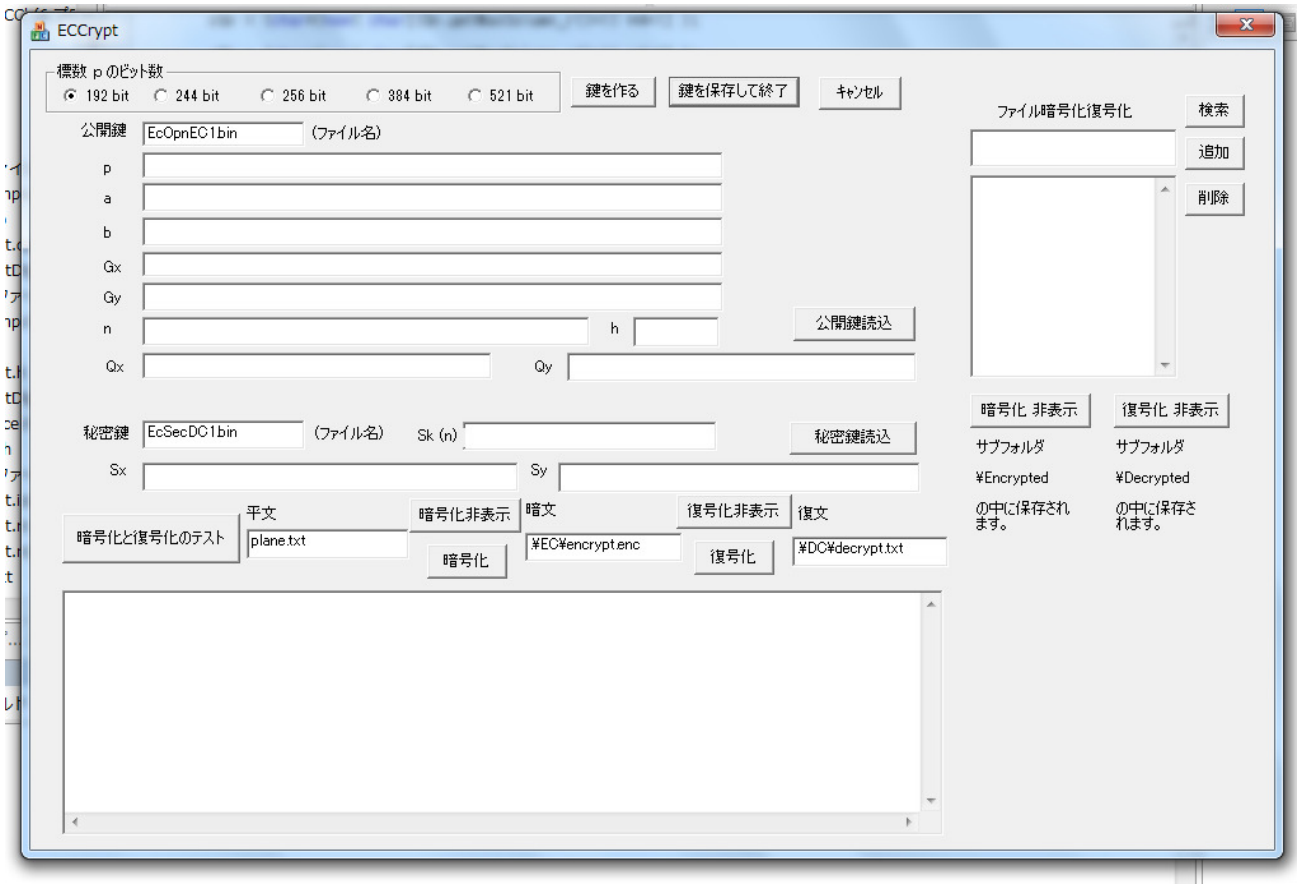
楕円曲線暗号の実装

(C) 宇山　靖政

2013.03.07

ECcrypt.exe　について。

使い方：



通信の仕方は次のようになります。

アリスの作業

1．素数 p のビット数を決定する。192,244,256,384,521 ビットから選ぶ。

2．鍵を作る　ボタンをクリックする。

3．公開鍵、秘密鍵の名前、Ao***, As***　を決める。

4．鍵を保存して終了する。

注意：

現在は、素数のビット数を選ぶと、楕円曲線のパラメータ　T = (p, a, b, G, n, h)　が 1 組決まります。

秘密の値　m　は乱数として決定されます。m　の値は、113 ビットから 392 ビットの間に設定しました。

この値　m　は秘密にしておきます。

5．相手（ボブ）に、(T , mG) を送る。（これが公開鍵）

ボブの作業

1．アリスから送られた公開鍵を読み込む。これで素数 p のビット数が決まる。

2．鍵を作る　ボタンをクリックする。

3．公開鍵、秘密鍵の名前、Bo***, Bs***　を決める。

４．鍵を保存して終了する。
５．アリスの公開鍵を読み込む。自分(ボブ)の秘密鍵を読み込む。
６．ECcrypt.exe　と同じフォルダに対称鍵をおく。
７．暗号化する対称鍵の名前を、平文のところに設定する。
８．暗文、復文の名前も設定する。
９．暗号化のボタンをクリックする。
１０．暗号化された対称鍵をアリスに送る。

アリスの作業
１．ボブから送られてきた暗号化された鍵を暗文のところにセットする。
２．自分（アリス）の秘密鍵を読み込む。
３．復号化のボタンをクリックする。
４．復文の作成し、メールもビトマで利用する。

//

ECcrypt のソースファイル

このソースコードについては、著作権を主張します。

このソフトは、楕円曲線暗号の公開鍵と秘密鍵を作成し、暗号化、復号化の様子を確認するためのソフトです。暗号化したものは、16 進数の列として表示します。

通信の構造は次のようになります。

アリスの作業
１．楕円曲線　T = (p, a, b, G, n, h)　を選ぶ。
２．秘密の値　m　を選ぶ。（これは秘密にしておく）
３．相手（ボブ）に、(T , mG) を送る。（これが公開鍵）

ボブの作業
４．ファイル全体を分割する。
５．分割した各部分を数値 X1　とみなす。
６．整数値　X1+α　に対して、P = (X1+α , Y) が楕円曲線の上に載るようにする。
７．秘密の値　k　を決める。
８．kG を作り、ファイルの先頭に置く。
９．アリスから受け取った、(T , mG) を利用して、P+kmG を計算してファイルに書き込む。
１０．　ファイルの終わりまで繰り返し終了したら、それをアリスに送る。

アリスの作業
１１．ファイルの先頭から、kG を取り出す。
１２．自分の持っている値　m を使って、mkG を計算する。
１３．ファイルの残りの各ブロックに対して、(P+kmG)－mkG ＝P　を計算する。
１４．P　の x 座標（X1+α）を取りだして、X1 = (X1+α)－α　を計算して並べる。

必要となるもの
１．多倍長整数の計算

２．ヤコビ記号の計算
３．平方根の計算
４．楕円曲線上の点の k 倍の計算

楕円曲線暗号を利用した公開鍵暗号のソフトです。“メールもビトマ”、"Cipher Web Mail” における共通鍵の交換のために作成しました。パラメーラーに関しては、

Standards for Efficient Cryptography
SEC 2: Recommended Elliptic Curve Domain Parameters
Certicom Research
Contact: Daniel R. L. Brown (dbrown@certicom.com)
January 27, 2010

にある値を利用しました。

GF p での p の値は、192 ビットから、521 ビットの間です。 k および n の値は、113 ビットから 392 ビットの間に設定しました。
ファイル全体を暗号化出来ますが、とても時間がかかります。RSA 暗号が速く見えるほどです。したがって、対称鍵の暗号化にしか利用できません。もちろん貿易管理令に違反しないようにソースコードを HP で公開します。

多倍長整数の計算は、複素数の配列と多倍長整数の変換を適宜行う方法で全体を扱っています。さらに、3 通りの乗法（複素数の普通の乗法、DFT による乗法、FFT による乗法）を用意して、扱う数の大きさによって切り替えて計算しています。除法と剰余は自分で考えた方法で計算しています。
ヤコビ記号の計算と平方根の計算、素数生成、最大公約数と逆数の計算は Menezes の Handbook of Applied Cryptography (Discrete Mathematics and Its Applications) にあった方法を少し変形して使っています。べき乗計算は FFT を主に利用しています。

全体的な流れは、

Journal of Applied Sciences 5 (4): 604-633, 2005
ISSN 1812-5654

Theory and Implementation of Elliptic Curve Cryptography

Kefa Rabah
Department of Physics, Eastern Mediterranean University, Gazimagusa, North Cyprus, via Mersin 10, Turkey

に沿って作成しました。ご指導いただいたことを感謝しております。

このソースコードについては、著作権を主張します。技術内容を公知の技術にするために、ソースファイルを公開します。

Header ファイル

```
/*
```

```
 * CmplxMPI.h
 * Complex Multi-Precision Integers (複素数による多倍長正整数クラス)
 *
 *
 * Copyright 2013 Uyama Yasumasa.
 */


// 定数
#define COL_LENGTH              16                      // u_short のビット数
#define COL_MASK        0xffff          // 1桁の有効部分取り出し用マスク
#define BASE_UNIT               0x10000         // 基数
#define HALF_UNIT               0x8000          // 1桁の半分
#define COL_CHAR        2                       // 1桁に入るchar数
#define DEF_COLS        4                       // デフォルト時の桁数

// 型の定義
typedef unsigned short  u_short;
typedef unsigned int    u_int;

// エラー種別
/*typedef*/ enum eErrType{
        noError,                                        // エラーなし
        notEnoughMemory,                        // メモリー不足、
        accessError,                                    // アクセスエラー
        divideByZero,                                   // 0による除算
        minusValue,                                             // 結果が負
        notXdigit                                               // 16進数以外の文字を指定
};

// エラー発生関数
/*typedef*/ enum eErrMethod {
        noMethod,                                               //      エラーなし
        C_void,                                                 //      CmplxMPI( void )
        C_int,                                                  //      CmplxMPI( const int )
        C_int2,                                                 //      CmplxMPI( const int, const
int )
        C_size,                                                 //      CmplxMPI( const u_short*,
const int, const int )
        C_str,                                                  //      CmplxMPI( const char* )
        C_CmplxMPI,                                             //      CmplxMPI( const CmplxMPI& )
        F_changeLength,                         //      int     changeLength( int
newLength )
        F_div,                                                  //      CmplxMPI div( const
CmplxMPI& a, const CmplxMPI& b, CmplxMPI& r )
        F_set,                                                  //      void CmplxMPI::set( int n,
u_short val )
        O_equal,                                //      CmplxMPI&
CmplxMPI::operator=( const CmplxMPI& a )
        O_minus                                                 //      CmplxMPI operator-( const
CmplxMPI& a, const CmplxMPI& b )
};
```

```
// エラー管理クラス
class aError
{
public:
        eErrTypec;                                              // エラー種別
        eErrMethod      f;                                              // エラー発生関数
        size_t          s;                                              // 要求したメモリサイズ

        // コンストラクタ
        aError( eErrType st, eErrMethod sf, size_t ss )
        {       c = st; f = sf; s = ss; };
};

// 多倍長正整数クラス
class CmplxMPI
{
public:
//protected:
        double* rv;                                             // 値(実正整数):領域はコンストラクタが割り
当てる。rv[0]がもっとも下位
        int                     rl;                                     // 桁数(rvが指す領域の長さ)
        double* iv;                                             // 値(虚正整数):領域はコンストラクタが割り
当てる。iv[0]がもっとも下位
        int                     il;                                     // 桁数(ivが指す領域の長さ)


public:
        int rs;                 //実部の符号
        int is, is2;            //虚部の符号および虚部があることを示す。


// コンストラクタ (型変換)
        CmplxMPI( void );                                       // デフォルト
        CmplxMPI(                                               // intからの型変換
                const int );                                    // int値
        CmplxMPI(                                               // 桁数を指定して実部のすべての桁
をvalでうめる(通常かffff)
                const int,                                      // 要求桁数
                const int );                                    // 値 (下位ビット有効)

        CmplxMPI(                                               // 桁数を指定して実部、虚部すべての
桁をvalでうめる(通常かffff)
                const int,
                const int,
                const int,                                      // 要求桁数
                const int );                                    // 値 (下位ビット有効)

        CmplxMPI(                                               // メモリ領域から
                const u_short*,                                 // 配列
                const int );                                    // 長さ
        CmplxMPI(                                               // 複素型のメモリ領域から
```

```
                const double*,
                const int,
                const double*,                          // 配列
                const int );                            // 長さ

        CmplxMPI(                                               // 16進数文字列から
                const char* );                          // 文字列
        CmplxMPI(                                               // コピー(領域を別に持つ)
                const CmplxMPI& );                      // 元の数値

// 桁数を指定してすべての桁をvalでうめる(通常かffff)
CmplxMPI::CmplxMPI(
        const int rsize,                                // 要求桁数
        const int rval,                 // 実部の値
        const int ival);                                        // 虚部の値（下位ビットのみ有効）

// デストラクタ
        ~CmplxMPI( void );                                      // デフォルト

// 値の操作・参照・出力
public:
        void    getText_r(                              // 実部の数値をストリングに変換
                                char* );                // 出力先
        void    getText(                        // 数値をストリングに変換
                                char* );                // 出力先
        int             copyTo_r(                               // 直接出力
                                char* );                // 出力先
        int             copyTo(                                 // 直接出力
                                char* );                // 出力先
        void    print( void );                  // 値の標準出力
        void    eprint( void );                 // 値のエラー出力
        void    sprint( char* );        // 値のストリング化
        int             getBitLength( void ) const;             // bit長(1である最高のbit位置)を返
す
        int             getMaxColumn_r( void ) const;   // 実部のでない最高桁を返す
        int             getMaxColumn_i( void ) const;   // 虚部のでない最高桁を返す
        int             getMaxColumn( void ) const;             // 0でない最高桁を返す

// 長さの変更

        int             changeLength(                   // データ長を指定桁数より大きいDEF_COLSの倍
数に変更する
                                int, int );                     // 実部および虚部の指定桁数
        void    adjustLength( void );   // データ長を最小のDEF_COLS倍にする
        void    torealpart(void);

        // 演算子
        CmplxMPI&               operator=( const CmplxMPI& );
        // 代入
        friend CmplxMPI  operator+( const CmplxMPI&, const CmplxMPI& );     // 加算
        friend CmplxMPI  operator-( const CmplxMPI&, const CmplxMPI& );     // 減算
        friend CmplxMPI  operator*( const CmplxMPI&, const CmplxMPI& );     // 乗算
```

```
//      friend CmplxMPI  operator/( const CmplxMPI&, const CmplxMPI& );     // 除算
//      friend CmplxMPI  operator%( const CmplxMPI&, const CmplxMPI& );     // 剰余算

// 除算の補助
/*
        friend CmplxMPI  div(                           // 正規化と除算の実行
                                const CmplxMPI&, // 被除数
                                const CmplxMPI&, // 除数
                                CmplxMPI& );            // 剰余(結果)
*/
        friend CmplxMPI  quo(                           // 除算の実行
                                const CmplxMPI&, // 被除数
                                const CmplxMPI&         // 除数
                                );                                          // 商を返す

        friend CmplxMPI  res(                           // 除算の実行
                                const CmplxMPI&, // 被除数
                                const CmplxMPI&         // 除数
                                 );                                         // 剰余を返す

        CmplxMPI operator-( void );             // 単項演算子(マイナス)

        CmplxMPI operator++( void );            // 前置インクリメント
        CmplxMPI operator++( int );             // 後置インクリメント
        CmplxMPI operator--( void );            // 前置デクリメント
        CmplxMPI operator--( int );             // 後置デクリメント

        friend CmplxMPI  operator>>( const CmplxMPI&, const int ); // 右シフト
        friend CmplxMPI  operator<<( const CmplxMPI&, const int ); // 左シフト

        CmplxMPI&       operator+=( const CmplxMPI& );  // 代入和
        CmplxMPI&       operator-=( const CmplxMPI& );  // 代入差
        CmplxMPI&       operator*=( const CmplxMPI& );  // 代入積
        CmplxMPI&       operator/=( const CmplxMPI& );  // 代入商
        CmplxMPI&       operator%=( const CmplxMPI& );  // 代入剰余

        CmplxMPI&       operator>>=( const int );// 代入右シフト
        CmplxMPI&       operator<<=( const int );// 代入左シフト

// 比較演算子
        friend int      operator==( const CmplxMPI&, const CmplxMPI& );         // 等しい
        friend int      operator!=( const CmplxMPI&, const CmplxMPI& );         // 等しくない
        friend int      operator>=( const CmplxMPI&, const CmplxMPI& );         // 大きいか等しい
        friend int      operator<=( const CmplxMPI&, const CmplxMPI& );         // 小さいか等しい
        friend int      operator>( const CmplxMPI&, const CmplxMPI& );          // 大きい
        friend int      operator<( const CmplxMPI&, const CmplxMPI& );          // 小さい

// 関数
        friend int      isEven(                                 // 偶数判定  偶数:1 奇数:0
                                const CmplxMPI& );      // 被検査数
        friend CmplxMPI  pow(                           // べき乗
                                const CmplxMPI&, // 被べき数
```

```
                                        const CmplxMPI& );      // べき数
        friend CmplxMPI  powFFT(                                // FFTでのべき乗
                                        const CmplxMPI&, // 被べき数
                                        const CmplxMPI& );      // べき数
        friend CmplxMPI  exp(                                   // mod付きべき乗
                                        const CmplxMPI&, // 被べき数
                                        const CmplxMPI&, // べき数
                                        const CmplxMPI& );      // 基数
        friend CmplxMPI  Sqrmp(                                                 // ヤコビ記号計算
                        const CmplxMPI& ,                       // 係数h
                        const CmplxMPI& );                      // 素数p

// 乱数
        friend void      randInitial( void );                   // 乱数生成の初期化
        friend CmplxMPI  random( void );                        // 乱数生成
        friend CmplxMPI  random( const int );      // 乱数生成（桁数指定）
        friend CmplxMPI  randomBit( const int );   // 乱数生成（bit長指定）

        friend CmplxMPI  randomSig( void );                     // 乱数生成
        friend CmplxMPI  randomSig( const int );                // 乱数生成（桁数指定）
        friend CmplxMPI  randomBitSig( const int );             // 乱数生成（bit長指定）

// FFT,DFT
        friend CmplxMPI  pFFT(                  // FFT による積
                        const CmplxMPI&,
                        const CmplxMPI& );
        friend CmplxMPI  pDFT(                                  // DFT による積
                        const CmplxMPI&,
                        const CmplxMPI& );

        friend CmplxMPI  expDFT(                                // DFTでのmod付きべき乗
                                        const CmplxMPI&, // 被べき数
                                        const CmplxMPI&, // べき数
                                        const CmplxMPI& );      // 基数
        friend CmplxMPI  expFFT(                                // FFTでのmod付きべき乗
                                        const CmplxMPI&, // 被べき数
                                        const CmplxMPI&, // べき数
                                        const CmplxMPI& );      // 基数

};      // CmplxMPIここまで


// 整数論
int millerrabin(                                        // 素数：  非素数：
        const CmplxMPI&,
        const int);                                                     // 安全助変数
int             rabin(                                                  // ラビン法(素数判定)素数：
非素数：
                        const CmplxMPI& );                      // 被判定数
int             fermat(                                                 // フェルマーテスト(素数判
定) 素数：  非素数：
                        const CmplxMPI&,                // 係数h
```

```
                        const CmplxMPI& );                  // 素数p
CmplxMPI gcd(                                               // 最大公約数
                        CmplxMPI,                                   // 演算数
                        CmplxMPI );                                 // 演算数
CmplxMPI lcm(                                               // 最小公倍数
                        const CmplxMPI&,                    // 演算数
                        const CmplxMPI& );                          // 演算数
CmplxMPI inv(                                               // 逆数(合同式の解)
                        const CmplxMPI&,                    // 演算数
                        const CmplxMPI& );                          // 法


CmplxMPI ugcd(                                              // 逆数(合同式の解)
                        const CmplxMPI&,                    // 演算数
                        const CmplxMPI& );                          // 法

CmplxMPI uinv(                                              // 逆数(合同式の解)
                        const CmplxMPI&,                    // 演算数
                        const CmplxMPI& );                          // 法

int                     Jacobi(                                     // ヤコビ記号計算（素数判
定）素数：　非素数：
                        const CmplxMPI&,                    // 係数h
                        const CmplxMPI&);                   // 素数p

class EcPoint{
public:
        CmplxMPI y;
        CmplxMPI x;

// コンストラクタ（型変換）
        EcPoint( void );                            // デフォルト
        EcPoint(                                            // intからの型変換
                const CmplxMPI,
                const CmplxMPI);
        EcPoint(                                            // 桁数を指定して実部のすべての桁をvalでう
める(通常かffff)
                const int,                                  // 要求桁数
                const int );                                // 値（下位ビット有効）
        EcPoint(                                            // 16進数文字列から
                const char*,
                const char*     );                          // 文字列
        EcPoint(                                            // コピー(領域を別に持つ)
                const EcPoint& );                           // 元の数値
// デストラクタ
        ~EcPoint( void );                                   // デフォルト

// 値の操作・参照・出力
public:
        void    getText_r(                                  // 実部の数値をストリングに変換
                                char* );                    // 出力先
        void    getText(                            // 数値をストリングに変換
```

```
                        char* );                // 出力先
        int             copyTo_r(                       // 直接出力
                        char* );                // 出力先
        int             copyTo(                         // 直接出力
                        char* );                // 出力先
        void    print( void );                  // 値の標準出力
        void    eprint( void );                 // 値のエラー出力
        void    sprint( char* );        // 値のストリング化
        int             getBitLength( void ) const;             // bit長(1である最高のbit位置)を返
す
        int             getMaxColumn_r( void ) const;   // 実部のでない最高桁を返す
        int             getMaxColumn_i( void ) const;   // 虚部のでない最高桁を返す
        int             getMaxColumn( void ) const;             // 0でない最高桁を返す

// 長さの変更

        int             changeLength(                   // データ長を指定桁数より大きいDEF_COLSの倍
数に変更する
                        int, int );                     // 実部および虚部の指定桁数
        void    adjustLength( void );   // データ長を最小のDEF_COLS倍にする
        void    torealpart(void);

        // 演算子
        EcPoint&        operator=( const EcPoint& );                                    // 代入
        friend EcPoint  operator+( const EcPoint&, const EcPoint& );    // 加算
        friend EcPoint  operator-( const EcPoint&, const EcPoint& );    // 減算
        friend EcPoint  operator*( const EcPoint&, const EcPoint& );    // 乗算
        friend EcPoint  operator/( const EcPoint&, const EcPoint& );    // 除算
        friend EcPoint  operator%( const EcPoint&, const EcPoint& );    // 剰余算

        friend EcPoint  operator*( const CmplxMPI&, const EcPoint& );
        // 加数

// 除算の補助
        friend EcPoint  div(                            // 正規化と除算の実行
                                const EcPoint&, // 被除数
                                const EcPoint&, // 除数
                                EcPoint& );             // 剰余(結果)

        friend EcPoint  quo(                            // 除算の実行
                                const EcPoint&, // 被除数
                                const EcPoint&          // 除数
                                );                                      // 商を返す

        friend EcPoint  res(                            // 除算の実行
                                const EcPoint&, // 被除数
                                const EcPoint&          // 除数
                                 );                                     // 剰余を返す

        friend EcPoint movetoEC (                       // 除算の実行
        //                      int chlen,
                                const CmplxMPI&,
```

```
//                                      int keylen,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&
                                        /*,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&,
                                        const CmplxMPI&*/
                                         );                                    // 剰余を返す

        EcPoint  operator-( void );              // 単項演算子(マイナス)

        EcPoint  operator++( void );             // 前置インクリメント
        EcPoint  operator++( int );              // 後置インクリメント
        EcPoint  operator--( void );             // 前置デクリメント
        EcPoint  operator--( int );              // 後置デクリメント

        friend EcPoint   operator>>( const EcPoint&, const int );  // 右シフト
        friend EcPoint   operator<<( const EcPoint&, const int );  // 左シフト

        EcPoint&operator+=( const EcPoint& );    // 代入和
        EcPoint&operator-=( const EcPoint& );    // 代入差
        EcPoint&operator*=( const EcPoint& );    // 代入積
        EcPoint&operator/=( const EcPoint& );    // 代入商
        EcPoint&operator%=( const EcPoint& );    // 代入剰余

        EcPoint&operator>>=( const int );// 代入右シフト
        EcPoint&operator<<=( const int );// 代入左シフト
};


/***
 * rsa.h
 * FFTRSA暗号関数
 *
 */


// 戻り値
#define NOERROR                  0// エラーなし
#define NOTENOUGHMEMORY  -1// メモリー不足
#define ACCESSERROR              -2// アクセスエラー
#define MATHERROR                -3// 数学的な誤り
#define KEYLENGTHERROR   -4// 鍵長不正
#define OTHERERROR       -5// その他のエラー

// 定数
#define MINKEYLENGTH     32// 最低鍵長（ビット）
```

```
/*********************/
/*** FFTRSA公開鍵暗号***/
/*********************/

// 暗号化
int FFTEccEncryption(// 0:成功　その他:エラー
                const int,// 鍵長
                                        const char* p,
                                        const char* a,
                                        const char* b,
                                        const char* cQx,
                                        const char* cQy,
                                        const char* cSk,
        //                              const char* cGx,
        //                              const char* cGy,
        const char* cSx,
        const char* cSy,
                const int,// 平文長
                const char*,// 平文
                int&,// 暗号文長
                char*& );// 暗号文

// 復号化
int FFTEccDecryption(// 0:成功　その他:エラー
                const int,// 鍵長
                                        const char* p,
                                        const char* a,
                //                      const char* b,
                //                      const char* cQx,
                //                      const char* cQy,
                                        const char* cSk,
                //                      const char* cSx,
                //                      const char* cSy,

                const int,// 暗号文長
                const char*,// 暗号文
                int&,// 平文長
                char*& );// 平文

// RSA公開鍵暗号用の鍵の組を作る (e,d,nを得る)
int     makeEccKeys(// 0:成功　その他:エラー
                        const int,                              // 鍵長k
                        char*&,                                 // 公開鍵e
                        char*&,                                 // 秘密鍵d
                        char*&,                         // 共通公開鍵n(法鍵)
                        char*&,                                 // 公開鍵e
                        char*&,                                 // 秘密鍵d
                        char*&,                         // 共通公開鍵n(法鍵)
                        char*&,                                 // 公開鍵e
                        char*&,                                 // 秘密鍵d
                        char*&,                                 // 公開鍵e
```

```
                        char*&,                                 // 秘密鍵d
                        char*&,                                 // 公開鍵e
                        char*& );                               // 共通公開鍵n(法鍵)



// ECCrypt.h : PROJECT_NAME アプリケーションのメインヘッダーファイルです。
//

#pragma once

#ifndef __AFXWIN_H__
        #error "PCH に対してこのファイルをインクルードする前に'stdafx.h' をインクルードしてください"
#endif

#include "resource.h"           // メインシンボル


// CECCryptApp:
// このクラスの実装については、ECCrypt.cpp を参照してください。
//

class CECCryptApp : public CWinApp
{
public:
        CECCryptApp();

// オーバーライド
        public:
        virtual BOOL InitInstance();

// 実装

        DECLARE_MESSAGE_MAP()
};

extern CECCryptApp theApp;


// ECCryptDlg.h : ヘッダーファイル
//

#pragma once


// CECCryptDlg ダイアログ
class CECCryptDlg : public CDialog
{
// コンストラクション
public:
        CECCryptDlg(CWnd* pParent = NULL);      // 標準コンストラクタ
```

```
// ダイアログデータ
        enum { IDD = IDD_ECCRYPT_DIALOG };


        DWORD blen;//=MAX_PATH+1;
        char bufpath[MAX_PATH+1];

    CString s_addbin;
        CString e_addbin;
    CListBox l_bin;

        int madekey, readeckey, readdckey;
        int k, i_KeyLen;
        CString keylength;

//      CString commonkey1;

        CString opensp;
        CString opensa;
        CString opensb;
        CString opensgx;
        CString opensgy;
        CString opensn;
        CString opensh;
        CString opensqx;
        CString opensqy;
//      CString opensqk;

        CString secretsk;
        CString secretsx;
        CString secretsy;

//      CString commonkey2;
 //     CString secretkey;
        CString planedata_file;
        CString encryptdata_file;
        CString decryptdata_file;
        CString encryptkey_file;
        CString decryptkey_file;
        CEdit   datadisp;

        // ClassWizard は仮想関数のオーバーライドを生成します。
        //{{AFX_VIRTUAL(CRSAcryptDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV のサポート
        virtual void OnOK();
        //}}AFX_VIRTUAL


private:
        void yDisplay(const char *);
```

```
// 実装
protected:
	HICON m_hIcon;

	// 生成された、メッセージ割り当て関数
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();

	afx_msg void MakeKey();
	afx_msg void TestKey();
	afx_msg void ECKRead();
	afx_msg void DCKRead();
	afx_msg void Search2();
	afx_msg void AddList2();
	afx_msg void DelList2();

	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()


public:
	afx_msg void OnEnChangeKeylength();
	afx_msg void OnEnChangeEncryptkeyFile();
	afx_msg void OnBnClickedCancel();
	afx_msg void OnBnClickedEckread();
	afx_msg void OnBnClickedMakekey();
	afx_msg void OnBnClickedOk();
	afx_msg void OnEnChangeCommonkey2();
	afx_msg void OnEnChangeSecretkey();
	afx_msg void OnEnChangeDisplay();
	afx_msg void OnEnChangeOpenkey();
	afx_msg void OnEnChangeCommonkey1();
	afx_msg void OnBnClickedRadio1();
	afx_msg void OnBnClickedRadio2();
	afx_msg void OnBnClickedRadio3();
	afx_msg void OnBnClickedRadio4();
	afx_msg void OnBnClickedRadio5();



	afx_msg void OnBnClickedDckread();
	afx_msg void OnBnClickedButton1();
	afx_msg void OnBnClickedButton2();
	afx_msg void OnEnChangeDecryptkeyFile();
	afx_msg void OnBnClickedButton3();
	afx_msg void OnBnClickedButton4();

	afx_msg void OnBnClickedButton5();
	afx_msg void OnBnClickedButton6();
```

```
};



//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by ECCrypt.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_ECCRYPT_DIALOG              102
#define IDR_MAINFRAME                   128
#define IDS_BSEARCH                     201
#define IDC_DISPLAY                     1000
#define IDC_DECRYPTDATA_FILE            1001
#define IDC_KEYLENGTH                   1002
#define ID_MAKEKEY                      1003
#define IDC_COMMONKEY1                  1004
#define IDC_OPENCA                      1004
#define IDC_OPENKEY                     1005
#define IDC_OPENCP                      1005
#define IDC_COMMONKEY2                  1006
#define IDC_SECRETSY                    1006
#define IDC_SECRETKEY                   1007
#define IDC_SECRETSX                    1007
#define ID_TESTKEY                      1008
#define IDC_PLANEDATA_FILE              1009
#define IDC_ENCRYPTDATA_FILE            1010
#define IDC_ENCRYPTKEY_FILE             1011
#define IDC_DECRYPTKEY_FILE             1012
#define IDC_ECKREAD                     1013
#define IDC_DCKREAD                     1014
#define IDC_RADIO1                      1015
#define IDC_RADIO2                      1016
#define IDC_RADIO3                      1017
#define IDC_BUTTON1                     1018
#define IDC_BUTTON2                     1019
#define IDC_RADIO4                      1020
#define IDC_BUTTON3                     1021
#define IDC_BUTTON4                     1022
#define IDC_BUTTON5                     1023
#define IDC_BUTTON6                     1024
#define IDC_RADIO15                     1025
#define IDC_RADIO16                     1026
#define IDC_RADIO17                     1027
#define IDC_RADIO18                     1028
#define IDC_RADIO19                     1029
#define IDC_EDIT1                       1030
#define IDC_OPENCB                      1030
#define IDC_BUTTON7                     1031
```

```
#define IDC_BUTTON8                     1032
#define IDC_BUTTON9                     1033
#define IDC_EDIT2                       1034
#define IDC_OPENCGX                     1034
#define IDC_BUTTON10                    1035
#define IDC_OPENCGY                     1037
#define IDC_OPENCN                      1038
#define IDC_OPENCH                      1039
#define IDC_OPENCQX                     1040
#define IDC_OPENCQK                     1041
#define IDC_SECRETSK                    1041
#define IDC_OPENCQY                     1042
#define IDC_ADDBIN2                     1132
#define IDC_LISTBIN2                    1133
#define IDC_BSEARCH2                    1134
#define IDC_BADDBIN2                    1135
#define IDC_BDELBIN2                    1136
#define IDC_RADIO5                      1137

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1042
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif



// stdafx.h : 標準のシステムインクルードファイルのインクルードファイル、または
// 参照回数が多く、かつあまり変更されない、プロジェクト専用のインクルードファイル
// を記述します。

#pragma once

#ifndef _SECURE_ATL
#define _SECURE_ATL 1
#endif

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Windows ヘッダーから使用されていない部分を除外します。
#endif

// 下で指定された定義の前に対象プラットフォームを指定しなければならない場合、以下の定義を変更してください。
// 異なるプラットフォームに対応する値に関する最新情報については、MSDN を参照してください。
#ifndef WINVER                          // Windows XP 以降のバージョンに固有の機能の使用を許可します。
#define WINVER 0x0501           // これをWindows の他のバージョン向けに適切な値に変更してください。
#endif
```

```
#ifndef _WIN32_WINNT              // Windows XP 以降のバージョンに固有の機能の使用を許可します。
#define _WIN32_WINNT 0x0501       // これをWindows の他のバージョン向けに適切な値に変更してください。
#endif

#ifndef _WIN32_WINDOWS            // Windows 98 以降のバージョンに固有の機能の使用を許可します。
#define _WIN32_WINDOWS 0x0410 // これをWindows Me またはそれ以降のバージョン向けに適切な値に変更してく
ださい。
#endif

#ifndef _WIN32_IE                         // IE 6.0 以降のバージョンに固有の機能の使用を許可します。
#define _WIN32_IE 0x0600 // これをIE の他のバージョン向けに適切な値に変更してください。
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS        // 一部のCString コンストラクタは明示的です。

// 一般的で無視しても安全なMFC の警告メッセージの一部の非表示を解除します。
#define _AFX_ALL_WARNINGS

//#include "dll.h"

#include <afxwin.h>         // MFC のコアおよび標準コンポーネント
#include <afxext.h>         // MFC の拡張部分


#include <afxdisp.h>        // MFC オートメーションクラス



#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>             // MFC のInternet Explorer 4 コモンコントロールサポート
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                       // MFC のWindows コモンコントロールサポート
#endif // _AFX_NO_AFXCMN_SUPPORT









#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker,"/manifestdependency:¥"type='win32' name='Microsoft.Windows.Common-Controls'
version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='*'¥"")
#elif defined _M_IA64
#pragma comment(linker,"/manifestdependency:¥"type='win32' name='Microsoft.Windows.Common-Controls'
version='6.0.0.0' processorArchitecture='ia64' publicKeyToken='6595b64144ccf1df' language='*'¥"")
#elif defined _M_X64
#pragma comment(linker,"/manifestdependency:¥"type='win32' name='Microsoft.Windows.Common-Controls'
```

```
version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='*'¥"")
#else
#pragma comment(linker,"/manifestdependency:¥"type='win32' name='Microsoft.Windows.Common-Controls'
version='6.0.0.0' processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'¥"")
#endif
#endif
```

C++ ファイル

```
/*
 * CmplxMPI.cpp
 * Class Multi-Precision Integers (複素数による多倍長正整数クラス)
 *
 *
 * Copyright 2013 Uyama Yasumasa.
 */
#include "stdafx.h"
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <math.h>
#include "CmplxMPI.h"

using namespace std;

/**********************/
/*** コンストラクタ***/
/**********************/

// デフォルト
CmplxMPI::CmplxMPI( void )
{
        rs = 1;
        is = 1;
        is2 = 0;

        rv = new( double[DEF_COLS] );     // デフォルトサイズで領域確保
        if( rv == 0 ){                                  // メモリ割り当て失敗時
                rl = 0;                                         // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                      // 上位のエラー処理へ
        }                                                               // DEF_COLS              4
                        // デフォルト時の桁数
        rl = DEF_COLS;                                  //  デフォルト時の桁数,1桁はcharで16 bit
        for(int i=0; i<DEF_COLS; i++)
```

```
                rv[i] = 0.0;                                    // 値の初期化


        iv = new( double[DEF_COLS] );     // デフォルトサイズで領域確保
        if( iv == 0 ){                                          // メモリ割り当て失敗時
                il = 0;                                                 // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                              // 上位のエラー処理へ
        }
        il = DEF_COLS;                                          // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                iv[i] = 0.0;                                    // 値の初期化
}

// 型変換int->CmplxMPI
CmplxMPI::CmplxMPI(
        const int rval)                                         // 初期値
{
        int trval;

        rs = 1;
        is = 1;
        is2 = 0;

        if(rval<0){
                trval = -rval;
                rs = -1;
        }
        else{trval = rval;}


        rv = new( double[DEF_COLS] );     // 実部のデフォルトサイズで領域確保
        if( rv == 0 ){                                          // メモリ割り当て失敗時
                rl = 0;                                                 // 無効
                aError  theError( notEnoughMemory, C_int, sizeof(double) * DEF_COLS );
                throw( theError );                              // 上位のエラー処理へ
        }
        rl = DEF_COLS;                                          // デフォルトの長さ
        for(int i=2; i<DEF_COLS; i++)     // 上位の桁の値を初期化
                rv[i] = 0.0;
        rv[0] = double( trval & COL_MASK );                     // 下位ビット
        rv[1] = double( trval >> COL_LENGTH );    // 上位ビット


        iv = new( double[DEF_COLS] );     // 虚部のデフォルトサイズで領域確保
        if( iv == 0 ){                                          // メモリ割り当て失敗時
                il = 0;                                                 // 無効
                aError  theError( notEnoughMemory, C_int, sizeof(double) * DEF_COLS );
                throw( theError );                              // 上位のエラー処理へ
        }
        il = DEF_COLS;                                          // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)     // 上位の桁の値を初期化
```

```
                iv[i] = 0.0;
}

// 型変換int->CmplxMPI
CmplxMPI::CmplxMPI(
        const int rval,
        const int ival)                                 // 初期値
{
        rs = 1;
        is = 1;
        is2 = 1;

        rv = new( double[DEF_COLS] );     // 実部のデフォルトサイズで領域確保
        if( rv == 0 ){                                  // メモリ割り当て失敗時
                rl = 0;                                         // 無効
                aError  theError( notEnoughMemory, C_int, sizeof(double) * DEF_COLS);
                throw( theError );                      // 上位のエラー処理へ
        }
        rl = DEF_COLS;                                  // デフォルトの長さ
        for(int i=2; i<DEF_COLS; i++)     // 上位の桁の値を初期化
                rv[i] = 0.0;
        rv[0] = double( rval & COL_MASK );              // 下位ビット
        rv[1] = double( rval >> COL_LENGTH );     // 上位ビット


        iv = new( double[DEF_COLS] );     // 虚部のデフォルトサイズで領域確保
        if( iv == 0 ){                                  // メモリ割り当て失敗時
                il = 0;                                         // 無効
                aError  theError( notEnoughMemory, C_int, sizeof(double) * DEF_COLS );
                throw( theError );                      // 上位のエラー処理へ
        }
        il = DEF_COLS;                                  // デフォルトの長さ
        for(int i=2; i<DEF_COLS; i++)     // 上位の桁の値を初期化
                iv[i] = 0.0;
        iv[0] = double( ival & COL_MASK );              // 下位ビット
        iv[1] = double( ival >> COL_LENGTH );     // 上位ビット
}

// 桁数を指定してすべての桁をvalでうめる(通常かffff)
CmplxMPI::CmplxMPI(
        const int rsize,                                // 要求桁数
        const int rval,
        const int ival)                                 // 値（下位ビットのみ有効）
{
        rs = 1;
        is = 1;
        is2 = 1;

        rl = (((rsize - 1) / DEF_COLS) + 1 ) * DEF_COLS;  // 実部の要求桁数を超える倍数
        rv = new( double[rl] );
        if( rv == 0 ){                                  // メモリ割り当て失敗時
                rl = 0;                                         // 無効
```

```
                aError  theError( notEnoughMemory, C_int2, sizeof(double) * rl );
                throw( theError );                          // 上位のエラー処理へ
        }
        for(int i=0; i<rl; i++)                     // すべての桁の値を初期化
                rv[i] = double( rval & COL_MASK );


        il = (((rsize - 1) / DEF_COLS) + 1 ) * DEF_COLS;   // 虚部の要求桁数を超える倍数
        iv = new( double[il] );
        if( iv == 0 ){                                      // メモリ割り当て失敗時
                il = 0;                                             // 無効
                aError  theError( notEnoughMemory, C_int2, sizeof(double) * il );
                throw( theError );                          // 上位のエラー処理へ
        }
        for(int i=0; i<il; i++)                     // すべての桁の値を初期化
                iv[i] = double( ival & COL_MASK );
}


// 桁数を指定してすべての桁をvalでうめる(通常かffff)
CmplxMPI::CmplxMPI(
        const int rsize,                                    // 要求桁数
        const int rval,                                     // 値（下位ビットのみ有効）
        const int isize,                                    // 要求桁数
        const int ival )                                    // 値（下位ビットのみ有効）
{
        rs = 1;
        is = 1;
        is2 = 1;

        rl = (((rsize - 1) / DEF_COLS) + 1 ) * DEF_COLS;   // 実部の要求桁数を超える倍数
        rv = new( double[rl] );
        if( rv == 0 ){                                      // メモリ割り当て失敗時
                rl = 0;                                             // 無効
                aError  theError( notEnoughMemory, C_int2, sizeof(double) * rl );
                throw( theError );                          // 上位のエラー処理へ
        }
        for(int i=0; i<rl; i++)                     // すべての桁の値を初期化
                rv[i] = double( rval & COL_MASK );


        il = (((isize - 1) / DEF_COLS) + 1 ) * DEF_COLS;   // 虚部の要求桁数を超える倍数
        iv = new( double[il] );
        if( iv == 0 ){                                      // メモリ割り当て失敗時
                il = 0;                                             // 無効
                aError  theError( notEnoughMemory, C_int2, sizeof(double) * il );
                throw( theError );                          // 上位のエラー処理へ
        }
        for(int i=0; i<il; i++)                     // すべての桁の値を初期化
                iv[i] = double( ival & COL_MASK );
}
```

```
CmplxMPI::CmplxMPI(                                                    // メモリ領域から
                const u_short* rvi,                                    // 配列
                const int li)                                  // 長さ
{
        int i;

        rs = 1;
        is = 1;
        is2 = 0;

        rv = new( double[li] );
        if( rv == 0 ){                                         // メモリ割り当て失敗時
                aError  theError( notEnoughMemory, C_str, sizeof(double) * li );
                throw( theError );                             // 上位のエラー処理へ
        }
        for(i=0; i<li; i++){
                rv[i] = rvi[i];
        }
    rl = li;

        iv = new( double[DEF_COLS] );     // 虚部はデフォルトサイズで領域確保
        if( iv == 0 ){                                         // メモリ割り当て失敗時
                il = 0;                                                // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                             // 上位のエラー処理へ
        }
        il = DEF_COLS;                                         // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                iv[i] = 0.0;                                           // 値の初期化
}


// メモリ領域から
CmplxMPI::CmplxMPI(
        const double*    rvi,                          // 配列
        const int                rli ,                 // 長さ
        const double*    ivi,                          // 配列
        const int                ili )                 // 長さ

{
        rs = 1;
        is = 1;
        is2 = 1;

        rv = new( double[rli] );
        if( rv == 0 ){                                         // メモリ割り当て失敗時
                aError  theError( notEnoughMemory, C_str, sizeof(double) * rli );
                throw( theError );                             // 上位のエラー処理へ
        }
        memcpy( rv, rvi, rli*(sizeof(double)) );  // bit数をbyte数へ変換してコピー
    rl = rli;
```

```
        iv = new( double[ili] );
        if( iv == 0 ){                                  // メモリ割り当て失敗時
                aError  theError( notEnoughMemory, C_str, sizeof(double) * ili );
                throw( theError );                      // 上位のエラー処理へ
        }
        memcpy( iv, ivi, ili*(sizeof(double)) );  // bit数をbyte数へ変換してコピー
    il = ili;
}


// 16進数文字列から
CmplxMPI::CmplxMPI(
        const char* str )                               // 文字列
{
        char *iptr;
        char *psptr;
        char *nsptr;
        int strLength;
        char rch[650];  // 2560/4 + おまけ= 640+10 = 650
        char ich[650];
        char tch[650];
        is2 = 1;

        strcpy_s(tch,str);

        iptr = (char *)strstr( str,"i" );

        if(iptr==NULL){// a 実数の場合
                strcpy_s(ich,"0"); is=1;
                // 符号を取る
                if( str[0] != '-' && str[0] != '+' ){
                        rs = 1;
                        strcpy_s(rch,str);
                }
                if( str[0] == '-' ){
                        rs = -1;
                        str++;
                        strcpy_s(rch,str);
                }
                if( str[0] == '+' ){
                        rs = 1;
                        str++;
                        strcpy_s(rch,str);
                }
        }
        else{// 複素数（虚部あり）
                        // 符号を取る
                if( str[0] != '-' && str[0] != '+' ){
                        rs = 1; is=1;
                        strcpy_s(tch,str);
                }
                if( str[0] == '-' ){
```

```
                rs = -1; is=-1;
                str++;
                strcpy_s(tch,str);
        }
        if( str[0] == '+' ){
                rs = 1; is=1;
                str++;
                strcpy_s(tch,str);
        }
        psptr = NULL;
        nsptr = NULL;
        psptr = (char *)strstr(tch,"+" );
        nsptr = (char *)strstr(tch, "-");
        if((nsptr==NULL)&&(psptr==NULL)){//純虚数の場合
                rs = 1;//純虚数なので実部の符号修正
                strcpy_s(rch,"0");
                iptr = (char *)strstr(tch,"i" );
                *iptr = NULL;
                if(0 != strlen( tch )){  strcpy_s(ich,tch);}
                else{strcpy_s(ich,"1");}
        }
        else{// a+bi
                if(psptr != NULL){
                        is = 1;
                        *psptr = NULL;
                        strcpy_s(rch,tch);
                        psptr++;
                        strcpy_s(ich,psptr);
                        iptr = (char *)strstr(ich,"i" );
                        *iptr = NULL;
                        if(0 == strlen( ich )){strcpy_s(ich,"1");}

                }
                if(nsptr != NULL){
                        is = -1;
                        *nsptr = NULL;
                        strcpy_s(rch,tch);
                        nsptr++;
                        strcpy_s(ich,nsptr);
                        iptr = (char *)strstr(ich,"i" );
                        *iptr = NULL;
                        if(0 == strlen( ich )){strcpy_s(ich,"1");}

                }
        }
}

// 実部領域の確保
strLength = strlen( rch );        // 文字列の長さ
rl = strLength/4 + ((strLength%4)?1:0);            // 文字桁でu_short1桁
rl = (((rl - 1) / DEF_COLS) + 1 ) * DEF_COLS;      // 指定桁数を超える倍数
rv = new( double[rl] );
```

```
	if( rv == 0 ){					// メモリ割り当て失敗時
		aError	theError( notEnoughMemory, C_str, sizeof(double) * rl );
		throw( theError );			// 上位のエラー処理へ
	}
	// 変換処理
	for( int i=0; i<rl; i++ ){
		char	h[5], e[5];			// strtol呼び出し用変換文字列、エラー文字列
		char	*ep = e;			// strtol呼び出し用ポインタ
		if( rl > 2 ){				// 2桁以上のときは以下の処理
			h[4] = '¥0';				// strtol()の検査終了文字null
			for( int j=3; j>=0; j-- ){		// ４文字ごとに
				if( strLength > 0 ){			// まだあれば
					char c = rch[--strLength];		// 1文字取りだし
					if( !isxdigit(c) ){		// 無効な文字があれば
						delete[] rv;
						aError	theError( notXdigit, C_str, 0 );
						throw( theError );		// 上位のエラー処
理へ
					}
					h[j] = c;
				}
				else
					h[j] = '0';		// もう文字がなければにする
			}
			rv[i] = double( 0xffff & strtol( h, &ep, 16 ) );	// longに変換してから
		}
		else					// 2桁未満の時は
			rv[i] = 0;
	}
	// 領域の確保
	strLength = strlen( ich );		// 文字列の長さ
	il = strLength/4 + ((strLength%4)?1:0);		// 文字桁でu_short1桁
	il = (((il - 1) / DEF_COLS) + 1 ) * DEF_COLS;	// 指定桁数を超える倍数
	iv = new( double[il] );
	if( iv == 0 ){				// メモリ割り当て失敗時
		aError	theError( notEnoughMemory, C_str, sizeof(double) * il );
		throw( theError );			// 上位のエラー処理へ
	}
	// 変換処理
	for( int i=0; i<il; i++ ){
		char	h[5], e[5];			// strtol呼び出し用変換文字列、エラー文字列
		char	*ep = e;			// strtol呼び出し用ポインタ
		if( il > 2 ){				// 2桁以上のときは以下の処理
			h[4] = '¥0';				// strtol()の検査終了文字null
			for( int j=3; j>=0; j-- ){		// ４文字ごとに
				if( strLength > 0 ){			// まだあれば
					char c = ich[--strLength];		// 1文字取りだし
					if( !isxdigit(c) ){		// 無効な文字があれば
						delete[] iv;
						aError	theError( notXdigit, C_str, 0 );
						throw( theError );		// 上位のエラー処
理へ
```

```
                                        }
                                        h[j] = c;
                                }
                                else
                                        h[j] = '0';             // もう文字がなければにする
                        }
                        iv[i] = double( 0xffff & strtol( h, &ep, 16 ) );  // longに変換してから
                }
                else                                            // 2桁未満の時は
                        iv[i] = 0;
        }
}


// コピー(領域を別に持つ)
CmplxMPI::CmplxMPI(
        const CmplxMPI& a )                                     // 元のMPI
{
        rs = a.rs;
        is = a.is;
        is2 = a.is2;

        if( a.rl ){                                             // 元のMPIの長さがでなければ
                rv = new( double[a.rl] );// 同じ大きさの領域を確保
                if( rv == 0 ){                                  // 領域確保失敗時
                        rl = 0;                                 // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.rl );
                        throw( theError );                      // 上位のエラー処理へ
                }
                memcpy( rv, a.rv, a.rl*(sizeof(double)) );      // bit数をbyte数へ変換してコピー
        }
        else
                rv = 0;                                         // 元のMPIの長さがならば
        rl = a.rl;                                              // 長さを同じに

        if( a.il ){                                             // 元のMPIの長さがでなければ
                iv = new( double[a.il] );// 同じ大きさの領域を確保
                if( iv == 0 ){                                  // 領域確保失敗時
                        il = 0;                                 // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.il );
                        throw( theError );                      // 上位のエラー処理へ
                }
                memcpy( iv, a.iv, a.il*(sizeof(double)) );      // bit数をbyte数へ変換してコピー
        }
        else
                iv = 0;                                         // 元のMPIの長さがならば
        il = a.il;                                              // 長さを同じに
}

/********************/
/*** デストラクタ***/
/********************/
```

```
CmplxMPI::~CmplxMPI( void )
{
	if( (rl == 0) && (il == 0) )		// 領域がなければ
		return;				// 終了
	if(rl!=0){delete[] rv;}			// 領域があれば解放
	if(il!=0){delete[] iv;}
}

/*****************************/
/*** 値の操作・参照・出力***/
/*****************************/

// 4bitの数値をHEXキャラクタに変換
// getText()で使用
char xtoc(
	char c )				// 4bitの数値
{
	if( c >= 0 && c <= 9 )		// 0～ならば
		return char( c + 0x30 );	// ASCIIに変換して返す
	else if( c >= 10 && c <= 15 )	// 10～ならば
		return char( c + 0x37 );	// A～FのASCIIを返す
	return ' ';				// 0～以外ならば空白文字にする
}

// 実部の数値をストリングに変換
void CmplxMPI::getText_r(
	char* str )				// 出力先
{
	char c;
	unsigned int rvi;
	int i,j;

	for( j=i=getMaxColumn_r(); i>=0; i-- ){	// すべての桁に対して
		rvi = (unsigned int)rv[i];
		c = char( 0xf & (rvi >> 12) );	// 上位ビット
		*str++ = xtoc( c );			// 文字に
		c = char( 0xf & ( rvi >> 8 ));	// 次のビット
		*str++ = xtoc( c );			// 文字に
		c = char( 0xf & ( rvi >> 4 ));	// その次のビット
		*str++ = xtoc( c );			// 文字に
		c = char( rvi & 0xf );		// 最下位ビット
		*str++ = xtoc( c );			// 文字に
	}
	*str = '¥0';				// nullで止める
	if(j==-1){str[0]='0'; str[1]='¥0';}
}

// 数値をストリングに変換
void CmplxMPI::getText(
	char* str )				// 出力先
{
```

```
        char c;
        unsigned int rvi,ivi;

        for( int i=getMaxColumn(); i>=0; i-- ){   // すべての桁に対して
                rvi = (unsigned int)rv[i];
                c = char( 0xf & (rvi >> 12) );          // 上位ビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( 0xf & ( rvi >> 8 ));          // 次のビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( 0xf & ( rvi >> 4 ));          // その次のビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( rvi & 0xf );                          // 最下位ビット
                *str++ = xtoc( c );                                     // 文字に
        }
//      *str = '¥0';                                    // nullで止める
        if(is >= 0){    *str = '+'; *str++;}
        else{*str = '-'; *str++;}

        for( int j=getMaxColumn(); j>=0; j-- ){   // すべての桁に対して
                ivi = (unsigned int)iv[j];
                c = char( 0xf & (ivi >> 12) );          // 上位ビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( 0xf & ( ivi >> 8 ));          // 次のビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( 0xf & ( ivi >> 4 ));          // その次のビット
                *str++ = xtoc( c );                                     // 文字に
                c = char( ivi & 0xf );                          // 最下位ビット
                *str++ = xtoc( c );                                     // 文字に
        }
        *str = 'i'; *str++;
        *str = '¥0';                                    // nullで止める
}

// 実部の直接出力
int CmplxMPI::copyTo_r(                                 // 有効データ量を返す[byte]
        char* distination )                              // 出力先
{
        unsigned int rvi;

        int r = (getMaxColumn_r()+1) * COL_CHAR;
        for( int i=0; i<r1; i++ ){
                rvi = (unsigned int)rv[i];// 下位からコピー
            *distination++ = ( char)(rvi & 0x00ff);     // 下位バイト
            *distination++ = ( char)((rvi >> 8) & 0x00ff);                      // 上位バイト
        }
        return r;                                               // コピーしたバイト数
}

// 直接出力
int CmplxMPI::copyTo(                                   // 有効データ量を返す[byte]
        char* distination )                              // 出力先
{
```

```
        unsigned int rvi,ivi;

        int r = (2*getMaxColumn()+2+1) * COL_CHAR;
        for( int i=0; i<rl; i++ ){
                rvi = (unsigned int)rv[i];// 下位からコピー
            *distination++ = (char)(rvi & 0x00ff);        // 下位バイト
            *distination++ = rvi >> 8;                          // 上位バイト
        }
        *distination++ = '+';
        for( int j=0; j<il; j++ ){
                ivi = (unsigned int)iv[j];// 下位からコピー
            *distination++ = (char)(ivi & 0x00ff);        // 下位バイト
            *distination++ = ivi >> 8;                          // 上位バイト
        }
        *distination++ = 'i';
        return r;                                               // コピーしたバイト数
}

// 値の標準出力
void CmplxMPI::print( void )
{
        int vi;

        if(rs<0){cout << "-";}

        for( int i=rl-1; i>=0; i-- ){                   // 桁ごとに
                cout.setf(ios::hex,ios::basefield);     // 16進数で
                cout.width(4);                                          // 4桁で
                cout.fill('0');                                         // 前にを充填
                vi = (int)rv[i];
                cout << vi;                                             // 出力
        }

        if(is>=0){cout << "+";}
        if(is<0){cout << "-";}

        for( int j=il-1; j>=0; j-- ){                   // 桁ごとに
                cout.setf(ios::hex,ios::basefield);     // 16進数で
                cout.width(4);                                          // 4桁で
                cout.fill('0');                                         // 前にを充填
                vi = (int)iv[j];
                cout << vi;                                             // 出力
        }
        cout << "i";

        cout.flush();
}


// 値のエラー出力
void CmplxMPI::eprint( void )
{
```

```
        for( int i=rl-1; i>=0; i-- ){                   // 桁ごとに
                cerr.setf(ios::hex,ios::basefield);     // 16進数で
                cerr.width(4);                                          // 4桁で
                cerr.fill('0');                                         // 前にを充填
                cerr << rv[i];                                          // 出力
        }
        cerr.flush();
}

// 値の文字列化
void CmplxMPI::sprint(
        char* mes )                                             // 格納先
{
        char    st[4];                                  // 作業用
        for( int i=rl-1; i>=0; i-- ){    // 桁ごとに
                sprintf( st, "%04X", rv[i]);    // 文字列に変換
                strcat( mes, st );                      // 格納
        }
}

// bit長(1である最高のbit位置)を返す
int CmplxMPI::getBitLength( void ) const
{
        int t;

        for( int i=rl-1; i>=0; i-- ){    // 上の桁から
                if( rv[i] != 0 ){                       // 0でなければビット位置を調べる
                        int     bits = 0;                       // ビット位置初期値をとし
                        t = (unsigned int)rv[i];        // 調べる値をコピー

                        do{
                                bits++;                         // ビット位置更新
                                t >>= 1;                // tをシフトして
                        }while( t != 0 );               // 0ならその桁内の最上位ビット確定

                        return bits + i * COL_LENGTH;   // 全体でのビット位置を返す
                }
        }
        return 0;                                               // 値がの時
}

// 実部のでない最高桁を返す
int     CmplxMPI::getMaxColumn_r( void ) const
{
        int i;                                                  // 桁
        for( i=rl-1; i>=0; i-- )                // 最上位から
                if( rv[i] != 0.0 )  break;              // 0でなければ停止
        return i;                                               // 値がなら-1
}
// 虚部のでない最高桁を返す
int     CmplxMPI::getMaxColumn_i( void ) const
{
```

```
        int j;                                                  // 桁
        for( j=il-1; j>=0; j-- )                // 最上位から
                if( iv[j] != 0.0 )  break;              // 0でなければ停止
        return j;                                               // 値がなら
}
// 0でない最高桁を返す
int     CmplxMPI::getMaxColumn( void ) const
{
        int i,j;                                                // 桁
        for( i=rl-1; i>=0; i-- )                // 最上位から
                if( rv[i] != 0.0 )  break;              // 0でなければ停止
        for( j=il-1; j>=0; j-- )                // 最上位から
                if( iv[j] != 0.0 )  break;              // 0でなければ停止
        return max(i,j);                                        // 値がなら
}


// データ長を変更する
int     CmplxMPI::changeLength(                         // 戻り値：:成功, 1:上位切捨発生, 失敗時
throw
        int newRealLength , int newImagLength )                                 // 実部､虚部の長さ
指定
{
        int ref=0, ief=0;
        is2 = 1;

        int     nlr = (((newRealLength - 1) / DEF_COLS) + 1 ) * DEF_COLS;  // 指定桁数を超える倍数
        int     nli = (((newImagLength - 1) / DEF_COLS) + 1 ) * DEF_COLS;  // 指定桁数を超える倍数

        if( (rl == nlr) && (il == nli) )  return 0;             // 長さが同じなら何もしない

////////////////////////////////////////////////////////////////////////////////
        if( rl < nlr ){                                         // 拡張
                double* p = new( double[nlr] );         // 新実領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nlr );
                        throw( theError );
                }
                int i;
                for( i=0; i<rl; i++ )                   // コピー
                        p[i] = rv[i];
                for( ; i<nlr; i++ )                     // 増加分初期化
                        p[i] = 0;
                delete[] rv;                                    // 旧領域の開放
                rv = p;                                         // ポインタの更新
                rl = nlr;                                               // サイズの更新
        }
        else{                                                   // 縮小
                ref = 0;                                // 戻り値フラグ
                double* p = new( double[nlr] );         // 新領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nlr );
```

```
                        throw( theError );
                }
                int i;
                for( i=0; i<nlr; i++ )              // コピー
                        p[i] = rv[i];
                for( ; i<rl; i++ )                          // 切り捨て部分がでない
                        if( rv[i] != 0 ) ref = 1;// 切り捨て発生
                delete[] rv;                                        //旧領域の開放
                rv = p;                                             // ポインタの更新
                rl = nlr;                                                   // サイズの更新
        }
////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
        if( il < nli ){                                     // 拡張
                double* p = new( double[nli] );             // 新実領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nli );
                        throw( theError );
                }
                int i;
                for( i=0; i<il; i++ )               // コピー
                        p[i] = iv[i];
                for( ; i<nli; i++ )                         // 増加分初期化
                        p[i] = 0;
                delete[] iv;                                        // 旧領域の開放
                iv = p;                                             // ポインタの更新
                il = nli;                                                   // サイズの更新
                return ref;                                         // 成功
        }
        else{                                                       // 縮小
                ief = 0;                                    // 戻り値フラグ
                double* p = new( double[nli] );             // 新領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nli );
                        throw( theError );
                }
                int i;
                for( i=0; i<nli; i++ )              // コピー
                        p[i] = iv[i];
                for( ; i<il; i++ )                          // 切り捨て部分がでない
                        if( iv[i] != 0 ) ief = 1;// 切り捨て発生
                delete[] iv;                                        //旧領域の開放
                iv = p;                                             // ポインタの更新
                il = nli;                                                   // サイズの更新
                return (ref+ief);                                           // 成功または切り捨て発生
        }
}


// データ長を最小のDEF_COLS倍にする
void    CmplxMPI::adjustLength( void )
```

```
{
        changeLength( getMaxColumn_r() + 1, getMaxColumn_i() + 1 );
}

// 実部のデータを最小のDEF_COLS倍にする。虚部は最小化する
void CmplxMPI::torealpart(void)
{
        int newLength = getMaxColumn_r() + 1 ;

        int     nl = (((newLength - 1) / DEF_COLS) + 1 ) * DEF_COLS;       // 指定桁数を超える倍数

        if( rl != nl){   // 長さが同じなら何もしない
////////////////////////////////////////////////////////////////////////////////
        if( rl < nl ){                                          // 拡張
                double* p = new( double[nl] );                  // 新実領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nl );
                        throw( theError );
                }
                int i;
                for( i=0; i<rl; i++ )               // コピー
                        p[i] = rv[i];
                for( ; i<nl; i++ )                              // 増加分初期化
                        p[i] = 0;
                delete[] rv;                                            // 旧領域の開放
                rv = p;                                                 // ポインタの更新
                rl = nl;                                                // サイズの更新
        }
        else{
                double* p = new( double[nl] );                  // 新領域の確保
                if( p == 0 ){
                        aError  theError( notEnoughMemory, F_changeLength, sizeof(double) * nl );
                        throw( theError );
                }
                int i;
                for( i=0; i<nl; i++ )               // コピー
                        p[i] = rv[i];
                delete[] rv;                                            // 旧領域の開放
                rv = p;                                                 // ポインタの更新
                rl = nl;                                                // サイズの更新
        }

        }

        if(is2 != 0){
        is = 1;
        is2 = 0;
        if(il > 0){delete[] iv;}
        iv = new( double[DEF_COLS] );     // デフォルトサイズで領域確保
        if( iv == 0 ){                                          // メモリ割り当て失敗時
                il = 0;                                                 // 無効
                aError  theError( notEnoughMemory, C_int, sizeof(double) * DEF_COLS );
```

```
                throw( theError );                              // 上位のエラー処理へ
        }
        il = DEF_COLS;                                          // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)       // 上位の桁の値を初期化
                iv[i] = 0.0;
        }
}



/**************/
/*** 演算子***/
/**************/

// 代入
CmplxMPI& CmplxMPI::operator=(                                  // 代入
        const CmplxMPI& a )                                             // 代入する値
{
        rs = a.rs;
        is = a.is;
        is2 = a.is2;

        if( rl && rv )                                          // すでに記憶領域があれば
                delete[] rv;                                            // 破棄する
        rv = new( double[a.rl] );
        if( rv == 0 ){                                          // メモリ割り当て失敗時
                rl = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.rl );
                throw( theError );
        }
        rl = a.rl;                                                      // 桁数を同じに
        for( int i=0; i<rl; i++ )           // 値のコピー
                rv[i] = a.rv[i];

        if( il && iv )                                          // すでに記憶領域があれば
                delete[] iv;                                            // 破棄する
        iv = new( double[a.il] );
        if( iv == 0 ){                                          // メモリ割り当て失敗時
                il = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.il );
                throw( theError );
        }
        il = a.il;                                                      // 桁数を同じに
        for( int j=0; j<il; j++ )           // 値のコピー
                iv[j] = a.iv[j];

        return *this;                                           // 自身を返して代入する
}



// 加算
```

```
CmplxMPI operator+(                                                    // 和
        const CmplxMPI& a,                                             // 被加数
        const CmplxMPI& b )                                            // 加数
{
        int     ll, ls;                                                // 戻り値の長さ、演算の範囲
        double *p;                                              // 代入用
        double *q;

        int     i, j;
        unsigned int tv;
        double c = 0.0, t;                                      // カウンタ、キャリー、中間結果

        int ll2 = max(a.rl,b.rl);
        int ll3 = max(a.il,b.il);
        CmplxMPI z(ll2+1,0,ll3+1,0);

        ////////////////////// 実部の計算開始/////////////////////////
        if(a.rs == b.rs){//real part 同符号の場合符号の変化なし、加算する
                if( a.rl > b.rl ){                                     // 大きな数を戻り値の参考にする
                        ll = a.rl;      ls = b.rl;      p = a.rv; q = a.rv;
                } else{
                        ll = b.rl;      ls = a.rl;      p = b.rv; q = b.rv;
                }

                c=0.0;
                for( i=0; i<ls; i++ ){                          // 1桁ずつ加える
                        t = a.rv[i] + b.rv[i] + c;              // i桁目と、i-1桁目の繰り上がりを加える
                        tv = (unsigned int)t;
                        z.rv[i] = (double)(tv & COL_MASK);                  // 繰り上がりを除いてi桁目
の和にする
                        c = (double)( (tv >> COL_LENGTH) & COL_MASK);               // 次の桁への繰り
上げ
                }
                for( ; i<ll; i++ ){                                    // 上位の残りの桁について
                        t = p[i] + c;                                  // 下の桁からの繰り上げを加える
                        tv = (unsigned int)t;
                        z.rv[i] = (double)( tv & COL_MASK) ;
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);                // 次の桁への繰り
上げ
                }
                z.rv[ll] = c ;                          // 繰り上がりをセット
                z.rs = a.rs;
        }
////////////////////////////////////////////////////////////////////////////
        if(a.rs != b.rs){//real part  異符号の場合は絶対値の大きなほうで符合が決まる。減算
                if( a.rl > b.rl ){                                     // 大きな数を戻り値の参考にする
                        ll = a.rl;      ls = b.rl;      p = a.rv; q = a.rv;
                } else{
                        ll = b.rl;      ls = a.rl;      p = b.rv; q = b.rv;
                }
                if( a >= b ){                                          // 符号は無視して実部のみ比較
                        z.rs = a.rs;
```

```
                int ls = b.getMaxColumn_r();                    // 演算の範囲
                c = 0.0;                                // カウンタ、キャリー、中間値
                for( i=0; i<=ls; i++ ){                         // 1桁ずつ減じる
                        t =  b.rv[i]  + c;                      // その桁の減数
                        if( a.rv[i] < t ){                      // 減じられないとき
                                z.rv[i] = BASE_UNIT + a.rv[i] - t ;     // 上の桁から借り
てくる
                                c = 1.0;                                // キャリーありs
                        }
                        else{                                           // 減じられるとき
                                z.rv[i] =  a.rv[i] - t ;
                                c = 0.0;                                // キャリーなし
                        }
                }
                for( ; i<a.rl; i++ ){                           // 上位の残りの桁について
                        if( a.rv[i] < c ){                              // キャリーがあるのにその
桁がなら
                                z.rv[i] = BASE_UNIT + a.rv[i] - c ;     // さらに上の桁か
ら借りる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // キャリーを減じ
られるか、キャリーがなければ
                                z.rv[i] =  a.rv[i] - c ; // その桁を求める
                                c = 0;                                  // キャリーなし
                        }
                }
        }
        else{ // b の実部の絶対値が大きい場合
                z.rs = b.rs;
                int ls = a.getMaxColumn_r();                    // 演算の範囲
                c=0.0;
                for( i=0; i<=ls; i++ ){                         // 1桁ずつ減じる
                        t =  a.rv[i] + c;                       // その桁の減数
                        if( b.rv[i] < t ){                      // 減じられないとき
                                z.rv[i] = BASE_UNIT + b.rv[i] - t ;     // 上の桁から借り
てくる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // 減じられるとき
                                z.rv[i] =  b.rv[i] - t ;
                                c = 0;                                  // キャリーなし
                        }
                }
                for( ; i<b.rl; i++ ){                           // 上位の残りの桁について
                        if( b.rv[i] < c ){                              // キャリーがあるのにその
桁がなら
                                z.rv[i] = BASE_UNIT + b.rv[i] - c ;     // さらに上の桁か
ら借りる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // キャリーを減じ
```

```
られるか、キャリーがなければ
                                        z.rv[i] =  b.rv[i] - c ; // その桁を求める
                                        c = 0;                                  // キャリーなし
                                }
                        }
                }// real par
        }
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
        if((a.is2!=0) && (b.is2!=0)){
////////////////////////////////////////////////////////////////////////////
        if(a.is==b.is){//虚部の計算+  同符号
                if( a.il > b.il ){                              // 大きな数を戻り値の参考にする
                        ll = a.il;      ls = b.il;      p = a.iv; q = a.iv;
                } else{
                        ll = b.il;      ls = a.il;      p = b.iv; q = b.iv;
                }
                c=0.0;
                for( j=0; j<ls; j++ ){                  // 1桁ずつ加える
                        t = a.iv[j] + b.iv[j] + c;      // i桁目と、i-1桁目の繰り上がりを加える
                        tv = (unsigned int)t;
                        z.iv[j] =  (double)(tv & COL_MASK)  ;           // 繰り上がりを除いてi桁目
の和にする
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);            // 次の桁への繰り
上げ
                }
                for( ; j<ll; j++ ){                             // 上位の残りの桁について
                        t = q[j] + c;                           // 下の桁からの繰り上げを加える
                        tv = (unsigned int)t;
                        z.iv[j] = (double)(tv & COL_MASK) ;
                        c = (double)( (tv >> COL_LENGTH) & COL_MASK);           // 次の桁への繰り
上げ
                }
                z.iv[ll] =  c ;                 // 繰り上がりをセ
                z.is = a.is;
                z.is2 = 1;
                z.adjustLength();                               // 桁長を調整
                return z;                                               // 和を返す
        }
////////////////////////////////////////////////////////////////////////////////
        // 虚部の計算+  異符号
        if(a.is != b.is){
                if( a.il > b.il ){                              // 大きな数を戻り値の参考にする
                        ll = a.il;      ls = b.il;      p = a.iv; q = a.iv;
                } else{
                        ll = b.il;      ls = a.il;      p = b.iv; q = b.iv;
                }

                int av = a.getMaxColumn_i();
                int bv = b.getMaxColumn_i();
                int aBb = 0;
                if(av > bv){aBb = 1;}
```

```
		if(av < bv){aBb = -1;}
		if(av == bv){
			aBb = 0;
			for(int i = av; i >=0; i--){
				if(a.iv[i] > b.iv[i]){aBb = 1; break;}
				if(a.iv[i] < b.iv[i]){aBb = -1; break;}
			}
		}
		if(aBb>=0 ){                                    // 負になるときは
			int ls = bv;              // 演算の範囲
			c = 0.0;                          // カウンタ、キャリー、中間値
			for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
				t =  b.iv[i] + c;               // その桁の減数
				if( int(a.iv[i]) < t ){         // 減じられないとき
					z.iv[i] = BASE_UNIT + a.iv[i] - t ;       // 上の桁から借りてくる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // 減じられるとき
					z.iv[i] =  a.iv[i] - t ;
					c = 0.0;                                  // キャリーなし
				}
			}
			for( ; i<a.il; i++ ){                   // 上位の残りの桁について
				if( a.iv[i] < c ){                        // キャリーがあるのにその桁がなら
					z.iv[i] = BASE_UNIT + a.iv[i] - c ;       // さらに上の桁から借りる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // キャリーを減じられるか、キャリーがなければ
					z.iv[i] =  a.iv[i] - c ; // その桁を求める
					c = 0.0;                                  // キャリーなし
				}
			}
			z.is = a.is;
			z.adjustLength();                               // 桁長を調整
			return z;                                       // 和を返す
		}
		else{ // 虚部が負の場合
			int ls = av;              // 演算の範囲
			c=0.0;
			for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
				t =  a.iv[i] + c;               // その桁の減数
				if( int(b.iv[i]) < t ){         // 減じられないとき
					z.iv[i] = BASE_UNIT + b.iv[i] - t ;       // 上の桁から借りてくる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // 減じられるとき
					z.iv[i] =  b.iv[i] - t ;
```

```
                                        c = 0.0;                                        // キャリーなし
                                }
                        }
                        for( ; i<b.il; i++ ){                   // 上位の残りの桁について
                                if( b.iv[i] < c ){                      // キャリーがあるのにその桁がなら
                                        z.iv[i] = BASE_UNIT + b.iv[i] - c ;     // さらに上の桁から借りる
                                        c = 1.0;                                        // キャリーあり
                                }
                                else{                                                   // キャリーを減じられるか、キャリーがなければ
                                        z.iv[i] =  b.iv[i] - c ; // その桁を求める
                                        c = 0.0;                                        // キャリーなし
                                }
                        }
                        z.is = b.is;
                        z.is2 = 1;
                        z.adjustLength();                                       // 桁長を調整
                        return z;                                                       // 和を返す
                }// imaginary par
        }
////////////////////////////////////////////////////////////////////////
        }// case (a.is2!=0) && (b.is2!=0)
////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
        if((a.is2!=0) && (b.is2==0)){
                ll = a.il;
                for( j=0; j<ll; j++ ){                  // 1桁ずつ加える
                        z.iv[j] = a.iv[j];
                }
                z.is = a.is;
                z.is2 = 1;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
        }
////////////////////////////////////////////////////////////////////////////////////
        if((a.is2==0) && (b.is2!=0)){
                ll = b.il;
                for( j=0; j<ll; j++ ){                  // 1桁ずつ加える
                        z.iv[j] = b.iv[j];
                }
                z.is = b.is;
                z.is2 = 1;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
        }
////////////////////////////////////////////////////////////////////////
        if((a.is2==0) && (b.is2==0)){
                if(z.il>0){ delete[] z.iv;}
                z.iv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
```

```
                if( z.iv == 0 ){                                        // メモリ割り当て失敗時
                        z.il = 0;                                                       // 無効
                        aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                        throw( theError );                              // 上位のエラー処理へ
                }
                z.il = DEF_COLS;                                        // デフォルトの長さ
                for(int i=0; i<DEF_COLS; i++)
                        z.iv[i] = 0.0;                                          // 値の初期化

                z.is = 1;
                z.is2 = 0;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
}
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
        z.adjustLength();                                       // 桁長を調整
        return z;                                                       // 和を返す
}


// 減算
CmplxMPI operator-(                                             // 差
        const CmplxMPI& a,                                      // 被減数
        const CmplxMPI& b )                                     // 減数
{
        int     ll, ls;                                                 // 戻り値の長さ、演算の範囲
        double *p;                                              // 代入用
        double *q;

        int     i, j;
        unsigned int tv;
        double c = 0.0, t;                                      // カウンタ、キャリー、中間結果

        int ll2 = max(a.rl,b.rl);
        int ll3 = max(a.il,b.il);
        CmplxMPI z(ll2+1,0,ll3+1,0);

        ////////////////////////////////////////////////////////////////////////////
        if(a.rs == b.rs){//実部計算  同符号
                if( a.rl > b.rl ){                                      // 大きな数を戻り値の参考にする
                        ll = a.rl;      ls = b.rl;      p = a.rv; q = a.rv;
                } else{
                        ll = b.rl;      ls = a.rl;      p = b.rv; q = b.rv;
                }
                // 実部から計算
                if( a >= b ){                                           // 符号は無視して実部のみ比較
                        z.rs = a.rs;
                        int ls = b.getMaxColumn_r();                    // 演算の範囲
                        c = 0.0;                                // カウンタ、キャリー、中間値
                        for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
                                t =  b.rv[i]  + c;              // その桁の減数
```

```
                        if( a.rv[i] < t ){              // 減じられないとき
                                z.rv[i] = BASE_UNIT + a.rv[i] - t ;     // 上の桁から借り
てくる
                                c = 1.0;                                // キャリーありs
                        }
                        else{                                           // 減じられるとき
                                z.rv[i] =  a.rv[i] - t ;
                                c = 0.0;                                // キャリーなし
                        }
                }
                for( ; i<a.rl; i++ ){                   // 上位の残りの桁について
                        if( a.rv[i] < c ){                      // キャリーがあるのにその
桁がなら
                                z.rv[i] = BASE_UNIT + a.rv[i] - c ;     // さらに上の桁か
ら借りる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // キャリーを減じ
られるか、キャリーがなければ
                                z.rv[i] =  a.rv[i] - c ; // その桁を求める
                                c = 0;                                  // キャリーなし
                        }
                }
        }
        else{ // 実部が負の場合
                z.rs = (-1)*b.rs;
                int ls = a.getMaxColumn_r();            // 演算の範囲
                c=0.0;
                for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
                        t =  a.rv[i] + c;               // その桁の減数
                        if( int(b.rv[i]) < t ){         // 減じられないとき
                                z.rv[i] = BASE_UNIT + b.rv[i] - t ;     // 上の桁から借り
てくる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // 減じられるとき
                                z.rv[i] =  b.rv[i] - t ;
                                c = 0;                                  // キャリーなし
                        }
                }
                for( ; i<b.rl; i++ ){                   // 上位の残りの桁について
                        if( b.rv[i] < c ){                      // キャリーがあるのにその
桁がなら
                                z.rv[i] = BASE_UNIT + b.rv[i] - c ;     // さらに上の桁か
ら借りる
                                c = 1;                                  // キャリーあり
                        }
                        else{                                           // キャリーを減じ
られるか、キャリーがなければ
                                z.rv[i] =  b.rv[i] - c ; // その桁を求める
                                c = 0;                                  // キャリーなし
                        }
```

```
                        }
                }// real par
        }
////////////////////////////////////////////////////////////////////////////////
        if(a.rs != b.rs){//実部計算　異符号
                if( a.rl > b.rl ){                              // 大きな数を戻り値の参考にする
                        ll = a.rl;      ls = b.rl;      p = a.rv; q = a.rv;
                } else{
                        ll = b.rl;      ls = a.rl;      p = b.rv; q = b.rv;
                }
                c=0.0;
                for( i=0; i<ls; i++ ){                  // 1桁ずつ加える
                        t = a.rv[i] + b.rv[i] + c;      // i桁目と、i-1桁目の繰り上がりを加える
                        tv = (unsigned int)t;
                        z.rv[i] = (double)( tv & COL_MASK);             // 繰り上がりを除いてi桁目
の和にする
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);            // 次の桁への繰り
上げ
                }
                for( ; i<ll; i++ ){                             // 上位の残りの桁について
                        t = p[i] + c;                           // 下の桁からの繰り上げを加える
                        tv = (unsigned int)t;
                        z.rv[i] = (double)(tv & COL_MASK) ;
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);            // 次の桁への繰り
上げ
                }
                z.rv[ll] = c ;                  // 繰り上がりをセット
                z.rs = a.rs;
        }

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
        if(a.is2!=0 && b.is2!=0){
////////////////////////////////////////////////////////////////////////////////
        if(a.is==b.is){//虚部の計算　同符号
                if( a.il > b.il ){                              // 大きな数を戻り値の参考にする
                        ll = a.il;      ls = b.il;      p = a.iv; q = a.iv;
                } else{
                        ll = b.il;      ls = a.il;      p = b.iv; q = b.iv;
                }
                int av = a.getMaxColumn_i();
                int bv = b.getMaxColumn_i();
                int aBb = 0;
                if(av > bv){aBb = 1;}
                if(av < bv){aBb = -1;}
                if(av == bv){
                        aBb = 0;
                        for(int i = av; i >=0; i--){
                                if(a.iv[i] > b.iv[i]){aBb = 1; break;}
                                if(a.iv[i] < b.iv[i]){aBb = -1; break;}
                        }
                }
```

```
		if(aBb >=0 ){                                   // 負になるときは
			int ls = bv;                // 演算の範囲
			c = 0.0;                            // カウンタ、キャリー、中間値
			for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
				t =  b.iv[i] + c;               // その桁の減数
				if( int(a.iv[i]) < t ){         // 減じられないとき
					z.iv[i] = BASE_UNIT + a.iv[i] - t ;       // 上の桁から借りてくる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // 減じられるとき
					z.iv[i] =  a.iv[i] - t ;
					c = 0.0;                                  // キャリーなし
				}
			}
			for( ; i<a.il; i++ ){                   // 上位の残りの桁について
				if( a.iv[i] < c ){                        // キャリーがあるのにその桁がなら
					z.iv[i] = BASE_UNIT + a.iv[i] - c ;       // さらに上の桁から借りる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // キャリーを減じられるか、キャリーがなければ
					z.iv[i] =  a.iv[i] - c ; // その桁を求める
					c = 0.0;                                  // キャリーなし
				}
			}
			z.is = a.is;
			z.adjustLength();                         // 桁長を調整
			return z;                                 // 和を返す
		}
		else{ // 虚部が負の場合
			z.is = (-1)*b.is;
			int ls = av;                // 演算の範囲
			c=0.0;
			for( i=0; i<=ls; i++ ){                 // 1桁ずつ減じる
				t =  a.iv[i] + c;               // その桁の減数
				if( int(b.iv[i]) < t ){         // 減じられないとき
					z.iv[i] = BASE_UNIT + b.iv[i] - t ;       // 上の桁から借りてくる
					c = 1.0;                                  // キャリーあり
				}
				else{                                     // 減じられるとき
					z.iv[i] =  b.iv[i] - t ;
					c = 0.0;                                  // キャリーなし
				}
			}
			for( ; i<b.il; i++ ){                   // 上位の残りの桁について
				if( b.iv[i] < c ){                        // キャリーがあるのにその桁がなら
```

```
                                        z.iv[i] = BASE_UNIT + b.iv[i] - c ;        // さらに上の桁から借りる
                                        c = 1.0;                                   // キャリーあり
                                }
                                else{                                              // キャリーを減じられるか、キャリーがなければ
                                        z.iv[i] =  b.iv[i] - c ; // その桁を求める
                                        c = 0.0;                                   // キャリーなし
                                }
                        }
                        z.adjustLength();                                  // 桁長を調整
                        return z;                                                  // 和を返す
                }// imaginary par
        }
        ////////////////////////////////////////////////////////////////////////
        // 虚部の計算異符号
        if(a.is != b.is){
                if( a.il > b.il ){                                 // 大きな数を戻り値の参考にする
                        ll = a.il;      ls = b.il;      p = a.iv; q = a.iv;
                } else{
                        ll = b.il;      ls = a.il;      p = b.iv; q = b.iv;
                }
                c=0.0;
                for( j=0; j<ls; j++ ){                     // 1桁ずつ加える
                        t = a.iv[j] + b.iv[j] + c;         // i桁目と、i-1桁目の繰り上がりを加える
                        tv = (unsigned int)t;
                        z.iv[j] =  (double)( tv & COL_MASK)  ;             // 繰り上がりを除いてi桁目の和にする
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);               // 次の桁への繰り上げ
                }
                for( ; j<ll; j++ ){                                // 上位の残りの桁について
                        t = q[j] + c;                              // 下の桁からの繰り上げを加える
                        tv = (unsigned int)t;
                        z.iv[j] = (double)(tv & COL_MASK) ;
                        c = (double)((tv >> COL_LENGTH) & COL_MASK);               // 次の桁への繰り上げ
                }
                z.iv[ll] =  c ;                    // 繰り上がりをセ
                z.is = a.is;
                z.adjustLength();                                  // 桁長を調整
                return z;                                                  // 和を返す
        }
////////////////////////////////////////////////////////////////////////////////
        }// case (a.is2!=0 && b.is2!=0){

////////////////////////////////////////////////////////////////
        if(a.is2!=0 && b.is2==0){
                ll = a.il;
                for( i=0; i<ll; i++ ){                     // 1桁ずつ減じる
                        z.iv[i] =  a.iv[i];
                }// その桁の減数
```

```
                z.is = a.is;
                z.is2 = 1;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
                }
////////////////////////////////////////////////////////////////
        if(a.is2==0 && b.is2!=0){
                ll = b.il;
                for( i=0; i<ll; i++ ){                          // 1桁ずつ減じる
                        z.iv[i] =  b.iv[i];
                }// その桁の減数
                z.is = (-1)*b.is;
                z.is2 = 1;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
                }
////////////////////////////////////////////////////////////////
        if(a.is2==0 && b.is2==0){
                if(z.il>0){ delete[] z.iv;}
                z.iv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
                if( z.iv == 0 ){                                        // メモリ割り当て失敗時
                        z.il = 0;                                                               // 無効
                        aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                        throw( theError );                              // 上位のエラー処理へ
                }
                z.il = DEF_COLS;                                        // デフォルトの長さ
                for(int i=0; i<DEF_COLS; i++)
                        z.iv[i] = 0.0;                                          // 値の初期化

                z.is = 1;
                z.is2 = 0;
                z.adjustLength();                                       // 桁長を調整
                return z;                                                       // 和を返す
                }
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
        z.adjustLength();                                       // 桁長を調整
        return z;                                                       // 和を返す
}


// 乗算
CmplxMPI operator*(                                                     // 積
        const CmplxMPI& a,                                      // 被乗数
        const CmplxMPI& b )                                     // 乗数
{
        int     rll = a.rl + b.rl;                                      // 積の長さは数の長さの和
        int ill = a.il + b.il;
        int llr = max(rll, ill) + 1;
        rll = a.rl + b.il ;
        ill = a.il + b.rl ;
        int lli = max(rll, ill) + 1;
```

```
        unsigned int tmp;
        CmplxMPI d( llr, 0, lli, 0 );                           // 戻り値
        CmplxMPI e( llr, 0, lli, 0 );                           // 戻り値
        CmplxMPI g( llr, 0, lli, 0 );                           // 戻り値

        int     i, j = 0;
        for( i=0; i<b.rl; i++ ){                // bの各桁について
                double k = 0;
                for( j=0; j<a.rl; j++ ){        // aの各桁(段)について
                        tmp = (unsigned int)( a.rv[j] * b.rv[i] + d.rv[i+j] + k);  // 1桁分の乗算
                        d.rv[i+j] = double(tmp & COL_MASK );    // 繰り上がりを除く
                        k = double( ( tmp >> COL_LENGTH ) & COL_MASK );          // 繰り上がりを格
納
                }
                d.rv[i+j] = k;                          // その段の繰り上がりを次の段へ
        }

        if((a.is2!=0) && (b.is2!=0)){
        for( i=0; i<b.il; i++ ){                // bの各桁について
                double k = 0;
                for( j=0; j<a.il; j++ ){        // aの各桁(段)について
                        tmp = (unsigned int)(a.iv[j] * b.iv[i] + e.rv[i+j] + k);   // 1桁分の乗算
                        e.rv[i+j] = double( tmp & COL_MASK );   // 繰り上がりを除く
                        k = double(( tmp >> COL_LENGTH ) & COL_MASK );          // 繰り上がりを格
納
                }
                e.rv[i+j] = k;                          // その段の繰り上がりを次の段へ
        }
        }

        d.rs=1; e.rs=1; d.is=1; e.is=1;

        if((a.rs==1) && (a.is==1)){
                if((a.is2!=0) && (b.is2!=0)){
                        if((b.rs== 1) && (b.is== 1)){   d -= e; }
                        if((b.rs== 1) && (b.is==-1)){   d += e; }
                        if((b.rs==-1) && (b.is== 1)){    d = -d -e;}
                        if((b.rs==-1) && (b.is==-1)){    d = e-d; }
                }else{
                        if(b.rs==-1){    d = -d;}
                }
        }
        if((a.rs==1) && (a.is==-1)){
                if((a.is2!=0) && (b.is2!=0)){
                        if((b.rs== 1) && (b.is== 1)){    d += e; }
                        if((b.rs== 1) && (b.is==-1)){    d -= e; }
                        if((b.rs==-1) && (b.is== 1)){    d = e-d; }
                        if((b.rs==-1) && (b.is==-1)){    d = -d -e;}
                }else{
                        if(b.rs==-1){    d = -d; }
                }
        }
```

```
	if((a.rs==-1) && (a.is==1)){
		if((a.is2!=0) && (b.is2!=0)){
			if((b.rs== 1) && (b.is== 1)){	d = -d -e;}
			if((b.rs== 1) && (b.is==-1)){	d = e - d; }
			if((b.rs==-1) && (b.is== 1)){	d -= e; }
			if((b.rs==-1) && (b.is==-1)){	d += e; }
		}else{
			if(b.rs== 1){	d = -d;}
		}
	}
	if((a.rs==-1) && (a.is==-1)){
		if((a.is2!=0) && (b.is2!=0)){
			if((b.rs== 1) && (b.is== 1)){	d =  e - d; }
			if((b.rs== 1) && (b.is==-1)){	d = -d - e;}
			if((b.rs==-1) && (b.is== 1)){	d +=  e; }
			if((b.rs==-1) && (b.is==-1)){	d -=  e; }
		}else{
			if(b.rs== 1){	d = -d; }
		}
	}

	e = g;  // 順序変更不可
	g = d;  // 順序変更不可
	d = e;  // 順序変更不可

	if(b.is2 != 0){
	for( i=0; i<b.il; i++ ){			// bの各桁について
		double k = 0;
		for( j=0; j<a.rl; j++ ){		// aの各桁(段)について
			tmp = (unsigned int)( a.rv[j] * b.iv[i] + d.iv[i+j] + k);  // 1桁分の乗算
			d.iv[i+j] = double( tmp & COL_MASK );	// 繰り上がりを除く
			k = double( ( tmp >> COL_LENGTH ) & COL_MASK );		// 繰り上がりを格
納
		}
		d.iv[i+j] = k;				// その段の繰り上がりを次の段へ
	}
	}

	if(a.is2 != 0){
	for( i=0; i<b.rl; i++ ){			// bの各桁について
		double k = 0;
		for( j=0; j<a.il; j++ ){		// aの各桁(段)について
			tmp = (unsigned int)(a.iv[j] * b.rv[i] + e.iv[i+j] + k);   // 1桁分の乗算
			e.iv[i+j] = double(  tmp & COL_MASK );	// 繰り上がりを除く
			k = double( ( tmp >> COL_LENGTH ) & COL_MASK );		// 繰り上がりを格
納
		}
		e.iv[i+j] = k;				// その段の繰り上がりを次の段へ
	}
	}

	d.rs=1; e.rs=1; d.is=1; e.is=1;
```

```
if((a.is2 != 0) || (b.is2 != 0)){
if((a.rs==1) && (a.is==1)){
        if((a.is2 != 0) && (b.is2 != 0)){
                if((b.rs== 1) && (b.is== 1)){     d += e; }
                if((b.rs== 1) && (b.is==-1)){     d = e-d; }
                if((b.rs==-1) && (b.is== 1)){     d -= e; }
                if((b.rs==-1) && (b.is==-1)){     d = -d -e;}
        }
        if(a.is2==0){
                if(b.is==-1){    d = -d; }
        }
        if(b.is2==0){
                if(b.rs== 1){    d = e; }
                if(b.rs==-1){    d = -e; }
        }
}
if((a.rs==1) && (a.is==-1)){
        if((a.is2 != 0) && (b.is2 != 0)){
                if((b.rs== 1) && (b.is== 1)){     d -= e; }
                if((b.rs== 1) && (b.is==-1)){     d = -d -e;}
                if((b.rs==-1) && (b.is== 1)){     d += e; }
                if((b.rs==-1) && (b.is==-1)){     d = e-d; }
        }
        if(a.is2==0){
                if(b.is==-1){    d = -d; }
        }
        if(b.is2==0){
                if(b.rs== 1){    d = -e; }
                if(b.rs==-1){    d = e; }
        }
}
if((a.rs==-1) && (a.is==1)){
        if((a.is2 != 0) && (b.is2 != 0)){
                if((b.rs== 1) && (b.is== 1)){     d = e-d; }
                if((b.rs== 1) && (b.is==-1)){     d += e; }
                if((b.rs==-1) && (b.is== 1)){     d = -d -e;}
                if((b.rs==-1) && (b.is==-1)){     d -= e; }
        }
        if(a.is2==0){
                if(b.is== 1){    d = -d; }
        }
        if(b.is2==0){
                if(b.rs== 1) {   d = e; }
                if(b.rs==-1) {   d = -e; }
        }
}
if((a.rs==-1) && (a.is==-1)){
        if((a.is2 != 0) && (b.is2 != 0)){
                if((b.rs== 1) && (b.is== 1)){     d = -d -e;}
                if((b.rs== 1) && (b.is==-1)){     d -= e; }
                if((b.rs==-1) && (b.is== 1)){     d = e-d; }
```

```
                        if((b.rs==-1) && (b.is==-1)){     d += e; }
                }
                if(a.is2==0){
                        if(b.is== 1){    d = -d; }
                }
                if(b.is2==0){
                        if(b.rs== 1){    d = -e; }
                        if(b.rs==-1){    d = e; }
                }
        }
        g += d;
        }

        if((a.is2==0) && (b.is2==0)){
                g.is = 1;
                g.is2 = 0;
                if(g.il>0){delete[] g.iv;}
                g.iv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
                if( g.iv == 0 ){                                        // メモリ割り当て失敗時
                        g.il = 0;                                                       // 無効
                        aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                        throw( theError );                              // 上位のエラー処理へ
                }
                g.il = DEF_COLS;                                        // デフォルトの長さ
                for(int i=0; i<DEF_COLS; i++)
                        g.iv[i] = 0.0;                                          // 値の初期化
        }
////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
        return g;                                                       // 積を返す
}


////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

// 除算商を求める
CmplxMPI quo(                                                           // 商
        const CmplxMPI& x,                                              // 被除数
        const CmplxMPI& y)                                              // 除数
{

        int n = x.getMaxColumn_r();
        int t = y.getMaxColumn_r();

        CmplxMPI q( x.rl, 0 , DEF_COLS, 0);                             // 商を

        if( y == 0 ){                                           // 0による除算
                aError  theError( divideByZero, F_div, 0 );
                throw( theError );
        }
        if( x < y ){                                            // 被除数< 除数
```

```
                return 0;                                               // 剰余(結果)
        }
        if( x == y ){                                           // 被除数= 除数
                return 1;                                               // 剰余(結果)
        }

        CmplxMPI x2( t+DEF_COLS , 0, DEF_COLS, 0); // Use From  0  to (t+1) for shift.

        for(int k=0; k<=t; k++){
                x2.rv[t-k] = x.rv[n-k];
        }

        for(int s =n; s >=t; s--){
                unsigned int q1;
                if(x2 >= y){
                        int w = x2.getMaxColumn_r();
                        if(w > t){
                                if(t >= 1){
                                q1 = (unsigned
int)( (x2.rv[t+1]*0x100000000+x2.rv[t]*0x10000+x2.rv[t-1])/(y.rv[t]*0x10000+y.rv[t-1]+1) );
                                }else{
                                q1 = (unsigned int)((x2.rv[t+1]*0x10000+x2.rv[t])/(y.rv[t]+1));
                                }
                        }else{
                                if(t >= 1){
                                q1 = (unsigned int)((x2.rv[t]*0x10000 + x2.rv[t-1])/ (y.rv[t]*0x10000
+ y.rv[t-1]+1) );
                                }else{
                                q1 = (unsigned int)((x2.rv[t])/(y.rv[t]+1));
                                }
                        }
                        while(x2 >= q1*y){
                                q1 = q1 + 1;
                        }
                        q1 = q1 - 1;
                        q.rv[s-t] = q1;
                        x2 = x2 - q1 * y;
                        if(s > t){
                                x2 = (x2 << 16);//* 0x10000;
                                x2.rv[0] = x.rv[s-t-1];
                        }
                }else{
                        if(s > t){
                                x2 = (x2 << 16);// * 0x10000;
                                x2.rv[0] = x.rv[s-t-1];
                        }
                }
        }
        q.torealpart();
        return q;
}
```

```
// 剰余計算
CmplxMPI res(                                                   // 剰余(結果)
        const CmplxMPI& x,                                      // 被除数
        const CmplxMPI& y)                              // 除数
{
        int s=0;
        CmplxMPI xx = x;

        if(xx.rs<0){s=-1; xx = -xx;}

        int n = x.getMaxColumn_r();
        int t = y.getMaxColumn_r();

        if( y == 0 ){                                   // 0による除算
                aError  theError( divideByZero, F_div, 0 );
                throw( theError );
        }
        if((x>0) && (x < y) ){                                  // 被除数< 除数
                return x;                                       // 剰余(結果)
        }
        if( x == y ){                                   // 被除数= 除数
                return 0;                                       // 剰余(結果)
        }

        CmplxMPI x2( t+DEF_COLS , 0, DEF_COLS, 0); // Use From  0  to (t+1) for shift.

        for(int k=0; k<=t; k++){
                x2.rv[t-k] = xx.rv[n-k];
        }

        for(int s =n; s >=t; s--){
                unsigned int q1;
                if(x2 >= y){
                        int w = x2.getMaxColumn_r();
                        if(w > t){
                                if(t >= 1){
                                q1 = (unsigned
int)( (x2.rv[t+1]*0x100000000+x2.rv[t]*0x10000+x2.rv[t-1])/(y.rv[t]*0x10000+y.rv[t-1]+1) );
                                }else{
                                q1 = (unsigned int)((x2.rv[t+1]*0x10000+x2.rv[t])/(y.rv[t]+1));
                                }
                        }else{
                                if(t >= 1){
                                q1 = (unsigned int)((x2.rv[t]*0x10000 + x2.rv[t-1])/ (y.rv[t]*0x10000
+ y.rv[t-1]+1) );
                                }else{
                                q1 = (unsigned int)((x2.rv[t])/(y.rv[t]+1));
                                }
                        }
```

```
                        while(x2 >= q1*y){
                                q1 = q1 + 1;
                        }
                        q1 = q1 - 1;
                        x2 = x2 - q1 * y;
                        if(s > t){
                                x2 = (x2 << 16);//* 0x10000;
                                x2.rv[0] = xx.rv[s-t-1];
                        }
                }else{
                        if(s > t){
                                x2 = (x2 << 16);// * 0x10000;
                                x2.rv[0] = x.rv[s-t-1];
                        }
                }
        }
        x2.torealpart();
        if(s == -1){
                return (y-x2);
        }
        return x2;
}


////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// 単項演算子  マイナス
CmplxMPI CmplxMPI::operator-( void )
{
        (*this).rs = (*this).rs * (-1);
        (*this).is = (*this).is * (-1);                         // 符号変換してから
        return *this;                                   // 自身を返す
}

////////////////////////////////////////////////////////////////////////////////

// 前置インクリメント
CmplxMPI CmplxMPI::operator++( void )
{
        *this = *this + 1;                              // 1加える
        return *this;                                   // 自身を返す
}

// 後置インクリメント
CmplxMPI CmplxMPI::operator++(
        int n )                                                 // 被演算数
{
        CmplxMPI r = *this;                                     // 戻り値をコピー
        *this = *this + 1;                              // 1加える
        return r;                                               // 戻り値を返す
```

```
}

// 前置デクリメント
CmplxMPI CmplxMPI::operator--( void )
{
        *this = *this - 1;                              // 1減じる
        return *this;                                   // 自身を返す
}

// 後置デクリメント
CmplxMPI CmplxMPI::operator--(
        int n )                                                 // 被演算数
{
        CmplxMPI r = *this;                                     // 戻り値をコピー
        *this = *this - 1;                              // 1減じる
        return r;                                               // 戻り値を返す
}

// 右シフト(実部のみ移動)
CmplxMPI operator>>(
        const CmplxMPI& a,                                      // 被演算数
        const int n )                                   // シフト数
{
        int vh, vl,temp;

        if( n < 0 )                                             // シフト数が負なら
                return a<<(-n);                         // 左シフト
        CmplxMPI r = a;                                 // 戻り値の初期化
        if( n == 0 )                                    // シフト数がなら
                return r;                                       // 何もしない

        if( n >= COL_LENGTH * r.rl )            // シフト数が桁数よりも多い
                return 0;

        int w = n / COL_LENGTH;                 // シフトする桁数
        int b = n - w * COL_LENGTH;             // 1桁あたりのシフトビット数
        int i;                                                  // カウンタ

        if(b != 0){
        for( i=0; i<r.rl-1-w; i++ ){            // 下位のシフト
                vh = (int)r.rv[i+w+1]; vl = (int)r.rv[i+w];
                temp = (( (vh << (COL_LENGTH-b))  | (vl >> b) ) & COL_MASK );
                r.rv[i] = double(COL_MASK & temp);
        }
        vl = (int)r.rv[i+w];
        temp = (( vl >> b ) & COL_MASK );
        r.rv[i] = double(COL_MASK & temp);      // 最後の桁の処置
        for( i++; i<r.rl; i++ )                 // 上位は
                r.rv[i] = 0;
        }

        if(b == 0){
```

```
        for( i=0; i<r.rl-1-w; i++ ){                // 下位のシフト
                r.rv[i] = r.rv[i+w];
        }
        for( ; i<r.rl; i++ )                        // 上位は
                r.rv[i] = 0;
        }

        return r;
}

// 左シフト(実部のみ移動)
CmplxMPI operator<<(
        const CmplxMPI& a,                                  // 被演算数
        const int n )                               // シフト数
{
        int vh, vl, temp;

        if( n < 0 )                                         // シフト数が負なら
                return a>>(-n);                     // 右シフト
        CmplxMPI r = a;                             // 戻り値の初期化
        if( n == 0 )                                // シフト数がなら
                return r;                                   // なにもしない

        int w = n / COL_LENGTH;                     // シフトする桁数
        int b = n - w * COL_LENGTH;                 // 1桁あたりのシフトビット数
        r.changeLength( w + 1 + r.rl, r.il );       // はみ出しが発生するなら桁を増やす

        int i;                                              // カウンタ

        if(b != 0){
        for( i=r.rl-1; i>w; i-- ){                  // 下位のシフト
                vh = (int)r.rv[i-w-1]; vl = (int)r.rv[i-w];
                temp = ((vh >> (COL_LENGTH-b))  | (vl << b)  & COL_MASK);
                r.rv[i] = double( (COL_MASK & temp) );
        }
        vl = (int)r.rv[i-w];
        temp = (( vl << b ) & COL_MASK);
        r.rv[i] = double( COL_MASK & temp );        // 最後の桁の処置
        for( i--; i>=0; i-- )                       // 下位は
                r.rv[i] = 0;
        }

        if(b == 0){
        for( i=r.rl-1; i>=w; i-- ){                 // 下位のシフト
                r.rv[i] = r.rv[i-w];
        }
        for( ; i>=0; i-- )                          // 下位は
                r.rv[i] = 0;
        }

        return r;
}
```

```
// 代入和
CmplxMPI& CmplxMPI::operator+=(
        const CmplxMPI& a )                                  // 加算数
{
        *this = *this + a;                              // 加える
        return *this;
}

// 代入差
CmplxMPI& CmplxMPI::operator-=(
        const CmplxMPI& a )                                  // 減算数
{
        *this = *this - a;                              // 減じる
        return *this;
}

// 代入積
CmplxMPI& CmplxMPI::operator*=(
        const CmplxMPI& a )                                  // 乗数
{
        *this = *this * a;                              // 乗ずる
        return *this;
}

// 代入商
CmplxMPI& CmplxMPI::operator/=(
        const CmplxMPI& a )                                  // 除数
{
        *this = quo(*this,a);                           // 除する
        return *this;
}

// 代入剰余
CmplxMPI& CmplxMPI::operator%=(
        const CmplxMPI& a )                                  // 除数
{
        *this = res(*this,a);                           // 剰余をとる
        return *this;
}

// 代入右シフト(実部のみ移動)
CmplxMPI& CmplxMPI::operator>>=(
        const int n )                                   // シフト数
{
        *this = *this >> n;
        return *this;
}

// 代入左シフト(実部のみ移動)
CmplxMPI& CmplxMPI::operator<<=(
        const int n )                                   // シフト数
```

```
{
        *this = *this << n;
        return *this;
}

/******************/
/*** 比較演算子***/
/******************/

// 等しい(複素数としての相等)
int     operator==(                                             // 真:  偽:
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        int     ll, ls;                                         // 長さ、比較の範囲
        double* p;                                      // 判定用
        if( a.rl > b.rl ){                              // 桁数の大きい方を基準とする
                ll = a.rl;      ls = b.rl;      p = a.rv;
        }
        else{
                ll = b.rl;      ls = a.rl;      p = b.rv;
        }
        int     i;
        for( i=0; i<ls; i++ )                   // 桁ごとに
                if( a.rv[i] != b.rv[i] )        // 違いがあれば
                        return 0;                               // 偽
        for( ; i<ll; i++ )                              // 万一上位の桁が
                if( p[i] != 0 )                         // 0でなければ
                        return 0;                               // 偽

        if( a.il > b.il ){                              // 桁数の大きい方を基準とする
                ll = a.il;      ls = b.il;      p = a.iv;
        }
        else{
                ll = b.il;      ls = a.il;      p = b.iv;
        }
        for( i=0; i<ls; i++ )                   // 桁ごとに
                if( a.iv[i] != b.iv[i] )        // 違いがあれば
                        return 0;                               // 偽
        for( ; i<ll; i++ )                              // 万一上位の桁が
                if( p[i] != 0 )                         // 0でなければ
                        return 0;                               // 偽

        return 1;                                               // そうでなければ真
}

// 等しくない(複素数として等しくない)
int     operator!=(                                             // 真:  偽:
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        return  !( a == b );                    // 等号の反対
```

```
}

// 大きいか等しい（実部の絶対値で比較）符号は無視する。
int     operator>=(                                             // 真：　偽：
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        int i, ll;                                              // カウンタ,共通長

        if( a.rl > b.rl ){                                  // aの方が桁数が多いなら
                ll = b.rl-1;
                for( i=a.rl-1; i>ll; i-- ){
                        if( a.rv[i] != 0 ){
                                return 1;
                        }
                }
        }
        else{                                                   // bの方が桁数が多いなら
                ll = a.rl-1;
                for( i=b.rl-1; i>ll; i-- ){
                        if( b.rv[i] != 0 ){
                                return 0;
                        }
                }
        }

        for( ; i>=0; i-- ){                                 // 長さが同じ部分
                if( a.rv[i] > b.rv[i] ){            // a.v[i] == b.v[i]のときはループを回る
                        return 1;
                        }
                else{
                        if( a.rv[i] < b.rv[i] ){
                                return 0;
                        }
                }
        }

        return 1;                                               // =のとき
}

// 小さいか等しい（実部の絶対値で比較）符号は無視する。
int     operator<=(                                             // 真：　偽：
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        int i, ll;                                              // カウンタ,共通長

        if( a.rl > b.rl ){                                  // aの方が桁数が多い
                ll = b.rl-1;
                for( i=a.rl-1; i>ll; i-- ){
                        if( a.rv[i] != 0 ){
                                return 0;
```

```
                        }
                }
        }
        else{                                                   // bの方が桁数が多い
                ll = a.rl-1;
                for( i=b.rl-1; i>ll; i-- ){
                        if( b.rv[i] != 0 ){
                                return 1;
                        }
                }
        }

        for( ; i>=0; i-- ){                                     // 長さが同じ部分
                if( a.rv[i] > b.rv[i] )  {          // a.v[i] == b.v[i]のときはループを回る
                        return 0;
                }
                else{
                        if( a.rv[i] < b.rv[i] )  {
                                return 1;
                        }
                }
        }

        return 1;                                               // =のとき
}

// 大きい（実部の絶対値で比較）
int     operator>(                                              // 真：  偽：
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        return !( a <= b );                                 // <=の反対
}

// 小さい（実部の絶対値で比較）
int     operator<(                                              // 真：  偽：
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b )                                     // 演算数
{
        return !( a >= b );                                 // >=の反対
}

/************/
/*** 関数***/
/************/

// 奇偶判定（実部のみ比較）
int     isEven(                                                 // 偶数:1,奇数:0
        const CmplxMPI& a )                                     // 判定数
{
        int rvl;
```

```
        rvl = (int)(a.rv[0]);
        return !( rvl & 0x1);           // 最下位ビットがなら偶数
}

// べき乗（実部のみ返す）
CmplxMPI pow(
        const CmplxMPI& a,                              // 被演算数
        const CmplxMPI& b )                             // 演算数
{
        CmplxMPI r = 1;                         // 結果の初期化
        if( b == 0 )                            // 0乗
                return r;                               // 1

        CmplxMPI aa = a, bb = b;                // 作業用コピー

        while( bb > 0 ){                        // 演算数がになるまで
                if( isEven( bb ) ){                     // 偶数ならば
                        bb >>= 1;                               // 演算数を/2にする
                        aa *= aa;                               // 被演算数を二乗する
                }
                else{                                   // 奇数ならば
                        bb -= 1 ;                                       // 演算数を減ずる
                        r *= aa;                        // 結果を更新する
                }
        }
        r.torealpart();
        return r;                                       // この時点でrがべき乗になっている
}

// べき乗（実部のみ返す）
CmplxMPI powFFT(
        const CmplxMPI& a,                              // 被演算数
        const CmplxMPI& b )                             // 演算数
{
        CmplxMPI r = 1;                         // 結果の初期化
        if( b == 0 )                            // 0乗
                return r;                               // 1

        CmplxMPI aa = a, bb = b;                // 作業用コピー
        aa.torealpart();
        bb.torealpart();

        while( bb > 0 ){                        // 演算数がになるまで
                if( isEven( bb ) ){                     // 偶数ならば
                        bb >>= 1;                               // 演算数を/2にする
                        aa = pFFT(aa,aa);
//                      aa *= aa;                               // 被演算数を二乗する
                }
                else{                                   // 奇数ならば
                        bb -= 1 ;                                       // 演算数を減ずる
                        r = pFFT(r,aa);
//                      r *= aa;                        // 結果を更新する
```

```
                }
        }
        r.torealpart();
        return r;                                       // この時点でrがべき乗になっている
}

// 剰余算べき乗（実部のみ返す）
CmplxMPI exp(
        const CmplxMPI& a,                              // 被演算数
        const CmplxMPI& b,                              // 演算数
        const CmplxMPI& c )                             // 基数
{
        CmplxMPI r = 1;                                 // 結果の初期化
        if( b == 0 )                                    // 0乗は
                return r;                                       // 1

        CmplxMPI aa = a, bb = b;                // 作業用コピー
        aa.torealpart(); bb.torealpart();

        while( bb > 0 ){                                // 演算数がになるまで
                if( isEven( bb ) ){                             // 偶数ならば
                        bb >>= 1;                               // 演算数を/2にする
                        aa *= aa;                               // 被演算数を二乗する
                        aa %= c;                        // 剰余をとる
                }
                else{                                           // 奇数ならば
                        bb -= 1;                                // 演算数を減ずる
                        r *= aa;                        // 結果を更新する
                        r %= c;                                 // 結果の剰余をとる
                }
        }

        r.torealpart();
        return r;                                       // この時点でrがべき乗(mod c)になっ
ている
}

// DFTによる、剰余算べき乗（実部のみ返す）
CmplxMPI expDFT(
        const CmplxMPI& a,                              // 被演算数
        const CmplxMPI& b,                              // 演算数
        const CmplxMPI& c )                             // 基数
{
        CmplxMPI r = 1;                                 // 結果の初期化
        if( b == 0 )                                    // 0乗は
                return r;                                       // 1

        CmplxMPI aa = a, bb = b;                // 作業用コピー
        aa.torealpart(); bb.torealpart();

        while( bb > 0 ){                                // 演算数がになるまで
                if( isEven( bb ) ){                             // 偶数ならば
```

```
                        bb >>= 1;                               // 演算数を/2にする
//                      aa *= aa;                               // 被演算数を二乗する
                        aa = pDFT(aa, aa);
                        aa = res(aa,c);
                        //aa %= c;                              // 剰余をとる
                }
                else{                                           // 奇数ならば
                        bb -= 1;                                // 演算数を減ずる
//                      r *= aa;                        // 結果を更新する
                        r = pDFT(r, aa);
                        r = res(r,c);
                        //r %= c;                                       // 結果の剰余をとる
                }
        }

        r.torealpart();
        return r;                                               // この時点でrがべき乗(mod c)になっ
ている
}

// FFTによる、剰余算べき乗（実部のみ返す）
CmplxMPI expFFT(
        const CmplxMPI& a,                                      // 被演算数
        const CmplxMPI& b,                                      // 演算数
        const CmplxMPI& c )                                     // 基数
{
        CmplxMPI r = 1;                                 // 結果の初期化
        if( b == 0 )                                    // 0乗は
                return r;                                       // 1

        CmplxMPI aa = a, bb = b;                // 作業用コピー
        aa.torealpart(); bb.torealpart();

        while( bb > 0 ){                                // 演算数がになるまで
                if( isEven( bb ) ){                     // 偶数ならば
                        bb >>= 1;                               // 演算数を/2にする
//                      aa *= aa;                               // 被演算数を二乗する
                        aa = pFFT(aa, aa);
                        aa = res(aa,c);
                        //aa %= c;                              // 剰余をとる
                }
                else{                                           // 奇数ならば
                        bb -= 1;                                // 演算数を減ずる
//                      r *= aa;                        // 結果を更新する
                        r = pFFT(r, aa);
                        r = res(r,c);
                        //r %= c;                                       // 結果の剰余をとる
                }
        }

        r.torealpart();
        return r;                                               // この時点でrがべき乗(mod c)になっ
```

```
ている
}



/************/
/*** 乱数***/
/************/

// 乱数生成の初期化
void randInitial( void )
{
        srand( (unsigned)time( NULL ) );
}

// 乱数生成（桁数指定）
CmplxMPI random(
        const int n )                                   // 乱数の桁数
{
        CmplxMPI a( n, 0 ,n , 0);
        for( int i=0; i<n; i++ )
                a.rv[i] = (short)rand();
        for( int i=0; i<n; i++ )
                a.iv[i] = (short)rand();
        return a;
}

// 乱数生成
CmplxMPI random( void )
{
        CmplxMPI a;
        a.is2 = 1;
        for( int i=0; i<a.rl; i++ )
                a.rv[i] = (short)rand();          // 鍵の設定
        for( int i=0; i<a.rl; i++ )
                a.iv[i] = (short)rand();          // 鍵の設定
        return a;
}

// 乱数生成（bit長指定）
CmplxMPI randomBit( const int l )
{
        CmplxMPI r = random( l / COL_LENGTH + 1 );
        r >>= r.getBitLength() - l;
        return  r;
}

////////////////////////////////////////////////////////////

// 符号付の乱数生成（桁数指定）
CmplxMPI randomSig(
        const int n )                                   // 乱数の桁数
```

```
{
        CmplxMPI a( n, 0 ,n , 0);
        for( int i=0; i<n; i++ )
                a.rv[i] = (short)rand();
        for( int i=0; i<n; i++ )
                a.iv[i] = (short)rand();

        int a0 = (int)(a.rv[0]);
        if(a0 & 0x01) a.rs = -1;
        int b0 = (int)(a.iv[0]);
        if(b0 & 0x01) a.is = -1;

        return a;
}

// 符号付の乱数生成
CmplxMPI randomSig( void )
{
        CmplxMPI a;
        a.is2 = 1;
        for( int i=0; i<a.rl; i++ )
                a.rv[i] = (short)rand();          // 鍵の設定
        for( int i=0; i<a.rl; i++ )
                a.iv[i] = (short)rand();          // 鍵の設定

        int a0 = (int)(a.rv[0]);
        if(a0 & 0x01) a.rs = -1;
        int b0 = (int)(a.iv[0]);
        if(b0 & 0x01) a.is = -1;

        return a;
}

// 符号付の乱数生成（bit長指定）
CmplxMPI randomBitSig( const int l )
{
        CmplxMPI r = randomSig( l / COL_LENGTH + 1 );
        r >>= r.getBitLength() - l;
        return  r;
}

/**************/
/*** 整数論***/
/**************/

// 素数判定：ミラー・ラビン法
int millerrabin(                                        // 素数：  非素数：
        const CmplxMPI& nn,
        const int t)                                    // 安全助変数
{
        CmplxMPI n = nn;
        if( (n == 1) || (n == 0) )      return 0;               // 0,1は素数ではない
```

```
        if(  n == 2 )    return 1;        // 2は素数
        if( isEven( n ) )        return 0;        // 偶数ならば素数ではない

        // n-1 == 2^s*r とおく
        CmplxMPI n1 = n - 1;                                    // n - 1
        CmplxMPI s = 0;                                         // 0から始める
        CmplxMPI r = n - 1;                                     // s=0 ならm=n-1
        CmplxMPI    v1(1);

        while( isEven( r ) ){                   // n-1 = 2^s*m のs,mを求める(mは奇数)
                r >>= 1;                                        // mを/2に
                s++;
        }

        for(int i=1; i<=t; i++){
                CmplxMPI a = 1;                                         // [2, n-1]の乱数生成
                while( a < 2 ){
                        a = random();
                        while( a > n - 2 )
                                a >>= 1;
                }
                a.torealpart(); r.torealpart(); n.torealpart();
                CmplxMPI y = expFFT( (a), (r), (n) ); // y = a^( (2^i)*r ) mod n
                if( (y!=v1) && (y!=n1) ){
                        int j = 1;
                        while((j<s) && (y!=n1)){
                                y = expFFT(y,2,n);
                                if( y == v1 )    return 0;                  // 素数ではない
                                j = j+1;
                        }
                        if( y != n1 )    return 0;                  // 素数ではない
                }
        }
        return 1;                                               // 素数
}


// 素数判定：ラビン法
int rabin(                                                      // 素数：　非素数：
        const CmplxMPI& nn )                                    // 被検査数
{
        CmplxMPI n = nn;
        if( (n == 1) || (n == 0) )        return 0;             // 0,1は素数ではない
        if(  n == 2 )    return 1;        // 2は素数
        if( isEven( n ) )        return 0;        // 偶数ならば素数ではない

        // n-1 == 2^s*m とおく
        CmplxMPI n1 = n - 1;                                    // n - 1
        CmplxMPI s = 0;                                         // 0から始める
        CmplxMPI m = n - 1;                                     // s=0 ならm=n-1
        CmplxMPI    v1(1);
```

```
        while( isEven( m ) ){                   // n-1 = 2^s*m のs,mを求める(mは奇数)
                m >>= 1;                                // mを/2に
                s++;
        }

        CmplxMPI a = 1;                                 // [2, n-1]の乱数生成
        while( a < 2 ){
                a = random();
                while( a > n )
                        a >>= 1;
        }

        a.torealpart(); m.torealpart(); n.torealpart();

//      CmplxMPI y = exp( (a), (m), (n) );      // y = a^( (2^i)*m ) mod n
//      CmplxMPI y = expDFT( (a), (m), (n) );           // y = a^( (2^i)*m ) mod n
        CmplxMPI y = expFFT( (a), (m), (n) );           // y = a^( (2^i)*m ) mod n

        if( y == 1 )    return 1;               // 素数
        if( y == n1 )   return 1;               // 素数

        CmplxMPI i = 1;
        while( i < s ){
//              y = exp( (y), 2, (n) );                 // y = y^2 mod n
//              y = expDFT( (y), 2, (n) );                      // y = y^2 mod n
                y = expFFT( (y), 2, (n) );                      // y = y^2 mod n

                if( y == n1 )   return 1;       // 素数
                if( y == v1 )   return 0;       // 素数ではない
                i++;
        }
        return 0;                                               // 素数ではない
}

// 素数判定：フェルマーテスト
// 判定できる数：v = h * p + 1;の形のもの
// 前提条件：p:素数, h<p, p>2
int     fermat(                                         // 素数：  非素数：
        const CmplxMPI& h,                              // 被検査数h
        const CmplxMPI& p )                             // 被検査数p
{
        CmplxMPI r;
        CmplxMPI d1(1);

        CmplxMPI rv = (h * p + 1);                      // 判定する素数候補
        rv.torealpart();
        int     a = 7 * 11;// 100以下の素数の積
        r = ugcd(rv, a);
        r.torealpart();
        if( r != d1 )           return  0;      // 素数ではない
        r = expFFT( 2, (rv-1), rv );
        if( r != d1 )   return  0;      // 素数ではない
```

```
        r = expFFT( 2, (h), rv );
        if( r == d1 )    return  0;       // 素数ではない
        return 1;                                                   // 素数
}

// 最大公約数
CmplxMPI gcd(
        CmplxMPI a,                                                     // 演算数
        CmplxMPI b )                                                // 演算数
{
        while( b > 0 ){
                CmplxMPI t = res(a,b);//a % b;
                a = b;
                b = t;
        }
        a.torealpart();
        return  a;
}

// 最小公倍数
CmplxMPI lcm(
        const CmplxMPI& a,                                          // 演算数
        const CmplxMPI& b )                                         // 演算数
{
        return  (quo(( a * b ), ugcd( a, b )));
}

// 逆数(合同式の解)
CmplxMPI inv(
        const CmplxMPI& a,                                          // 演算数
        const CmplxMPI& n )                                         // 法
{
        if( ugcd( a, n ) != 1 )  return  0;      // 互いに素でなければ逆数はない

        CmplxMPI q, s = a, t = n, u = 1, v = 0, w, x;       // 作業用
        int             qs, ws, ts, ss, vs, us, xs;                             // 負号  :正0:負
        qs = ws = ts = ss = vs = us = xs = 1;

// 中間結果は負になり得ることに注意

        while( ss != 0 && s > 0 ){              // s > 0
                q = quo(t,s)/* t / s*/;          qs = ts == ss;
                x = q * s;              xs = qs == ss;
                if( ts ){
                        if( xs )                        // + - +
                                if( t >= x ){
                                        w = t - x;      ws = 1;
                                }
                                else{
                                        w = x - t;      ws = 0;
                                }
                        else{                                   // + - -
```

```
                        w = t + x;      ws = 1;
                }
        }
        else{
                if( xs ){                               // - - +
                        w = t + x;      ws = 0;
                }
                else{                                   // - - -
                        if( t >= x ){
                                w = t - x;      ws = 0;
                        }
                        else{
                                w = x - t;      ws = 1;
                        }
                }
        }

        t = s;          ts = ss;
        s = w;          ss = ws;

        x = q * u;      xs = qs == us;
        if( vs ){
                if( xs ){                               // + - +
                        if( v >= x ){
                                w = v - x;      ws = 1;
                        }
                        else{
                                w = x - v;      ws = 0;
                        }
                }
                else{                                   // + - -
                        w = v + x;      ws = 1;
                }
        }
        else{
                if( xs ){                               // - - +
                        w = v + x;      ws = 0;
                }
                else{                                   // - - -
                        if( v >= x ){
                                w = v - x;      ws = 0;
                        }
                        else{
                                w = x - v;      ws = 1;
                        }
                }
        }

        v = u;          vs = us;
        u = w;          us = ws;
}
if( ! vs ){                                             // 負のとき
```

```
                v = n - v;
                v.torealpart();
                return  v;//n - v;
        }

        v.torealpart();
        return v;
}


// 最大公約数
CmplxMPI ugcd(
        const CmplxMPI& xx,                                     // 演算数
        const CmplxMPI& yy )                                    // 法
{

        CmplxMPI x, y , g = 1,t;  // 作業用

        if(xx >= yy){ x = xx; y = yy;}
        else{ x = yy; y = xx; }

        while(isEven(x) && isEven(y)){
                x >>= 1; y >>= 1; g <<= 1;
        }
        while( x != 0){
                while(isEven(x)){ x >>= 1;}
                while(isEven(y)){  y >>= 1;}
                t = (x-y)>>1; t.rs = 1;
                if(x>=y){x = t;}
                else{ y = t;}
        }
        return (g*y);
}

// 逆数(合同式の解)
CmplxMPI uinv(
        const CmplxMPI& xx,                                     // 演算数
        const CmplxMPI& yy )                                    // 法
{
        if( ugcd( xx, yy ) != 1 )return  0;       // 互いに素でなければ逆数はない

        CmplxMPI x =xx, y = yy, g = 1,a,b, u,v,A,B,C,D;     // 作業用

        while(isEven(x) && isEven(y)){
                x >>= 1; y >>= 1; g <<= 1;
        }
        u = x; v = y; A = 1; B = 0; C = 0; D = 1;
S4:
        while(isEven(u)){
                u = u>>1;
                if(isEven(A) && isEven(B)){
                        A >>= 1; B >>= 1;
```

```
                }else{
                        A = (A+y)>>1; B = (B-x)>>1;
                }
        }
        while(isEven(v)){
                v = v>>1;
                if(isEven(C) && isEven(D)){
                        C >>= 1; D >>= 1;
                }else{
                        C = (C+y)>>1; D = (D-x)>>1;
                }
        }
        if(u>=v){
                u = u-v; A = A-C; B = B-D;
        }else{
                v = v-u; C = C-A; D = D-B;
        }
        if(u == 0){
                a = C; b = D;
                if(a.rs<0){a = a+yy;}
                return a;
        }else{
                goto S4;
        }
}



//////////////////////////////////////////
CmplxMPI pDFT(
                        const CmplxMPI& a,                          // DFT による積
                        const CmplxMPI& b ){

        #define PI 3.141592653589793

        double c = 0.0;
        double de, t;
        unsigned int tv, tv1, tv2;
        int i,j;
        int la = max(a.rl,a.il);
        int lb = max(b.rl,b.il);
        int ml = max(la,lb);
        int ll = ml * 2;
        CmplxMPI za(ll+1, 0, ll+1, 0);
        CmplxMPI zb(ll+1, 0, ll+1, 0);
        CmplxMPI r(ll+1, 0, ll+1, 0);

        for(i=0; i<ll; i++){
                double iP = i*PI/ml;
                for(j=0; j<ml; j++){
                        double jiP = j*iP;
                        double co = cos(jiP);
```

```
                        double si = sin(jiP);
                        if(j<a.rl){
                        za.rv[i] += a.rv[j]*co;//s(j*iP);
                        za.iv[i] += a.rv[j]*si;//n(j*iP);
                        }
                        if(j<b.rl){
                        zb.rv[i] += b.rv[j]*co;//s(j*iP);
                        zb.iv[i] += b.rv[j]*si;//n(j*iP);
                        }
                }
        }

        for(i=0; i<ll; i++){
                double iP = i*PI/ml;
                for(j=0; j<ll; j++){
                                r.rv[i] +=  (za.rv[j]*zb.rv[j] - za.iv[j]*zb.iv[j])*cos(j*iP)
                                          +  (za.rv[j]*zb.iv[j] + za.iv[j]*zb.rv[j])*sin(j*iP);
                }
                r.rv[i] /= ll;
                if(r.rv[i]<0.0){r.rv[i] = 0.0;}
        }

        ////////////////////////////////////////
/*
        for(i=0; i<ll; i++){
                while(r.rv[i] > 0xffff0000){
                        r.rv[i] -= 0xffff0000;
                        r.rv[i+1] += 0xffff;
                }
        }
*/

                for(i=0; i<ll; i++){
                        unsigned int tmp = (unsigned int)(r.rv[i]/0xffff0000);
                        r.rv[i] = r.rv[i] - (double)(tmp)*0xffff0000;
                        r.rv[i+1] += (double)(tmp)*0xffff;
                        }
////////////////////////////////////


        c = 0.0;
        for( i=0; i<ll; i++ ){                          // 1桁ずつ加える
                t = r.rv[i] + c ;               // i桁目と、i-1桁目の繰り上がりを加える
                tv = (unsigned int)t;
                tv1 = tv & COL_MASK;
                tv2 = (tv >> COL_LENGTH) & COL_MASK;
                de = t - tv;
                if(de >0.9){
                        if(tv1 != 0xffff){
                                tv1 += 1;
                        }
                        else{
```

```
                                tv1 = 0x0000;
                                tv2 += 1;
                        }
                }
                r.rv[i] = (double)(tv1);          // 繰り上がりを除いてi桁目の和にする
                c =  (double)(tv2);               // 次の桁への繰り上げ
        }
        r.rv[ll] = c;

        r.torealpart();
        return r;
}

////////////////////////////////////
CmplxMPIpFFT(
                                 const CmplxMPI& a,
                                const CmplxMPI& b ){                    // FFTによる積

        #define PI 3.141592653589793

        double c = 0.0;
        double de, t;
        unsigned int tv, tv1,tv2;
        int i,j;
        int la = max(a.rl,a.il);
        int lb = max(b.rl,b.il);
        int ml = max(la,lb);
        int ll = ml * 2;


        CmplxMPI z(ll+1, 0, ll+1, 0);

        int it,xp,xp2,k,j1,j2,im1,jm1,iter ;
        double sign,w,wr,wi,dr1,dr2,di1,di2,tr,ti,arg ;


        if (ll < 2){ printf("n<2 ¥n");}
        iter = 0;
        if (iter <= 0)
                {
                iter = 0 ;
                i = ll ;
                while (1)
                        {
                        if ((i /= 2) == 0) break ;
                        iter++ ;
                        }
                }
        int jj = 1 ;
        for (i = 0; i < iter; i++) jj *= 2 ;

////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////
        if(ll != jj){                   // ２のべき乗でなければDFT
                return (pDFT(a,b));
        }
////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////

        if(ll == jj){
                CmplxMPI aa = a;
                CmplxMPI bb = b;
                aa.changeLength(ll+1,ll+1);
                bb.changeLength(ll+1,ll+1);
////////////////////////////////////
        sign =1.0 ;
        xp2 = ll ;
        for (it = 0; it < iter ; it++)
                {
                xp = xp2 ;
                xp2 = xp / 2 ;
                w = PI / xp2 ;
                for (k = 0; k < xp2; k++)
                        {
                        arg = k * w ;
                        wr = cos(arg) ;
                        wi = sign * sin(arg) ;
                        i = k - xp ;
                        for (j = xp; j <= ll; j += xp)
                                {
                                j1 = j + i ;
                                j2 = j1 + xp2 ;
                                dr1 = aa.rv[j1] ;
                                dr2 = aa.rv[j2] ;
                                di1 = aa.iv[j1] ;
                                di2 = aa.iv[j2] ;
                                tr = dr1 - dr2 ;
                                ti = di1 - di2 ;
                                aa.rv[j1] = dr1 + dr2 ;
                                aa.iv[j1] = di1 + di2 ;
                                aa.rv[j2] = tr * wr - ti * wi ;
                                aa.iv[j2] = ti * wr + tr * wi ;

                                dr1 = bb.rv[j1] ;
                                dr2 = bb.rv[j2] ;
                                di1 = bb.iv[j1] ;
                                di2 = bb.iv[j2] ;
                                tr = dr1 - dr2 ;
                                ti = di1 - di2 ;
                                bb.rv[j1] = dr1 + dr2 ;
                                bb.iv[j1] = di1 + di2 ;
                                bb.rv[j2] = tr * wr - ti * wi ;
                                bb.iv[j2] = ti * wr + tr * wi ;
                                }
```

```
                        }
                }
        j1 = ll / 2 ;
        j2 = ll - 1 ;
        j = 1 ;
        for (i = 1; i <= j2; i++)
                {
                if (i < j)
                        {
                        im1 = i - 1 ;
                        jm1 = j - 1 ;
                        tr = aa.rv[jm1] ;
                        ti = aa.iv[jm1] ;
                        aa.rv[jm1] = aa.rv[im1] ;
                        aa.iv[jm1] = aa.iv[im1] ;
                        aa.rv[im1] = tr ;
                        aa.iv[im1] = ti ;

                        tr = bb.rv[jm1] ;
                        ti = bb.iv[jm1] ;
                        bb.rv[jm1] = bb.rv[im1] ;
                        bb.iv[jm1] = bb.iv[im1] ;
                        bb.rv[im1] = tr ;
                        bb.iv[im1] = ti ;

                        }
                k = j1 ;
                while (k < j)
                        {
                        j -= k ;
                        k /= 2 ;
                        }
                j += k ;
                }
////////////////////////////////////////
                for(i=0; i<ll; i++){
                        z.rv[i] = aa.rv[i]*bb.rv[i] - aa.iv[i]*bb.iv[i];
                        z.iv[i] = aa.rv[i]*bb.iv[i] + aa.iv[i]*bb.rv[i];
                }
////////////////////////////////////////////
                sign = -1.0 ;
                xp2 = ll ;
                for (it = 0; it < iter ; it++)
                        {
                        xp = xp2 ;
                        xp2 = xp / 2 ;
                        w = PI / xp2 ;
                        for (k = 0; k < xp2; k++)
                                {
                                arg = k * w ;
                                wr = cos(arg) ;
                                wi = sign * sin(arg) ;
```

```
                                i = k - xp ;
                                for (j = xp; j <= ll; j += xp)
                                        {
                                        j1 = j + i ;
                                        j2 = j1 + xp2 ;
                                        dr1 = z.rv[j1] ;
                                        dr2 = z.rv[j2] ;
                                        di1 = z.iv[j1] ;
                                        di2 = z.iv[j2] ;
                                        tr = dr1 - dr2 ;
                                        ti = di1 - di2 ;
                                        z.rv[j1] = dr1 + dr2 ;
                                        z.iv[j1] = di1 + di2 ;
                                        z.rv[j2] = (tr * wr - ti * wi) ;
                                        z.iv[j2] = (ti * wr + tr * wi) ;
                                }
                        }
                }
                j1 = ll / 2 ;
                j2 = ll - 1 ;
                j = 1 ;
                for (i = 1; i <= j2; i++){
                        if (i < j){
                                im1 = i - 1 ;
                                jm1 = j - 1 ;
                                tr = z.rv[jm1] ;
                                ti = z.iv[jm1] ;
                                z.rv[jm1] = z.rv[im1] ;
                                z.iv[jm1] = z.iv[im1] ;
                                z.rv[im1] = tr ;
                                z.iv[im1] = ti ;
                        }
                        k = j1 ;
                        while (k < j){
                                j -= k ;
                                k /= 2 ;
                        }
                        j += k ;
                }
                for(i=0;i<ll;i++){
                        z.rv[i] /= ll;
                        if(z.rv[i]<0.0){z.rv[i] = 0.0;}
                }
////////////////////////////////////////////////////////////
/*
                for(i=0; i<ll; i++){
                        while(z.rv[i] > 0xffff0000){
                                z.rv[i] -= 0xffff0000;
                                z.rv[i+1] += 0xffff;
                        }
                }
*/
```

```
                for(i=0; i<ll; i++){
                        unsigned int tmp = (unsigned int)(z.rv[i]/0xffff0000);
                        z.rv[i] = z.rv[i] - (double)(tmp)*0xffff0000;
                        z.rv[i+1] += (double)(tmp)*0xffff;
                        }

////////////////////////////////////////////////////////////
                for( i=0; i<ll; i++ ){                  // 1桁ずつ加える
                        t = z.rv[i] + c ;       // i桁目と、i-1桁目の繰り上がりを加える
                        tv = (unsigned int)t;
                        tv1 = tv & COL_MASK;
                        tv2 = (tv >> COL_LENGTH) & COL_MASK;
                        de = t - tv;
                        if(de >0.9){
                                if(tv1 != 0xffff){
                                        tv1 += 1;
                                }
                                else{
                                        tv1 = 0x0000;
                                        tv2 += 1;
                                }
                        }
                        z.rv[i] = (double)(tv1);        // 繰り上がりを除いてi桁目の和にする
                        c =  (double)(tv2);             // 次の桁への繰り上げ
                }
                z.rv[ll] = c;

                z.torealpart();
                return z;
                }
        z.torealpart();
        return z;
        }


////////////////////////////////////////////////////////////////////////////////////
int             Jacobi(                                                 // ヤコビ記号計算
                        const CmplxMPI& a,                      // 係数h
                        const CmplxMPI& n)                      // 素数p
{
        int s=1,ss=1, e=0;
        if(a == 0) return 0;
        if(a == 1) return 1;
        CmplxMPI aa, nn=n, tmp;                 // 作業用コピー
//      aa.torealpart(); nn.torealpart();

    aa = res(a,n);//%= nn;
A:
        while(isEven(aa)){
                aa >>= 1;
                e +=1;
```

```
        }
        if(e%2 == 0){
                s = 1;
        }
        else{
                tmp = res(nn,8);
                if(tmp==1 || tmp==7){s = 1;}//(res(nn,8)==1) || (res(nn,8)==7)){s = 1;}
                if(tmp==3 || tmp==5){s = -1;}//(res(nn,8)==3) || (res(nn,8)==5)){s = -1;}
        }
        if((res(nn,4)==3) && (res(aa,4)==3)){s = -s;}

        if(aa == 1){
                return(ss*s);
        }
        else{
                tmp = res(nn,aa);
                nn = aa;
                aa = tmp;

                ss = ss*s; e=0;
                goto A;
//              return(s*Jacobi(aa,nn));
        }
}

CmplxMPI Sqrmp(                                                 // ヤコビ記号計算
                        const CmplxMPI& a,                      // 係数h
                        const CmplxMPI& p)                      // 素数p
{
        int r, e=0;
        CmplxMPI aa = a,y,x,b,q,t;                      // 作業用コピー
        CmplxMPI n = 1;
        while(Jacobi(n,p) != -1){
                n = n+1;
        }
        q = p-1;
        while(isEven(q)){
                q >>= 1;
                e = e+1;
        }
        y = exp(n,q,p);
        r = e;
        x = exp(aa,(q-1)>>=1,p);
        aa = res(aa*x,p);//
        b = res(aa*x,p);//(aa*x)*x), p);
        x = aa;//res((aa*x), p);
        while(res(b,p) != 1){
                int k = 2, m=1;
                while(exp(b,k,p) != 1){
                        k = k*2;
                        m++;
                }
```

```
                t = res((y<<=(r-m-1)), p);
                y = res((t*t), p);
                r = m;
                x = res((x*t), p);
                b = res((b*y), p);
        }
        return x;
}



/********************/
/*** デストラクタ***/
/********************/

EcPoint::~EcPoint( void )
{
        /*
        if( (x.rl == 0) && (x.il == 0) && (y.rl == 0) && (y.il == 0))       // 領域がなければ
                return;                                                 // 終了
        if(x.rl!=0){delete[] x.rv;}                                                 // 領域があれば解
放
        if(x.il!=0){delete[] x.iv;}
        if(y.rl!=0){delete[] y.rv;}                                                 // 領域があれば解
放
        if(y.il!=0){delete[] y.iv;}
        */
}



//class EcPoint  // コンストラクタ（型変換）
EcPoint::EcPoint(void)
{
        x.rs = 1;
        x.is = 1;
        x.is2 = 0;

        y.rs = 1;
        y.is = 1;
        y.is2 = 0;

        x.rv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
        if( x.rv == 0 ){                                        // メモリ割り当て失敗時
                x.rl = 0;                                                       // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                              // 上位のエラー処理へ
        }
        x.rl = DEF_COLS;                                        // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                x.rv[i] = 0.0;                                          // 値の初期化
```

```
        x.iv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
        if( x.iv == 0 ){                                    // メモリ割り当て失敗時
                x.il = 0;                                                   // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                          // 上位のエラー処理へ
        }
        x.il = DEF_COLS;                                    // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                x.iv[i] = 0.0;                                      // 値の初期化

        ////////////////////////////

        y.rv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
        if( y.rv == 0 ){                                    // メモリ割り当て失敗時
                y.rl = 0;                                                   // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                          // 上位のエラー処理へ
        }
        y.rl = DEF_COLS;                                    // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                y.rv[i] = 0.0;                                      // 値の初期化


        y.iv = new( double[DEF_COLS] );   // デフォルトサイズで領域確保
        if( y.iv == 0 ){                                    // メモリ割り当て失敗時
                y.il = 0;                                                   // 無効
                aError theError( notEnoughMemory, C_void, sizeof(double) * DEF_COLS );
                throw( theError );                          // 上位のエラー処理へ
        }
        y.il = DEF_COLS;                                    // デフォルトの長さ
        for(int i=0; i<DEF_COLS; i++)
                y.iv[i] = 0.0;                                      // 値の初期化

}

EcPoint::EcPoint(                                           // intからの型変換
                const CmplxMPI a,
                const CmplxMPI b)
{
        x =     a;
        y = b;
}

EcPoint::EcPoint(                                           // 桁数を指定して実部のすべての桁
をvalでうめる(通常かffff)
                const int a,                                        // 要求桁数
                const int b)                                // 値（下位ビット有効）
                {
                        x = a;
                        y = b;
                }
```

```
EcPoint::EcPoint(                                               // 16進数文字列から
                const char* pa,
                const char*      pb)                    // 文字列
{
        x = pa;
        y = pb;
}

// コピー(領域を別に持つ)
EcPoint::EcPoint(
        const EcPoint& a )                                      // 元のMPI
{
        x.rs = a.x.rs;
        x.is = a.x.is;
        x.is2 = a.x.is2;
        y.rs = a.y.rs;
        y.is = a.y.is;
        y.is2 = a.y.is2;

        if( a.x.rl ){                                           // 元のMPIの長さがでなければ
                x.rv = new( double[a.x.rl] );     // 同じ大きさの領域を確保
                if( x.rv == 0 ){                        // 領域確保失敗時
                        x.rl = 0;                                       // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.x.rl );
                        throw( theError );              // 上位のエラー処理へ
                }
                memcpy( x.rv, a.x.rv, a.x.rl*(sizeof(double)) );  // bit数をbyte数へ変換してコピー
        }
        else
                x.rv = 0;                                               // 元のMPIの長さがならば
        x.rl = a.x.rl;                                          // 長さを同じに

        if( a.x.il ){                                           // 元のMPIの長さがでなければ
                x.iv = new( double[a.x.il] );     // 同じ大きさの領域を確保
                if( x.iv == 0 ){                        // 領域確保失敗時
                        x.il = 0;                                       // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.x.il );
                        throw( theError );              // 上位のエラー処理へ
                }
                memcpy( x.iv, a.x.iv, a.x.il*(sizeof(double)) );  // bit数をbyte数へ変換してコピー
        }
        else
                x.iv = 0;                                               // 元のMPIの長さがならば
        x.il = a.x.il;                                          // 長さを同じに
////////////////////////////////
        if( a.y.rl ){                                           // 元のMPIの長さがでなければ
                y.rv = new( double[a.y.rl] );     // 同じ大きさの領域を確保
                if( y.rv == 0 ){                        // 領域確保失敗時
                        y.rl = 0;                                       // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.y.rl );
                        throw( theError );              // 上位のエラー処理へ
```

```
                }
                memcpy( y.rv, a.y.rv, a.y.rl*(sizeof(double)) );   // bit数をbyte数へ変換してコピー
        }
        else
                y.rv = 0;                                                   // 元のMPIの長さがならば
        y.rl = a.y.rl;                                              // 長さを同じに

        if( a.y.il ){                                               // 元のMPIの長さがでなければ
                y.iv = new( double[a.y.il] );     // 同じ大きさの領域を確保
                if( y.iv == 0 ){                            // 領域確保失敗時
                        y.il = 0;                                           // 無効
                        aError  theError( notEnoughMemory, C_CmplxMPI, sizeof(double) * a.y.il );
                        throw( theError );                  // 上位のエラー処理へ
                }
                memcpy( y.iv, a.y.iv, a.y.il*(sizeof(double)) );   // bit数をbyte数へ変換してコピー
        }
        else
                y.iv = 0;                                                   // 元のMPIの長さがならば
        y.il = a.y.il;                                              // 長さを同じに

}

EcPoint& EcPoint::operator=(                                // 代入
        const EcPoint& a )                                          // 代入する値
{
        x.rs = a.x.rs;
        x.is = a.x.is;
        x.is2 = a.x.is2;
        y.rs = a.y.rs;
        y.is = a.y.is;
        y.is2 = a.y.is2;

////////////////////////////////////////////

        if( x.rl && x.rv )                                          // すでに記憶領域があれば
                delete[] x.rv;                                      // 破棄する
        x.rv = new( double[a.x.rl] );
        if( x.rv == 0 ){                                    // メモリ割り当て失敗時
                x.rl = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.x.rl );
                throw( theError );
        }
        x.rl = a.x.rl;                                              // 桁数を同じに
        for( int i=0; i<x.rl; i++ )                 // 値のコピー
                x.rv[i] = a.x.rv[i];

        if( x.il && x.iv )                                          // すでに記憶領域があれば
                delete[] x.iv;                                      // 破棄する
        x.iv = new( double[a.x.il] );
        if( x.iv == 0 ){                                    // メモリ割り当て失敗時
                x.il = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.x.il );
```

```
                throw( theError );
        }
        x.il = a.x.il;                                          // 桁数を同じに
        for( int j=0; j<x.il; j++ )             // 値のコピー
                x.iv[j] = a.x.iv[j];

////////////////////////////////////////////////////////////

        if( y.rl && y.rv )                                      // すでに記憶領域があれば
                delete[] y.rv;                                  // 破棄する
        y.rv = new( double[a.y.rl] );
        if( y.rv == 0 ){                                // メモリ割り当て失敗時
                y.rl = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.y.rl );
                throw( theError );
        }
        y.rl = a.y.rl;                                          // 桁数を同じに
        for( int i=0; i<y.rl; i++ )             // 値のコピー
                y.rv[i] = a.y.rv[i];

        if( y.il && y.iv )                                      // すでに記憶領域があれば
                delete[] y.iv;                                  // 破棄する
        y.iv = new( double[a.y.il] );
        if( y.iv == 0 ){                                // メモリ割り当て失敗時
                y.il = 0;
                aError  theError( notEnoughMemory, 0_equal, sizeof(double) * a.y.il );
                throw( theError );
        }
        y.il = a.y.il;                                          // 桁数を同じに
        for( int j=0; j<y.il; j++ )             // 値のコピー
                y.iv[j] = a.y.iv[j];
////////////////////////////////////////////

        return *this;                                   // 自身を返して代入する


}


// 加算
extern CmplxMPI extc1, extp;                                                    // x の係数
EcPoint operator+(                                              // 和
        const EcPoint& a,                                       // 被加数
        const EcPoint& b )                                      // 加数
{
        CmplxMPI lam, tmp;
        EcPoint z;

        if(a.x > b.x){
                lam = (a.y-b.y)*uinv((a.x-b.x),extp);
        }
        if(a.x < b.x){
```

```
                lam = (b.y-a.y)*uinv((b.x-a.x),extp);
        }
        if((a.x == b.x) && (a.y!=0)){
                lam = (3*res(a.x*a.x, extp)+extc1)*uinv(2*a.y,extp);
        }
        z.x = res((lam*lam-a.x-b.x), extp);
        z.y = res((a.x-z.x)*lam-a.y, extp);

        return z;
}

EcPoint operator-(                                              // 和
        const EcPoint& a,                                       // 被加数
        const EcPoint& bb )                                     // 加数
{
        CmplxMPI lam, tmp;
        EcPoint z, b = bb;

        if(a.x > b.x){
                lam = (a.y+b.y)*uinv((a.x-b.x),extp);
        }
        if(a.x < b.x){
                lam = (-b.y-a.y)*uinv((b.x-a.x),extp);
        }
        if((a.x == b.x) && (a.y!=0)){
                lam = (3*res(a.x*a.x,extp)+extc1)*uinv(2*a.y,extp);
        }
        z.x = res((lam*lam-a.x-b.x), extp);
        z.y = res((a.x-z.x)*lam-a.y, extp);

        return z;
}


EcPoint operator*(                                              // 和
        const CmplxMPI& k,                                      // 被加数
        const EcPoint& a )                                      // 加数
{
        EcPoint tmp = a, tmp2=a;
        CmplxMPI i=1, j, kk=k;

        if(k<=1){ return a;}
START:
        if((kk.rs<0) || (kk<1)){ return tmp2-a;}
        if(kk==1){
//              tmp2 = tmp2+a;
                tmp2.x = res(tmp2.x, extp);
                tmp2.y = res(tmp2.y, extp);
                return tmp2;//-a;}
        }
B:
        j = i;
```

```
        i = 2*i;
        if(i <= kk) {
                tmp = tmp +tmp;
                tmp.x = res(tmp.x, extp);
                tmp.y = res(tmp.y, extp);
                goto B;
        }
        else{
                kk = kk - j;
                tmp2 = tmp2+tmp;
                i =1;
                tmp2.x = res(tmp2.x, extp);
                tmp2.y = res(tmp2.y, extp);
                goto START;
        }
}

EcPoint movetoEC (
  const CmplxMPI& ch,
  const CmplxMPI& p,
  const CmplxMPI& a,
  const CmplxMPI& b
                                        )
{
        int i,j;
        EcPoint PR;
        CmplxMPI Rx = ch, tmp;
        Rx <<= 8;
        for(i = 0; i<256; i++){
                Rx = Rx+i;
                tmp = res(( (a+res(Rx*Rx,p))*Rx + b), p);
                j = Jacobi(tmp,p);
                if(j == 1){break;}
        }
        PR.y = Sqrmp(tmp,p);
        PR.x=Rx;
        return PR;
}



/*
 * rsa.cpp
 * FFTRSA暗号関数
 *
 */
#include "stdafx.h"
#include <stdlib.h>

#include "cmplxmpi.h"
#include "ecc.h"
extern CmplxMPI extc1, extp;
```

```
// 暗号化
int FFTEccEncryption(                           // 0:正常　その他:エラー種別
        const int       keyLength,              // 鍵長

        const char* cp,
        const char* ca,
        const char* cb,
        const char* cQx,
        const char* cQy,
        const char* cSk,
        const char* cSx,
        const char* cSy,
        const int       mesLength,              // 平文長
        const char*     mes,                    // 平文
        int&            ciphLength,             // 暗号文長
        char*&          ciphMes )               // 暗号文
{
        if( keyLength < MINKEYLENGTH )    // 鍵長のチェック
                return KEYLENGTHERROR;

        char* pmes = (char*)mes;          // 平文の先頭
        int rest = mesLength;                   // 残り平文長

        int baseByte=25;// = keyLength / 8 + ((keyLength % 8)?1:0);         // 鍵バイト長
        int mesByte=22;// = baseByte - 1;         // 処理単位バイト数（暗号化ではbyte減らしておく）

        if(keyLength==192){                     // 192/8=24
                baseByte = 25;          mesByte = 22;  // 24より大きな数,    を加えても24 より小さな数
        }
        if(keyLength==244){  // 224/8 = 28
                baseByte = 29;          mesByte = 26;// 28より大きな数,    を加えても28 より小さな数
        }
        if(keyLength==256){// 256/8 =32
                baseByte = 33;          mesByte = 30;
        }
        if(keyLength == 384){// 384/8=48
                baseByte = 49;          mesByte = 46;
        }
        if(keyLength == 521){// 521/8=65.125
                baseByte = 67;          mesByte = 63;
        }

        ciphLength = 0;

        int safebyte = (mesByte > (DEF_COLS * COL_CHAR)) ?// 暗号文は平文より長くなるので多めにとる
                                        mesByte : DEF_COLS * COL_CHAR;
        int bufsize = ( (1+ mesLength/mesByte) + 1)*baseByte*2 + safebyte+300;      // 暗号文用領域サ
イズ
    ciphMes = (char*)new(char[bufsize]);  // 暗号文領域確保
        if( ciphMes == 0 )                              // エラーなら
```

```
		return NOTENOUGHMEMORY;
	for( int n=0; n<bufsize; n++ )	// 暗号文初期化
		ciphMes[n] = 0;
	char* cmes = ciphMes;			// 結果格納場所

	extp = cp;
	extc1 = ca;
	CmplxMPI k = cSk;
	CmplxMPI p=cp,a=ca,b=cb;
	EcPoint Pc;

	EcPoint Pb(cQx,cQy);
	EcPoint kG(cSx,cSy);
	EcPoint kPb = k*Pb;

	kG.x.getText_r( cmes );		// 暗号文の格納
	cmes += 150;//baseByte;		// 出力ポインタ更新
	ciphLength += 150;//baseByte;	// 暗号文長の更新
	kG.y.getText_r( cmes );		// 暗号文の格納
	cmes += 150;//baseByte;		// 出力ポインタ更新
	ciphLength += 150;//baseByte;	// 暗号文長の更新

	try{
		while( rest > 0 ){			// 平文が残っている間
			char *block = new( char[baseByte] );	// 平文ブロック領域確保
			if( block == 0 )		// 失敗時
				return NOTENOUGHMEMORY;

			for( int i=0; i<mesByte; i++ )	// 平文からブロック分コピー
				block[i] = (char)(*pmes++ & 0x00ff);
			block[mesByte] = 0;			// 最高位(baseByte目)をに

			CmplxMPI lm( (u_short*)block, (mesByte/2)+((mesByte&1)?1:0) );	// 平文ブロックのCmplxMPI化
			rest -= mesByte;		// 残り平文長更新

			EcPoint Pm = movetoEC (   lm, p,   a,      b );
			Pc = Pm+kPb;

			Pc.x.copyTo_r( cmes );		// 暗号文の格納
			cmes += baseByte;			// 出力ポインタ更新
			ciphLength += baseByte;	// 暗号文長の更新

			Pc.y.copyTo_r( cmes );		// 暗号文の格納
			cmes += baseByte;			// 出力ポインタ更新
			ciphLength += baseByte;	// 暗号文長の更新
		}
		return  NOERROR;			// 正常終了

	}
	catch( aError theErr ){			// throw()発生時
		switch( theErr.c ){
```

```
                        case notEnoughMemory:     // メモリー不足
                                return NOTENOUGHMEMORY;
                        case accessError:                 // アクセスエラー
                                return ACCESSERROR;
                        case divideByZero:                // 0による除算
                        case minusValue:          // 結果が負
                                return MATHERROR;
                        case notXdigit:                   // 16進数以外の文字を指定
                        default:                          // その他のエラー
                                return OTHERERROR;
                }
        }
}

// 復号化
int FFTEccDecryption(
        const int       keyLength,                      // 鍵長
        const char* cp,
        const char* ca,
        const char* cSk,
        const int       ciphLength,                     // 暗号文長
        const char*     ciphMes,                // 暗号文
        int&            mesLength,                      // 平文長
        char*&          mes     )                               // 平文
{
        if( keyLength < MINKEYLENGTH )    // 鍵長のチェック
                return KEYLENGTHERROR;

        char* pmes = (char*)ciphMes;      // 暗号文の先頭
        int rest = ciphLength;                  // 残り暗号文長


        int baseByte=25;// = keyLength / 8 + ((keyLength % 8)?1:0);        // 鍵バイト長
        int mesByte=22;// = baseByte - 1;          // 処理単位バイト数（暗号化ではbyte減らしておく）

        if(keyLength==192){                     // 192/8=24
                baseByte = 25;          mesByte = 22;  // 24より大きな数,    を加えても24 より小さな数
        }
        if(keyLength==244){  // 224/8 = 28
                baseByte = 29;          mesByte = 26;// 28より大きな数,    を加えても28 より小さな数
        }
        if(keyLength==256){// 256/8 =32
                baseByte = 33;          mesByte = 30;
        }
        if(keyLength == 384){// 384/8=48
                baseByte = 49;          mesByte = 46;
        }
        if(keyLength == 521){// 521/8=65.125
                baseByte = 67;          mesByte = 63;
        }

        mesLength = 0;                                  // 平文長初期化
```

```
        int safebyte = (baseByte > DEF_COLS * COL_CHAR) ?          // 平文は暗号文より短いが作業上はみ出す可能性がある
                                                baseByte : DEF_COLS * COL_CHAR;
        int bufsize = ciphLength/2 /* + ( ciphLength / baseByte )*/ + safebyte;     // 平文用領域サイズ

    mes = (char*)new(char[bufsize]);       // 平文用領域
        if( mes == 0 )
                return NOTENOUGHMEMORY;
        for( int n=0; n<bufsize; n++ )     // 平文初期化
                mes[n] = 0;
        char* mesp = mes;
        extp = cp;
        extc1 = ca;
        CmplxMPI nn(cSk);
        EcPoint nGk, P, Pm;

        char* block = new( char[150]);//baseByte+2] );     // 暗号文領域確保
        if( block == 0 )          // 失敗時
                return NOTENOUGHMEMORY;
        int i;
        for( i=0; i<150/*baseByte*/; i++ )         // 暗号文から処理分コピー
                block[i] = (char)(*pmes++ & 0x00ff);
        if( i & 1 )                                // 奇数バイトの処理
                block[i] = 0;

        CmplxMPI cm((char*)block);//, baseByte/2+((baseByte&1)?1:0) );    // 暗号文ブロックのCmplxMPI化
        rest -= 150;//baseByte;           // 残り暗号文長更新
        nGk.x = cm;

                        for( i=0; i<150/*baseByte*/; i++ )         // 暗号文から処理分コピー
                                block[i] = (char)(*pmes++ & 0x00ff);
                        if( i & 1 )                                // 奇数バイトの処理
                                block[i] = 0;

        CmplxMPI cm2((char*)block);//, baseByte/2+((baseByte&1)?1:0) );    // 暗号文ブロックのCmplxMPI化
        rest -= 150;//baseByte;           // 残り暗号文長更新
        nGk.y = cm2;

        nGk = nn*nGk;


        try{

                while( rest > 0 ){                         // 暗号文が残っている間
                        char* block = new( char[baseByte+2] );     // 暗号文領域確保
                        if( block == 0 )          // 失敗時
                                return NOTENOUGHMEMORY;

                        int i;
                        for( i=0; i<baseByte; i++ )        // 暗号文から処理分コピー
```

```
                        block[i] = (char)(*pmes++ & 0x00ff);
                if( i & 1 )                             // 奇数バイトの処理
                        block[i] = 0;

                CmplxMPI cm((u_short*)block, baseByte/2+((baseByte&1)?1:0) );    // 暗号文ブロックのCmplxMPI化
                rest -= baseByte;               // 残り暗号文長更新
                P.x = cm;

                for( i=0; i<baseByte; i++ )     // 暗号文から処理分コピー
                        block[i] = (char)(*pmes++ & 0x00ff);
                if( i & 1 )                             // 奇数バイトの処理
                        block[i] = 0;

                CmplxMPI cm2((u_short*)block, baseByte/2+((baseByte&1)?1:0) );    // 暗号文ブロックのCmplxMPI化
                rest -= baseByte;               // 残り暗号文長更新
                P.y = cm2;

                Pm = P-nGk;

                CmplxMPI lm = Pm.x;     // 復号化
                lm >>= 8;

                lm.copyTo_r( mesp );            // 平文の格納

                mesp += mesByte;//baseByte-1;           // 出力ポインタ更新
                mesLength += mesByte;//baseByte -1;     // 平文長の更新
        }
        *mesp = '¥0';                           // 最後のブロックにはゴミがついている

    }
    catch( aError theErr ){                 // throw()発生時
        switch( theErr.c ){
                case notEnoughMemory:   // メモリー不足
                        return NOTENOUGHMEMORY;
                case accessError:               // アクセスエラー
                        return ACCESSERROR;
                case divideByZero:              // 0による除算
                case minusValue:        // 結果が負
                        return MATHERROR;
                case notXdigit:                 // 16進数以外の文字を指定
                default:                        // その他のエラー
                        return OTHERERROR;
        }
    }
    return  NOERROR;                        // 正常終了
}

// 素数を作る(2)
CmplxMPI makePrimeNumber2(                      // 素数
        const int k )                           // ビット数
```

```
{
        CmplxMPI p;                                                     // 素数
        do{
                p = randomBit( k );                     // 元となる乱数
                if( isEven( p ) ){                      // 奇数を使う
                        p--;
                }
                while( ! millerrabin( p, 1 ) ){         // 確率的素数判定
                        p +=2;
                }
        }while( p.getBitLength() != k );  // ビット長を確認
        p.torealpart();
        return (p);
}


// 素数を作る
CmplxMPI makePrimeNumber(                       // 素数
                const int k )                           // ビット数
{
        CmplxMPI p;                                                     // 素数
        do{
                p = randomBit( k );                     // 元となる乱数
                if( isEven( p ) )                       // 奇数を使う
                        p--;
                while( ! rabin( (p) ) )         // 確率的素数判定
                        p +=2;
        }while( p.getBitLength() != k );  // ビット長を確認
        p.torealpart();
        return (p);
}

// 小さい素数の生成（r=2at+1形式）
CmplxMPI makeShortPrimeNumber(                  // 素数
                const CmplxMPI& t,                                      // 元になる素数t
                const int k )                           // ビット数
{
        int     k2 = k / 5;                                             // ビット数の/10（Kは最終的にほしいビット数の半分）
        CmplxMPI p;                                                     // 素数

        int cnt=100;                                            // カウンタ
        do{
                if( !(--cnt) ){
                        cnt = 100;
                        randInitial();                  // 乱数を初期化してやり直す
                }

                CmplxMPI a = randomBit( k2 );   // aを用意する
                a.torealpart();
                while( ! fermat((a), t ) )      // 確定的素数判定
                        a++;                                            // 素数になるまで
```

```
                p = ((a << 1) * t) + 1;           // 2at+1
        }while( p.getBitLength() != k );  // ビット数確認
        p.torealpart();
        return  (p);
}


// 素数の生成(p = 1+2(bs-R)r形式)
CmplxMPI makeSafetyPrimeNumber(                 // 素数
                const CmplxMPI& r,                              // r
                const CmplxMPI& s,                              // s
                const int k )                           // ビット数
{
        CmplxMPI R = uinv( r, s );                      // S1) Rはrの逆数 (法s)
        CmplxMPI p;                                                     // 素数

        do{
                CmplxMPI b = randomBit( k / 10 - 1 );
                b.torealpart();
                p =     ((b * s - R) << 1) * r + 1;             // S2') p=1+2(bs-R)r
                p.torealpart();

                int n = p.getBitLength();
                while( n != k ){                // ビット長がkになるまで
                        b <<= k - n;                    // bの長さを調整
                        p =     (((pFFT(b , s) - R) << 1) * r) + 1;             // pを再計算
                        n = p.getBitLength();
                }                                                               // ここでpの長さはkビット
になっている
        }while( ! fermat( p, r ) );             // 素数判定
        p.torealpart();
        return  p;
}


/*
 * RSA用素数生成
 *      kビットの素数を生成する
 */

CmplxMPI makeRsaSafetyPrimeNumber(              // 素数
                const int k )                           // ビット数
{
        // 素数t(0.4kbit)を生成
        CmplxMPI t = makePrimeNumber( int( k * 0.4 ) );
        // r=2at+1の形の素数r(0.5kbit)を生成
        CmplxMPI r = makeShortPrimeNumber( t, k>>1 );
        // 素数s(0.4kbit)を生成
        CmplxMPI s = makePrimeNumber( int( k * 0.4 ) );
        // 整数Rをinv(r,s)として、p=1+2(bs-R)rの形の素数pを生成
        CmplxMPI p = makeSafetyPrimeNumber( r, s, k );
        p.torealpart();
```

```
        return  p;
}


// 素数の生成(Gordon's Strong prime)
CmplxMPI makeGordonPrimeNumber(                 // 素数
                const CmplxMPI& s,                              // s
                const CmplxMPI& t,                              // t
                const int k )                                   // ビット数
{
        CmplxMPI p, p0,j,r;                                     // 素数

//      do{
                CmplxMPI ip = randomBit( k / 10 - 1 );
                ip.torealpart();
                while(!rabin((2*ip*t + 1))){
                        ip  = ip + 1;
                }
                r = ip;
                p0 = 2*expFFT(s, r-2, r)*s - 1;
                j = 1;//t;
                p = p0 + 2*j*r*s;
//              int n = p.getBitLength();

                while(/* (n<=k) && */(!rabin(p)) ){
//                      ip = j*r*s;
//                      int m = ip.getBitLength();
//                      j <<= k-m;
                        j += 1;
                        p = p0 + 2*j*r*s;
//                      n = p.getBitLength();
                }
//      }while( p.getBitLength() < k);//! fermat( p, s ) );               // 素数判定
        p.torealpart();
        return  p;
}


/*
 * RSA用素数生成
 *      kビットの素数を生成する
 */

CmplxMPI makeRsaGordonPrimeNumber(              // 素数
                const int k )                           // ビット数
{
        // 素数t(0.4kbit)を生成
        CmplxMPI t = makePrimeNumber2( int( k * 0.4 ) );
        // r=2at+1の形の素数r(0.5kbit)を生成
        CmplxMPI r = makeShortPrimeNumber( t, k>>1 );
        // 素数s(0.4kbit)を生成
        CmplxMPI s = makePrimeNumber2( int( k * 0.4 ) );
```

```
        // 整数Rをinv(r,s)として、p=1+2(bs-R)rの形の素数pを生成
        CmplxMPI p = makeGordonPrimeNumber( r, s, k );
        p.torealpart();
        return  p;
}



// ECC公開鍵暗号用の鍵の組を作る (e,d,nを得る)
int     makeEccKeys(
                        const int k,                    // 鍵長k
                        char*& cp,                              // 公開鍵e
                        char*& ca,                              // 秘密鍵d
                        char*& cb,                              // 共通法鍵n
                        char*& cGx,                             // 公開鍵e
                        char*& cGy,                             // 秘密鍵d
                        char*& cn,                              // 共通法鍵n
                        char*& ch,                              // 公開鍵e
                        char*& cQx,                             // 秘密鍵d
                        char*& cQy,                             // 共通法鍵n
                        char*& cSk,                             // 共通法鍵n
                        char*& cSx,                             // 秘密鍵d
                        char*& cSy)                             // 共通法鍵n
{
        if( k < MINKEYLENGTH )                  // 鍵長のチェック
                return KEYLENGTHERROR;

        try{
                randInitial();
                CmplxMPI p,a,b,Gx,Gy,n,h;                                       // 素数
                CmplxMPI Qx,Qy,Sk,Sx,Sy; // 鍵の素Lを求める

                CmplxMPI d,tmp;                                                 // 鍵d

                if(k == 192){
                        p = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFEE37";
                        a = "0";
                        b = "3";
                        Gx = "DB4FF10EC057E9AE26B07D0280B7F4341DA5D1B1EAE06C7D";
                        Gy = "9B2F2F6D9C5628A7844163D015BE86344082AA88D95E2F9D";
                        n = "FFFFFFFFFFFFFFFFFFFFFFFE26F2FC170F69466A74DEFD8D";
                        h = "1";
                }
                if(k == 244){
                        p = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000000000000000000001";
                        a = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFE";
                        b = "B4050A850C04B3ABF54132565044B0B7D7BFD8BA270B39432355FFB4";
                        Gx = "B70E0CBD6BB4BF7F321390B94A03C1D356C21122343280D6";
                        Gy = "115C1D21BD376388B5F723FB4C22DFE6CD4375A05A07476444D5819985007E34";
                        n = "FFFFFFFFFFFFFFFFFFFFFFFFFFFF16A2E0B8F03E13DD29455C5C2A3D";
                        h = "1";
                }
```

```
                if(k == 256){
                        p = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F";
                        a = "0";
                        b = "7";
                        Gx = "79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D9";
                        Gy =
"59F2815B16F81798483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8";
                        n = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141";
                        h = "1";
                }
                if(k == 384){
                        p =
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFF";
                        a =
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFF0000000000000000FFFFFFFC";
                        b =
"B3312FA7E23EE7E4988E056BE3F82D19181D9C6EFE8141120314088F5013875AC656398D8A2ED19D2A85C8EDD3EC2AEF";
                        Gx = "AA87CA22BE8B05378EB1C71EF320AD746E1D3B628BA79B98";
                        Gy =
"59F741E082542A385502F25DBF55296C3A545E3872760AB73617DE4A96262C6F5D9E98BF9292DC29F8F41DBD289A147CE9DA3
113B5F0B8C00A60B1CE1D7E819D7A431D7C90EA0E5F";
                        n =
"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC7634D81F4372DDF581A0DB248B0A77AECEC196ACCC52973";
                        h = "1";
                }
                if(k == 521){
                        p =
"01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
                        a =
"01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFC";
                        b =
"0051953EB9618E1C9A1F929A21A0B68540EEA2DA725B99B315F3B8B489918EF109E156193951EC7E937B1652C0BD3BB1BF073
573DF883D2C34F1EF451FD46B503F00";
                        Gx = "00C6858E06B70404E9CD9E3ECB662395B4429C648139053F";
                        Gy =
"B521F828AF606B4D3DBAA14B5E77EFE75928FE1DC127A2FFA8DE3348B3C1856A429BF97E7E31C2E5BD66011839296A789A3BC
0045C8A5FB42C7D1BD998F54449579B446817AFBD17273E662C97EE72995EF42640C550B9013FAD0761353C7086A272C24088B
E94769FD16650";
                        n =
"01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFA51868783BF2F966B7FCC0148F709A5D03
BB5C9B8899C47AEBB6FB71E91386409";
                        h = "1";
                }

                EcPoint G(Gx,Gy);
                extp = p;
                extc1 = a;
                Sk = randomBit(k/2+32);
                EcPoint Q =  Sk*G;
                Qx = res((Q.x), extp);
```

```
		Qy = res((Q.y), extp);
		Sx = res((Q.x), extp);
		Sy = res((Q.y), extp);

		//出力
		cp = (char*)new( char[(p.getMaxColumn_r()+1) *4+1] );
		ca = (char*)new( char[(a.getMaxColumn_r()+1) *4+1] );
		cb = (char*)new( char[(b.getMaxColumn_r()+1) *4+1] );
		cGx = (char*)new( char[(Gx.getMaxColumn_r()+1) *4+1] );
		cGy = (char*)new( char[(Gy.getMaxColumn_r()+1) *4+1] );
		cn = (char*)new( char[(n.getMaxColumn_r()+1) *4+1] );
		ch = (char*)new( char[(h.getMaxColumn_r()+1) *4+1] );
		cQx = (char*)new( char[(Qx.getMaxColumn_r()+1) *4+1] );
		cQy = (char*)new( char[(Qy.getMaxColumn_r()+1) *4+1] );

		cSk = (char*)new( char[(Sk.getMaxColumn_r()+1) *4+1] );
		cSx = (char*)new( char[(Qx.getMaxColumn_r()+1) *4+1] );
		cSy = (char*)new( char[(Qy.getMaxColumn_r()+1) *4+1] );

		if( cp==0 || ca==0 || cb==0 || cGx==0 || cGy==0 || cn==0 || ch==0 || cQx==0 || cQy==0 ||
cSk==0)
			return NOTENOUGHMEMORY;

		p.getText_r( cp );	a.getText_r( ca );	b.getText_r( cb );
		Gx.getText_r( cGx );	Gy.getText_r( cGy ); n.getText_r( cn ); h.getText_r( ch );
		Qx.getText_r( cQx );	Qy.getText_r( cQy );	Sk.getText_r( cSk );
		Sx.getText_r( cSx ); Sy.getText_r( cSy );



	}
	catch( aError theErr ){				// throw()発生時
		switch( theErr.c ){
			case notEnoughMemory:		// メモリー不足
				return NOTENOUGHMEMORY;
			case accessError:				// アクセスエラー
				return ACCESSERROR;
			case divideByZero:				// 0による除算
			case minusValue:			// 結果が負
				return MATHERROR;
			case notXdigit:				// 16進数以外の文字を指定
			default:					// その他のエラー
				return OTHERERROR;
		}
	}
	return NOERROR;
}


// ECCrypt.cpp : アプリケーションのクラス動作を定義します。
//
```

```
#include "stdafx.h"
#include "ECCrypt.h"
#include "ECCryptDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CECCryptApp

BEGIN_MESSAGE_MAP(CECCryptApp, CWinApp)
	ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()


// CECCryptApp コンストラクション

CECCryptApp::CECCryptApp()
{
	// TODO: この位置に構築用コードを追加してください。
	// ここにInitInstance 中の重要な初期化処理をすべて記述してください。
}


// 唯一のCECCryptApp オブジェクトです。

CECCryptApp theApp;


// CECCryptApp 初期化

BOOL CECCryptApp::InitInstance()
{
	// アプリケーションマニフェストがvisual スタイルを有効にするために、
	// ComCtl32.dll Version 6 以降の使用を指定する場合は、
	// Windows XP にInitCommonControlsEx() が必要です。さもなければ、ウィンドウ作成はすべて失敗します。
	INITCOMMONCONTROLSEX InitCtrls;
	InitCtrls.dwSize = sizeof(InitCtrls);
	// アプリケーションで使用するすべてのコモンコントロールクラスを含めるには、
	// これを設定します。
	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);

	CWinApp::InitInstance();

	AfxEnableControlContainer();

	// 標準初期化
	// これらの機能を使わずに最終的な実行可能ファイルの
	// サイズを縮小したい場合は、以下から不要な初期化
```

```
	// ルーチンを削除してください。
	// 設定が格納されているレジストリキーを変更します。
	// TODO: 会社名または組織名などの適切な文字列に
	// この文字列を変更してください。
	SetRegistryKey(_T("アプリケーションウィザードで生成されたローカルアプリケーション"));

	CECCryptDlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: ダイアログが<OK> で消された時のコードを
		//  記述してください。
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: ダイアログが<キャンセル> で消された時のコードを
		//  記述してください。
	}

	// ダイアログは閉じられました。アプリケーションのメッセージポンプを開始しないで
	//  アプリケーションを終了するためにFALSE を返してください。
	return FALSE;
}


// ECCryptDlg.cpp : 実装ファイル
//

#include "stdafx.h"
#include "ECCrypt.h"
#include "ECCryptDlg.h"
#include "fstream"
#include <iostream>
#include "cmplxmpi.h"
#include "ecc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

using namespace std;

#define file_len(x) (unsigned long)x

FILE *stream;
FILE *stream1;
FILE *stream2;
FILE *stream3;
FILE *stream4;
```

```
CmplxMPI extc1, extp;

// アプリケーションのバージョン情報に使われるCAboutDlg ダイアログ

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// ダイアログデータ
	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV サポート

// 実装
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()



// 暗号文のHEX表示用
char toChar( int c )
{
	if( c >= 0 && c <= 9 )                    // 0～ならば
		return char( c + 0x30 ); // ASCIIに変換して返す
	else if( c >= 10 && c <= 15 )     // 10～ならば
		return char( c + 0x37 ); // A～FのASCIIを返す
	else
		return ' ';
}


//////////////////////////////////////////////////////////////////////////
// CRSAcryptDlg ダイアログ

CECCryptDlg::CECCryptDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CECCryptDlg::IDD, pParent)
```

```
{
	//{{AFX_DATA_INIT(CRSAcryptDlg)
		// メモ: この位置にClassWizard によってメンバの初期化が追加されます。
	blen=MAX_PATH+1;
	//char bufpath[MAX_PATH+1];
	GetCurrentDirectory(blen, bufpath);

	madekey = 0;
	readeckey=0;
	readdckey=0;
	i_KeyLen = 0;

//	commonkey1 = "";

	opensp = "";
	opensa = "";
	opensb = "";
	opensgx = "";
	opensgy = "";
	opensn = "";
	opensh = "";
	opensqx = "";
	opensqy= "";

	secretsk = "";
	secretsx = "";
	secretsy = "";

	planedata_file = "plane.txt";
	encryptdata_file = ".¥¥EC¥¥encrypt.enc";
	decryptdata_file = ".¥¥DC¥¥decrypt.txt";
	encryptkey_file = "EcOpnEC1.bin";
	decryptkey_file = "EcSecDC1.bin";

	//}}AFX_DATA_INIT
	// メモ: LoadIcon はWin32 のDestroyIcon のサブシーケンスを要求しません。
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CECCryptDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CRSAcryptDlg)

    DDX_Text(pDX, IDC_ADDBIN2, s_addbin);
    DDV_MaxChars(pDX, s_addbin, 100);
    DDX_Control(pDX, IDC_LISTBIN2, l_bin);

	DDX_Radio(pDX, IDC_RADIO1, i_KeyLen);


//	DDX_Text(pDX, IDC_COMMONKEY1, commonkey1);
```

```
//    DDV_MaxChars(pDX, commonkey1, 650);  // 2560/4 + おまけ= 640+10 = 650

    DDX_Text(pDX, IDC_OPENCP, opensp);
    DDV_MaxChars(pDX, opensp, 150);                         // 2560/4 + おまけ= 640+10 = 150
    DDX_Text(pDX, IDC_OPENCA, opensa);
    DDV_MaxChars(pDX, opensa, 150);
    DDX_Text(pDX, IDC_OPENCB, opensb);
    DDV_MaxChars(pDX, opensb, 150);
    DDX_Text(pDX, IDC_OPENCGX, opensgx);
    DDV_MaxChars(pDX, opensgx, 150);
    DDX_Text(pDX, IDC_OPENCGY, opensgy);
    DDV_MaxChars(pDX, opensgy, 150);
    DDX_Text(pDX, IDC_OPENCN, opensn);
    DDV_MaxChars(pDX, opensn, 150);
    DDX_Text(pDX, IDC_OPENCH, opensh);
    DDV_MaxChars(pDX, opensh, 150);
    DDX_Text(pDX, IDC_OPENCQX, opensqx);
    DDV_MaxChars(pDX, opensqx, 150);
    DDX_Text(pDX, IDC_OPENCQY, opensqy);
    DDV_MaxChars(pDX, opensqy, 150);

    DDX_Text(pDX, IDC_SECRETSK, secretsk);
    DDV_MaxChars(pDX, secretsk, 150);
    DDX_Text(pDX, IDC_SECRETSX, secretsx);
    DDV_MaxChars(pDX, secretsx, 150);
    DDX_Text(pDX, IDC_SECRETSY, secretsy);
    DDV_MaxChars(pDX, secretsy, 150);

    DDX_Text(pDX, IDC_PLANEDATA_FILE, planedata_file);
    DDV_MaxChars(pDX, planedata_file, 128);
    DDX_Text(pDX, IDC_ENCRYPTDATA_FILE, encryptdata_file);
    DDV_MaxChars(pDX, encryptdata_file, 128);
    DDX_Text(pDX, IDC_DECRYPTDATA_FILE, decryptdata_file);
    DDV_MaxChars(pDX, decryptdata_file, 128);
        DDX_Text(pDX, IDC_ENCRYPTKEY_FILE, encryptkey_file);
    DDV_MaxChars(pDX, encryptkey_file, 128);
    DDX_Text(pDX, IDC_DECRYPTKEY_FILE, decryptkey_file);
    DDV_MaxChars(pDX, decryptkey_file, 128);
        DDX_Control(pDX, IDC_DISPLAY, datadisp);
                // メモ: この場所にはClassWizard によってDDX とDDV の呼び出しが追加されます。
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CECCryptDlg, CDialog)
        //{{AFX_MSG_MAP(CRSAcryptDlg)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_COMMAND(ID_TESTKEY, TestKey)
        ON_COMMAND(ID_MAKEKEY, MakeKey)
        ON_COMMAND(IDC_ECKREAD, ECKRead)
        ON_COMMAND(IDC_DCKREAD, DCKRead)
```

```
        ON_COMMAND(IDC_BSEARCH2, Search2)
        ON_COMMAND(IDC_BADDBIN2, AddList2)
        ON_COMMAND(IDC_BDELBIN2, DelList2)


        //}}AFX_MSG_MAP
//      ON_EN_CHANGE(IDC_KEYLENGTH, &CECCryptDlg::OnEnChangeKeylength)
        ON_EN_CHANGE(IDC_ENCRYPTKEY_FILE, &CECCryptDlg::OnEnChangeEncryptkeyFile)
        ON_BN_CLICKED(IDCANCEL, &CECCryptDlg::OnBnClickedCancel)
        ON_BN_CLICKED(IDC_ECKREAD, &CECCryptDlg::OnBnClickedEckread)
        ON_BN_CLICKED(ID_MAKEKEY, &CECCryptDlg::OnBnClickedMakekey)
        ON_BN_CLICKED(IDOK, &CECCryptDlg::OnBnClickedOk)
        ON_EN_CHANGE(IDC_COMMONKEY2, &CECCryptDlg::OnEnChangeCommonkey2)
        ON_EN_CHANGE(IDC_SECRETKEY, &CECCryptDlg::OnEnChangeSecretkey)
        ON_EN_CHANGE(IDC_DISPLAY, &CECCryptDlg::OnEnChangeDisplay)
        ON_EN_CHANGE(IDC_OPENKEY, &CECCryptDlg::OnEnChangeOpenkey)
        ON_EN_CHANGE(IDC_COMMONKEY1, &CECCryptDlg::OnEnChangeCommonkey1)

        ON_BN_CLICKED(IDC_RADIO1, &CECCryptDlg::OnBnClickedRadio1)
        ON_BN_CLICKED(IDC_RADIO2, &CECCryptDlg::OnBnClickedRadio2)
        ON_BN_CLICKED(IDC_RADIO3, &CECCryptDlg::OnBnClickedRadio3)
        ON_BN_CLICKED(IDC_RADIO4, &CECCryptDlg::OnBnClickedRadio4)
        ON_BN_CLICKED(IDC_RADIO5, &CECCryptDlg::OnBnClickedRadio5)


        ON_BN_CLICKED(IDC_DCKREAD, &CECCryptDlg::OnBnClickedDckread)
        ON_BN_CLICKED(IDC_BUTTON1, &CECCryptDlg::OnBnClickedButton1)
        ON_BN_CLICKED(IDC_BUTTON2, &CECCryptDlg::OnBnClickedButton2)
        ON_EN_CHANGE(IDC_DECRYPTKEY_FILE, &CECCryptDlg::OnEnChangeDecryptkeyFile)
        ON_BN_CLICKED(IDC_BUTTON3, &CECCryptDlg::OnBnClickedButton3)
        ON_BN_CLICKED(IDC_BUTTON4, &CECCryptDlg::OnBnClickedButton4)

        ON_BN_CLICKED(IDC_BUTTON5, &CECCryptDlg::OnBnClickedButton5)
        ON_BN_CLICKED(IDC_BUTTON6, &CECCryptDlg::OnBnClickedButton6)


END_MESSAGE_MAP()

////////////////////////////////////////////
void CECCryptDlg::yDisplay(const char *cp)
{
  int nEditLength = datadisp.GetWindowTextLength();
  datadisp.SetSel(nEditLength, nEditLength); //テキストの終端にカーソルを移動する
  datadisp.ReplaceSel(cp); //新しいテキストを追加する
}

void CECCryptDlg::ECKRead()
{
        CString strNewFileName;
        CString sFName;
        CWnd* pwnd;
        int i, sl;
```

```
	strNewFileName = "*.*";
	CFileDialog fileDlg(TRUE, NULL,"*.bin", OFN_HIDEREADONLY, strNewFileName);
	if(fileDlg.DoModal() == IDOK){
		strNewFileName = fileDlg.GetPathName();

		// If file doesn't already exist, then create it.
		CFile file;
		CFileStatus status;
		if (!file.GetStatus(strNewFileName, status)){
			CString strMessage;
//			AfxFormatString1(strMessage, IDS_MAKENEWFILE,
//				strNewFileName);
			if(AfxMessageBox(strMessage, MB_YESNO) == IDNO)	{
				return;
			}
			if (!file.Open(strNewFileName, CFile::modeCreate)){
				CString strMessage;
//				AfxFormatString1(strMessage, IDS_NOADBOOKMAILBOXTEMPLATE,
//					strNewFileName);
				AfxMessageBox(strMessage);
				return;
			}
			file.Close();
		}
	}

	FILE *fkey = 0;
	char c_klen[8], c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150], c_ch[150];
	char c_cQx[150], c_cQy[150];

	if((fkey = fopen(strNewFileName, "rt")) == NULL){
		MessageBox( "Can not find key file for encryption. ¥n");
		return ;
	}

	fgets( c_klen, 8, fkey );

	fgets( c_cp, 150, fkey );
	fgets( c_ca, 150, fkey );
	fgets( c_cb, 150, fkey );

	fgets( c_cGx, 150, fkey );
	fgets( c_cGy, 150, fkey );
	fgets( c_cn, 150, fkey );
	fgets( c_ch, 150, fkey );

	fgets( c_cQx, 150, fkey );
	fgets( c_cQy, 150, fkey );
	if(fkey){ fclose(fkey);  }

	sl = strlen(c_cp);
```

```
	c_cp[sl-1] = NULL;
	sl = strlen(c_ca);
	c_ca[sl-1] = NULL;
	sl = strlen(c_cb);
	c_cb[sl-1] = NULL;

	sl = strlen(c_cGx);
	c_cGx[sl-1] = NULL;
	sl = strlen(c_cGy);
	c_cGy[sl-1] = NULL;
	sl = strlen(c_cn);
	c_cn[sl-1] = NULL;
	sl = strlen(c_ch);
	c_ch[sl-1] = NULL;

	sl = strlen(c_cQx);
	c_cQx[sl-1] = NULL;
	sl = strlen(c_cQy);
	c_cQy[sl-1] = NULL;


	i = atoi(c_klen);
	if(i == 192 ){
		k = 192; keylength = "192";
		CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO1);
	}else{
		if(i==244){
			k = 244; keylength = "244";
			CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO2);
		}else{
			if(i==256){
				k = 256; keylength = "256";
				CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO3);
			}else{
				if(i==384){
					k = 384; keylength = "384";
					CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO4);
				}else{
					if(i==521){
						k = 521; keylength = "521";
						CheckRadioButton(IDC_RADIO1, IDC_RADIO5,
IDC_RADIO5);
					}else{
						MessageBox( "鍵の長さが不正です。¥n");
						return ;
					}
				}
			}
		}
	}
```

```
        sFName = strNewFileName;
        int j = sFName.ReverseFind( '¥¥' );
        sFName.Delete(0,j+1);
        pwnd = GetDlgItem(IDC_ENCRYPTKEY_FILE);
        pwnd->SetWindowText(sFName);

        pwnd = GetDlgItem(IDC_OPENCP);    pwnd->SetWindowText(c_cp);
        pwnd = GetDlgItem(IDC_OPENCA);    pwnd->SetWindowText(c_ca);
        pwnd = GetDlgItem(IDC_OPENCB);    pwnd->SetWindowText(c_cb);

        pwnd = GetDlgItem(IDC_OPENCGX);   pwnd->SetWindowText(c_cGx);
        pwnd = GetDlgItem(IDC_OPENCGY);   pwnd->SetWindowText(c_cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->SetWindowText(c_cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->SetWindowText(c_ch);

        pwnd = GetDlgItem(IDC_OPENCQX);   pwnd->SetWindowText(c_cQx);
        pwnd = GetDlgItem(IDC_OPENCQY);   pwnd->SetWindowText(c_cQy);


        readeckey = 1;

        return ;

        // Open the file now that it has been created.
//      strcpy(tmppath, strNewFileName);
//      OpenDocumentFile(strNewFileName);

}

void CECCryptDlg::DCKRead()
{
        CString strNewFileName;
        CString sFName;
        CWnd* pwnd;
        int i, sl;

        strNewFileName = "*.*";
        CFileDialog fileDlg(TRUE, NULL,"*.bin", OFN_HIDEREADONLY, strNewFileName);
        if(fileDlg.DoModal() == IDOK){
                strNewFileName = fileDlg.GetPathName();

                // If file doesn't already exist, then create it.
                CFile file;
                CFileStatus status;
                if (!file.GetStatus(strNewFileName, status)){
                        CString strMessage;
//                      AfxFormatString1(strMessage, IDS_MAKENEWFILE,
//                              strNewFileName);
                        if(AfxMessageBox(strMessage, MB_YESNO) == IDNO)    {
                                return;
                        }
                        if (!file.Open(strNewFileName, CFile::modeCreate)){
```

```
                                CString strMessage;
//                              AfxFormatString1(strMessage, IDS_NOADBOOKMAILBOXTEMPLATE,
//                                      strNewFileName);
                                AfxMessageBox(strMessage);
                                return;
                        }
                        file.Close();
                }
        }

        FILE *fkey = 0;
        char c_klen[8],c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150],c_Sk[150], c_Sx[150], c_Sy[150];  // 2560/4 + おまけ= 640+10 = 650

        if((fkey = fopen(strNewFileName, "rt")) == NULL){
                MessageBox( "Can not find key file for encryption. ¥n");
                return ;
        }

        fgets( c_klen, 8, fkey );
        fgets( c_cp, 150, fkey );
        fgets( c_ca, 150, fkey );
        fgets( c_cb, 150, fkey );

        fgets( c_cGx, 150, fkey );
        fgets( c_cGy, 150, fkey );
        fgets( c_cn, 150, fkey );
        fgets( c_ch, 150, fkey );

        fgets( c_Sk, 150, fkey );
        fgets( c_Sx, 150, fkey );
        fgets( c_Sy, 150, fkey );

        if(fkey){ fclose(fkey);  }

        sl = strlen(c_cp);
        c_cp[sl-1] = NULL;
        sl = strlen(c_ca);
        c_ca[sl-1] = NULL;
        sl = strlen(c_cb);
        c_cb[sl-1] = NULL;

        sl = strlen(c_cGx);
        c_cGx[sl-1] = NULL;
        sl = strlen(c_cGy);
        c_cGy[sl-1] = NULL;
        sl = strlen(c_cn);
        c_cn[sl-1] = NULL;
        sl = strlen(c_ch);
        c_ch[sl-1] = NULL;

        sl = strlen(c_Sk);
```

```
	c_Sk[sl-1] = NULL;
	sl = strlen(c_Sx);
	c_Sx[sl-1] = NULL;
	sl = strlen(c_Sy);
	c_Sy[sl-1] = NULL;

	i = atoi(c_klen);
	if(i == 192 ){
		k = 192; keylength = "192";
		CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO1);
	}else{
		if(i==244){
			k = 244; keylength = "244";
			CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO2);
		}else{
			if(i==256){
				k = 256; keylength = "256";
				CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO3);
			}else{
				if(i==384){
					k = 384; keylength = "384";
					CheckRadioButton(IDC_RADIO1, IDC_RADIO5, IDC_RADIO4);
				}else{
					if(i==521){
						k = 521; keylength = "521";
						CheckRadioButton(IDC_RADIO1, IDC_RADIO5,
IDC_RADIO5);
					}else{
						MessageBox( "鍵の長さが不正です。¥n");
						return ;
					}
				}
			}
		}
	}


//	c_dckey[k/4] = NULL;
//	c_comkey[k/4] = NULL;

	sFName = strNewFileName;
	int j = sFName.ReverseFind( '¥¥' );
	sFName.Delete(0,j+1);
	pwnd = GetDlgItem(IDC_DECRYPTKEY_FILE);
	pwnd->SetWindowText(sFName);
	pwnd = GetDlgItem(IDC_OPENCP);
	pwnd->SetWindowText(c_cp);
	pwnd = GetDlgItem(IDC_OPENCA);
	pwnd->SetWindowText(c_ca);
	pwnd = GetDlgItem(IDC_OPENCB);
	pwnd->SetWindowText(c_cb);
```

```
        pwnd = GetDlgItem(IDC_OPENCGX);   pwnd->SetWindowText(c_cGx);
        pwnd = GetDlgItem(IDC_OPENCGY);   pwnd->SetWindowText(c_cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->SetWindowText(c_cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->SetWindowText(c_ch);

        pwnd = GetDlgItem(IDC_SECRETSK);
        pwnd->SetWindowText(c_Sk);
        pwnd = GetDlgItem(IDC_SECRETSX);
        pwnd->SetWindowText(c_Sx);
        pwnd = GetDlgItem(IDC_SECRETSY);
        pwnd->SetWindowText(c_Sy);

        readdckey = 1;

        return ;

        // Open the file now that it has been created.
//      strcpy(tmppath, strNewFileName);
//      OpenDocumentFile(strNewFileName);


}

void CECCryptDlg::MakeKey()
{
        CString cstr;
        CWnd* pwnd;

        yDisplay("鍵を作成中です。しばらくお待ちください。¥n");
        yDisplay("乱数を使っていますので遅いこともあります。¥n");
        yDisplay("¥r¥n");

        if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 192; keylength = "192";
        }
        if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 244; keylength = "244";
        }
        if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 256; keylength = "256";
        }
        if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 384; keylength = "384";
        }
        if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 521; keylength = "521";
        }


        char *cp, *ca, *cb, *cGx, *cGy, *cn, *ch, *cQx, *cQy, *cSk, *cSx, *cSy;
    int f= makeEccKeys( k, cp, ca, cb, cGx, cGy, cn, ch, cQx, cQy, cSk, cSx, cSy );
        // 共通法);
```

```
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥n");
                case ACCESSERROR:
                        yDisplay("アクセスエラー¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥n");
            case OTHERERROR:
                        yDisplay("その他のエラー¥n");
    }

        yDisplay("鍵を作成しました。¥n");
        yDisplay("¥r¥n");

        pwnd = GetDlgItem(IDC_OPENCP);    pwnd->SetWindowText(cp);
        pwnd = GetDlgItem(IDC_OPENCA);    pwnd->SetWindowText(ca);
        pwnd = GetDlgItem(IDC_OPENCB);    pwnd->SetWindowText(cb);
        pwnd = GetDlgItem(IDC_OPENCGX);   pwnd->SetWindowText(cGx);
        pwnd = GetDlgItem(IDC_OPENCGY);   pwnd->SetWindowText(cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->SetWindowText(cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->SetWindowText(ch);
        pwnd = GetDlgItem(IDC_OPENCQX);   pwnd->SetWindowText(cQx);
        pwnd = GetDlgItem(IDC_OPENCQY);   pwnd->SetWindowText(cQy);

        pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->SetWindowText(cSk);
        pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->SetWindowText(cSx);
        pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->SetWindowText(cSy);

        madekey = 1;
}



void CECCryptDlg::TestKey()
{
        CWnd* pwnd;
        CFileStatus fs;
        int j,f;
        CString Sse, Ssd, Ssn;
        char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
        char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

        datadisp.SetLimitText(INT_MAX);

        if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 192; keylength = "192";
        }
        if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 244; keylength = "244";
```

```
	}
	if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 256; keylength = "256";
	}
	if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 384; keylength = "384";
	}
	if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 521; keylength = "521";
	}


////////////////////////////////////////////////////////////

	if(madekey == 0){
		MessageBox( "＜鍵を作る＞ボタンをクリックして¥n鍵を作成してください。¥n" );
		return ;
	}

	pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
	pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
	pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
	pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
	pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
	pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
	pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
	pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
	pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

	pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
	pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
	pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

		strcpy_s(c_cp, opensp);
		strcpy_s(c_ca, opensa);
		strcpy_s(c_cb, opensb);
		strcpy_s(c_cGx, opensgx);
		strcpy_s(c_cGy, opensgy);
		strcpy_s(c_cn, opensn);
		strcpy_s(c_ch, opensh);
		strcpy_s(c_cQx, opensqx);
		strcpy_s(c_cQy, opensqy);
		strcpy_s(c_cSk, secretsk);
		strcpy_s(c_cSx, secretsx);
		strcpy_s(c_cSy, secretsy);

	yDisplay("¥r¥n");

////////////////////////////////////////////////////


   int numclosed;
```

```
    /* 読み出すファイルを開く
     *（ファイルが存在しないときは、呼び出しが失敗）
     */
          pwnd = GetDlgItem(IDC_PLANEDATA_FILE);
          pwnd->GetWindowText(planedata_file);
          if( (stream1  = fopen( planedata_file, "rb" )) == NULL ){
                    yDisplay("ファイル"); yDisplay(planedata_file); yDisplay(" は開けませんでした。¥r¥n");
                    return;//(-1);
                    }
    else
                    {yDisplay("ファイル"); yDisplay(planedata_file); yDisplay(" は開けました。¥r¥n");}

    /* 暗号文を書き込むファイルを開く*/
          pwnd = GetDlgItem(IDC_ENCRYPTDATA_FILE);
          pwnd->GetWindowText(encryptdata_file);
          if( (stream2 = fopen( encryptdata_file, "w+b" )) == NULL ){
                    yDisplay("ファイル"); yDisplay(encryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                    return;//(-1);
                    }
    else
                    {yDisplay("ファイル"); yDisplay(encryptdata_file); yDisplay(" は開けました。¥r¥n");}

    /* 復号文を書き込むファイルを開く*/
          pwnd = GetDlgItem(IDC_DECRYPTDATA_FILE);
          pwnd->GetWindowText(decryptdata_file);
          if( (stream4 = fopen( decryptdata_file, "w+b" )) == NULL ){
                    yDisplay("ファイル"); yDisplay(decryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                    return;//(-1);
                    }
    else
                    {yDisplay("ファイル"); yDisplay(decryptdata_file); yDisplay(" は開けました。¥r¥n");}


     // 平文
          CFile::GetStatus(planedata_file, fs);
          unsigned long filelen = fs.m_size;
          int c;
          int i = 0;
          char* bufp;
          bufp = (char*)new(char[filelen +1+ int(k/4 + 2) + sizeof(long)]);
          if(bufp == NULL){
                    yDisplay("メモリ不足¥r¥n");
                    return;//(-1);
                    }
          *(unsigned long*)bufp = filelen;
    // 平文
          fseek(stream1,0,0);
          i = sizeof(long);
          do{
```

```
                c = fgetc(stream1);
                bufp[i]=c;
                i=i+1;
        }while(c!=EOF);
        bufp[i-1]=NULL;

        for(int j=0; j<(k/4+2); j++){
                bufp[j+i] = 0;
        }

        int mesLength = i-1; // 平文長（バイト）

        if( fclose( stream1 ) )
      MessageBox( "エラー：　平文ファイルは閉じられませんでした。¥n" );


        char buff[16];
        itoa(mesLength, buff, 10);
    yDisplay("平文長　= "); yDisplay(buff); yDisplay(" byte¥r¥n");
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");

    yDisplay("平文　　= ¥r¥n");
        yDisplay(bufp+sizeof(long));
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");


    // 暗号化実行
        int ciphMesLength;                              // 暗号文長
        char* cstr;                                             // 暗号文格納場所へのポインタ


    f = FFTEccEncryption( k, c_cp, c_ca, c_cb, c_cQx, c_cQy, c_cSk, /*c_cGx, c_cGy,*/ c_cSx, c_cSy,mesLength,
bufp, ciphMesLength, cstr );
    switch( f ){                                        // エラーチェック
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                        yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                        yDisplay("その他のエラー¥r¥n");
    }

        // 暗号文を進数で表示 0xa4　は　A4　と表示される
    yDisplay("暗号文(HEX)= ");
        for( int cnt = 0; cnt<ciphMesLength; cnt++ ){
                char c1, c2;
```

```
                if(cnt%30 ==0){yDisplay("¥r¥n");}

                c1 = toChar((int(cstr[cnt]) >> 4) & 0xf);
                c2 = toChar(int(cstr[cnt]) & 0xf);
                CString sc = (CString)c1;
                sc += c2;
                yDisplay(sc);
                fwrite( &(cstr[cnt]),sizeof(char),1, stream2);
        }
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");

        if( fclose( stream2 ) )
      MessageBox( "エラー：暗文ファイルは閉じられませんでした。¥n" );

   /* 暗号化したデータのファイルを読み込むために開く*/
        if( (stream2 = fopen( encryptdata_file, "rb" )) == NULL ){
                MessageBox( "暗文ファイルを再度開く事ができませんでした。¥n" );
                numclosed = _fcloseall( );
                return;//(-1);
        }

        char* ostr;                                             // 復号文へのポインタ
////////////////////////////////////
        CFile::GetStatus(encryptdata_file, fs);
        filelen = fs.m_size;
        char* bufc;

        bufc = (char*)new(char[filelen + int(k/4 + 3)]);
        if(bufc == 0){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        fseek(stream2,0,0);
        int cc;
        i=0;
        do{
                cc = fgetc(stream2);
                bufc[i]=cc;
                i=i+1;
        }while(cc!=EOF);
        bufc[i-1]=NULL;

        for(j=0; j<(k/4+3); j++){
                bufc[j+i] = 0;
        }

        ciphMesLength = i-1;

        if( fclose( stream2 ) )
      MessageBox( "エラー：暗文ファイルは閉じられませんでした。¥n" );
```

```
//////////////////////////////////////////

    f = FFTEccDecryption( k, c_cp, c_ca, /* c_cb, c_cQx, c_cQy,*/ c_cSk, /*c_cSx, c_cSy,*/ ciphMesLength,
bufc, mesLength, ostr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                    yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                    yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                    yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                    yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                    yDisplay("その他のエラー¥r¥n");
    }

        // 復号文出力
        ostr[mesLength] = '¥0';           // 文字列の終わりを記録

    yDisplay("復号文  = ¥r¥n");
        yDisplay(ostr+sizeof(long));
        yDisplay("¥r¥n");

        filelen = *(long *)ostr;
        for( int ont = sizeof(long); ont<mesLength; ont++ ){
                char c1, c2;
                c1 = toChar((int(ostr[ont]) >> 4) & 0xf);
                c2 = toChar(int(ostr[ont]) & 0xf);
                CString sc = (CString)c1;
                sc += c2;
                if(ont < filelen + sizeof(long)){
                        fwrite( &(ostr[ont]),sizeof(char),1, stream4);
                }
        }

        if( fclose( stream4 ) )
      MessageBox( "復号化ファイルは閉じられませんでした。¥n" );


   /* 他のすべてのファイルを閉じる*/
   numclosed = _fcloseall( );

   delete[] bufp;
   delete[] ostr;
   delete[] cstr;

        return;// 0;
}
```

```
void CECCryptDlg::OnOK()
{
	if(madekey==0){
		MessageBox("鍵を作成してください。");
		return;
	}

	CDialog::OnOK();

	ofstream ofs(encryptkey_file, ios::out);
	ofs << keylength << endl;
	ofs << opensp << endl;
	ofs << opensa << endl;
	ofs << opensb << endl;

	ofs << opensgx << endl;
	ofs << opensgy << endl;
	ofs << opensn << endl;
	ofs << opensh << endl;

	ofs << opensqx << endl;
	ofs << opensqy << endl;

	ofs.close();

	ofstream ofs2(decryptkey_file, ios::out);
	ofs2 << keylength << endl;
	ofs2 << opensp << endl;
	ofs2 << opensa << endl;
	ofs2 << opensb << endl;

	ofs2 << opensgx << endl;
	ofs2 << opensgy << endl;
	ofs2 << opensn << endl;
	ofs2 << opensh << endl;

	ofs2 << secretsk << endl;
	ofs2 << secretsx << endl;
	ofs2 << secretsy << endl;
	ofs2.close();
}


/////////////////////////////////////////////////////////////////////////////
// CRSAcryptDlg メッセージハンドラ

BOOL CECCryptDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// "バージョン情報..." メニュー項目をシステムメニューへ追加します。
```

```
	// IDM_ABOUTBOX はコマンドメニューの範囲でなければなりません。
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// このダイアログ用のアイコンを設定します。フレームワークはアプリケーションのメイン
	// ウィンドウがダイアログでない時は自動的に設定しません。
	SetIcon(m_hIcon, TRUE);			// 大きいアイコンを設定
	SetIcon(m_hIcon, FALSE);		// 小さいアイコンを設定

	// TODO: 特別な初期化を行う時はこの場所に追加してください。

	return TRUE;  // TRUE を返すとコントロールに設定したフォーカスは失われません。
}

void CECCryptDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// もしダイアログボックスに最小化ボタンを追加するならば、アイコンを描画する
// コードを以下に記述する必要があります。MFC アプリケーションはdocument/view
// モデルを使っているので、この処理はフレームワークにより自動的に処理されます。

void CECCryptDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // 描画用のデバイスコンテキスト

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// クライアントの矩形領域内の中央
```

```
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // アイコンを描画します。
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// システムは、ユーザーが最小化ウィンドウをドラッグしている間、
// カーソルを表示するためにここを呼び出します。
HCURSOR CECCryptDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}


void CECCryptDlg::OnEnChangeKeylength()
{
        // TODO:  これがRICHEDIT コントロールの場合、
        // まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
        // フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
        // コントロールは、この通知を送信しません。

        // TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnEnChangeEncryptkeyFile()
{
        // TODO:  これがRICHEDIT コントロールの場合、
        // まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
        // フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
        // コントロールは、この通知を送信しません。

        // TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnBnClickedCancel()
{
        // TODO: ここにコントロール通知ハンドラコードを追加します。
        OnCancel();
}

void CECCryptDlg::OnBnClickedEckread()
{
```

```
	// TODO: ここにコントロール通知ハンドラコードを追加します。
}

void CECCryptDlg::OnBnClickedMakekey()
{
	// TODO: ここにコントロール通知ハンドラコードを追加します。
}

void CECCryptDlg::OnBnClickedOk()
{
	// TODO: ここにコントロール通知ハンドラコードを追加します。
	OnOK();
}

void CECCryptDlg::OnEnChangeCommonkey2()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnEnChangeSecretkey()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnEnChangeDisplay()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnEnChangeOpenkey()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
```

```
}

void CECCryptDlg::OnEnChangeCommonkey1()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
}



void CECCryptDlg::OnBnClickedRadio1()
{	// TODO: ここにコントロール通知ハンドラコードを追加します。
		k = 192; keylength = "192";
}

void CECCryptDlg::OnBnClickedRadio2()
{	// TODO: ここにコントロール通知ハンドラコードを追加します。
		k = 244; keylength = "244";
}

void CECCryptDlg::OnBnClickedRadio3()
{	// TODO: ここにコントロール通知ハンドラコードを追加します。
		k = 256; keylength = "256";
}

void CECCryptDlg::OnBnClickedRadio4()
{	// TODO: ここにコントロール通知ハンドラコードを追加します。
		k = 384; keylength = "384";
}

void CECCryptDlg::OnBnClickedRadio5()
{	// TODO: ここにコントロール通知ハンドラコードを追加します。
		k = 521; keylength = "521";
}




void CECCryptDlg::OnBnClickedDckread()
{
	// TODO: ここにコントロール通知ハンドラコードを追加します。
}

void CECCryptDlg::OnBnClickedButton1() // 暗号化　表示する
{	// TODO: ここにコントロール通知ハンドラコードを追加します。

	CWnd* pwnd;
	CFileStatus fs;
```

```
	int f;
	CString Sse, Ssn;
	char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
	char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

	datadisp.SetLimitText(INT_MAX);


	if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 192; keylength = "192";
	}
	if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 244; keylength = "244";
	}
	if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 256; keylength = "256";
	}
	if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 384; keylength = "384";
	}
	if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 521; keylength = "521";
	}



////////////////////////////////////////////////////////////

	if((madekey == 0) && (readeckey==0)){
		MessageBox( "＜鍵を作る＞ボタンをクリックして¥n鍵を作成してください。¥n" );
		return ;
	}
	if((madekey == 0) && (readdckey==0)){
		MessageBox( "あなたの秘密鍵を読み込んでください。¥n" );
		return ;
	}

	else{

	pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
	pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
	pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
	pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
	pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
	pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
	pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
	pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
	pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

	pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
	pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
```

```
        pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

                strcpy_s(c_cp, opensp);
                strcpy_s(c_ca, opensa);
                strcpy_s(c_cb, opensb);
                strcpy_s(c_cGx, opensgx);
                strcpy_s(c_cGy, opensgy);
                strcpy_s(c_cn, opensn);
                strcpy_s(c_ch, opensh);
                strcpy_s(c_cQx, opensqx);
                strcpy_s(c_cQy, opensqy);
                strcpy_s(c_cSk, secretsk);
                strcpy_s(c_cSx, secretsx);
                strcpy_s(c_cSy, secretsy);

        }
        yDisplay("¥r¥n");
////////////////////////////////////////////////


//   int numclosed;

   /* 読み出すファイルを開く
    *（ファイルが存在しないときは、呼び出しが失敗）
    */
        pwnd = GetDlgItem(IDC_PLANEDATA_FILE);
        pwnd->GetWindowText(planedata_file);
        if( (stream1  = fopen( planedata_file, "rb" )) == NULL ){
                yDisplay("ファイル"); yDisplay(planedata_file); yDisplay(" は開けませんでした。¥r¥n");
                return;//(-1);
                }

   /* 暗号文を書き込むファイルを開く*/
        pwnd = GetDlgItem(IDC_ENCRYPTDATA_FILE);
        pwnd->GetWindowText(encryptdata_file);
        if( (stream2 = fopen( encryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(encryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                return;//(-1);
                }

    // 平文
        CFile::GetStatus(planedata_file, fs);
        unsigned long filelen = fs.m_size;
        int c;
        int i = 0;
        char* bufp;
        bufp = (char*)new(char[filelen +1+ int(k/4 + 2) + sizeof(long)]);
        if(bufp == NULL){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
```

```
        *(unsigned long*)bufp = filelen;
  // 平文
        fseek(stream1,0,0);
        i = sizeof(long);
        do{
                c = fgetc(stream1);
                bufp[i]=c;
                i=i+1;
        }while(c!=EOF);
        bufp[i-1]=NULL;

        for(int j=0; j<(k/4+2); j++){
                bufp[j+i] = 0;
        }
        int mesLength = i-1; // 平文長（バイト）

        char buff[16];
        itoa(mesLength, buff, 10);
    yDisplay("平文長  = "); yDisplay(buff); yDisplay(" byte¥r¥n");
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");

    yDisplay("平文    = ¥r¥n");
        yDisplay(bufp+sizeof(long));
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");


    // 暗号化実行
        int ciphMesLength;                              // 暗号文長
        char* cstr;                                             // 暗号文格納場所へのポインタ

    f = FFTEccEncryption( k, c_cp, c_ca, c_cb, c_cQx, c_cQy, c_cSk, /*c_cGx, c_cGy,*/ c_cSx, c_cSy,mesLength,
bufp, ciphMesLength, cstr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                        yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                        yDisplay("その他のエラー¥r¥n");
    }

        // 暗号文を進数で表示 0xa4  は  A4  と表示される
    yDisplay("暗号文(HEX)= ");
        for( int cnt = 0; cnt<ciphMesLength; cnt++ ){
                char c1, c2;
                if(cnt%30 ==0){yDisplay("¥r¥n");}
```

```
                c1 = toChar((int(cstr[cnt]) >> 4) & 0xf);
                c2 = toChar(int(cstr[cnt]) & 0xf);
                CString sc = (CString)c1;
                sc += c2;
                yDisplay(sc);
                fwrite( &(cstr[cnt]),sizeof(char),1, stream2);
        }
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");

        if( fclose( stream1 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );

    delete[] bufp;
    delete[] cstr;

        yDisplay("¥r¥n");
        yDisplay("暗号化終了。¥n");
        yDisplay("¥r¥n");

        return;
}

void CECCryptDlg::OnBnClickedButton2() // 復号化　表示する
{       // TODO: ここにコントロール通知ハンドラコードを追加します。

        CWnd* pwnd;
        CFileStatus fs;
        int i, j;
//      char *sd, *sn;
        int f;
        CString Ssd, Ssn;
        char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
        char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

        datadisp.SetLimitText(INT_MAX);


        if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 192; keylength = "192";
        }
        if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 244; keylength = "244";
        }
        if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 256; keylength = "256";
        }
        if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
```

```
                k = 384; keylength = "384";
        }
        if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 521; keylength = "521";
        }


////////////////////////////////////////////////////////////

        if((madekey == 0) && (readdckey==0)){
                MessageBox( "秘密鍵を読込んで下さい。¥nまたは、鍵を作成してください。¥n" );
                return ;
        }

        pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
        pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
        pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
        pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
        pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
        pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
        pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

        pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
        pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
        pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

                strcpy_s(c_cp, opensp);
                strcpy_s(c_ca, opensa);
                strcpy_s(c_cb, opensb);
                strcpy_s(c_cGx, opensgx);
                strcpy_s(c_cGy, opensgy);
                strcpy_s(c_cn, opensn);
                strcpy_s(c_ch, opensh);
                strcpy_s(c_cQx, opensqx);
                strcpy_s(c_cQy, opensqy);
                strcpy_s(c_cSk, secretsk);
                strcpy_s(c_cSx, secretsx);
                strcpy_s(c_cSy, secretsy);


        yDisplay("¥r¥n");

    int numclosed;

    /* 暗号化したデータのファイルを読み込むために開く*/
        if( (stream2 = fopen( encryptdata_file, "rb" )) == NULL ){
                MessageBox( "暗文ファイルは開けませんでした。¥n" );
                numclosed = _fcloseall( );
                return;//(-1);
        }
```

```
 /* 復号文を書き込むファイルを開く*/
        pwnd = GetDlgItem(IDC_DECRYPTDATA_FILE);
        pwnd->GetWindowText(decryptdata_file);
        if( (stream4 = fopen( decryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(decryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                return;//(-1);
                }


        CFile::GetStatus(encryptdata_file, fs);
        unsigned long filelen = fs.m_size;

        char* bufc;
        bufc = (char*)new(char[filelen + int(k/4 + 3)+ sizeof(long)]); // 暗号文格納場所へのポインタ
        if(bufc == 0){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        fseek(stream2,0,0);
        int cc;
        i=0;
        do{
                cc = fgetc(stream2);
                bufc[i]=cc;
                i=i+1;
        }while(cc!=EOF);
        bufc[i-1]=NULL;

        for(j=0; j<(k/4+3); j++){
                bufc[j+i] = 0;
        }
        int ciphMesLength = i-1;

        // 暗号文を進数で表示 0xa4  は  A4  と表示される
    yDisplay("暗号文(HEX)= ");
        for( int cnt = 0; cnt<ciphMesLength; cnt++ ){
                char c1, c2;
                if(cnt%30 ==0){yDisplay("¥r¥n");}

                c1 = toChar((int(bufc[cnt]) >> 4) & 0xf);
                c2 = toChar(int(bufc[cnt]) & 0xf);
                CString sc = (CString)c1;
                sc += c2;
                yDisplay(sc);
        }
        yDisplay("¥r¥n");
        yDisplay("¥r¥n");
```

```
////////////////////////////////////////////////////////////////////////////

        int mesLength;
        char* ostr;                                                     // 復号文へのポインタ
    f = FFTEccDecryption( k, c_cp, c_ca,/* c_cb, c_cQx, c_cQy,*/ c_cSk, /*c_cSx, c_cSy,*/ ciphMesLength,
bufc, mesLength, ostr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                    yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                    yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                    yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                    yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                    yDisplay("その他のエラー¥r¥n");
    }

        // 復号文出力

        filelen = *(long *)ostr;

        ostr[filelen+ sizeof(long)] = '¥0';                 // 文字列の終わりを記録
    yDisplay("復号文  = ¥r¥n");
        yDisplay(ostr+sizeof(long));
        yDisplay("¥r¥n");

        for(unsigned int ont = sizeof(long); ont<filelen + sizeof(long); ont++ ){
                char c1, c2;
                c1 = toChar((int(ostr[ont]) >> 4) & 0xf);
                c2 = toChar(int(ostr[ont]) & 0xf);
                CString sc = (CString)c1;
                sc += c2;
                if(ont < filelen + sizeof(long)){
                        fwrite( &(ostr[ont]),sizeof(char),1, stream4);
                }
        }

        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream4 ) )
      MessageBox( "復号化ファイルは閉じられませんでした。¥n" );

        delete[] bufc;
        delete[] ostr;

        yDisplay("¥r¥n");
        yDisplay("復号化終了。¥n");
        yDisplay("¥r¥n");
```

```
	return;// 0;
}

void CECCryptDlg::OnEnChangeDecryptkeyFile()
{
	// TODO:  これがRICHEDIT コントロールの場合、
	// まず、CDialog::OnInitDialog() 関数をオーバーライドして、OR 状態のENM_CHANGE
	// フラグをマスクに入れて、CRichEditCtrl().SetEventMask() を呼び出さない限り、
	// コントロールは、この通知を送信しません。

	// TODO:  ここにコントロール通知ハンドラコードを追加してください。
}

void CECCryptDlg::OnBnClickedButton3() // 暗号化　非表示
{
	// TODO: ここにコントロール通知ハンドラコードを追加します。

	CWnd* pwnd;
	CFileStatus fs;
	int f;
	CString Sse, Ssn;
	char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
	char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

	datadisp.SetLimitText(INT_MAX);


	if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 192; keylength = "192";
	}
	if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 244; keylength = "244";
	}
	if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 256; keylength = "256";
	}
	if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 384; keylength = "384";
	}
	if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 521; keylength = "521";
	}


////////////////////////////////////////////////////////////

	if((madekey == 0) && (readeckey==0)){
		MessageBox( "＜鍵を作る＞ボタンをクリックして¥n鍵を作成してください。¥n" );
		return ;
	}
	if((madekey == 0) && (readdckey==0)){
```

```
                MessageBox( "あなたの秘密鍵を読み込んでください。¥n" );
                return ;
        }

        pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
        pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
        pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
        pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
        pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
        pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
        pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

        pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
        pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
        pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

                strcpy_s(c_cp, opensp);
                strcpy_s(c_ca, opensa);
                strcpy_s(c_cb, opensb);
                strcpy_s(c_cGx, opensgx);
                strcpy_s(c_cGy, opensgy);
                strcpy_s(c_cn, opensn);
                strcpy_s(c_ch, opensh);
                strcpy_s(c_cQx, opensqx);
                strcpy_s(c_cQy, opensqy);
                strcpy_s(c_cSk, secretsk);
                strcpy_s(c_cSx, secretsx);
                strcpy_s(c_cSy, secretsy);

        yDisplay("¥r¥n");
        yDisplay("暗号化開始、しばらくお待ちください。¥r¥n");
////////////////////////////////////////////////////

//   int numclosed;

   /* 読み出すファイルを開く
    *（ファイルが存在しないときは、呼び出しが失敗）
    */
        pwnd = GetDlgItem(IDC_PLANEDATA_FILE);
        pwnd->GetWindowText(planedata_file);
        if( (stream1  = fopen( planedata_file, "rb" )) == NULL ){
                yDisplay("ファイル"); yDisplay(planedata_file); yDisplay(" は開けませんでした。¥r¥n");
                return;//(-1);
                }

   /* 暗号文を書き込むファイルを開く*/
        pwnd = GetDlgItem(IDC_ENCRYPTDATA_FILE);
        pwnd->GetWindowText(encryptdata_file);
        if( (stream2 = fopen( encryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(encryptdata_file); yDisplay(" は開けませんでした。
```

```
¥r¥n");
                return;//(-1);
                }


    // 平文
        CFile::GetStatus(planedata_file, fs);
        unsigned long filelen = fs.m_size;
        int c;
        int i = 0;
        char* bufp;
        bufp = (char*)new(char[filelen +1+ int(k/4 + 2) + sizeof(long)]);
        if(bufp == NULL){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        *(unsigned long*)bufp = filelen;
  // 平文
        fseek(stream1,0,0);
        i = sizeof(long);
        do{
                c = fgetc(stream1);
                bufp[i]=c;
                i=i+1;
        }while(c!=EOF);
        bufp[i-1]=NULL;

        for(int j=0; j<(k/4+2); j++){
                bufp[j+i] = 0;
        }
        int mesLength = i-1; // 平文長（バイト）

        char buff[16];
        itoa(mesLength, buff, 10);



    // 暗号化実行
        int ciphMesLength;                                      // 暗号文長
        char* cstr;                                                     // 暗号文格納場所へのポインタ
    f=FFTEccEncryption(k, c_cp, c_ca, c_cb, c_cQx, c_cQy, c_cSk,/*c_cGx, c_cGy,*/c_cSx, c_cSy, mesLength,
bufp, ciphMesLength, cstr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                        yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
```

```
                        yDisplay("その他のエラー¥r¥n");
    }

        for( int cnt = 0; cnt<ciphMesLength; cnt++ ){
                fwrite( &(cstr[cnt]),sizeof(char),1, stream2);
        }

////////////////////////////////////////////////////////////////////////

        if( fclose( stream1 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );

    delete[] bufp;
    delete[] cstr;

        yDisplay("¥r¥n");
        yDisplay("暗号化終了。¥n");
        yDisplay("¥r¥n");

        return;




}

void CECCryptDlg::OnBnClickedButton4() // 復号化　非表示
{
        // TODO: ここにコントロール通知ハンドラコードを追加します。

        CWnd* pwnd;
        CFileStatus fs;
        int i, j;
//      char *sd, *sn;
        int f;
        CString Ssd, Ssn;
        char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
        char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

        datadisp.SetLimitText(INT_MAX);


        if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 192; keylength = "192";
        }
        if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 244; keylength = "244";
        }
        if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
```

```
		k = 256; keylength = "256";
	}
	if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 384; keylength = "384";
	}
	if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
		k = 521; keylength = "521";
	}


////////////////////////////////////////////////////////////
	/*
	if((madekey == 0) && (readeckey==0)){
		MessageBox( "送信者の公開鍵を読込んで下さい。¥n" );
		return ;
	}
*/
	if((madekey == 0) && (readdckey==0)){
		MessageBox( "＜鍵を作る＞ボタンをクリックして¥n鍵を作成してください。¥n" );
		return ;
	}


	pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
	pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
	pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
	pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
	pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
	pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
	pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
	pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
	pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

	pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
	pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
	pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

		strcpy_s(c_cp, opensp);
		strcpy_s(c_ca, opensa);
		strcpy_s(c_cb, opensb);
		strcpy_s(c_cGx, opensgx);
		strcpy_s(c_cGy, opensgy);
		strcpy_s(c_cn, opensn);
		strcpy_s(c_ch, opensh);
		strcpy_s(c_cQx, opensqx);
		strcpy_s(c_cQy, opensqy);
		strcpy_s(c_cSk, secretsk);
		strcpy_s(c_cSx, secretsx);
		strcpy_s(c_cSy, secretsy);


	yDisplay("¥r¥n");
```

```
        yDisplay("復号化開始、しばらくお待ちください。¥r¥n");

    int numclosed;

    /* 暗号化したデータのファイルを読み込むために開く*/
        if( (stream2 = fopen( encryptdata_file, "rb" )) == NULL ){
                MessageBox( "暗文ファイルは開けませんでした。¥n" );
                numclosed = _fcloseall( );
                return;//(-1);
        }

  /* 復号文を書き込むファイルを開く*/
        pwnd = GetDlgItem(IDC_DECRYPTDATA_FILE);
        pwnd->GetWindowText(decryptdata_file);
        if( (stream4 = fopen( decryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(decryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                return;//(-1);
                }

////////////////////////////////////////////////////////////////////////////////

        CFile::GetStatus(encryptdata_file, fs);
        unsigned long filelen = fs.m_size;

        char* bufc;
        bufc = (char*)new(char[filelen + int(k/4 + 3)+ sizeof(long)]); // 暗号文格納場所へのポインタ
        if(bufc == 0){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        fseek(stream2,0,0);
        int cc;
        i=0;
        do{
                cc = fgetc(stream2);
                bufc[i]=cc;
                i=i+1;
        }while(cc!=EOF);
        bufc[i-1]=NULL;

        for(j=0; j<(k/4+3); j++){
                bufc[j+i] = 0;
        }
        int ciphMesLength = i-1;

        int mesLength;
        char* ostr;                                             // 復号文へのポインタ
    f = FFTEccDecryption( k, c_cp, c_ca,/* c_cb, c_cQx, c_cQy,*/ c_cSk, /*c_cSx, c_cSy,*/ ciphMesLength,
bufc, mesLength, ostr );

        switch( f ){                                    // エラーチェック
```

```
            case NOTENOUGHMEMORY:
                    yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                    yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                    yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                    yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                    yDisplay("その他のエラー¥r¥n");
    }

        // 復号文出力

        filelen = *(long *)ostr;

        ostr[filelen+ sizeof(long)] = '¥0';                // 文字列の終わりを記録

        for(unsigned int ont = sizeof(long); ont<filelen + sizeof(long); ont++ ){
                if(ont < filelen + sizeof(long)){
                    fwrite( &(ostr[ont]),sizeof(char),1, stream4);
                }
        }

        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream4 ) )
      MessageBox( "復号化ファイルは閉じられませんでした。¥n" );

        delete[] bufc;
        delete[] ostr;

        yDisplay("¥r¥n");
        yDisplay("復号化終了。¥n");
        yDisplay("¥r¥n");

        return;// 0;

}


void CECCryptDlg::Search2(){
        CString strNewFileName;

        strNewFileName.LoadString(IDS_BSEARCH);
        CFileDialog fileDlg(TRUE, NULL, NULL, OFN_HIDEREADONLY, strNewFileName);
        if(fileDlg.DoModal() == IDOK){
                strNewFileName = fileDlg.GetPathName();
            l_bin.AddString(strNewFileName);
        }
}
```

```
void CECCryptDlg::AddList2(){
        CWnd* pwnd;
        pwnd = GetDlgItem(IDC_ADDBIN2);
        pwnd->GetWindowText(s_addbin);
    l_bin.AddString(s_addbin);
        s_addbin = "";
        pwnd->SetWindowText(s_addbin);
}

void CECCryptDlg:: DelList2(){
        int index = l_bin.GetCurSel();
        l_bin.DeleteString((UINT)index);
}

void CECCryptDlg::OnBnClickedButton5(){// 暗号化　連続
        CWnd* pwnd;
        CFileStatus fs;
//      char *se, *sn;
        int f;
        CString Sse, Ssn;
        char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
        char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

        datadisp.SetLimitText(INT_MAX);


        if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 192; keylength = "192";
        }
        if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 244; keylength = "244";
        }
        if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 256; keylength = "256";
        }
        if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 384; keylength = "384";
        }
        if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
                k = 521; keylength = "521";
        }



        if((madekey == 0) && (readeckey==0)){
                MessageBox( "＜鍵を作る＞ボタンをクリックして¥n鍵を作成してください。¥n" );
                return ;
        }
        if((madekey == 0) && (readdckey==0)){
                MessageBox( "あなたの秘密鍵を読み込んでください。¥n" );
                return ;
```

```
        }

        pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
        pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
        pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
        pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
        pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
        pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
        pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
        pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
        pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

        pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
        pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
        pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

                strcpy_s(c_cp, opensp);
                strcpy_s(c_ca, opensa);
                strcpy_s(c_cb, opensb);
                strcpy_s(c_cGx, opensgx);
                strcpy_s(c_cGy, opensgy);
                strcpy_s(c_cn, opensn);
                strcpy_s(c_ch, opensh);
                strcpy_s(c_cQx, opensqx);
                strcpy_s(c_cQy, opensqy);
                strcpy_s(c_cSk, secretsk);
                strcpy_s(c_cSx, secretsx);
                strcpy_s(c_cSy, secretsy);

        yDisplay("¥r¥n");
        yDisplay("暗号化開始、しばらくお待ちください。¥r¥n");
////////////////////////////////////////////////////

        SetCurrentDirectory(bufpath);

        char buf[256];
        int i, n;
        n = l_bin.GetCount();

        for(i=0; i<n; i++){
        l_bin.SetCurSel(i);
                l_bin.GetText(i,buf);
////////////////////////////////////////////////////

//   int numclosed;

   /* 読み出すファイルを開く
    *（ファイルが存在しないときは、呼び出しが失敗）
    */
        planedata_file = buf;
        if( (stream1  = fopen( planedata_file, "rb" )) == NULL ){
                yDisplay("ファイル"); yDisplay(planedata_file); yDisplay(" は開けませんでした。¥r¥n");
```

```
                return;//(-1);
                }

    /* 暗号文を書き込むファイルを開く*/
        CString sFName = planedata_file;
        int j = sFName.ReverseFind( '¥¥' );
        sFName.Delete(0,j+1);
        encryptdata_file = "Encrypted¥¥";
        encryptdata_file += sFName;

        if( (stream2 = fopen( encryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(encryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                return;//(-1);
                }

     // 平文
        CFile::GetStatus(planedata_file, fs);
        unsigned long filelen = fs.m_size;
        int c;
        int i = 0;
        char* bufp;
        bufp = (char*)new(char[filelen +1+ int(k/4 + 2) + sizeof(long)]);
        if(bufp == NULL){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        *(unsigned long*)bufp = filelen;
   // 平文
        fseek(stream1,0,0);
        i = sizeof(long);
        do{
                c = fgetc(stream1);
                bufp[i]=c;
                i=i+1;
        }while(c!=EOF);
        bufp[i-1]=NULL;

        for(int j=0; j<(k/4+2); j++){
                bufp[j+i] = 0;
        }
        int mesLength = i-1; // 平文長（バイト）

        char buff[16];
        itoa(mesLength, buff, 10);

     // 暗号化実行
        int ciphMesLength;                                  // 暗号文長
        char* cstr;                                                 // 暗号文格納場所へのポインタ
     f = FFTEccEncryption( k, c_cp, c_ca, c_cb, c_cQx, c_cQy, c_cSk, /*c_cGx, c_cGy,*/ c_cSx, c_cSy, mesLength,
bufp, ciphMesLength, cstr );
     switch( f ){                                   // エラーチェック
```

```
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥r¥n");
                    case ACCESSERROR:
                        yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                        yDisplay("その他のエラー¥r¥n");
    }

        for( int cnt = 0; cnt<ciphMesLength; cnt++ ){
                fwrite( &(cstr[cnt]),sizeof(char),1, stream2);
        }

        if( fclose( stream1 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );

    delete[] bufp;
    delete[] cstr;

        }
////////////////////////////////////////////////////////////////////////


        yDisplay("¥r¥n");
        yDisplay("暗号化終了。¥n");
        yDisplay("¥r¥n");

        return;


}


void CECCryptDlg::OnBnClickedButton6(){// 復号化  連続

        CWnd* pwnd;
        CFileStatus fs;
        int i;
        int f;
        CString Ssd, Ssn;
        char /*c_klen[8],*/ c_cp[150], c_ca[150], c_cb[150], c_cGx[150], c_cGy[150], c_cn[150],
c_ch[150];
        char c_cQx[150], c_cQy[150], c_cSk[150], c_cSx[150], c_cSy[150];

        datadisp.SetLimitText(INT_MAX);
```

```
if(IDC_RADIO1 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
        k = 192; keylength = "192";
}
if(IDC_RADIO2 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
        k = 244; keylength = "244";
}
if(IDC_RADIO3 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
        k = 256; keylength = "256";
}
if(IDC_RADIO4 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
        k = 384; keylength = "384";
}
if(IDC_RADIO5 == GetCheckedRadioButton(IDC_RADIO1, IDC_RADIO5) ){
        k = 521; keylength = "521";
}


if((madekey == 0) && (readeckey==0)){
        MessageBox( "送信者の公開鍵を読込んで下さい。¥n" );
        return ;
}

if((madekey == 0) && (readdckey==0)){
        MessageBox( "秘密鍵を読み込むか、¥n新しく鍵を作成してください。¥n" );
        return ;
}

pwnd = GetDlgItem(IDC_OPENCP);    pwnd->GetWindowText(opensp);//c_cp);
pwnd = GetDlgItem(IDC_OPENCA);    pwnd->GetWindowText(opensa);//c_ca);
pwnd = GetDlgItem(IDC_OPENCB);    pwnd->GetWindowText(opensb);//c_cb);
pwnd = GetDlgItem(IDC_OPENCGX); pwnd->GetWindowText(opensgx);//c_cGx);
pwnd = GetDlgItem(IDC_OPENCGY); pwnd->GetWindowText(opensgy);//c_cGy);
pwnd = GetDlgItem(IDC_OPENCN);    pwnd->GetWindowText(opensn);//c_cn);
pwnd = GetDlgItem(IDC_OPENCH);    pwnd->GetWindowText(opensh);//c_ch);
pwnd = GetDlgItem(IDC_OPENCQX); pwnd->GetWindowText(opensqx);//c_cQx);
pwnd = GetDlgItem(IDC_OPENCQY); pwnd->GetWindowText(opensqy);//c_cQy);

pwnd = GetDlgItem(IDC_SECRETSK);  pwnd->GetWindowText(secretsk);//c_cSk);
pwnd = GetDlgItem(IDC_SECRETSX);  pwnd->GetWindowText(secretsx);//c_cQx);
pwnd = GetDlgItem(IDC_SECRETSY);  pwnd->GetWindowText(secretsy);//c_cQy);

        strcpy_s(c_cp, opensp);
        strcpy_s(c_ca, opensa);
        strcpy_s(c_cb, opensb);
        strcpy_s(c_cGx, opensgx);
        strcpy_s(c_cGy, opensgy);
        strcpy_s(c_cn, opensn);
        strcpy_s(c_ch, opensh);
        strcpy_s(c_cQx, opensqx);
        strcpy_s(c_cQy, opensqy);
        strcpy_s(c_cSk, secretsk);
        strcpy_s(c_cSx, secretsx);
```

```
                strcpy_s(c_cSy, secretsy);

        yDisplay("¥r¥n");
        yDisplay("復号化開始、しばらくお待ちください。¥r¥n");

        ////////////////////////////////////////////////////////////////////


        SetCurrentDirectory(bufpath);

        char buf[256];
        int n,m;
        n = l_bin.GetCount();

        for(m=0; m<n; m++){
        l_bin.SetCurSel(m);
                l_bin.GetText(m,buf);
////////////////////////////////////////////////////
   int numclosed;

   /* 暗号化したデータのファイルを読み込むために開く*/
        encryptdata_file = buf;

        if( (stream2 = fopen( encryptdata_file, "rb" )) == NULL ){
                MessageBox( "暗文ファイルは開けませんでした。¥n" );
                numclosed = _fcloseall( );
                return;//(-1);
        }

 /* 復号文を書き込むファイルを開く*/
        CString sFName = encryptdata_file;
        int j = sFName.ReverseFind( '¥¥' );
        sFName.Delete(0,j+1);
        decryptdata_file = "Decrypted¥¥";
        decryptdata_file += sFName;

        if( (stream4 = fopen( decryptdata_file, "w+b" )) == NULL ){
                yDisplay("ファイル"); yDisplay(decryptdata_file); yDisplay(" は開けませんでした。
¥r¥n");
                return;//(-1);
                }

        CFile::GetStatus(encryptdata_file, fs);
        unsigned long filelen = fs.m_size;

        char* bufc;
        bufc = (char*)new(char[filelen + int(k/4 + 3)+ sizeof(long)]); // 暗号文格納場所へのポインタ
        if(bufc == 0){
                yDisplay("メモリ不足¥r¥n");
                return;//(-1);
                }
        fseek(stream2,0,0);
```

```
        int cc;
        i=0;
        do{
                cc = fgetc(stream2);
                bufc[i]=cc;
                i=i+1;
        }while(cc!=EOF);
        bufc[i-1]=NULL;

        for(j=0; j<(k/4+3); j++){
                bufc[j+i] = 0;
        }
        int ciphMesLength = i-1;

        int mesLength;
        char* ostr;                                                    // 復号文へのポインタ
    f = FFTEccDecryption( k, c_cp, c_ca,/* c_cb, c_cQx, c_cQy,*/ c_cSk, /*c_cSx, c_cSy,*/ ciphMesLength,
bufc, mesLength, ostr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        yDisplay("メモリ不足¥r¥n");
                case ACCESSERROR:
                        yDisplay("アクセスエラー¥r¥n");
            case MATHERROR:
                        yDisplay("計算エラー¥r¥n");
            case KEYLENGTHERROR:
                        yDisplay("鍵長不正¥r¥n");
            case OTHERERROR:
                        yDisplay("その他のエラー¥r¥n");
    }

        // 復号文出力

        filelen = *(long *)ostr;

        ostr[filelen+ sizeof(long)] = '¥0';                // 文字列の終わりを記録

        for(unsigned int ont = sizeof(long); ont<filelen + sizeof(long); ont++ ){
                if(ont < filelen + sizeof(long)){
                        fwrite( &(ostr[ont]),sizeof(char),1, stream4);
                }
        }

        if( fclose( stream2 ) )
      MessageBox( "ファイル'data' は閉じられませんでした。¥n" );
        if( fclose( stream4 ) )
      MessageBox( "復号化ファイルは閉じられませんでした。¥n" );

        delete[] bufc;
        delete[] ostr;

        }
```

```
        ////////////////////////////////////////

        yDisplay("¥r¥n");
        yDisplay("復号化終了。¥n");
        yDisplay("¥r¥n");

        return;// 0;

}
```