# Chapter 16 high precision package

Speaker: Lung-Sheng Chien

Reference: high precision package in http://crd.lbl.gov/~dhbailey/mpdist/
and http://www.cs.berkeley.edu/~yozo/

# OutLine

- How to add high precision package into program
- Embed high precision package in vc2005
- Embed high precision package in Linux through qmake

# Recall: relaxation for high precision package

global.h

```
#ifndef  GLOBAL_H

// in 32-bit machine, integer is 4 byte
// in 64-bit machine, integer is 8 byte (recommand)
typedef  long int  integer ;

// we may use high precision representation,
// double: 16 digits
// dd (double-double): 32 digits
// qd (quadruple-double: 64 bits
// arprec (arbitray precision): up to 1000 digits
typedef  double   doublereal ;

enum orderVar { ROW_MAJOR , COL_MAJOR } ;

// {ONE_NORM, TWO_NORM, INF_NORM} is used in function "norm"
enum matrix_keyword {
        ONE_NORM = 300 ,
        TWO_NORM = 301 ,
        INF_NORM = 302
    } ;

#endif  //GLOBAL_H
```

Large 1D array, “int” is 4-byte, only supports up to 2G elements

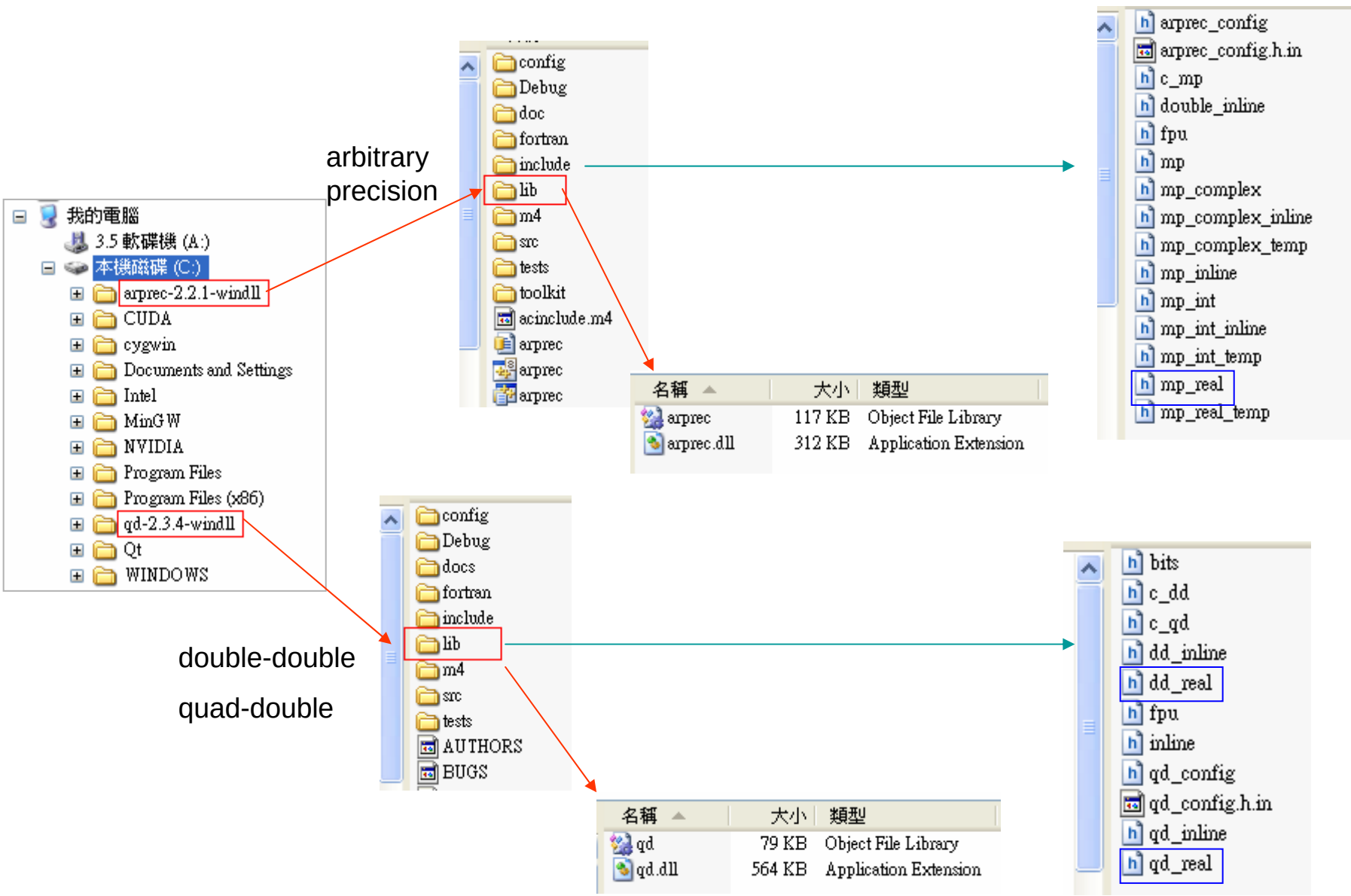
We use col-major, however you can implement both.

How to use high precision package

http://crd.lbl.gov/~dhbailey/mpdist/

- **ARPREC** (C++/Fortran-90 arbitrary precision package)
  Unix-based systems (including Apple Macintosh systems): arprec-2.2.2.tar.gz (version date 2008-07-24)
  Windows systems: arprec-2.2.1-windll.zip (version date 2008-01-23)
  Before installing either version, please see the "release notes" in the README file.
- **QD** (C++/Fortran-90 double-double and quad-double package)
  Unix-based systems (including Apple Macintosh systems): qd-2.3.7.tar.gz (version date 2008-07-22)
  Windows systems: qd-2.3.4-windll.zip (version date 2008-01-23)
  Before installing either version, please see "release notes" in the README file.

# Installation directory of high precision package



# Modify global.h [1]

global.h

```
/*------------------ macro for High precision package -------------------------
 *
 *  if DO_ARPREC is defined, then do Arbitrary precision operation for double
 *  at this time, we define
 *      typedef mp_real doublereal;
 *
 *  if DO_DOUBLE_DOUBLE is defined, then double-double datatype
 *  (approx. 32 decimal digits) is executed, we define
 *      typedef dd_real doublereal;
 *
 *  if DO_QUAD_DOUBLE is defined, then  quad-double datatype (approx. 64 decimal digits)
 *  is executed, we define
 *      typedef qd_real doublereal;
 *
 *  precedence of three macro, DO_ARPREC, DO_DOUBLE_DOUBLE and DO_QUAD_DOUBLE
 *  DO_DOUBLE_DOUBLE > DO_QUAD_DOUBLE > DO_ARPREC
 */

#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
//#define DO_ARPREC

/*------------------ END macro for High precision package ------------------ */
```

user-configurable

***Objective:*** easy to change different precision, you only choose one and comment the others.

1 double

```
//#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

2 double-double

```
#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

3 quad-double

```
//#define DO_DOUBLE_DOUBLE
#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

4 arbitrary-precision

```
//#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
#define DO_ARPREC
```

# Modify global.h [2]

global.h

```
//#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
#define DO_ARPREC

/*------------------- END macro for High precision package ------------------- */

/* ------------------------------------------------------------------------
 *              macro to make sure precedence of three macro
 *              DO_ARPREC, DO_DOUBLE_DOUBLE and DO_QUAD_DOUBLE
 *  DO_DOUBLE_DOUBLE > DO_QUAD_DOUBLE > DO_ARPREC
 *
 *------------------------------------------------------------------------*/
#ifdef DO_DOUBLE_DOUBLE

#ifdef DO_QUAD_DOUBLE
#undef DO_QUAD_DOUBLE
#endif

#ifdef DO_ARPREC
#undef DO_ARPREC
#endif

#endif // DO_DOUBLE_DOUBLE

#ifdef DO_QUAD_DOUBLE

#ifdef DO_ARPREC
#undef DO_ARPREC
#endif

#endif // DO_QUAD_DOUBLE

/* -------------------- END macro to make sure precedence of three macro ------ */
```

1 double-double

```
#define DO_DOUBLE_DOUBLE
#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

2 double-double

```
#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
#define DO_ARPREC
```

3 double-double

```
#define DO_DOUBLE_DOUBLE
#define DO_QUAD_DOUBLE
#define DO_ARPREC
```

4 quad-double

```
//#define DO_DOUBLE_DOUBLE
#define DO_QUAD_DOUBLE
#define DO_ARPREC
```

Precedence (優先次序): double-double > quad-double > arbitrary-precision

# Modify global.h [3]

global.h

```
/* ------------------- macro to add include PATH ---------------
 *                     of High Precision Package
 *-----------------------------------------------------------------*/

#ifdef DO_DOUBLE_DOUBLE
    #define DD_NDIGITS      32
    #include <qd/dd_real.h>
    #include <qd/fpu.h>
#endif //DO_DOUBLE_DOUBLE


#ifdef DO_QUAD_DOUBLE
    #define QD_NDIGITS      64
    #include <qd/qd_real.h>
    #include <qd/fpu.h>
#endif // DO_QUAD_DOUBLE

#ifdef DO_ARPREC
 /*
  *  ARPREC_NDIGITS decides the precision of arbitrary precision
  *
  *  in main.cpp, we initilize the precision of arprec by
  *
  *  #ifdef DO_ARPREC
  *      mp::mp_init(ARPREC_NDIGITS);
  *  #endif
  */
    #define     ARPREC_NDIGITS      128
    #include <arprec/c_mp.h>
    #include <arprec/mp_real.h>
    #include <arprec/fpu.h>
#endif // DO_ARPREC
```

precision of double-double is 32 digits

precision of quad-double is 64 digits

precision of arbitrary-precision is up to 1024 digits

user-configurable

These header files are essential during compilation. Moreover these header files work for windows and unix system (e.g. Linux).

# Modify global.h [4]

global.h

```
#if defined(DO_DOUBLE_DOUBLE) || defined(DO_QUAD_DOUBLE) || defined(DO_ARPREC)

#define HIGH_PRECISION_PACKAGE

#endif

#ifdef  HIGH_PRECISION_PACKAGE

#if defined(DO_DOUBLE_DOUBLE)
    #define HIGH_PRECISION_NDIGITS  DD_NDIGITS

#elif defined(DO_QUAD_DOUBLE)
    #define HIGH_PRECISION_NDIGITS  QD_NDIGITS

#elif defined(DO_ARPREC)
    #define HIGH_PRECISION_NDIGITS  ARPREC_NDIGITS

#else
    #define HIGH_PRECISION_NDIGITS  16

#endif

#endif // HIGH_PRECISION_PACKAGE
```

If one uses high precision package, then we define a new macro ***HIGH_PRECISION_PACKAGE***, you can use this macro in your code.

# Modify global.h [5]

## global.h

```
/* --------------  macro of type declaration -------------------
 *
 *      these macros mainly come from f2c.h
 *
 *  here we only focus on type "doublereal"
 *  1. normal
 *      typedef double doublereal;
 *  2. ARPREC
 *      typedef mp_real doublereal;
 *  3. DD (32 digits)
 *      typedef dd_real doublereal;
 *  4. QD (64 digits)
 *      typedef qd_real doublereal;
 *
 *-----------------------------------------------------------*/

typedef long int integer;
typedef unsigned long int uinteger;
typedef short int shortint;
typedef float real;

#if defined(DO_DOUBLE_DOUBLE)
    typedef dd_real doublereal;

#elif defined(DO_QUAD_DOUBLE)
    typedef qd_real doublereal;

#elif defined(DO_ARPREC)
    typedef mp_real doublereal;

#else
    typedef double doublereal;

#endif
```

```
struct QD_API dd_real {
  double x[2];

  dd_real(double hi, double lo) { x[0] = hi; x[1] = lo; }
  dd_real() {}
  dd_real(double h) { x[0] = h; x[1] = 0.0; }
  dd_real(int h) {
    x[0] = (static_cast<double>(h));
    x[1] = 0.0;
  }

  dd_real (const char *s);
  explicit dd_real (const double *d) {
    x[0] = d[0]; x[1] = d[1];
  }
```

```
struct QD_API qd_real {
  double x[4];    /* The Components. */

  /* Eliminates any zeros in the middle component(s). */
  void zero_elim();
  void zero_elim(double &e);

  void renorm();
  void renorm(double &e);

  void quick_accum(double d, double &e);
  void quick_prod_accum(double a, double b, double &e);

  qd_real(double x0, double x1, double x2, double x3);
  explicit qd_real(const double *xx);
```

```
/*
 *  This class represents MP real numbers.
 */
struct ARPREC_API mp_real: public mp {
  double *mpr;
  bool    alloc;

  static mp_real _pi;
  static mp_real _log2;
  static mp_real _log10;
  static mp_real _eps;
```

variable-length

# Recall: data structure of matrix and its method

matrix.h

```
#ifndef  MATRIX_H

#include "global.h"
#include <stdio.h>

typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ; // col-major or row-major
    doublereal **A ;
} matrix ;

typedef  matrix*   matrixHandler ;

typedef struct int_matrix
{
    integer  m ;
    integer  n ;
    orderVar sel ; // col-major or row-major
    integer  **A ;
} int_matrix ;

typedef  int_matrix*   int_matrixHandler ;
```

```
// matrix constructor
matrixHandler  zeros( integer m, integer n, orderVar sel ) ;

int_matrixHandler  int_zeros( integer m, integer n, orderVar sel ) ;

// in MATLAB we don't need memory management, but in C,
// we must take care it
void  dealloc( matrixHandler Ah ) ;
void int_dealloc( int_matrixHandler Ah ) ;

// show content of matrix in standard form
void  disp( matrixHandler Ah, FILE* fp ) ;
void int_disp( int_matrixHandler Ah, FILE* fp ) ;

// the same as "norm" in MATLAB
doublereal  norm( matrixHandler Ah, matrix_keyword p ) ;

// y = A*x
void matvec( matrixHandler Ah, matrixHandler xh, matrixHandler yh) ;

// C = A - B
void matsub( matrixHandler Ah, matrixHandler Bh, matrixHandler Ch) ;

// copy A to A_shadow
void duplicate( matrixHandler Ah, matrixHandler Ah_shadow ) ;

#endif // MATRIX_H
```

1. constructor (建構子): zeros
2. destructor (解構子): dealloc
3. Index rule: we do it explicitly
4. Arithmetic: matvec, matsub, norm
5. I/O: disp

# Change I/O from FILE* to iostream

```
#include "global.h"  // declaration of doublereal

#include <stdio.h>
#include <iostream> // type of I/O
#include <iomanip>  // manipulate I/O
using namespace std ;

typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ; // col-major or row-major
    doublereal **A ;
} matrix ;

typedef  matrix*   matrixHandler ;

typedef struct int_matrix
{
    integer  m ;
    integer  n ;
    orderVar sel ; // col-major or row-major
    integer  **A ;
} int_matrix ;

typedef  int_matrix*   int_matrixHandler ;
```

```
// matrix constructor
matrixHandler  zeros( integer m, integer n, orderVar sel ) ;

int_matrixHandler  int_zeros( integer m, integer n, orderVar sel ) ;

// in MATLAB we don't need memory management, but in C,
// we must take care it
void dealloc( matrixHandler Ah ) ;

void int_dealloc( int_matrixHandler Ah ) ;

// show content of matrix in standard form

void disp( matrixHandler Ah,  ostream& out ) ;

void int_disp( int_matrixHandler Ah, FILE* fp ) ;

// the same as "norm" in MATLAB
doublereal  norm( matrixHandler Ah, matrix_keyword p ) ;

// y = A*x
void matvec( matrixHandler Ah, matrixHandler xh, matrixHandler yh) ;

// C = A - B
void matsub( matrixHandler Ah, matrixHandler Bh, matrixHandler Ch) ;

// copy A to A_shadow
void duplicate( matrixHandler Ah, matrixHandler Ah_shadow ) ;

#endif // MATRIX_H
```

An output stream object is a destination for bytes. The three most important output stream classes are ostream, ofstream, and ostrstream.

The **ostream** class, through the derived class basic_ostream, supports the predefined stream objects:

- **cout** standard output
- **cerr** standard error with limited buffering
- **clog** similar to **cerr** but with full buffering

# constructor

```
#include "matrix.h"

// constructor: set up 0-matrix
matrixHandler  zeros( integer m, integer n, orderVar sel )
{
    integer j ;
    doublereal **A ;
    doublereal *memA ; // contiguous memory block of A
    integer size ; // number of entries in matrix A

    assert( 0 < m ) ;  assert( 0 < n ) ;
// alolocate an empty handler
    matrixHandler  Ah = (matrixHandler) malloc( sizeof(matrix) ) ;
    assert( Ah ) ;

    if ( COL_MAJOR == sel ){
// A[0] is useless, A[j] means pointer of column j
        A = (doublereal **) malloc( sizeof(doublereal *)*(n+1) ) ;
        assert( A ) ;
        size = m*n ;

#ifdef  HIGH_PRECISION_PACKAGE

        memA = new doublereal [size] ;

#else
        memA = (doublereal*) malloc( sizeof(doublereal)*size ) ;
#endif

        assert( memA ) ;
        for ( j=0 ; j < size ; j++){
            memA[j] = 0.0 ;  // reset matrix A as zero matrix
        }

        for ( j=1 ; j<=n ; j++){
// A[j][0] is useless, A[j][i] means A(i,j)
            A[j] = (memA - 1) + (j-1)*m ;
        }
    }else{
        printf("Error: we don't support row-major so far\n");
        exit(1) ;
    }
// set parameter of a matrix
    Ah->m = m  ;  Ah->n = n ; Ah->sel = sel ; Ah->A = A ;

    return Ah ;
}
```

```
/*
 *  This class represents MP real numbers.
 */
struct ARPREC_API mp_real: public mp {
  double *mpr;
  bool    alloc;

  static mp_real _pi;
  static mp_real _log2;
  static mp_real _log10;
  static mp_real _eps;
```

variable-length

"new" is C++ keyword, it is not only ***malloc*** but do initialization of object.

The *allocation-expression* — the expression containing the **new** operator — does three things:

- Locates and reserves storage for the object or objects to be allocated. When this stage is complete, the correct amount of storage is allocated, but it is not yet an object.
- Initializes the object(s). Once initialization is complete, enough information is present for the allocated storage to be an object.
- Returns a pointer to the object(s) of a pointer type derived from *new-type-name* or *type-name*. The program uses this pointer to access the newly allocated object.

# destructor

```
// The procedure holds for row-major or col-major
void dealloc( matrixHandler Ah )
{
// step 1: dealloc contiguous memory block

#ifdef  HIGH_PRECISION_PACKAGE
    delete [] ( (doublereal*)(Ah->A[1]) + 1 ) ;
#else
    free( (doublereal*)(Ah->A[1]) + 1 ) ;
#endif


// step 2: dealloc column-index array
    free( Ah->A ) ;

// step 3: dealloc matrix handler
    free( Ah ) ;
}
```

When we use “new” operator for allocation, we must use “delete” operator for deallocation

**How delete Works**

See Also

Collapse All  Language Filter: Multiple

The delete operator invokes the function operator delete. For objects of class types (class, struct, and union), the delete operator invokes the destructor for an object prior to deallocating memory (if the pointer is not null). For objects not of class type, the global delete operator is invoked. For objects of class type, the delete operator can be defined on a per-class basis; if there is no such definition for a given class, the global operator is invoked.

# I/O: output operator <<

```
//void disp( matrixHandler Ah,  FILE* fp )
void disp( matrixHandler Ah, ostream& out )
{
    int i, j ;
    doublereal **A ;

    assert( Ah ) ;
    assert( COL_MAJOR == Ah->sel ) ;
1
    out.precision(4) ;
2   out << std::fixed ;

3   out << "dimension of matrix is (" << Ah->m  << "," << Ah->n  << ") with col-major"
        << endl ;

    A = Ah->A ;
    for( i=1 ; i <= Ah->m ; i++ ){
        for( j=1 ; j <= Ah->n ; j++ ){
            //fprintf(fp, "%8.4f  ",  A[j][i]  );
      4     out << setw(10) << A[j][i]  ;
        }
        out << endl ;  5
    }
    out.precision(16) ;
    out << std::scientific ;
}
```

1. only print 4 digits after decimal point,
   It is the same as 4 in “%10.**4**f”.
2. Use fixed point format, it is the same as flag *f* in “%10.4**f**”
3. operator **<<** sends string “dimension of matrix is (” to ostream object *out,*
   then send object *Ah->m* to *out,*
4. setw(10) means “set width as 10 characters”,
   it is the same as 10 in “%**10**.4f”
5. ***endl*** means linefeed (new line)

```
basic_ostream& operator<<(short _Val);
basic_ostream& operator<<(unsigned short _Val);
basic_ostream& operator<<(int _Val);
basic_ostream& operator<<(unsigned int __w64 _Val);
basic_ostream& operator<<(long _Val);
basic_ostream& operator<<(unsigned long _Val);
basic_ostream& operator<<(const void *_Val);
```

Question: why cannot we use ***printf*** ? Try ***printf*** and what happens?

# y = Ax (no modification)

```
// y = A*x
void matvec( matrixHandler Ah, matrixHandler xh, matrixHandler yh)
{
    integer  m, n, s ;
    integer  i, j, k ;
    doublereal **A ;
    doublereal **x ;
    doublereal **y ;

    assert( Ah ) ;  assert( xh ) ;  assert( yh ) ;
    assert( COL_MAJOR == Ah->sel ) ;
    assert( COL_MAJOR == xh->sel ) ;
    assert( COL_MAJOR == yh->sel ) ;

    m = Ah->m ; n = Ah->n ;  s = xh->n ;

    assert(n == xh->m) ;
    assert(m == yh->m) ; assert(s == yh->n) ;

    A = Ah->A ; x = xh->A ; y = yh->A ;
// y = x(1)*A(:,1) + sum_{j}( x(j)*A(:,j) )
    for ( k=1 ; k <= s ; k++){
        for ( i=1 ; i <= m ; i++ ){
            y[k][i] = A[1][i] * x[k][1] ;
        }
        for ( j=2 ; j <= n ; j++){
            for ( i=1 ; i <= m ; i++ ){
                y[k][i] += A[j][i] * x[k][j] ;
            }
        }// for each col-j
    }// for each vector x(:,k)
}
```

***dd_real.h***

```
QD_API dd_real operator*(const dd_real &a, double b);
QD_API dd_real operator*(double a, const dd_real &b);
QD_API dd_real operator*(const dd_real &a, const dd_real &b);

        dd_real &operator+=(double a);
        dd_real &operator+=(const dd_real &a);

        dd_real &operator=(double a);
        dd_real &operator=(const char *s);
```

***mp_real.h***

```
        // Assignment operator.
        mp_real& operator=(const int&);
        mp_real& operator=(const double&);
        mp_real& operator=(const mp_real&);
        mp_real& operator=(const mp_real_temp&);
        mp_real& operator=(const char*);
        mp_real& operator+=(const mp_real&);
        mp_real& operator-=(const mp_real&);
        mp_real& operator*=(const mp_real&);
        mp_real& operator*=(double);
        mp_real& operator/=(const mp_real&);
        mp_real& operator/=(double);

ARPREC_API mp_real_temp operator-(const mp_real &a, const mp_real &b);
ARPREC_API mp_real_temp operator+(const mp_real &a, const mp_real &b);
ARPREC_API mp_real_temp operator*(const mp_real &a, const mp_real &b);
ARPREC_API mp_real_temp operator*(const mp_real &a, const double b);
ARPREC_API mp_real_temp operator*(const double b, const mp_real& a);
ARPREC_API mp_real_temp operator/(const mp_real &a, const mp_real &b);
ARPREC_API mp_real_temp operator/(const mp_real &a, const double b);
ARPREC_API mp_real_temp operator/(const double b, const mp_real &a);
```

Arithmetic operators have been defined in class dd_real, qd_real and mp_real.

# *Matrix / vector norm (no modification)*

matrix.cpp

```
// implement p = 1, 2 and inf
doublereal  norm( matrixHandler Ah, matrix_keyword p )
{
    integer m, n ;
    doublereal  sum ;
    doublereal  sumMax ;
    doublereal  **A ;
    int i, j ;

    assert( Ah ) ;
    assert( COL_MAJOR == Ah->sel ) ;
    m = Ah->m ; n = Ah->n ;
    A = Ah->A ;
    if ( INF_NORM == p ){
        sumMax = 0.0 ;
        for ( i = 1 ; i <= m ; i++ ){
            sum = 0.0 ;
            for ( j = 1 ; j <= n ; j++){
                sum += fabs( A[j][i] ) ;
            }
            if ( sum > sumMax ){
                sumMax = sum ;
            }
        }// for each row
    }else if ( ONE_NORM == p ){
        sumMax = 0.0 ;
        for ( j = 1 ; j <= n ; j++ ){
            sum = 0.0 ;
            for ( i = 1 ; i <= m ; i++){
                sum += fabs( A[j][i] ) ;
            }
            if ( sum > sumMax ){
                sumMax = sum ;
            }
        }// for each col
    }else{
        printf("TWO_NORM is NOT implement\n");
        exit(1) ;
    }
    return sumMax ;
}
```

***dd_real.h***

```
QD_API dd_real fabs(const dd_real &a);
QD_API dd_real abs(const dd_real &a);   /* same as fabs */
```

***mp_real.h***

```
ARPREC_API mp_real_temp pow(const mp_real& a, int n);
ARPREC_API mp_real_temp pow(const mp_real& a, const mp_real& b);
ARPREC_API mp_real_temp pow(const mp_real& a, double b);
ARPREC_API mp_real_temp abs(const mp_real& a);
ARPREC_API mp_real_temp sqrt(const mp_real& a);
ARPREC_API mp_real_temp aint(const mp_real& a);
```

***global.h***

```
#ifdef DO_ARPREC

#ifndef ARPREC_SUPPLEMENT

    #define ARPREC_SUPPLEMENT

    inline mp_real_temp fabs(const mp_real& a){ return abs(a);}

    #if  defined(_WIN32) || defined(__WIN32__)
        inline mp_real_temp pow(const mp_real& a, const mp_real& b) {
            return exp(b * log(a)) ;
        }

        inline mp_real_temp pow(const mp_real& a, double b_db) {
            mp_real  b = b_db ;
            return exp(b * log(a)) ;
        }
    #endif // _WIN32 || __WIN32__

#endif // ARPREC_SUPPLEMENT

#endif // DO_ARPREC
```

# *Prototype of main function*

main.cpp

```
#include "global.h"
#include <stdio.h>
#include "matrix.h"
#include "GaussianEliminate.h"

#include <iostream>
using namespace std ; // use cout

void  test_PA_LU ( void ) ;

int main( int argc, char* argv[])
{

#ifdef  HIGH_PRECISION_PACKAGE
    unsigned int old_cw;
    fpu_fix_start(&old_cw);
#endif

#ifdef DO_ARPREC
    mp::mp_init(ARPREC_NDIGITS);
#endif

// ---------- main code ---------------

    test_PA_LU() ;

// ---------- end main code ------------

#ifdef DO_ARPREC
    mp::mp_finalize();
#endif

#ifdef  HIGH_PRECISION_PACKAGE
    fpu_fix_end(&old_cw);
#endif

    return 0 ;
}
```

README in directory C:\qd-2.3.4-windll

```
The algorithms in this library assume IEEE double precision floating
point arithmetic.  Since Intel x86 processors have extended (80-bit)
floating point registers, the round-to-double flag must be enabled in
the control word of the FPU for this library to function properly
under x86 processors.  The following functions contains appropriate
code to facilitate manipulation of this flag.  For non-x86 systems
these functions do nothing (but still exist).

fpu_fix_start    This turns on the round-to-double bit in the
                 control word.
fpu_fix_end      This restores the control flag.

These functions must be called by the main program, as follows:

    int main() {
      unsigned int old_cw;
      fpu_fix_start(&old_cw);

      ... user code using quad-double library ...

      fpu_fix_end(&old_cw);
    }
```

README in directory C:\arprec-2.2.1-windll

```
Before any multiprecision variables are created, the user should call
the function "mp_init()", which has one required argument: the needed
precision in decimal digits. For example, calling mp_init(300) will
initialize the library for arithmetic at or below 300 decimal digits.
```

# Test driver [1]

main.cpp

```
void  test_PA_LU( void )
{
    integer m = 4 ;
    integer n = 4 ;
    matrixHandler Ah ;
    int_matrixHandler Ph ;
    matrixHandler bh ;
    matrixHandler xh ;  // x = inv(A)*b
    matrixHandler Ah_dup ;
    matrixHandler bh_hat ; // b_hat = A*x
    matrixHandler rh ; // residual r = b - Ax
    doublereal  r_supnorm ;
    int conti_flag = 1 ;
    int end_flag ;
    doublereal **A ;
    doublereal **b ;

// step 1: set up a matrix A
    Ah = zeros( m, n, COL_MAJOR ) ;
    A  = Ah->A ;
    A[1][1] =  6. ; A[1][2] = 12. ; A[1][3] =  3.  ; A[1][4] = -6.  ;
    A[2][1] = -2. ; A[2][2] = -8. ; A[2][3] = -13. ; A[2][4] =  4.  ;
    A[3][1] =  2. ; A[3][2] =  6. ; A[3][3] =  9.  ; A[3][4] =  1.  ;
    A[4][1] =  4. ; A[4][2] = 10. ; A[4][3] =  3.  ; A[4][4] = -18. ;

        printf("matrix A = \n");

        disp( Ah, cout ) ;

// duplicate A since L,U reuse storage of A
    Ah_dup = zeros( m, n, COL_MAJOR ) ;
    duplicate( Ah, Ah_dup ) ;

    Ph = int_zeros( m, 1, COL_MAJOR ) ;
// step 2: factorization, PA = LU
    end_flag = lu_partialPivot( Ah, Ph, conti_flag ) ;

        printf("\nPA = LU : store L and U into A:\n");

        disp( Ah, cout ) ;

        printf("\npermutation matrix P:\n");
        int_disp( H, stdout ) ;
```

$$A = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{pmatrix}$$

We have modified function ***disp***

**The std namespace**

The ANSI/ISO C++ standard requires you to explicitly declare the namespace in the standard library. For example, when using iostream, you must specify the namespace of **cout** in one of the following ways:

- std::cout (explicitly)
- using std::cout (**using** declaration)
- using namespace std (**using** directive)

# Test driver [2]

main.cpp

```
// step 3: generate right hand side vector b
    bh = zeros( m, 1, COL_MAJOR ) ;
    b  = bh->A ;
    b[1][1] =  5. ; b[1][2] = 6. ; b[1][3] =  7.  ; b[1][4] = 8.  ;
        printf("\nright hand side vector b:\n");

        disp( bh, cout ) ;

    xh = zeros( m, 1, COL_MAJOR ) ;
// step 4: solve x = inv(A)*b
    lu_partialPivot_lin_sol( Ah, Ph, bh, xh ) ;

        printf("\nsolution x = inv(A)*b\n");

        disp( xh, cout ) ;

// step 5: verify residual r = b - A*x
    bh_hat = zeros( m, 1, COL_MAJOR ) ;
    rh     = zeros( m, 1, COL_MAJOR ) ;

    matvec( Ah_dup, xh, bh_hat ) ; // b_hat = A*x
    matsub( bh, bh_hat, rh) ; // r = b - A*x
    r_supnorm = norm( rh, INF_NORM ) ;

    cout << "\nsupnorm(r = b - A*x ) = " << r_supnorm << endl ;

    dealloc( Ah ) ;
    int_dealloc( Ph ) ;
    dealloc( bh ) ;
    dealloc( xh ) ;
    dealloc( Ah_dup ) ;
    dealloc( bh_hat ) ;
    dealloc( rh ) ;
}
```

$$b = \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix}$$

$$x = A^{-1}b = \begin{pmatrix} -6.9306 \\ 17.9583 \\ 26.5833 \\ 7.3333 \end{pmatrix}$$

We cannot use ***printf***, why?

# *Test driver [3]*

## global.h

```
//#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

```
#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

```
//#define DO_DOUBLE_DOUBLE
#define DO_QUAD_DOUBLE
//#define DO_ARPREC
```

```
//#define DO_DOUBLE_DOUBLE
//#define DO_QUAD_DOUBLE
#define DO_ARPREC
```

## Execution results

```
supnorm(r = b - A*x ) = 2.8421709430404007e-014
請按任意鍵繼續 . . .
```

```
supnorm(r = b - A*x ) = 1.5777218104420236e-30
請按任意鍵繼續 . . .
```

```
supnorm(r = b - A*x ) = 2.4308653429145085e-63
請按任意鍵繼續 . . .
```

```
supnorm(r = b - A*x ) = 1.0819896817717659e-129
請按任意鍵繼續 . . .
```

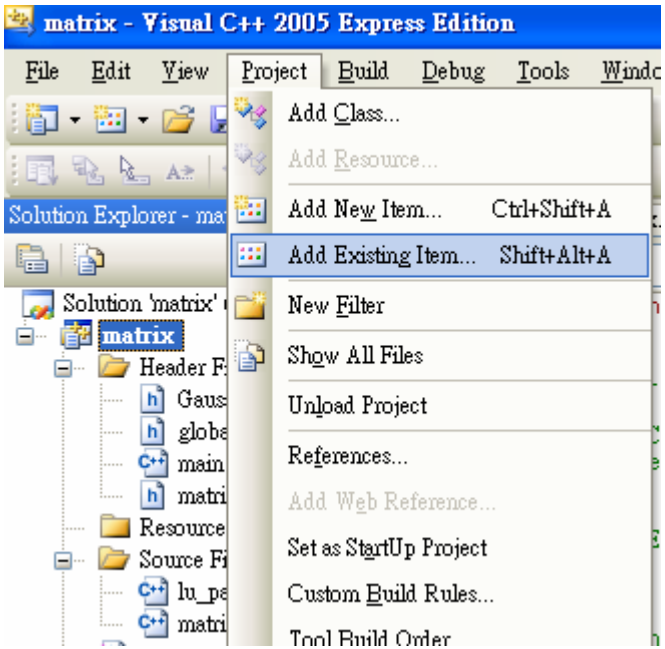
1. Are these four results reasonable?
2. How can we embed high precision package into vc 2005?
3. How can we embed high precision package into Linux machine?

# OutLine

- How to add high precision package into program
- Embed high precision package in vc2005
- Embed high precision package in Linux through qmake

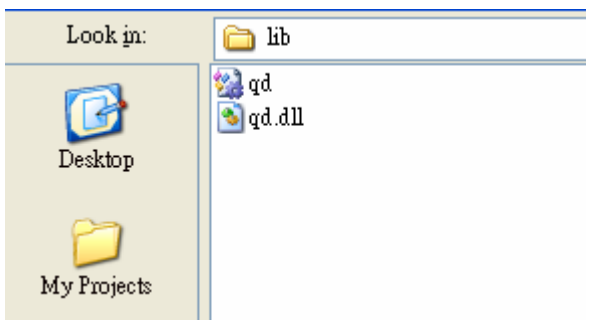
# *Visual Studio 2005: How To add high precision package [1]*

1 Project → Add Existing Item

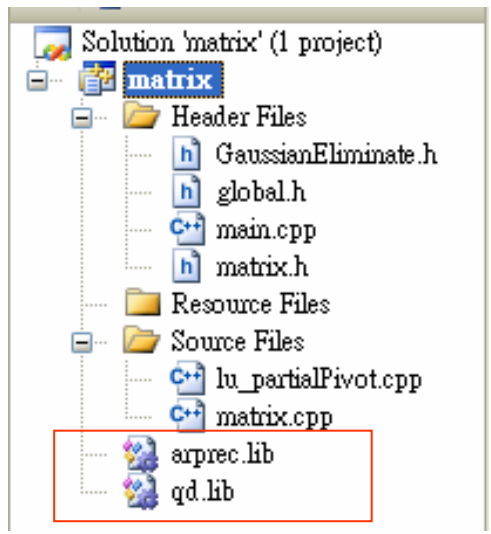



2 Add arprec.lib which is in *C:\arprec-2.2.1-windll\lib*


3 Add qd.lib which is in *C:\qd-2.3.4-windll\lib*

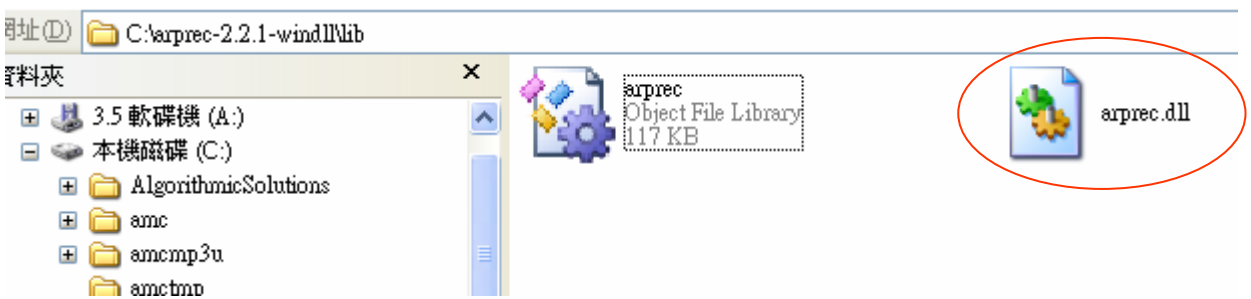


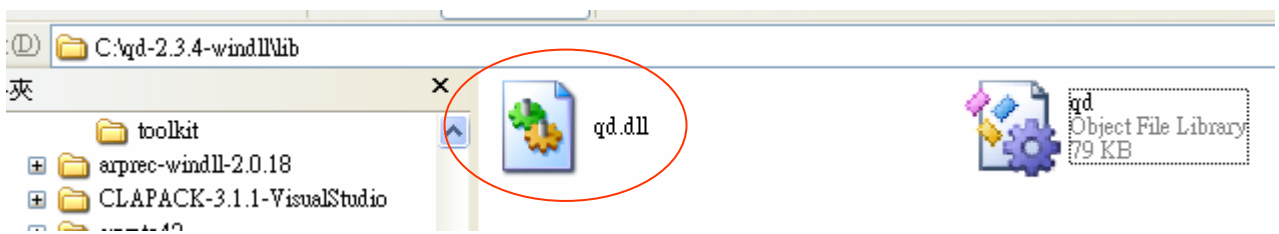

4 arprec.lib and qd.lib appears in project manager



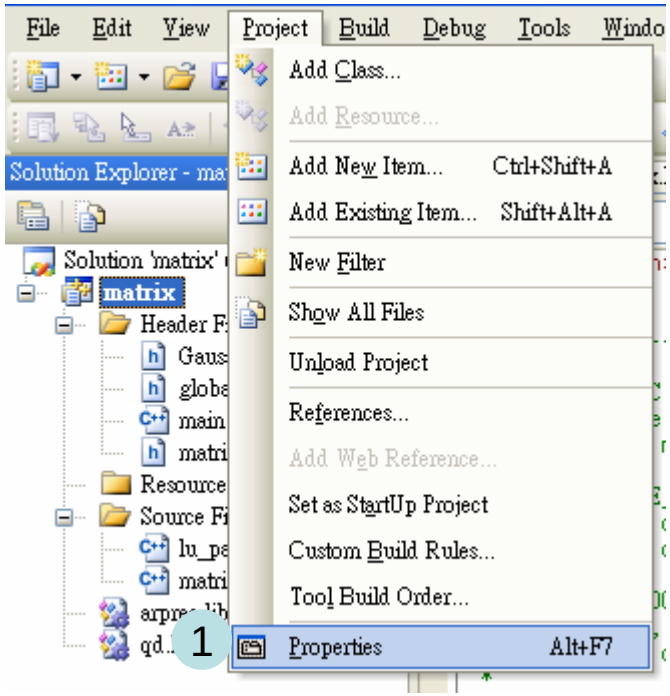
# *Visual Studio 2005: How To add high precision package [2]*

copy **C:\arprec-2.2.1-windll\lib\arprec.dll** and **C:\qd-2.3.4-windll\lib\qd.dll**
to directory *matrix*

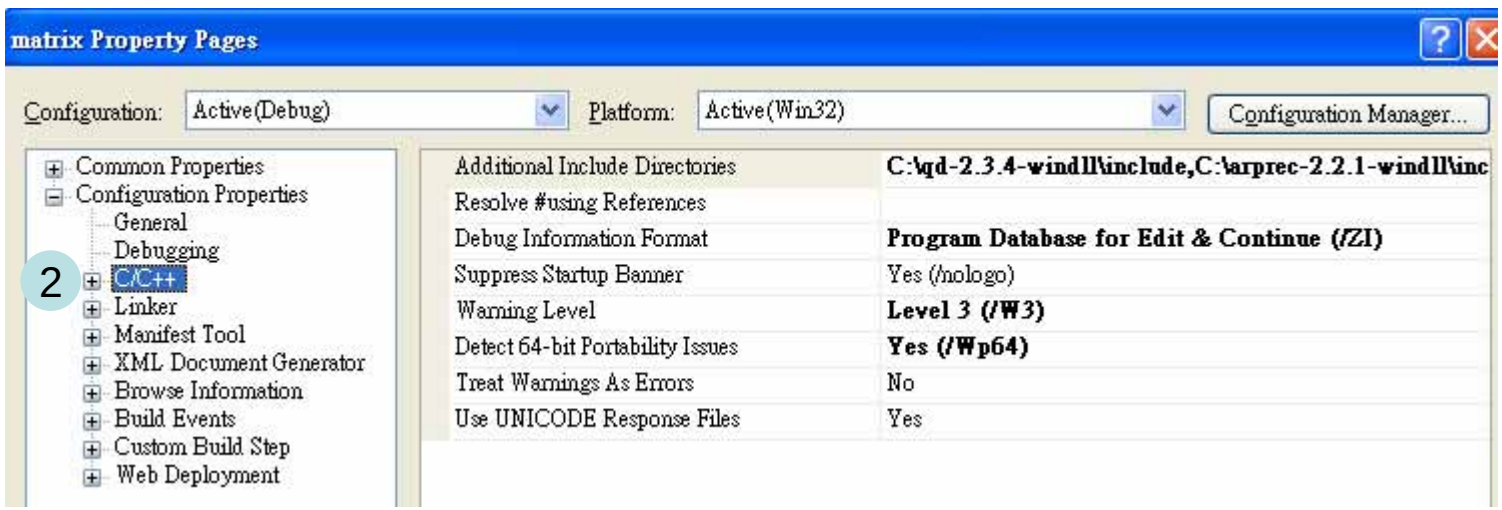
# *Visual Studio 2005: How To add high precision package [3]*



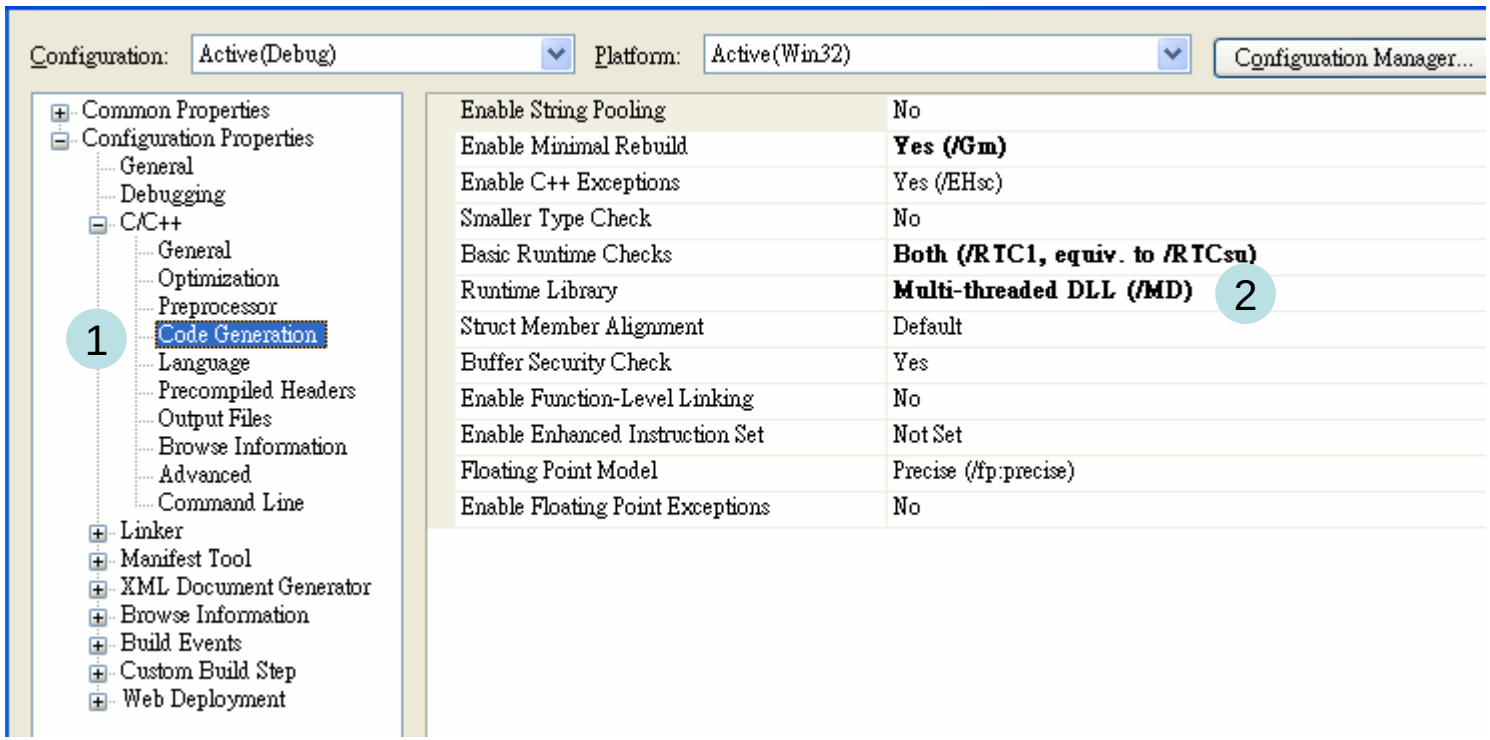

1. Project → properties
2. Choose C/C++ item
3. In field (欄位) **Additional Include Directories**

   add *C:\qd-2.3.4-windll\include* and

   *C:\arprec-2.2.1-windll\include*



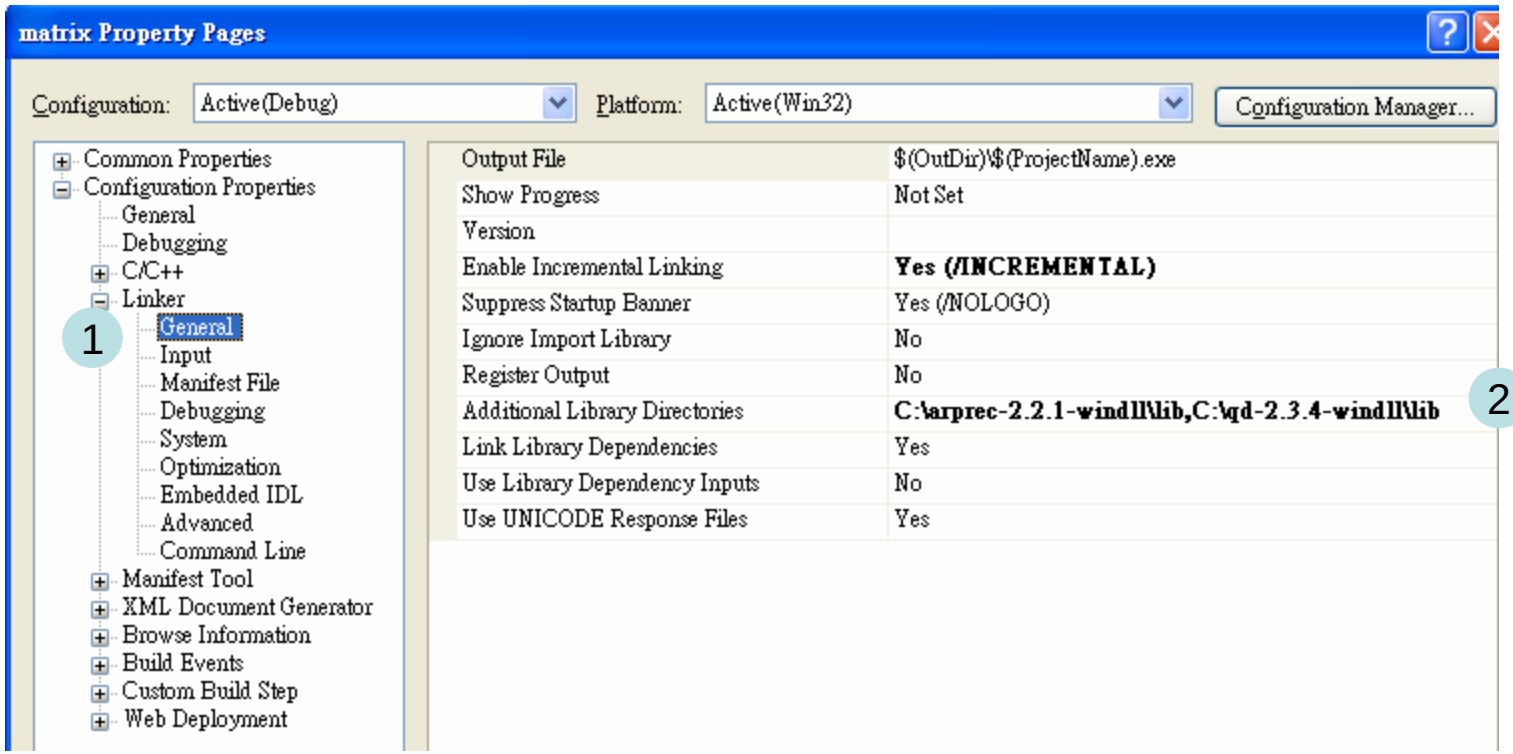
# *Visual Studio 2005: How To add high precision package [4]*




1. Choose C/C++ → Code Generation
2. In field (欄位) **Runtime library,** choose Multi-threaded DLL (/MD)

   多執行緒 DLL (/MD)

# *Visual Studio 2005: How To add high precision package [5]*




1. Choose Liner → General
2. In field **Additional Library Directories,** add *C:\qd-2.3.4-windll\lib* and *C:\arprec-2.2.1-windll\lib*

Now You can compile your code, good luck

# OutLine

- How to add high precision package into program
- Embed high precision package in vc2005
- Embed high precision package in Linux through qmake

# *Linux machine: How To add high precision package [1]*

1. Suppose we put all source file into directory ***matrix*** and upload to workstation.

```
[macrold@quartet2 matrix]$ ls
Debug                lu_partialPivot.cpp  matrix.vcproj
GaussianEliminate.h  main.cpp             matrix.vcproj.MATH-238DAC3744.LungShengChien.user
arprec.dll           matrix.cpp           matrix.vcproj.MATH-719229FB76.lschien.user
global.h             matrix.h             qd.dll
[macrold@quartet2 matrix]$
```

2. Remove all VC related files and DLL files.

```
[macrold@quartet2 matrix]$
[macrold@quartet2 matrix]$ rm matrix.vc*
rm: remove regular file `matrix.vcproj'? y
rm: remove regular file `matrix.vcproj.MATH-238DAC3744.LungShengChien.user'? y
rm: remove regular file `matrix.vcproj.MATH-719229FB76.lschien.user'? y
[macrold@quartet2 matrix]$ rm qd.dll
rm: remove regular file `qd.dll'? y
[macrold@quartet2 matrix]$ rm arprec.dll
rm: remove regular file `arprec.dll'? y
[macrold@quartet2 matrix]$ rm -r -f Debug/
[macrold@quartet2 matrix]$ ls
GaussianEliminate.h  global.h  lu_partialPivot.cpp  main.cpp  matrix.cpp  matrix.h
[macrold@quartet2 matrix]$
```

3. use command “qmake -project” to generate project file

```
[macrold@quartet2 matrix]$ qmake -project
[macrold@quartet2 matrix]$ ls
GaussianEliminate.h  lu_partialPivot.cpp  matrix.cpp  matrix.pro
global.h             main.cpp             matrix.h
[macrold@quartet2 matrix]$
```

# *Linux machine: How To add high precision package [2]*

4 use command “vi matrix.pro” to edit project file

```
#############################################################################
# Automatically generated by qmake (1.06c) Thu Nov 13 22:52:03 2008
#############################################################################

TEMPLATE = app
INCLUDEPATH += .

# Input
HEADERS += GaussianEliminate.h global.h matrix.h
SOURCES += lu_partialPivot.cpp main.cpp matrix.cpp
~
```

```
#############################################################################
# Automatically generated by qmake (1.06c) Thu Nov 13 22:52:03 2008
#############################################################################

TEMPLATE = app
INCLUDEPATH += .

INCLUDEPATH += /opt/arprec/include
INCLUDEPATH += /opt/qd/include

LIBS += -L/opt/arprec/lib -larprec -larprecmod
LIBS += -L/opt/qd/lib -lqd -lqdmod

# Input
HEADERS += GaussianEliminate.h global.h matrix.h
SOURCES += lu_partialPivot.cpp main.cpp matrix.cpp
~
```

Add include file path

Add linking library

***-larprec*** means libarprec.a

***-lqd*** means libqd.a

# *Linux machine: How To add high precision package [3]*

5 use command “qmake matrix.pro” to generate Makefile

```
[macrold@quartet2 matrix]$ qmake matrix.pro
[macrold@quartet2 matrix]$ ls
GaussianEliminate.h  global.h             main.cpp    matrix.h
Makefile             lu_partialPivot.cpp  matrix.cpp  matrix.pro
[macrold@quartet2 matrix]$
```

6 use command “make” to compile your codes and generate executable file

```
[macrold@quartet2 matrix]$
[macrold@quartet2 matrix]$ make
icpc -c -pipe -w -O2 -mp  -DQT_NO_DEBUG -DQT_SHARED -DQT_THREAD_SUPPORT -I/opt/qt/mkspecs/def
ault -I. -I. -I/opt/arprec-2.2.2/include -I/opt/qd-2.3.7/include -I/opt/qt/include -o lu_part
ialPivot.o lu_partialPivot.cpp
icpc -c -pipe -w -O2 -mp  -DQT_NO_DEBUG -DQT_SHARED -DQT_THREAD_SUPPORT -I/opt/qt/mkspecs/def
ault -I. -I. -I/opt/arprec-2.2.2/include -I/opt/qd-2.3.7/include -I/opt/qt/include -o main.o
main.cpp
icpc -c -pipe -w -O2 -mp  -DQT_NO_DEBUG -DQT_SHARED -DQT_THREAD_SUPPORT -I/opt/qt/mkspecs/def
ault -I. -I. -I/opt/arprec-2.2.2/include -I/opt/qd-2.3.7/include -I/opt/qt/include -o matrix.
o matrix.cpp
matrix.cpp(80): (col. 3) remark: LOOP WAS VECTORIZED.
matrix.cpp(42): (col. 3) remark: LOOP WAS VECTORIZED.
icpc -Wl,-rpath,/opt/qt/lib -o matrix lu_partialPivot.o main.o matrix.o   -L/opt/qt/lib -L/us
r/X11R6/lib -L/opt/arprec/lib -larprec -larprecmod -L/opt/qd/lib -lqd -lqdmod -lqt-mt -lXext
-lX11 -lm
[macrold@quartet2 matrix]$ ls
GaussianEliminate.h  global.h             lu_partialPivot.o  main.o  matrix.cpp  matrix.o
Makefile             lu_partialPivot.cpp  main.cpp           matrix  matrix.h    matrix.pro
[macrold@quartet2 matrix]$
```

7 Execute executable file by “./matrix”

# Exercise

- Given global.h, modify your matrix.cpp (constructor, destructor and I/O)
- Check if you need to modify subroutine *PA = LU or not*
- Check if you need to modify forward and backward or not.
- How can you use high precision package to test your linear solver?