CFRSA を利用した暗号ソフト

（C）宇山　靖政
2012.04.10

CFRSA 暗号　について

CFRSAEC.exe と CFRSADC.exe のソースコードです。

公開鍵暗号 RSA 方式のソフトです。“メールもビトマ”、"Cipher Web Mail” における共通鍵（対称鍵）の交換のために作成しました。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。
他の作者の方の作品では、鍵の長さが不十分なので、512 ビット、1024 ビット、1536 ビット、2048 ビットの鍵を扱えるようにしました。（鍵を作成するのに要する時間はそれぞれ、30 秒、10 分、40 分、2 時間です。メモリーの量や CPU の性能で異なる。）
公開鍵暗号は処理速度が遅いので、メールの送信途中での暗号化に利用すると、時間がかかりすぎてサーバーとの接続が切れてしまう恐れがあります。小さなデータの交換に利用するか、事前に暗号化したものを添付ファイルとして送信するようにするひつようがあります。
２セット用意して、普通のメール用と、鍵交換専用のものに分けて使うのがべんりなほうほうだとかんがえます。“メール用”、“鍵交換用”の二つのフォルダを用意してその中にそれぞれ“BtmMailSys”をコピーします。

特徴は、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ることです。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。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、橋本晋之介　氏の
”RSA 暗号技術の基礎から C++による実装まで”
の流れ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。

貿易管理令では

（４）例外規定の確認
貿易外省令第９条の規定に基づき、例外的に、経済産業大臣の許可を取得せずに技術の提供を行うことができる場合があります。貨物の輸出とは異なり、大学・研究機関における技術の提供については公知の技術など、例外規定を適用できるものが多いと思われます。その主な例は以下のとおりです。
○ 公知の技術及び公知とするために提供する技術
・・・公知の技術を提供する場合又は技術を公知とするために当該技術を提供する場合で、次のいずれかに該当する場合は許可が不要です（第２項第９号)。
（イ）新聞、書籍、雑誌、電気通信ネットワーク上のファイル等により、既に不特定多数の者に対して公開されている技術を提供する取引
（ロ）学会誌、公開特許情報等不特定多数の者が入手可能な技術を提供する取引
（ハ）講演会、展示会等において不特定多数の者が入手又は聴講可能な技術を提供する取引
（ニ）ソースコードが公開されているプログラムを提供する取引

（ホ）学会発表用の原稿の送付や雑誌への投稿等、不特定多数の者が入手
又は閲覧可能とすることを目的とする取引など

となっていますので、ソースコードを公開して例外規定に合うようにします。

したがって、著作権を放棄するわけではありません。

以下、ソースコードを掲載いたします。

1. 共通なファイル

```
/*
 * 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
        }
        rl = DEF_COLS;                                  // デフォルトの長さ
        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[520];  // 2048/4 + おまけ= 520
        char ich[520];
        char tch[520];
        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;

        for( int 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
}

// 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<rl; 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 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 < 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] = 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;
                        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];
                        }
                }
        }
        x2.torealpart();
        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 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
}



// 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 = pDFT(aa, aa);
                        aa = res(aa,c);
                }
                else{                                           // 奇数ならば
                        bb -= 1;                                        // 演算数を減ずる
                        r = pDFT(r, aa);
                        r = res(r,c);
                }
        }

        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 = pFFT(aa, aa);
                        aa = res(aa,c);
                }
                else{                                                   // 奇数ならば
                        bb -= 1;                                        // 演算数を減ずる
                        r = pFFT(r, aa);
                        r = res(r,c);
                }
        }

        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は素数ではない
        if(  n == 2 )    return 1;       // 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 = 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 = 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 )));
}



// 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;
	}
}



//////////////////////////////////////
CmplxMPIpDFT(
			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;
        }


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


/*
 * rsa.cpp
 * FFTRSA暗号関数
 *
 */
//
// Copyright 2013 Uyama Yasumasa.
//

#include "stdafx.h"
#include <stdlib.h>
#include "rsa.h"
#include "cmplxmpi.h"


// 暗号化
int FFTRsaEncryption(                           // 0:正常  その他:エラー種別
                const int       keyLength,              // 鍵長
                const char*     modKey,                 // 鍵法数
                const char*     expKeyEC,               // 鍵指数暗号化
                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 = keyLength / 8 + ((keyLength % 8)?1:0);	// 鍵バイト長
	int mesByte = baseByte - 1;		// 処理単位バイト数（暗号化ではbyte減らしておく）
	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 = mesLength + ( mesLength / mesByte ) + safebyte;	// 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

	try{
		CmplxMPI lmk = modKey;			// 法鍵
		CmplxMPI lek = expKeyEC;		// 指数鍵

		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

			CmplxMPI cm = expFFT( (lm), (lek), (lmk) );		// 暗号化
			cm.torealpart();

			cm.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 FFTRsaDecryption(
        const int       keyLength,                      // 鍵長
        const char*     modKey,                         // 鍵法数
        const char*     expKeyDC,                       // 鍵指数復号化
        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 = keyLength / 8 + ((keyLength % 8)?1:0);   // 鍵バイト長
        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 + ( 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;

        try{
                CmplxMPI lmk = modKey;                  // 法鍵
                CmplxMPI ldk = expKeyDC;                // 指数鍵

                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

                        CmplxMPI lm = expFFT( (cm), (ldk), (lmk) );         // 復号化
                        lm.torealpart();

                        lm.copyTo_r( mesp );                // 平文の格納
                        mesp += baseByte-1;                 // 出力ポインタ更新
                        mesLength += 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(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
LOOP:
	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 = randomBit( k / 30 - 1 );
        j.torealpart();
        p = p0 + 2*j*r*s;
        int n = p.getBitLength();
        while( !millerrabin(p,5) ){
                n = p.getBitLength();
                if(n>k) goto LOOP;
                if(n<k) j<<= k-n;
                j += 1;
                p = p0 + 2*j*r*s;
        }
        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;
}


// 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;
}


// RSA公開鍵暗号用の鍵の組を作る (e,d,nを得る)
int     makeRsaKeys(
                        const int k,                    // 鍵長k
                        char*& ce,                              // 公開鍵e
                        char*& cd,                              // 秘密鍵d
                        char*& cn )                             // 共通法鍵n
{
        if( k < MINKEYLENGTH )                  // 鍵長のチェック
                return KEYLENGTHERROR;

        try{
                randInitial();
                CmplxMPI p, q;                                  // 素数
                CmplxMPI n;                                             // 共通法鍵
                int k2 = k >> 1;

                do{
////////////////////////////////////////////////////////////////////////////////
```

```
/*                      p = makePrimeNumber( k2 );
                        do{
                                q = makePrimeNumber( k - k2 );          // kが奇数の場合にbit長く
                        }while( p == q );               // 同一素数は不可
*/
////////////////////////////////////////////////////////////////////////////////
                        p = makePrimeNumber2( k2 );
                        do{
                                q = makePrimeNumber2( k - k2 );         // kが奇数の場合にbit長く
                        }while( p == q );               // 同一素数は不可

////////////////////////////////////////////////////////////////////////////////
                        // 安全性を高めたい場合は以下の行を使用
/*
                        p = makeRsaSafetyPrimeNumber( k2 );
                        do{
                                q = makeRsaSafetyPrimeNumber( k - k2 );  // kが奇数の場合にbit長く
                        }while( p == q );               // 同一素数は不可
*/
////////////////////////////////////////////////////////////////////////////////
                        // 安全性を高めたい場合は以下の行を使用
/*
                        p = makeRsaGordonPrimeNumber( k2 );
                        do{
                                q = makeRsaGordonPrimeNumber( k - k2 );  // kが奇数の場合にbit長く
                        }while( p == q );               // 同一素数は不可
*/
////////////////////////////////////////////////////////////////////////////////


                n = pFFT( p , q);                               // 共通公開鍵nを求める
                }while( n.getBitLength() != k );  // ビット長を確認

                CmplxMPI L = lcm( (p-1), (q-1) ); // 鍵の素Lを求める
                CmplxMPI e = L;                                 // 鍵eの初期値
                CmplxMPI d;                                             // 鍵d


                if( isEven( e ) )                       // 奇数を使う
                        e = e - 1;

                do {
                        // 秘密鍵eを求める
                        e = e - 2;
                        while( ugcd( e, L ) != 1 )
                                e = e - 2;

                        // 公開鍵dを求める
                        d = uinv( (e), (L) );
                                if( (res((pFFT(d , e)) , L)) != 1 )     // dとeが互いに素でなけれ
ば
                                        continue;                       // やり直し
```

```
                } while( d == 0 || d == e );      // 真ならやり直し

                //出力
                ce = (char*)new( char[(e.getMaxColumn_r()+1) *4+1] );
                cd = (char*)new( char[(d.getMaxColumn_r()+1) *4+1] );
                cn = (char*)new( char[(n.getMaxColumn_r()+1) *4+1] );
                if( ce == 0 || cd == 0 || cn == 0 )
                        return NOTENOUGHMEMORY;

                e.getText_r( ce );      d.getText_r( cd );      n.getText_r( cn );

        }
        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;
}


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

// stdafx.cpp : 標準インクルードCFRSADC.pch のみを
// 含むソースファイルは、プリコンパイル済みヘッダーになります。
// stdafx.obj にはプリコンパイル済み型情報が含まれます。

#include "stdafx.h"

// TODO: このファイルではなく、STDAFX.H で必要な
// 追加ヘッダーを参照してください。


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

/*
 * 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:
        eErrType c;                                     // 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
{
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

// 除算の補助

        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  powFFT(                        // FFTでのべき乗
                                const CmplxMPI&, // 被べき数
                                const CmplxMPI& );      // べき数

// 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(                                        // ミラー・ラビン法（素数判定）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 lcm(                                           // 最小公倍数
                        const CmplxMPI&,                // 演算数
                        const CmplxMPI& );                      // 演算数

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


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


/***
 * rsa.h
 * FFTRSA暗号関数
```

```
 */
//
// Copyright 2013 Uyama Yasumasa.
//

// 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）

/*********************/
/*** RSA公開鍵暗号***/
/*********************/

// 暗号化
int FFTRsaEncryption(// 0:成功　その他:エラー
                const int,// 鍵長
                const char*,// 鍵法数
                const char*,// 鍵指数暗号化
                const int,// 平文長
                const char*,// 平文
                int&,// 暗号文長
                char*& );// 暗号文

// 復号化
int FFTRsaDecryption(// 0:成功　その他:エラー
                const int,// 鍵長
                const char*,// 鍵法数
                const char*,// 鍵指数復号化
                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     makeRsaKeys(// 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*& );                               // 共通法鍵n

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

// 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 _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

#include <stdio.h>
#include <tchar.h>



// TODO: プログラムに必要な追加ヘッダーをここで参照してください。

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

2. CFRSAEC.exe だけにあるファイル

```
// CFRSAEC.cpp : コンソールアプリケーションのエントリポイントを定義します。
//

#include "stdafx.h"

#include <iostream>
#include <string.h>
#include <stdlib.h>
#include "rsa.h"
#include <stdio.h>

using namespace std;

FILE *stream;
FILE *stream1;
FILE *stream2;

        char seb[2050];
        char snb[2050];

// 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 ' ';
```

```
}

// メイン
int main(                                                       // 正常終了時:0、異常時:1
                int argc,                                       // 引数の数
                char** argv )                           // 引数へのポインタ
{
        char key[16];
//      char seb[2050];
//      char snb[2050];
        int kk, c, block, i;
        long mesLength; // 平文長（バイト）
        char* bufp;
        int ciphMesLength;                              // 暗号文長
        char* cstr;                                             // 暗号文格納場所へのポインタ

        printf("Start!¥n" );
        // 引数チェック
        if( argc != 4 ){                        // 使い方の誤り
                exit(1);
        }

   int numclosed;
    /* 鍵ファイルを開く*/
   if( (stream  = fopen( argv[1], "rb" )) == NULL )
      printf( "Can not open key file for encryption.¥n" );
  /* 平文を読み出すファイルを開く*/
   if( (stream1  = fopen( argv[2], "rb" )) == NULL )
      printf( "Can not open plane text file.¥n");
   /* 暗号文を書き込むファイルを開く*/
   if( (stream2 = fopen( argv[3], "wb" )) == NULL )
      printf( "Can not open file for encrypted data.¥n" );
        fseek(stream,0,0);
        fscanf(stream,"%s",key);
        fscanf(stream,"%s",seb);
        fscanf(stream,"%s",snb);
        kk = int(atol(key));
        if( kk < MINKEYLENGTH ){                // 鍵長不正
                exit(1);
        }
////////////////////////////////////////////////////////////////
   // 平文
        fseek(stream1, 0, SEEK_END);
        long filelen = ftell(stream1);
        fseek(stream1,0,0);
        int head = sizeof(long);
        mesLength = filelen + head;

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

        //暗号化
```

```
if(mesLength <= mesByte*20){//63が暗号化の作業サイズ　*20＝
        bufp = (char*)new(char[filelen +1+ int(kk/8 + 2)+head]);
        if(bufp == NULL){
            cout << "No Memory" <<"¥n";
                cout.flush();
                return(-1);
                }
  // 平文
        *(long *)bufp = filelen;
        int i = head;
        do{
                c = fgetc(stream1);
                bufp[i]=c;
                i=i+1;
        }while(c!=EOF);
        bufp[i-1]=NULL;
        for(int j=0; j<(kk/8+2); j++){
                bufp[j+i] = NULL;
        }
        mesLength = i-1; // 平文長（バイト）+ head

    // 暗号化実行
    int f = FFTRsaEncryption( kk, snb, seb, mesLength, bufp, ciphMesLength, cstr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        cerr << "Not enough memory.¥n";   cerr.flush();    exit(1);
                case ACCESSERROR:
                        cerr << "Access error.¥n";        cerr.flush();    exit(1);
            case MATHERROR:
                        cerr << "Calculation error.¥n";   cerr.flush();    exit(1);
            case KEYLENGTHERROR:
                        cerr << "Key length error.¥n";    cerr.flush();    exit(1);
            case OTHERERROR:
                        cerr << "Another error.¥n";       cerr.flush();    exit(1);
    }
        fseek(stream2,0,0);
        fwrite( cstr,sizeof(char),ciphMesLength, stream2);
}
else{
                long rBlen = mesLength;
                bufp = (char*)new(char[mesByte*20+1+ int(kk/8 + 2)+head]);//63が暗号化の作業サイズ　*20
＝
                if(bufp == NULL){
                        printf("メモリ不足¥r¥n");
                        return(-1);
                        }
                (*(long*)(bufp)) = filelen;

                int r = 0;
                do{
                // 平文
                        if(r == 0){
```

```
                        i = head;
                        fseek(stream1, r*mesByte*20, 0);//63が暗号化の作業サイズ　*20＝
                }
                if(r > 0 ){
                        i = 0;
                        fseek(stream1, r*mesByte*20-4, 0);//63が暗号化の作業サイズ　*20＝
                }
                do{
                        c = fgetc(stream1);
                        bufp[i]=c;
                        i=i+1;
                }while((c!=EOF) && (i<=mesByte*20));//63が暗号化の作業サイズ　*20＝
                bufp[i-1]=NULL;
                for(int j=0; j<(kk/8+2); j++){
                        bufp[j+i] = NULL;
                }

                if(rBlen >= mesByte*20){ block = mesByte*20; }//63が暗号化の作業サイズ　*20＝
                if(rBlen < mesByte*20){ block = rBlen;}

    // 暗号化実行
    int f = FFTRsaEncryption( kk, snb, seb, block, bufp, ciphMesLength, cstr );
    switch( f ){                                    // エラーチェック
            case NOTENOUGHMEMORY:
                        cerr << "Not enough memory.¥n";   cerr.flush();    exit(1);
                case ACCESSERROR:
                        cerr << "Access error.¥n";        cerr.flush();    exit(1);
            case MATHERROR:
                        cerr << "Calculation error.¥n";   cerr.flush();    exit(1);
            case KEYLENGTHERROR:
                        cerr << "Key length error.¥n";    cerr.flush();    exit(1);
            case OTHERERROR:
                        cerr << "Another error.¥n";       cerr.flush();    exit(1);
    }
        // 暗号文を書き込む
                        fwrite( cstr,sizeof(char),ciphMesLength, stream2);
                        r += 1;
                        rBlen -= block;
                }while(rBlen>0);
        }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
        /* ストリームを閉じる*/
    numclosed = _fcloseall( );
    delete[] bufp;
    delete[] cstr;
        printf("End!¥n" );
        return 0;
}
```

3. CFRSADC.exe だけにあるファイル

```
// CFRSADC.cpp : コンソールアプリケーションのエントリポイントを定義します。
//

#include "stdafx.h"


#include <iostream>
#include <string.h>
#include <stdlib.h>
#include "rsa.h"
#include <stdio.h>

using namespace std;

FILE *stream;
FILE *stream1;
FILE *stream2;

        char sdb[2050];
        char snb[2050];


// 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 ' ';
}


// メイン
int main(                                                       // 正常終了時:0、異常時:1
                int argc,                                               // 引数の数
                char** argv )                                   // 引数へのポインタ
{
        int i, block;
        int head;
        long lenp;
        int numclosed;
        char* bufc;
        char* ostr;                                                     // 平分へのポインタ
        char key[16];
//      char sdb[2050];
//      char snb[2050];
```

```
        int ciphMesLength;
        int mesLength;
        int cc;

        printf("Start!¥n" );
        // 引数チェック
        if( argc != 4 ){                        // 使い方の誤り
                exit(1);
        }

 // 鍵ファイルを開く
 if( (stream  = fopen( argv[1], "rb" )) == NULL )
    printf( "Can not open key file for decryption.¥n" );
/* 平分を書き込むファイルを開く*/
 if( (stream1  = fopen( argv[3], "wb" )) == NULL )
    printf( "Can not open file for decrypted data.¥n");
 /* 暗号文ファイルを開く*/
 if( (stream2 = fopen( argv[2], "rb" )) == NULL )
    printf( "Can not open encrypted data file.¥n" );

        fseek(stream,0,0);
        fscanf(stream,"%s",key);
        fscanf(stream,"%s",sdb);
        fscanf(stream,"%s",snb);
        int kk= int(atol(key));
        int baseByte = kk / 8 + ((kk % 8)?1:0);   // 鍵バイト長

        fseek(stream2, 0, SEEK_END);
        long filelen = ftell(stream2);
        fseek(stream2,0,0);

        if(filelen <= baseByte*20){//64が暗号化の作業サイズ  *20＝
                bufc = (char*)new(char[filelen +1+ int(kk/8 + 2)]);
                int cc;
                int i=0;
                do{
                        cc = fgetc(stream2);
                        bufc[i]=cc;
                        i=i+1;
                }while(cc!=EOF);
                bufc[i-1]=NULL;
                for(int j=0; j<(kk/8+2); j++){
                        bufc[j+i] = NULL;
                }
                ciphMesLength = i-1;

  // 復号化の実行
        int f = FFTRsaDecryption( kk, snb, sdb, ciphMesLength, bufc , mesLength, ostr );

          switch( f ){                                  // エラーチェック
                  case NOTENOUGHMEMORY:
                                  cerr << "Not enough memory.¥n";   cerr.flush();    exit(1);
```

```
                    case ACCESSERROR:
                            cerr << "Access error.¥n";        cerr.flush();    exit(1);
                case MATHERROR:
                            cerr << "Calculation error.¥n";   cerr.flush();    exit(1);
                    case KEYLENGTHERROR:
                            cerr << "Key length error.¥n";    cerr.flush();    exit(1);
                case OTHERERROR:
                            cerr << "Another error.¥n";       cerr.flush();    exit(1);
             }

        // 復号文出力
                head = sizeof(long);
                lenp = *((long*)ostr);
                ostr[lenp + head] = '¥0';          // 文字列の終わりを記録
                fseek(stream1,0,0);
                int wc = fwrite( ostr+head,sizeof(char),lenp, stream1);
                fclose(stream1);
        }
        else{
                long rBlen = filelen;
                bufc = (char*)new(char[baseByte*20+1+ int(kk/8 + 2)]);//64が暗号化の作業サイズ  *20＝
                if(bufc == NULL){
                        printf("メモリ不足¥r¥n");
                        return(-1);
                        }

                int r = 0;
                do{
                        fseek(stream2, r*baseByte*20, 0);//63が暗号化の作業サイズ  *20＝
                        i = 0;
                        do{
                                cc = fgetc(stream2);
                                bufc[i]=cc;
                                i=i+1;
                        }while((cc!=EOF) && (i<=baseByte*20));
                        bufc[i-1]=NULL;
                        for(int j=0; j<(kk/8+2); j++){
                                bufc[j+i] = NULL;
                        }
                        int ciphMesLength = i-1;
                        int mesLength;

                        if(rBlen >= baseByte*20){ block = baseByte*20; }//63が暗号化の作業サイズ  *20
＝
                        if(rBlen < baseByte*20){ block = rBlen;}

    // 復号化実行
                int f = FFTRsaDecryption( kk, snb, sdb, ciphMesLength, bufc , mesLength, ostr );

                switch( f ){                                    // エラーチェック
                        case NOTENOUGHMEMORY:
                                        cerr << "Not enough memory.¥n";   cerr.flush();    exit(1);
```

```
                                case ACCESSERROR:
                                        cerr << "Access error.¥n";        cerr.flush();    exit(1);
                                case MATHERROR:
                                        cerr << "Calculation error.¥n";   cerr.flush();    exit(1);
                                case KEYLENGTHERROR:
                                        cerr << "Key length error.¥n";    cerr.flush();    exit(1);
                                case OTHERERROR:
                                        cerr << "Another error.¥n";       cerr.flush();    exit(1);
                        }
        // 復号文を書き込む
                        if(r==0){
                                int head = sizeof(long);
                                lenp = *((long*)ostr);
        // 復号文出力
                                int wc = fwrite( ostr+head,sizeof(char),mesLength-head, stream1);
                                lenp -= mesLength-head;
                        }
                        if(r>=1){
                                if(lenp >= mesLength){
                                        fwrite( ostr,sizeof(char),mesLength, stream1);
                                        lenp -= mesLength;
                                }
                                else {
                                        if(lenp < mesLength){
                                                fwrite( ostr,sizeof(char),lenp, stream1);
                                                lenp -= lenp;
                                        }
                                }
                        }
                        r += 1;
                        rBlen -= block;
                }while(rBlen>0);
        }

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

        delete[] bufc;
        delete[] ostr;
        printf("End!¥n" );
        return 0;
}


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

終わり。

                                2013.01.27              宇山靖政
```