```
  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 ﾒﾝﾊﾞの宣言と定義
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 = home()->TimeUnit().serial() / NkDate::YearLength();

    //double age    = (home()->Now()serial() -home()->Start() -home()->Option()->StartAge()).yearf();
    double age    = (double)( home()->Now().serial()
                              -home()->Start().serial()
                              -home()->Option()->StartAge().serial()
                            ) / NkDate::YearLength();
```

```
    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, " I D"          );
    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)
//                  ( dayname() = "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/(SimMode()?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,month(),day());
    return *this;
}
// プロパティ取得関数：month
unsigned NkDate::month(void)
{
    if( Serial<=NULL_DATE ) return 0;
    unsigned y = year();
    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(year(),m,day());
    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(year(),month()))+1;
}
NkDate& NkDate::day(unsigned d)
{
    // 引数が無効な日付のとき
    if( d<1 || d>monthsize(year(),month()) ) Serial=NULL_DATE;
    // 有効な日付のとき
    Serial= serialdate(year(),month(),d);
    return *this;
}
// プロパティ取得関数：dayw
unsigned NkDate::dayw()
{
    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 = year();
    unsigned m = month();
    unsigned d = day();
    if(m<=2){y--;m+=12;}
    return (y+y/4-y/100+y/400+(13*m+8)/5+d)%7;
}
char *NkDate::daywname()
{
    // if( Serial<=NULL_DATE ) return NULL_WEEKDAY;
    return DAYWEEK[dayw()];
}
void NkDate::printform(unsigned form)
{
    switch(form){
        case 1:
                printf("%4d/%2d/%2d/",year(),month(),day());
```

```
                printf("(%s)",daywname());
                printf(" [%ld]¥n",Serial);
                break;
        default:
                printf("%4d 年%2d 月%2d 日",year(),month(),day());
                printf("(%s)",daywname());
                printf(" [%ld]¥n",Serial);
    }
}
NkDate &NkDate::setDate(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; return *this; }
    if( m<1 || m>12               ){ Serial=NULL_DATE; return *this; }
    if( d<1 || d>monthsize(y,m) ){ Serial=NULL_DATE; return *this; }

    // 有効な日付のとき
    Serial= serialdate(y,m,d);
    return *this;
}
NkDate &NkDate::setDate(long s)
{
    if( s<=0 ) Serial = NULL_DATE;
    else       Serial = s;
    return *this;
}
/*...[ Slaves ].....*/
unsigned NkDate::leapyear(unsigned y)
{
    // 引数が無効な日付のとき
    if( y<START_YEAR ) return 0;

    // 有効な日付のとき

    // Simulation 用(うるう年なし)
    if( SimMode() ) return 0;

    // 0:平年,1：うるう年(4 で割り切れて、かつ 100 で割れないか 400 で割り切れる年)
    return ((y%4==0 && y%100!=0) || y%400==0);
}
// 指定月の日数を返す(28,29,30,31 のどれか)。
unsigned NkDate::monthsize(unsigned y,unsigned m)
{
    if(m==2 && leapyear(y)) return 29;
    if( SimMode() ) return MONTHSIZE_SIM[m-1];
    else            return MONTHSIZE[m-1];
}
// 指定年月日の元旦からの通算日を返す（1～366, 無効=0)。
unsigned NkDate::dayyear(unsigned y,unsigned m,unsigned d)
{
    // 引数が無効な日付のとき
    if( y<START_YEAR || y==0      ) return 0;
    if( m<1 || m>12               ) return 0;
    if( d<1 || d>monthsize(y,m) ) return 0;

    // 有効な日付のとき
    unsigned    td=0;
    for(unsigned i=0; i<(m-1); i++){
        if( SimMode() ) td+=MONTHSIZE_SIM[i];
        else            td+=MONTHSIZE[i];
    }
    td+=d;
    if(m>2 && leapyear(y)) td++;
    return td;
}
// START_YEAR 年 1 月 1 日＝1 とした年月日の通算日の計算
long NkDate::serialdate(unsigned y,unsigned m,unsigned d)
```

```
{
    // 引数が無効な日付のとき
    if( y<START_YEAR || y==0      ) return NULL_DATE;
    if( m<1 || m>12               ) return NULL_DATE;
    if( d<1 || d>monthsize(y,m) ) return NULL_DATE;

    // 有効な日付のとき
    // Sim モードのとき
    if( SimMode() ){
        return 360L*(long)(y-START_YEAR) + (long)dayyear(y,m,d);
    }

    // 通常モードのとき
    unsigned py = y-1;  // 前年まで(前年を含める)のうるう年の回数を求める
    unsigned n_leap=
        ((unsigned)floor(py/4)-(unsigned)floor(py/100)+(unsigned)floor(py/400))
        - START_LEAP;  // y年がうるう年の場合も dayyear() がそれを考慮する
    return 365L*(long)(y-START_YEAR) + (long)n_leap + (long)dayyear(y,m,d);
}

//===[ クラス実装 : 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
//-----------------------------------------------------------------------------
// コンストラクタ
Duration::Duration(void)
:Serial(0)
{
}
Duration::Duration(unsigned y,unsigned m,unsigned d)
{
    Serial= serialdate(y,m,d);
}
// コピーコンストラクタ
Duration::Duration(Duration &d)
:Serial(d.Serial)
{
}
// 演算子 : dura = drua
Duration &Duration::operator=(Duration &d)
{
    if(this!=&d) Serial=d.Serial;
    return *this;
}
// 演算子 : dura = long
Duration &Duration::operator=(long s)
{
    Serial= s;
    return *this;
}
// 演算子 : + (long)
NkDate      &Duration::operator+(NkDate &d)
{
    static NkDate datestat((long)0);

    long dif=Serial+d.serial();
    return datestat.setDate(dif);
}
// 演算子 : + (Duration)
Duration    &Duration::operator+(Duration &dur)
```

```
{
    static Duration durstat((long)0);

    long dif=Serial+dur.serial();
    return durstat.setDuration(dif);
}
Duration    &Duration::operator-(Duration &dur)
{
    static Duration durstat((long)0);

    long dif=Serial-dur.serial();
    return durstat.setDuration(dif);
}
Duration    &Duration::operator++(void)
{
    Serial++;
    return *this;
}
Duration    &Duration::operator--(void)
{
    Serial--;
    return *this;
}
Duration    &Duration::operator+=(long d)
{
    Serial+=d;
    return *this;
}
Duration    &Duration::operator+=(Duration &dur)
{
    Serial+=dur.Serial;
    return *this;
}
Duration    &Duration::operator-=(long d)
{
    Serial-=d;
    return *this;
}
Duration    &Duration::operator-=(Duration &dur)
{
    Serial-=dur.Serial;
    return *this;
}

// プロパティ設定
Duration &Duration::setDuration(unsigned y,unsigned m,unsigned d)
{
    Serial= serialdate(y,m,d);
    return *this;
}
Duration &Duration::setDuration(long s)
{
    Serial=s;
    return *this;
}

// プロパティ取得

// （年月日）の年を返す
unsigned Duration::year(void)
{
    return (unsigned)floor(Serial/(NkDate::SimMode()?360L:365L));
}
// 年単位に換算した長さを返す
double Duration::yearf()
{
    return (double)Serial/(NkDate::SimMode()?360L:365L);
}
```

```
// （年月日）の年を設定
Duration& Duration::year(unsigned y)
{
    Serial= serialdate(y,month0,day0);
    return *this;
}
// （年月日）の月を返す
unsigned Duration::month(void)
{
    unsigned y = year0;
    unsigned td=(unsigned)(Serial-serialdate(y)); // 年初からの通算日計算
    unsigned m;
    for(m=0; td>monthsize(m); m++) td-=monthsize(m); // 月を順に除去
    return m;
}
// （年月日）の月を設定
Duration& Duration::month(unsigned m)
{
    Serial= serialdate(year0,m,day0);
    return *this;
}
// （年月日）の 日を返す
unsigned Duration::day(void)
{
    return (unsigned)(Serial-serialdate(year0,month0));
}
// （年月日）の 日を設定
Duration& Duration::day(unsigned d)
{
    Serial= serialdate(year0,month0,d);
    return *this;
}
void Duration::printform(unsigned form)
{
    switch(form){
        case 1:
                printf("%4d/%2d/%2d/",year0,month0,day0);
                printf(" [%ld]¥n",Serial);
                break;
        default:
                printf("%4d 年%2d 月%2d 日",year0,month0,day0);
                printf(" [%ld]¥n",Serial);
    }
}
/*...[ Slaves ].....*/
// 指定月(0ヶ月目～11ヶ月目)の日数を返す(28,29,30,31 のどれか)。
unsigned Duration::monthsize(unsigned m)
{
    if( NkDate::SimMode() ) return MONTHSIZE_SIM[m];
    else                    return MONTHSIZE[m];
}
// （月日)表示の日数の通算日数の計算．(15 月 1 日)、(0 月 151 日)なども受付
unsigned Duration::dayyear(unsigned m,unsigned d)
{
    unsigned    td=0;
    for(unsigned i=0; i<m; i++){
        if( NkDate::SimMode() ) td+=MONTHSIZE_SIM[i%12];
        else                    td+=MONTHSIZE[i%12];
    }
    td+=d;
    return td;
}
// (年月日)表示の期間の計算．(1 年 15 月 1 日)、(1 年 0 月 151 日)なども受付
long Duration::serialdate(unsigned y,unsigned m,unsigned d)
{
    return (NkDate::SimMode()?360L:365L)*(long)y + (long)dayyear(m,d);
}
```

Ⅱ. 個別研究報告

（分析研究：子育て環境をライフコースの観点から探る）

【出生児調査】

11　21世紀出生児縦断調査に基づく子供の成長パターンの測定（III）

北村行伸*

概 要

本論文では、厚生労働省（大臣官房統計情報部）によって始められた21世紀出生児縦断調査を用いて、新生児の生育（身長・体重）を時間とともに追い、子供の成長のパターンが個人の初期条件（出生時の体重・身長など）、その後の条件（養育費）や個人差（男女、生年月）などによってどのように違ってくるかを分析した。パネルデータの特徴を生かして推定するとほとんどの場合、固定効果推定が選択されることがわかり、産まれた時の初期値の違いだけではなく、親からの遺伝情報や経済状態も影響を与えていることが推測された。また、初期条件の違いが、子供のその後の成長にどのような影響を与えているかを体重と身長の一日当たりの成長で見ると、初期値が小さい子供ほど成長率が高く、キャッチアップが行われていることが判った。

Key words: 身体成長、新生児、初期条件、パネル調査

1．はじめに

21世紀の幕開けとともに始められた『21世紀出生児縦断調査』は2001年1月と7月に生まれたそれぞれ2万人以上の子供の成長を継続的に追っていくことにより、少子化対策等厚生労働行政施策の企画立案、実施等のための基礎資料を得ることを目的としたプロジェクトである。調査の対象は平成13年（2001年）1月10日-17日生まれかあるいは同年7月10日-17日に生まれた全ての子供である。調査時期は1月出生児は平成13年8月1日現在、7月出生児は平成14年2月1日現在としてある[1]。調査事項は保育者、同居者、就業状況、労働時間、父母の家事・育児の分担状況、住居の状況、子育てで意識して行っていること、子供をもってよかったと思うこと、子供をもって負担に思うこと、子育ての不安や悩みの状況、授乳の状況、収入の状況等多岐にわたっている。調査方法は厚生労働省が人口動態調査出生票を基に調査対象を抽出し、対象世帯に対して質問票を配布し、回収は郵送によって行った。調査票の回収状況は1月出生児26620人に対して回収数23421人で

* 本論文は国立社会保障人口問題研究所内で組織された厚生労働省科学研究費補助金統計情報総合研究事業『パネル調査（縦断調査）に関する統合的高度統計分析システムの開発研究』（課題番号H20-統計-003）で行なわれた研究成果をまとめたものである。本論分で用いたプログラムおよび論文の構成は北村（2007, 2008）に基づくものであるが、『第6回21世紀出生時縦断調査』の結果を加えて、再推計し、新しい解釈を加えている。

[1]しかし、後に論じるように身体測定の日付は分散しており、調査日をもって全てのデータの記録日であると判断するのは間違いである。

あり、回収率は88.0% 、7月出生児26955人に対して、23589人が回答し、回収率は87.5%となっている。両月出生児を合計した第1回調査全体の標本数は47010人である。さらに、第2回1月出生児の回収数は21923人、7月出生児の回収数は22002人、合計43925人となっている。第3回1月出生児の回収数は21365人、7月出生児の回収数は21447人、合計42812人。第4回1月出生児の回収数は20699人、7月出生児の回収数は20860人、合計41559人。第5回1月出生児の回収数は19824人、7月出生児の回収数は19993人、合計39817人。第6回1月出生児の回収数は19154人、7月出生児の回収数は19381人、合計38535人である。第1回を100% とすると第6回で82% の標本が残っており、極めて高い回収率を維持している。

本論文では、身体発育に関するデータを分析するが、これに関しては厚生労働省雇用均等・児童家庭局が『乳幼児身体発育調査』を昭和35年より平成12年まで10年毎にこれまで5回行ってきている。この調査は全国的に乳幼児の身体発育の状態を調査し、新たに我が国の乳幼児の身体発育値を定めて、乳幼児保健指導の改善に資することを目的としている。調査対象は一般調査として、全国の乳幼児を対象として国勢調査地区のなかの3000地区内から調査実施日において生後14日以上2歳未満の乳幼児および3000地区のうちから抽出した900地区内の2歳以上小学校就学前の幼児から選んだ。これに加えて、病院調査として、全国の産科病床を有する病院のうち、医療施設基本ファイルから抽出した病院で出生し、1ヶ月健診を受診した乳幼児から選んだ。調査事項は身長、体重、胸囲、頭囲、運動・言語能力、現症・既往症、栄養状況、妊娠・出産時の状況、出産場所、母親の身長・体重、年齢、雇用状況などを含んでいる。調査方法は一般調査に関しては保健所における乳幼児の一斉健診に合わせて集団調査を行った。病院調査に関しては、病院が被調査乳幼児の調査を実施した。平成12年調査では調査対象は、一般調査で対象者10285世帯、12312人の内、8104世帯、10021人が回答した（回収率81.4% ）。病院調査では136病院、4094人が回答した。

本論文ではクロスセクション調査ではなくパネル調査であることの意義を、上述の2つの調査を用いながら論じたい。言うまでもなく、パネルデータでは同一個人の時間を通した成長を追跡できることが最大の利点である。これまでのパネルデータ分析の経験から言えることであるが、同一の経済主体を継続的に追うことができるというのは、クロスセクションで平均を見るのとは情報量が格段に違い、また、対象として分析できる問題の範囲が広がるという意味でも意義がある。もっとも、パネルデータでは同一のサンプルを追い続けるメリットの反面、それらのサンプルが調査から脱落してしまうと、新しいサンプルを安易に加えることが出来ず、脱落の仕方によっては社会全体から無作為に抽出したサンプルから次第に離れて特定のバイアスをもったサンプルのみが残ることになる。このような問題はクロスセクション調査では起こらない。また、継続して調査しているデータに入力ミスがあると、一時点のミスの影響だけではなく、前後のデータからの変化を見た場合に、異常な動きをすることがある。これは、毎回の調査で確認し、事後的にミスを見つけ