの指摘がなされており、これに対して、21 世紀縦断調査という優れた現実の事象データを用いることができることから、こうした分野の発展にも寄与することが大いに期待できる。

図２に示した調査結果データのクラスの実装モデル（C++）については、本事業の先行事業である「パネル調査（縦断調査）に関する総合的分析システムの開発研究」（厚生労働科学研究費補助金（統計情報総合研究事業））平成 18 年度総括報告書、ならびに平成 18～19 年総合報告書に所収したところである。

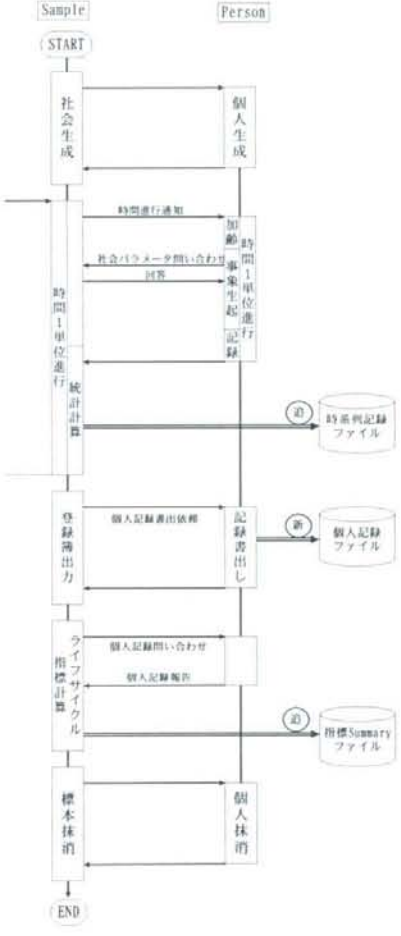
(2) モデルのシーケンス

次に図２には、本シミュレーションの基本的なシーケンス図を示した。調査対象者の個人を順に生成することによって、標本を再現する。その際、個人には実際の調査から得られた各種の個人属性が割り当てられ、時間経過にしたがって検証するモデル（規則）によって行動が発生することになる。その間に、必要であれば、個人間においての相互作用が発生する。これら相互作用の過程は、通常の統計分析法では推定がほぼ不可能と考えられる領域である。出生等による新たな個人の参入については、新たなエージェントの発生と

して扱うことができる。これにより現実に近い状況を実現できるが、系の進行状況はきわめて複雑なものとなりうる。これを避けるためには、新たな個人を増やすのではなく新たな関係の発生と考え、属性の変化として扱うことも可能である。その方が状況の見通しはよくなると考えられる。

マイクロシミュレーションは、既存の統計手法の当該データ（21 世紀縦断調査データ）への適用妥当性の検討や、選択的脱落・不詳の効果の推定や将来的な帰結についての予見に用いることができる。そのためには、対象標本について詳細な統計指標を算出することが必要となる。統計指標の算出は、時間単位ごと行われるものと、ライフサイクル指標のように一定の期間終了後に算出されるものがある。いずれにせよ、それら指標はすべて記録され、標本に対する検討対象の統計手法の適用結果と詳細に比較されることによって、手法の妥当性が調べられることになる。

図2　プロトタイプモデルのシーケンス図



(2) 標本個体クラスモジュールの開発

図1標本モデルのクラス図に見られるように、本シミュレーションにおいて個人クラス Person は中心的な役割を持つものである。個人クラスは継承によって再生産する個人クラス PersonReproduce や、さらにこれを継承した女性バージョンとなる PersonReproduceF、ならびに男性バージョンとなる PersonReproduceM が派生される。また、PersonReproduce からは、別の派生クラスである子どもクラス PersonChild のインスタンスが生成される。

今回、上記のようなすべての標本個体クラスの基底となる Person のクラスモジュールの開発を行った。個人クラス Person は、図1、図2に示すとおり、個人属性クラスや調査記録配列として調査個票 Record クラスを保有する。また、Person は集合して標本クラス Sample に保有される。Person クラスは機能（メソッド）として、誕生、加齢、死亡などの基本的なライフコースの形成を行うほか、別クラスとして用意される社会経済基本属性クラス PerProperty を維持管理する。結婚や出産といったライフイベントは上述のとおり、本クラスから派生された女性クラス PersonReproduceF、男性クラス PersonReproduceM において追加される機能である。

(3) 日付処理クラスモジュールの開発

マイクロシミュレーションモデル分析においては、コンピュータ内にシミュレートする人口（標本）の年月・時間の進行を独自に管理する必要がある。すなわち、人口内の擬似的な個人（エージェント）は、単位計算サイクルごとにたとえば、1ヶ月加齢し、結婚、出生、就業、死亡等のライフコース事象を経験するので、加齢や事象発生時間の記録、管理、事象間の時間の計算などは、シミュレーション過程においてきわめて重要な機能である。また、シミュレーションに用いられる暦は、うるう年などを用いず、1ヶ月を常に 30 日とするなど、日常用いられている暦（グレゴリオ暦）と異なる日・時間体系を操作できる必要がある。このため、先行研究に引き続き、本シミュレーションモデルに親和する日付処理クラスモジュールの開発を行った。

本クラスモジュールでは、通常の日付、時間計算の管理をベースとしながらも、シミュレーション時間モードを導入し、仮想的な時間進行が可能となるように設計された。また、各種時間計算の勘弁に行えるようにするため、加減演算の演算子を定義し、日付・時刻と期間・時間との間で、直感的な加減式によって演算が行えるようにした。

まとめ

本研究では、21 世紀縦断調査データを用いたライフコース事象のマイクロシミュレーション分析を行うための基礎的システムの検討・設計を行った。マイクロシミュレーション分析は、パネル調査との親和性が強く、既存の統計分析ではできない非線形現象としての事象メカニズムの分析や、脱落等の評価が行えるため、統計分析との併用によって縦断調査データの活用範囲を広げるとともに、提供する情報の信頼性向上に資することが期待できる。本年度は、標本個体クラスモジュール、日付処理クラスモジュールの開発を行った。

参照文献

Citro, C. F. and Hanushek, E. A. (eds.) 1991, *The Uses of Microsimulation Modelling. Vol. 1: Review and Recommendations*. National Academy Press, Washington, DC.

Harding, A.(ed.), 1991, *Microsimulation and Public Policy*, Contributions to Economic Analysis, vol.232, Elsevier, Amsterdam.

Lewis, G. H. and Michel, R. C. (eds.) 1990, *Microsimulation Techniques for Tax and Transfer Analysis*. Urban Institute Press, Washington, DC.

付[1]　標本個体クラスモジュール

```
//*************************************************************************
//*****              Person   Ver 1.0 : 個人クラス                        *******
//*****                                          Kaneko <09/02/18>   *******
//*************************************************************************
// ((person 定義 file ))
// person.h
#if !defined(person_h)     //多重インクルード防止
#define     person_h

#include "..¥NkDate¥NkDate.h"
#include "..¥Society¥society.h"

//…[ マクロ ]-----------------------------------------------------------
// 一様乱数の発生
//(RAND_MAX = 0x7FFFU = 32,767) : 正規乱数発生のため rand()=0 を除外
//#define UNIFORM       ((double)(rand()+1)/(double)(RAND_MAX+1))

//こちらは、0 から LONG_RAND_MAX(=2^31-1 =2,147,483,647)までの一様乱数
#define LONG_RAND_MAX   2147483647
#define UNIFORM       ((double)(_lrand())/(double)(LONG_RAND_MAX))
// 一様事象の生起
#define TRIAL_U(p)   (UNIFORM<(p)?1:0)

#define LOCATION_ORIGIN (0)       // 座標の原点
#define LOCATION_CYCLE   (360)     // 座標の周期

//===[ クラス定義 : Person ]=============================================
class Person {

  public://公開関数-----------------------------------------------------

    // クラス初期化、終了処理
    static void       ClassInitialize(Society* s);       // クラス全体の初期化
    static void       ClassFinalize(void);                 // クラス全体の終了処理
    static void       ClassCommonParameter(void);          // クラス全体の設定（各期）

    // ｺﾝｽﾄﾗｸﾀ
    Person(void);
    Person(long idn);
    Person(long idn, NkDate birthdate );
    Person(long idn, NkDate birthdate, double location);

    // ﾃﾞｽﾄﾗｸﾀ
    ~Person(void);

    //行動操作
    long    proceed(void);                          // 単位時間前へ
    long    retreat(void);                          // 単位時間後へ

    // 所属
```

```
  long          ID(void){ return id; }                  // 個人ＩＤ番号取得

  // 生存関連の属性操作
  bool    Alive(void){return alive;}                    // 生存状態取得(true=生存, false=死亡)
  NkDate BirthDate(void){return birth_date;}            // 誕生日取得
  NkDate DeathDate(void){return death_date;}            // 死亡日取得
  double Age(void);                                     // 年齢取得
  double Age(NkDate t);                                 // 指定年月年齢(算出のみ設定はしない)
  int     CompletedAge(void);                           // 満年齢
  int     CompletedAge(NkDate t);                       // 指定年月満年齢(算出のみ設定はしない)

  // ロケーション
  double Location(void){return location;}               // 位置座標取得

  // 出 力
  static void OutputHead(FILE*);                        // 出力表のヘッダを出力(クラス関数)
  void          output(FILE*);                          // 出力表の個人データを出力
  void          ProcessOut(FILE*,int);                  // 途中状態の出力
  void          showProperties(void);

  double        Frailty(void)          { return frailty; }
  double        Frailty(double f)      { return frailty=f; }
  void          baptizeFrailty(double* p);

protected://内部関数-----------------------------------------------------------

  long     ID(long idn){ return id=idn; }               // 個人ＩＤ番号設定

  // 生存関係
  void    Birth(NkDate bd);                             // 出生発生：形式１
  void    Birth(unsigned y,                             // 出生発生：形式２
                unsigned m,
                unsigned d);
  void    Death(NkDate bd);                             // 死亡発生：形式１
  void    Death(unsigned y,                             // 死亡発生：形式２
                unsigned m,
                unsigned d);
  bool    Alive(bool a){ return (alive=a);}             // 生存状態設定(true=生存, false=死亡)
  NkDate BirthDate(NkDate bd);                          // 誕生日設定：形式１
  NkDate BirthDate(long ld);                            // 誕生日設定：形式２
  NkDate BirthDate(unsigned y,                          // 誕生日設定：形式３
                   unsigned m,
                   unsigned d);

  NkDate DeathDate(NkDate dd);                          // 死亡日設定：形式１
  NkDate DeathDate(long ld);                            // 死亡日設定：形式２
  NkDate DeathDate(unsigned y,                          // 死亡日設定：形式３
                   unsigned m,
                   unsigned d);

  // ロケーション
  double Location(double l);

  // 属性操作
  void    aging(void);                                  // 加齢

  // 事象確率
  bool die();                                           // 死亡判定
  double probDeath(void);                               // 死亡確率

  // ユーティリティ
  NkDate    now(void){return home_society->Now();}      // 現在時刻
  static Society* home(void){return home_society;}      // 所属社会取得
  static SimOpt*  Option(void){return home_society->Option();} // オプションクラス*取得
```

```
  private://私的変数-----------------------------------------------------------

    // 識別属性
    static Society*   home_society;                  // 所属社会(クラス属性)
    long              id;                             // ＩＤ番号

    // 基本属性
    int               alive;                          // 生存状態(1=生存, 0=死亡)
    NkDate            birth_date;                       // 出生年月
    NkDate            death_date;                       // 死亡年月(生存なら NULL_DATE)

    double            location;                       // ロケーション( 0<= location < 360 )
    double            frailty;                        // Frailty

    // クラス属性
    static double* qx;                                // 基準死亡確率（クラス属性）
    static double  hx;                                // 今期の(ベースライン)ハザード
};
#endif                //person_h

//*************************************************************************
//*****              Person   Ver 1.0 : 個人クラス                        *******
//*****                                          Kaneko <09/02/18>   *******
//*************************************************************************
// ((person 実装 file ))
//person.cpp
#include <stdio.h>          // 一時的 : getchar()
#include <stdlib.h>         // exit(),rand(),RAND_MAX,atoi()
#include <iostream.h>       // cout
#include <string.h>         // stfchr()
//#include <ctype.h>        //
#include <math.h>           // pow()
//#include <fastmath.h>     // pow()...invalid floating point operation(2001/11/15)

//#include "..¥NkStr¥NkStr.h"      // for string 操作
#include "..¥NkDate¥NkDate.h"      // for class NkDate,Duration,
#include "..¥Person¥person.h"      // for class Person,
                                   // Society(society.h),
                                   // NkDate(NkDate.h), macro YEAR_LENGTH(NkDate.h)

//…[ マクロ ]-----------------------------------------------------------
// 一様乱数の発生
//(RAND_MAX = 0x7FFFU = 32,767) : 正規乱数発生のため rand()=0 を除外
//#define UNIFORM       ((double)(rand()+1)/(double)(RAND_MAX+1))

// 一様事象の生起
//#define TRIAL_U(p)   (UNIFORM<(p)?1:0)


//===[ クラス実装 : Person ]=================================================

//…[ グローバル ]-------------------------------------------------------
double Qx[]={
/*
// 1990 年完全生命表女子 qx:0-109 歳、110 歳は、1.0000 に設定
//      0,        1,        2,        3,        4,        5,        6,        7,        8,        9,
0.00417,0.00064,0.00042,0.00027,0.00019,0.00016,0.00016,0.00014,0.00013,0.00012,   //0
0.00011,0.00010,0.00010,0.00011,0.00014,0.00017,0.00020,0.00024,0.00027,0.00029,   //1
0.00030,0.00031,0.00032,0.00033,0.00033,0.00033,0.00032,0.00034,0.00036,0.00039,   //2
0.00042,0.00044,0.00045,0.00049,0.00053,0.00058,0.00063,0.00068,0.00073,0.00079,   //3
0.00089,0.00099,0.00109,0.00117,0.00125,0.00134,0.00147,0.00163,0.00180,0.00198,   //4
0.00217,0.00234,0.00248,0.00264,0.00285,0.00309,0.00338,0.00369,0.00403,0.00441,   //5
0.00481,0.00522,0.00567,0.00619,0.00681,0.00752,0.00834,0.00930,0.01044,0.01176,   //6
0.01324,0.01495,0.01694,0.01917,0.02161,0.02435,0.02757,0.03145,0.03616,0.04156,   //7
0.04785,0.05502,0.06302,0.07176,0.08119,0.09125,0.10249,0.11498,0.12856,0.14327,   //8
0.15876,0.17517,0.19222,0.21044,0.22972,0.24947,0.27002,0.29135,0.31346,0.33631,   //9
```

```
0.35990,0.38417,0.40909,0.43461,0.46065,0.48716,0.51405,0.54122,0.56857,0.59600,   //10
1.00000                                                                              //11
*/
// 1950 年完全生命表女子 qx:0-109 歳、110 歳は、1.0000 に設定
//      0,          1,          2,          3,          4,          5,          6,          7,          8,          9,
0.054628923,0.014389101,0.009788420,0.007410140,0.004431008,0.003264859,0.002252178,0.001717248,0.001477
447,0.001280735,
0.001149641,0.001128818,0.001163358,0.001186911,0.001421701,0.001635241,0.002106389,0.002446299,0.002990
591,0.003415961,
0.003608399,0.004211194,0.004593624,0.004969781,0.005040647,0.004973632,0.005207850,0.005106502,0.005261
986,0.004982743,
0.005090806,0.004866325,0.004926108,0.004962570,0.005120464,0.005134617,0.005210049,0.005138960,0.005400
340,0.005690765,
0.005760814,0.005857121,0.006182653,0.006310310,0.006363160,0.006868432,0.007019907,0.007279084,0.007952
469,0.008415389,
0.009117891,0.009472917,0.010618142,0.010648850,0.012066323,0.012071810,0.013512741,0.014762767,0.016169
850,0.016616563,
0.018032962,0.020164730,0.021587975,0.025737500,0.026333311,0.029301020,0.033107321,0.035523719,0.038976
557,0.044169402,
0.045829628,0.052415501,0.056423312,0.062130394,0.070468315,0.076625994,0.082197176,0.088770089,0.103225
338,0.106696248,
0.125418969,0.134898117,0.151845238,0.161735014,0.181014541,0.195970879,0.217183478,0.236243468,0.258756
395,0.266564339,
0.308688245,0.325070159,0.367899604,0.351325758,0.409405256,0.385567010,0.403669725,0.504807692,0.441860
465,0.631578947,
0.625000000,0.416666667,0.437500000,0.363636364,0.428571429,0.000000000,0.000000000,0.000000000,0.000000
000,0.000000000,
0.666666667
};
//=============================================================================
// static メンバの宣言と定義
Society*   Person::home_society=0;                  // 所属社会(クラス属性)
double*    Person::qx=Qx;                           // 死亡確率ベクトル
double     Person::hx=0.0L;                         // (ベースライン)ハザード

//=============================================================================
// クラス初期化
void Person::ClassInitialize(Society* s)
{
    home_society=s;
}
// クラス終了処理
void Person::ClassFinalize(void)
{
    //cout << "class Person is Finalized!" << endl;
}

#define YI_MEAN   (0.0L)
#define YI_SD     (0.03L)
#define YI_K      (10.0L)
void Person::ClassCommonParameter(void)
{
    // Hannerz,2001(fitted for Sweden, male, 1982)
    static double a0= -4.627L;
    static double a1=   0.208L;
    static double a2=   1.541E-3L;
    static double a3=   4.715E-7L;
    static double c =   0.1379L;

    // f(= 年に換算した時間ユニット)
    double f = home0->TimeUnit0.serial0 / NkDate::YearLength0;

    //double age    = (home0->Now0serial0 -home0->Start0 -home0->Option0->StartAge0).yearf0;
    double age    = (double)( home0->Now0.serial0
                             -home0->Start0.serial0
                             -home0->Option0->StartAge0.serial0
                            ) / NkDate::YearLength0;
```

```
    double x      = age + f/2;
    double Gx     = a0 - (a1/x) + (a2/2)*x*x + (a3/c)*exp(c*x);
    double expGx = exp( Gx );
    double dGdx   = (a1/(x*x)) + a2*x + a3*exp(c*x);
    double Sx     = 1 / ( 1+expGx );
    double fx     = dGdx * (expGx/((1+expGx)*(1+expGx)));

          hx      = fx/Sx; // ベースラインハザードとして格納
}
//=============================================================================
// コンストラクタ
Person::Person(void)
{
    static long i=0;

    // Identification
    ID(i++);

    // 出生
    Birth(2000,1,1);

    // 死亡年月初期化
    DeathDate(NULL_DATE);

    // ロケーション
    Location(0.0L);

    // Frailty
    double p[10];
    p[0]=YI_MEAN; p[1]=YI_SD;
    baptizeFrailty(p);
}
Person::Person(long idn)
{
    // Identification
    ID(idn);

    // 出生
    Birth(2000,1,1);

    // 死亡年月初期化
    DeathDate(NULL_DATE);

    // ロケーション
    Location(0.0L);

    // Frailty
    double p[10];
    p[0]=YI_MEAN; p[1]=YI_SD;
    baptizeFrailty(p);
}
Person::Person(long idn, NkDate birthdate)
{
    // Identification
    ID(idn);

    // 出生
    Birth(birthdate);

    // 死亡年月初期化
    DeathDate(NULL_DATE);

    // ロケーション
    Location(0.0L);

    // Frailty
    double p[10];
```

```
    p[0]=YI_MEAN; p[1]=YI_SD;
    baptizeFrailty(p);
}

// ﾃﾞｽﾄﾗｸﾀ
Person::~Person(void)
{
}
//=============================================================================
// 出生発生
void Person::Birth(NkDate d)
{
    if( BirthDate(d)!=NULL_DATE ) Alive(true);
    else                          Alive(false);
}
void Person::Birth(unsigned y, unsigned m, unsigned d){
    NkDate t(y,m,d);
    if( BirthDate(t)!=NULL_DATE ) Alive(true);
    else                          Alive(false);
}
// 死亡発生
void Person::Death(NkDate d)
{
    if( DeathDate(d)!=NULL_DATE ) Alive(false);
}
void Person::Death(unsigned y, unsigned m, unsigned d){
    NkDate t(y,m,d);
    if( DeathDate(t)!=NULL_DATE ) Alive(false);
}
// -------------------------------------------------------
// 誕生日設定
NkDate Person::BirthDate(NkDate bd){
    return birth_date=bd;
}
NkDate Person::BirthDate(long ld){
    NkDate bd(ld);
    return birth_date=bd;
}
NkDate Person::BirthDate(unsigned y, unsigned m, unsigned d){
    NkDate dd(y,m,d);
    return birth_date=dd;
}
// 死亡日設定
NkDate Person::DeathDate(NkDate dd){
    return death_date=dd;
}
NkDate Person::DeathDate(long ld){
    NkDate dd(ld);
    return death_date=dd;
}
NkDate Person::DeathDate(unsigned y, unsigned m, unsigned d){
    NkDate dd(y,m,d);
    return death_date=dd;
}
// -------------------------------------------------------
// 現在年齢
double Person::Age(void){ return (now() - birth_date).yearf(); }
double Person::Age(NkDate t){
    if( t==NULL_DATE || birth_date==NULL_DATE ) return 0L;
    return (t - birth_date).yearf();
}
// 現在の満年齢
int Person::CompletedAge(void){ return (int)( (now() - birth_date).year() ); }
int Person::CompletedAge(NkDate t){
    if( t==NULL_DATE || birth_date==NULL_DATE ) return 0;
    return (int)( (t - birth_date).year() );
}
```

```
// ------------------------------------------------------
// ロケーション
double Person::Location(double lc)
{
    if( lc>=LOCATION_ORIGIN && lc< LOCATION_CYCLE ) return (location=lc);

    double lf=lc-(long)lc;
    double lx=((long)lc % LOCATION_CYCLE)+lf;
    if( lx< LOCATION_ORIGIN ) lx=LOCATION_CYCLE+lx;

    return (location=lx);
}
// frailty を授ける
void Person::baptizeFrailty(double* p)
{
        // 標準正規乱数
        double u1=UNIFORM, u2=UNIFORM;
        double z=sqrt(-2L*log(u1))*sin(2L*M_PI*u2);

        // Log-Normal 乱数
        Frailty( exp( p[0]+p[1]*z ) );
}
// 時間進行 ==========================================================================
long Person::proceed(void)
{
    int hit=0;

    if( Alive() ){ // 健在なら

        home()->registerAlive();
        hit++;

        // 加齢(aging)
        //aging();

        // 死亡判定
        if( die() ){  // 死亡？
            Death( now() );  // 死亡発生処理
            home()->registerDeath();
        }
    }

    return hit;
}
// 時間後退
long Person::retreat(void)
{
    int exposured=0;
    return exposured;
;
}
// 加齢 ------------------------------------------------------------------
void Person::aging()
{
}

// 死亡判定 ----------------------------------------------------------------
bool Person::die()
{
    return TRIAL_U( probDeath() )? true : false ;
}
//====================================================================================
// 死亡確率
double Person::probDeath(void)
/*
{   // ハザード所与
    double prob = qx[ CompletedAge() ];
```

```
    double f = home()->TimeUnit().serial() / NkDate::YearLength();
    return 1.0L-pow(1.0L-prob,f);
}
{   // Frailty Model
    double mi      = Frailty() * hx;

    double f = home()->TimeUnit().serial() / NkDate::YearLength();
    return 1.0L-exp( -f*mi );  // タイムユニット内の死亡確率に変換して値を戻す
}
*/
{   // Kaneko Model
    double mi      = exp( YI_K*(1.0L-Frailty()) ) * pow( hx, Frailty() );

    return 1.0L-exp( - home()->TUinYear() * mi );
}
//=============================================================================
// 途中状態の出力
void Person::ProcessOut(FILE* fp, int var)
{
    switch(var){
        case  0:  fprintf(fp,",%2d",Alive()); // 生存状態
        default:  fprintf(fp,",   ");              // 空
    }
}
// 出力表のヘッダを出力(静的メンバー関数)
void Person::OutputHead(FILE* fp)
{
    char rcd[2048],tmp[256];

    sprintf(rcd, " ＩＤ"          );
    sprintf(tmp,",生存状態"     );  strcat(rcd,tmp);
    sprintf(tmp,",出生年"       );  strcat(rcd,tmp);
    sprintf(tmp,",出生月"       );  strcat(rcd,tmp);

    sprintf(tmp,",死亡年"       );  strcat(rcd,tmp);
    sprintf(tmp,",死亡月"       );  strcat(rcd,tmp);
    sprintf(tmp,",死亡年齢"     );  strcat(rcd,tmp);

    fprintf(fp,rcd);
}
// 出力表の個人データを出力：上記ヘッダ出力関数と同時に更新すること
void Person::output(FILE* fp)
{
    char rcd[2048],tmp[256];

    sprintf(rcd,"%06ld",ID());
    sprintf(tmp,",%1d",Alive());                    strcat(rcd,tmp); // 最終生存状態
    sprintf(tmp,",%4d",BirthDate().year());         strcat(rcd,tmp); // 出生年
    sprintf(tmp,",%2d",BirthDate().month());        strcat(rcd,tmp); // 出生月

    sprintf(tmp,",%4d",DeathDate().year());         strcat(rcd,tmp); // 死亡年
    sprintf(tmp,",%2d",DeathDate().month());        strcat(rcd,tmp); // 死亡月
    sprintf(tmp,",%5.1f",Age(DeathDate()));         strcat(rcd,tmp); // 死亡年齢

    fprintf(fp,rcd);
}

//=============================================================================
// ﾒﾝﾊﾞ表示
void Person::showProperties(void)
{
    cout << "ID number          = " << ID()                          << endl;
    cout << "Suvival Status     = " << (Alive()?"alive":"dead")      << endl;
    cout << "Date of Birth      = " << &BirthDate()                  << endl;
    cout << "Date of Death      = " << &DeathDate()                  << endl;
    cout << "Completed Age      = " << CompletedAge()                << endl;
```

```
    cout << "Exact       Age     = " << Age()                          << endl;
    cout << "Location            = " << Location()                     << endl;
    cout << "=============================================================" << endl;
}
```

付[1]　日付処理クラスモジュール

```
//*************************************************************************
//*****              NkDate   Ver 1.0 : 日付処理クラス                  *******
//*****                                          Kaneko <08/12/26>   *******
//*************************************************************************
#if !defined(nk_nkdate_h)     //多重インクルード防止
#define nk_nkdate_h

#include <iostream.h>

//===[ Macro & Global 定義 ]=================================================
#define START_YEAR (1500)              // serial 値の起点となる年 1582
                                       // (参考)グレゴリオ暦の起点=1582 年
#define YEAR_LENGTH 365.24219879       // 1 年の日数
#define NULL_DATE   (0L)               // 無効な日付を表す Serial 値、0 であることを保証
                                       // (参考)long の最小値=-2147483648
#define NULL_WEEK   (7)                // 無効な日付の曜日コード(暫定で 7)
//===[ Prototypes ]==========================================================
class Duration;
class NkDate;

//===[ クラス定義 : NkDate ]=================================================
// class NkDate
//     機能 : 日付の処理
//     説明 : Serial 値は、START_YEAR 年 1 月 1 日=1 とした通算日
//            日付の時は、NkDate d1(10,10,10), d2((long)1826);
//                        d1.setNkDate(11,11,11);              など
//            NkDate どうしの引き算(d2-d1)で、Duration 生成(dur=d2-d1)
//            NkDate と Duration の足し算(d1+dur)で、NkDate 生成(d2=d1+dur)
//                date  = date          ,  date  = long
//                date  = date + dur ,  date  = date + long
//                date  = date - dur ,  date  = date - long
//                dur   = date - date, (dur   = date - long)
//                date += dura          ,  date += long
//                date -= dura          ,  date -= long
//                date ++               ,  date --
//                date == date          ,  date == long
//                date  > date          ,  date >= date
//                date  < date          ,  date <= date
//                ( )は、未サポート ... 必要なときに追加(このリストも変更のこと)
//            無効な日付( NULL_DATE )になるのは、
//                (1)年次が START_YEAR 以前を示すとき
//                (2)月が、1 ～12 以外を示すとき
//                (3)日が、1 ～※以外を示すとき(※は各月の最後の日)
//                (4)Serial<=0 のとき
//                以上により無効な日付のときは、Serial = NULL_DATE に設定され、
//                メンバー関数は次の値を返す。ｺﾝｽﾄﾗｸﾄ、代入、単項演算など Serial 変更の
//                際にチェックされ、該当のとき Serial = NULL_DATE に設定される。
//                    serial()    = NULL_DATE
//                    year()      = 0
//                    month()     = 0
//                    day()       = 0
//                    dayw()      = NULL_WEEK (=7, できれば NULL_WEEK=-1 にしたいが)
//                  ( daywname() = "VOID" )
//                ( )は、未サポート ... 必要なときに追加(このリストも変更のこと)
//                NkDate に NULL_DATE を設定するには、NkDate d(0L); NkDate d=0L; など
//-------------------------------------------------------------------------
class NkDate{
```

```
//friend class Duration;

public: // 公開関数

  // コンストラクタ
  NkDate(void);
  NkDate(unsigned y,unsigned m=1,unsigned d=1);
  NkDate(long s){ Serial=s; }
  // コピーコンストラクタ
  NkDate(NkDate &);

  // 演算子
  NkDate    &operator= (NkDate    &d   );
  NkDate    &operator= (long        s   );
  NkDate    &operator+ (Duration &dur);
  NkDate    &operator+ (long        dur);
  NkDate    &operator- (Duration &dur);
  NkDate    &operator- (long        dur);
  Duration &operator- (NkDate    &date);
  NkDate    &operator+=(Duration &dur);
  NkDate    &operator+=(long        d   );
  NkDate    &operator-=(Duration &dur);
  NkDate    &operator-=(long        d   );
  NkDate    &operator++(void            );
  NkDate    &operator--(void            );
  int       operator==(NkDate &d1){return (Serial==d1.Serial)?1:0;}
  int       operator!=(NkDate &d1){return (Serial!=d1.Serial)?1:0;}
  int       operator==(long     d1){return (Serial==d1          )?1:0;}
  int       operator!=(long     d1){return (Serial!=d1          )?1:0;}
  int       operator> (NkDate &d1){return (Serial> d1.Serial)?1:0;}
  int       operator>=(NkDate &d1){return (Serial>=d1.Serial)?1:0;}
  int       operator< (NkDate &d1){return (Serial< d1.Serial)?1:0;}
  int       operator<=(NkDate &d1){return (Serial<=d1.Serial)?1:0;}

  // プロパティ設定
  NkDate&       setDate(unsigned y,unsigned m=1,unsigned d=1);
  NkDate&       setDate(long s);

  // プロパティ取得
  long          serial(void){return Serial;}
  unsigned      year(void);                 // （年月日）の年を返す
  NkDate&       year(unsigned y);           // （年月日）の年を設定
  unsigned      month(void);                // （年月日）の月を返す
  NkDate&       month(unsigned m);          // （年月日）の月を設定
  unsigned      day(void);                  // （年月日）の 日を返す
  NkDate&       day(unsigned d);            // （年月日）の 日を設定
  unsigned      dayw(void);                 // （年月日）の曜日を数値で返す(0 ＝日曜～6＝土曜)
  char          *dayname(void);             // （年月日）の曜日を文字列で返す(Sun,Mon,...)

  // 設定値取得
  static double   YearLength()  { return (sim_mode?360.0L:YEAR_LENGTH); }
  static double   MonthLength() { return (sim_mode? 30.0L:(YEAR_LENGTH/12)); }

  // シミュレーション・プロパティの設定
  // （一度設定するとすべてのインスタンスに適用される。したがってすべてのインス
  //  タンスを生成する前に設定する必要がある・・・途中で変えると動作保証しない）
  static int  setSimMode  (void) { return sim_mode=1; }
  static int  resetSimMode(void) { return sim_mode=0; }
  static int  SimMode      (void) { return sim_mode  ; }
  static int  SimMode      (int m){ return sim_mode=(m==0?0:1); }

  // ストリーム用プリント操作
  void          print(ostream *os){*os << year() << "/" << month() << "/" << day();}

  // プロパティ・プリント操作
  void          printform(unsigned form);
```

```
  private:// 私的関数

    unsigned leapyear(unsigned y);
    unsigned monthsize(unsigned y,unsigned m);
    unsigned dayyear(unsigned y,unsigned m=1,unsigned d=1);
    long     serialdate(unsigned y,unsigned m=1,unsigned d=1);

  private:// 私的変数

    long          Serial;       // START_YEAR 年 1 月 1 日(=1)よりの通算日
    static int    sim_mode;     // (=0)通常、(=1)シミュレーション用
};
// Sim モードの定義
int NkDate::sim_mode=0;

//===[ クラス定義 : Duration ]=================================================
// class Duration
//     機能 : 期間(日数)の処理
//     説明 : Serial 値は、日数
//            (年, 月, 日)表示のとき、月の日数は 31,28,31,30,... の順
//               → このようにしないと 1 年の日数が 365 日にならない
//               → ただし、うるう月(29 日)はない
//            また、0 年、0 月、0 日のように、0 もありうる
//               → 31 日=1 月 0 日、365 日=1 年 0 月 0 日など
//                 dur   = dur          ,  dur   = long
//                 dur   = dur  + dur , (dur   = dur  + long)
//                 date  = dur  + date, (date  = dur  + long)
//                 dur   = dur  - dur , (dur   = dur  - date)
//                 dur  += dur          ,  dur  += long
//                (dur  += date)        , (dur  += long)
//                 dur  -= dur          ,  dur  -= long
//                 dur  ++              ,  dur  --
//                 dur  == dur          , (dur  == long)
//                 dur   > dur          ,  dur  >= dur
//                 dur   < dur          ,  dur  <= dur
//                 ()は、未サポート ... 必要なときに追加(このリストも変更のこと)
//-----------------------------------------------------------------------------
class Duration{
  public: // 公開関数

    // コンストラクタ
    Duration(void);
    Duration(unsigned y,unsigned m=0,unsigned d=0);
    Duration(long s){ Serial=s;}
    // コピーコンストラクタ
    Duration(Duration &d);

    // 演算子
    Duration   &operator=  (Duration &d  );
    Duration   &operator=  (long s       );
    NkDate     &operator+  (NkDate &d    );
    Duration   &operator+  (Duration &dur);
    Duration   &operator-  (Duration &dur);
    Duration   &operator+= (Duration &dur);
    Duration   &operator+= (long d       );
    Duration   &operator-= (long d);
    Duration   &operator-= (Duration &dur);
    Duration   &operator++ (void);
    Duration   &operator-- (void);
    int         operator==(Duration &d){return (Serial==d.Serial)?1:0;}
    int         operator!=(Duration &d){return (Serial!=d.Serial)?1:0;}
    int         operator> (Duration &d){return (Serial> d.Serial)?1:0;}
    int         operator>=(Duration &d){return (Serial>=d.Serial)?1:0;}
    int         operator< (Duration &d){return (Serial< d.Serial)?1:0;}
    int         operator<=(Duration &d){return (Serial<=d.Serial)?1:0;}
```

```
    // プロパティ設定
    Duration      &setDuration(unsigned y,unsigned m=0,unsigned d=0);
    Duration      &setDuration(long s);

    // プロパティ取得
    long          serial(){ return Serial; }
    unsigned      year(void);                 //  (年月日) の年を返す
    Duration&     year(unsigned y);           //  (年月日) の年を設定
    unsigned      month(void);                //  (年月日) の月を返す
    Duration&     month(unsigned m);          //  (年月日) の月を設定
    unsigned      day(void);                  //  (年月日) の 日を返す
    Duration&     day(unsigned d);            //  (年月日) の 日を設定
    double        yearf();                    // 年単位に換算した長さを返す

    // ストリーム用プリント操作
    void          print(ostream *os){*os << year() << "/" << month() << "/" << day();}

    // プロパティ・プリント操作
    void          printform(unsigned form);

  private:// 私的関数
    unsigned monthsize(unsigned m);
    unsigned dayyear(unsigned m,unsigned d=0);
    long      serialdate(unsigned y,unsigned m=0,unsigned d=0);

  private:// 私的変数
    long      Serial;       // 日数
};

//===[ Global 演算子 ]=================================================================
ostream &operator<<(ostream &os, Duration &d);//{ d.print(&os); return os; }
ostream &operator<<(ostream &os, NkDate     &d);//{ d.print(&os); return os; }

//===[ 説明 ]==========================================================================
/*
NkDate では、START_YEAR 以降をグレゴリオ暦として曜日計算している。
<< グレゴリオ暦 >>
  現在使用されている暦は、地球が太陽を1周するのに要する日数を平年において365日とするため、
実際の1回帰年=365.2422日都の間に、毎年0.2422日分の誤差を生ずる。これを修正するために、
4で割り切れる年を閏年（1年=366日)、100で割り切れる年を平年、さらに400で割り切れる年を
閏年とする（グレゴリオ暦)。
  かつてヨーロッパで用いられたユリウス暦においては、同様に4で割り切れる年を必ず閏年としたが、
それ以外の修正を行わなかったため、誤差が累積し季節とのずれが甚だしくなった。このためユリウス
暦1582年10月5日に、これをグレゴリオ暦の10月15日として切り替えが行われた（ただし、国や
地域によって採用の時期は異なる)。
*/
//=====================================================================================

#endif                //nk_nkdate_h 多重インクルード防止


//*************************************************************************
//*****              NkDate   Ver 1.0 : 日付処理クラス                  *******
//*****                                          Kaneko <08/12/26>   *******
//*************************************************************************
// (( nkdate 実装 file ))

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <iostream.h>

#include "..¥NkDate¥nkdate.h"
```

```
//===[ Macro]=================================================================
#define START_LEAP (floor(START_YEAR/4)-floor(START_YEAR/100)+floor(START_YEAR/400))
                                        // START_YEAR 年までの閏年の数(1500 年なら=363)
                                        // serialdate()で使用
/*-------------------------------------------------------------------------*/
char        *MONTH[]={"Jan","Feb","Mar","Apr","May","Jun",
                      "Jul","Aug","Sep","Oct","Nov","Dec","VOID"};
unsigned MONTHSIZE[]      ={31,28,31,30,31,30,31,31,30,31,30,31};
unsigned MONTHSIZE_SIM[]={30,30,30,30,30,30,30,30,30,30,30,30};
char        *DAYWEEK[]={"Sun","Mon","Tue","Wed","Thu","Fri","Stu","VOID"};

ostream &operator<<(ostream &os, Duration &d){ d.print(&os); return os; }
ostream &operator<<(ostream &os, NkDate     &d){ d.print(&os); return os; }

//===[ クラス実装 : NkDate ]==================================================
// class NkDate
//      機能：日付の処理
//      説明：Serial 値は、1900 年 1 月 1 日=1 とした通算日
//              期間の時は、NkDate d(10,10,10,DURATION); の様に最後に DURATION を!
//              日付の時は、NkDate d(10,10,10,DATE); の様に最後に DATE を(省略化)!
//-------------------------------------------------------------------------
// コンストラクタ：形式 1
NkDate::NkDate(void)
  :Serial(NULL_DATE){}

// コンストラクタ：形式 2
NkDate::NkDate(unsigned y,unsigned m,unsigned d)
{
    // 引数が無効な日付のとき
    if( y<START_YEAR || y==0      ) Serial=NULL_DATE;
    if( m<1 || m>12               ) Serial=NULL_DATE;
    if( d<1 || d>monthsize(y,m) ) Serial=NULL_DATE;

    // 有効な日付のとき
    Serial= serialdate(y,m,d);
}

// コピーコンストラクタ
NkDate::NkDate(NkDate &d)
  :Serial(d.Serial){}

// 演算子：date = date
NkDate &NkDate::operator=(NkDate &d)
{
    if(this!=&d) Serial=d.Serial;
    return *this;
}
// 演算子：date = long
NkDate &NkDate::operator=(long s)
{
    if( s<=0 ) Serial = NULL_DATE;
    else       Serial = s;
    return *this;
}
// 演算子：date + dur
NkDate &NkDate::operator+(Duration &dur)
{
    static NkDate datestat((long)0);
    long result=Serial+dur.serial();
    return datestat.setDate(result);
}
// 演算子：date + days
NkDate &NkDate::operator+(long days)
{
    static NkDate datestat((long)0);
    long result=Serial+days;
```

```
    return datestat.setDate(result);
}
// 演算子：date - dur
NkDate &NkDate::operator-(Duration &dur)
{
    static NkDate datestat((long)0);
    long result=Serial-dur.serial();
    if( result<=0 ) result=NULL_DATE;
    return datestat.setDate(result);
}
// 演算子：date - days
NkDate &NkDate::operator-(long days)
{
    static NkDate datestat((long)0);
    long result=Serial-days;
    if( result<=0 ) result=NULL_DATE;
    return datestat.setDate(result);
}
// 演算子：date - date
Duration &NkDate::operator-(NkDate &date)
{
    static Duration durstat((long)0);
    long result=Serial-date.Serial;
    return durstat.setDuration(result);   // (result>0?result:-result);
}
// 演算子：date ++
NkDate &NkDate::operator++()
{
    Serial++;
    return *this;
}
// 演算子：date --
NkDate &NkDate::operator--()
{
    Serial--;
    if( Serial<NULL_DATE ) Serial=NULL_DATE;
    return *this;
}
NkDate &NkDate::operator+=(long d)
{
    Serial+=(long)d;
    if( Serial<NULL_DATE ) Serial=NULL_DATE;
    return *this;
}
NkDate &NkDate::operator+=(Duration &dur)
{
    Serial+=dur.serial();
    if( Serial<NULL_DATE ) Serial=NULL_DATE;
    return *this;
}
NkDate &NkDate::operator-=(long d)
{
    Serial-=(long)d;
    if( Serial<NULL_DATE ) Serial=NULL_DATE;
    return *this;
}
NkDate &NkDate::operator-=(Duration &dur)
{
    Serial-=dur.serial();
    if( Serial<NULL_DATE ) Serial=NULL_DATE;
    return *this;
}

// プロパティ取得関数：year
unsigned NkDate::year(void)
{
    if( Serial<=NULL_DATE ) return 0;
```

```
    unsigned y=(unsigned)floor( Serial/(SimMode0?360L:365L) )+START_YEAR;
    while( serialdate(y)>Serial ) y--;
    return y;
}
NkDate &NkDate::year(unsigned y)
{
    // 引数が無効な日付のとき
    if( y<START_YEAR || y==0      ) Serial=NULL_DATE;
    // 有効な日付のとき
    Serial= serialdate(y,month0,day0);
    return *this;
}
// プロパティ取得関数：month
unsigned NkDate::month(void)
{
    if( Serial<=NULL_DATE ) return 0;
    unsigned y = year0;
    unsigned td=(unsigned)(Serial-serialdate(y))+1; // 年初からの通算日計算
    unsigned m;
    for(m=1; td>monthsize(y,m); m++) td-=monthsize(y,m); // 月を順に除去
    return m;
}
NkDate& NkDate::month(unsigned m)
{
    // 引数が無効な日付のとき
    if( m<1 || m>12                ) Serial=NULL_DATE;
    // 有効な日付のとき
    Serial= serialdate(year0,m,day0);
    return *this;
}
// プロパティ取得関数：day
unsigned NkDate::day(void)
{
    if( Serial<=NULL_DATE ) return 0;
    return (unsigned)(Serial-serialdate(year0,month0))+1;
}
NkDate& NkDate::day(unsigned d)
{
    // 引数が無効な日付のとき
    if( d<1 || d>monthsize(year0,month0) ) Serial=NULL_DATE;
    // 有効な日付のとき
    Serial= serialdate(year0,month0,d);
    return *this;
}
// プロパティ取得関数：dayw
unsigned NkDate::dayw0
{
    if( Serial<=NULL_DATE ) return NULL_WEEK;

    // 以下の式は 1582 年 10 月 15 日以降で有効
    unsigned y = year0;
    unsigned m = month0;
    unsigned d = day0;
    if(m<=2){y--;m+=12;}
    return (y+y/4-y/100+y/400+(13*m+8)/5+d)%7;
}
char *NkDate::daywname0
{
    // if( Serial<=NULL_DATE ) return NULL_WEEKDAY;
    return DAYWEEK[dayw0];
}
void NkDate::printform(unsigned form)
{
    switch(form){
        case 1:
                printf("%4d/%2d/%2d/",year0,month0,day0);
```