Chapter 15 C-implementation

PA = LU

Speaker: Lung-Sheng Chien

OutLine

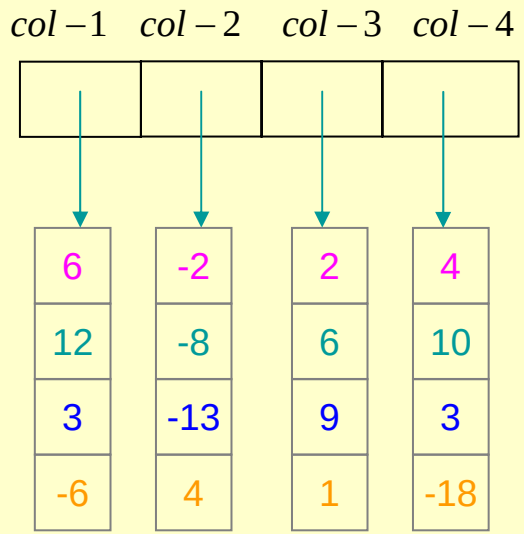
- Data structure of full matrix
- Implementation of *PA=LU*
- Visual Studio 2005: How To
- Linux machine: How to compile

row-major versus col-major

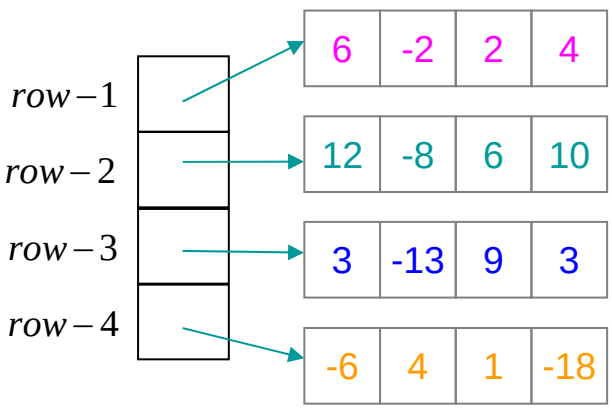
Logical index : 2D

6	-2	2	4
12	-8	6	10
3	-13	9	3
-6	4	1	-18

col-major based



row-major based



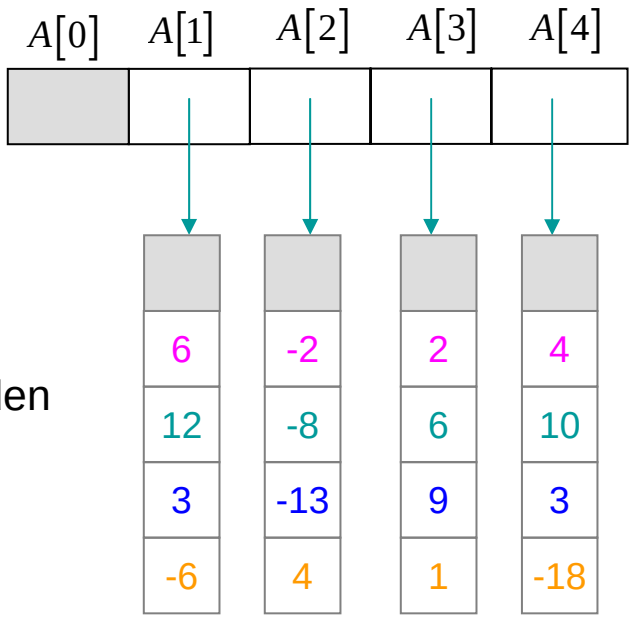
We choose col-major representation

col-major: C starts index from 0

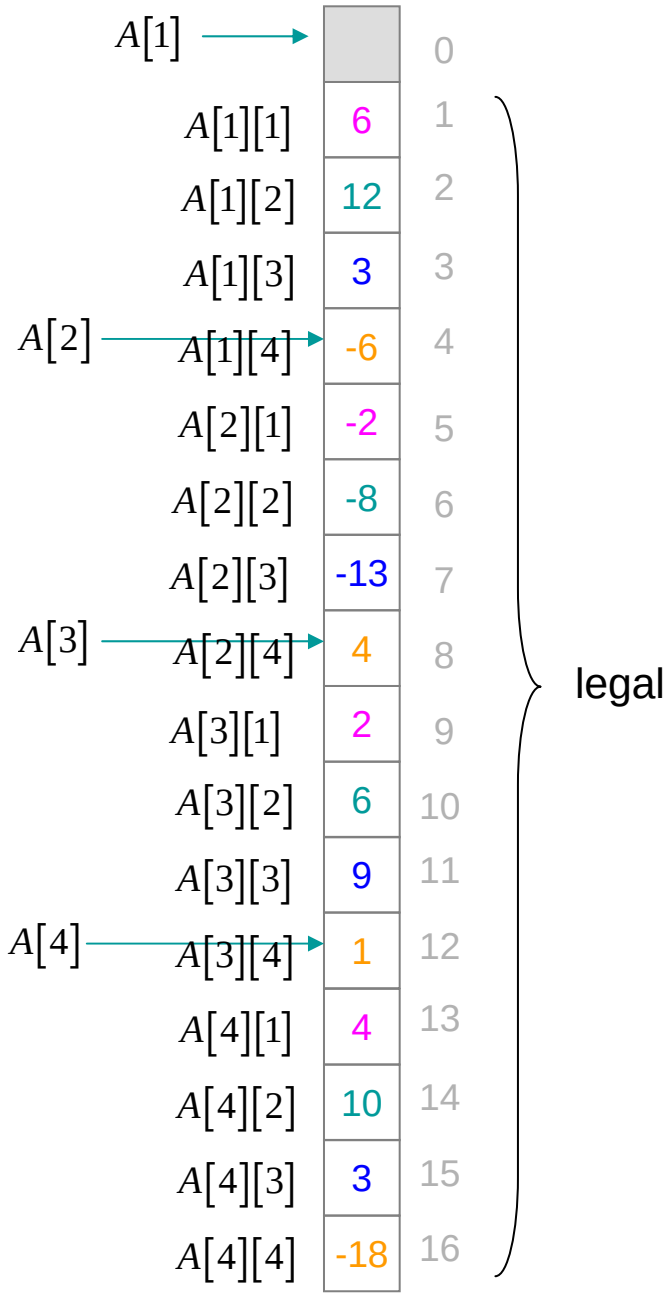
matrix.h

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

typedef  matrix*   matrixHandler ;
```

col-major based



1. $A[0]$ is useless
2. $A[k][0]$ is forbidden
3. $A[j][i] \equiv A(i,j)$



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.

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.

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

matrix.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>

#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
1   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) ) ;
2       assert( A ) ;
        size = m*n ;
3       memA = (doublereal*) malloc( sizeof(doublereal)*size ) ;
        assert( memA ) ;
        for ( j=0 ; j < size ; j++){
            memA[j] = 0 ;  // reset matrix A as zero matrix
        }
4       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
5   Ah->m = m  ;  Ah->n = n ; Ah->sel = sel ; Ah->A = A ;

    return Ah ;
}
```

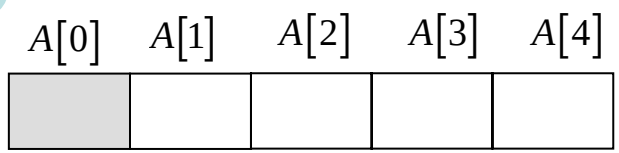
constructor [1]

1 empty handler

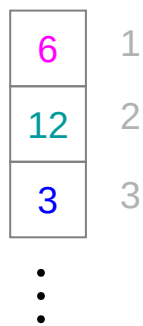
```
typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ;
    doublereal **A ;
} matrix ;
```

5 set parameter

2



3 contiguous memory block



matrix.cpp

constructor [2]

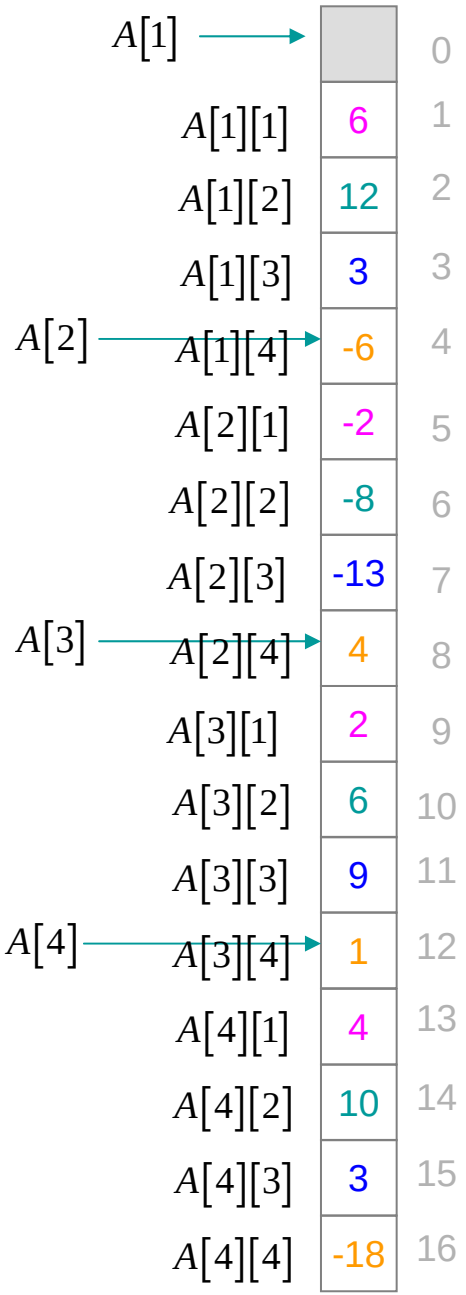
```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>

#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 ;
        memA = (doublereal*) malloc( sizeof(doublereal)*size ) ;
        assert( memA ) ;
        for ( j=0 ; j < size ; j++){
            memA[j] = 0 ;  // reset matrix A as zero matrix
        }
    4   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 ;
}
```

4

low

high

pointer arithmetic

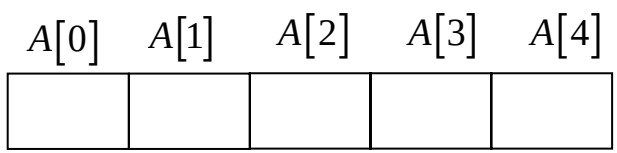


destructor

matrix.cpp

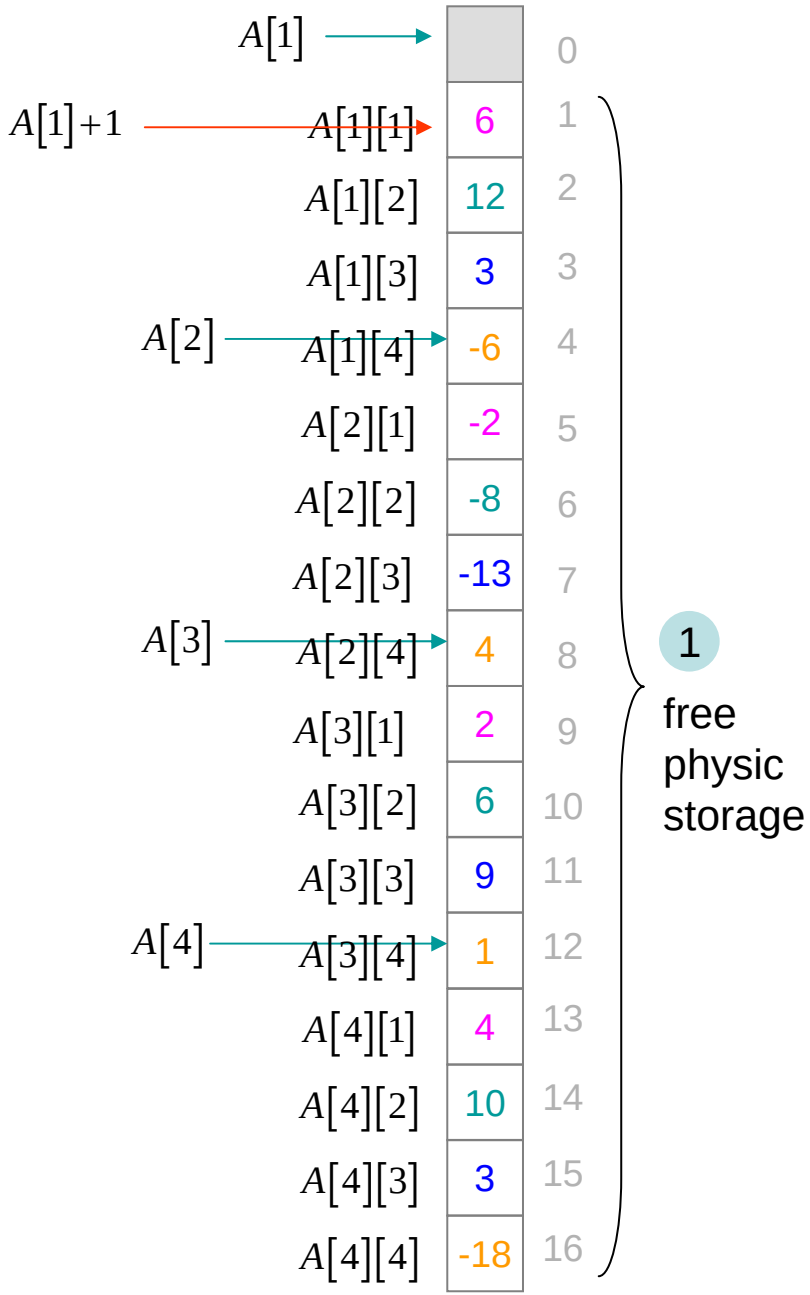
```
// The procedure holds for row-major or col-major
void dealloc( matrixHandler Ah )
{
// step 1: dealloc contiguous memory block
1  free( (doublereal*)(Ah->A[1]) + 1 ) ;

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

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

2 free pointer array



3 free matrix handler

```
typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ;
    doublereal **A ;
} matrix ;
```



Input/Output

matrix.cpp

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

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

    fprintf(fp, "dimension of matrix is (%d,%d) with col-major\n",
           Ah->m, Ah->n );

    A = Ah->A ;
    for( i=1 ; i <= Ah->m ; i++ ){
        for( j=1 ; j <= Ah->n ; j++ ){

            fprintf(fp, "%8.4f  ",  A[j][i]  );
        }
        fprintf(fp, "\n");
    }
}
```

```
typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ;
    doublereal **A ;
} matrix ;
```

$$A[j][i] \equiv A(i, j)$$

6	-2	2	4
12	-8	6	10
3	-13	9	3
-6	4	1	-18

→ Show in standard form: human readable

Simple driver

main.cpp

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

void  test_matrix( void ) ;

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

    return 0 ;
}
```

```
void  test_matrix( void )
{
    integer m = 4 ;
    integer n = 4 ;
    matrixHandler Ah ;
    doublereal **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. ;

    disp( Ah, stdout ) ;
    dealloc( Ah ) ;
}
```

Execution result

```
dimension of matrix is (4,4) with col-major
  6.0000   -2.0000    2.0000    4.0000
 12.0000   -8.0000    6.0000   10.0000
  3.0000  -13.0000    9.0000    3.0000
 -6.0000    4.0000    1.0000  -18.0000
Press any key to continue_
```

$$A[j][i] \equiv A(i,j)$$

duplicate of a matrix / vector

matrix.cpp

```
void duplicate( matrixHandler Ah, matrixHandler Ah_shadow )
{
    integer  m, n ;
    integer  i, j ;
    doublereal **A ;
    doublereal **A_shadow ;

    assert( Ah ) ;  assert( Ah_shadow ) ;
    assert( COL_MAJOR == Ah->sel ) ;
    assert( COL_MAJOR == Ah_shadow->sel ) ;

    m = Ah->m ; n = Ah->n ;
// check Ah_shadow has the same configuration as Ah
    assert( m == Ah_shadow->m ) ;
    assert( n == Ah_shadow->n ) ;

    A = Ah->A ;
    A_shadow = Ah_shadow->A ;
    for ( j = 1 ; j <= n ; j++){
        for ( i = 1 ; i <= m ; i++ ){
            A_shadow[j][i] = A[j][i] ;
        }// for each row
    }// for each col
}
```

verification

$$A[j][i] \equiv A(i,j)$$

```
typedef struct matrix
{
    integer  m ;
    integer  n ;
    orderVar   sel ;
    doublereal **A ;
} matrix ;
```

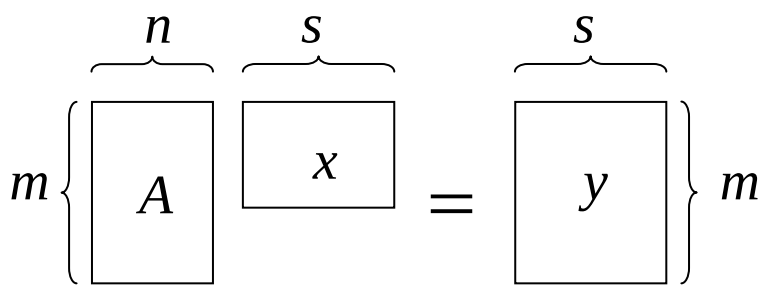
We require matrix handler to verify configuration, this is very important for debugging.

$y = Ax$

matrix.cpp

```
// 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)
}
```

$$A \in R^{m \times n}, x \in R^{n \times s}, y \in R^{m \times s}$$



Outer-product formulation

$$y = x(1)\,A(:,1) + \sum_{j=2}^{n} x(j)\,A(:,j)$$

Matrix / vector norm

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 ;
}
```

$$\|A\|_1 = \sup_{x\neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{1\le j\le i} \sum_{i=1}^{n} |a_{ij}|$$

$=$ Maximum of absolute column sum

$$\|A\|_\infty = \sup_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{1\le i\le i} \sum_{j=1}^{n} |a_{ij}|$$

$=$ Maximum of absolute row sum

$$\|A\|_2 = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\rho\left(A^T A\right)}$$

2-norm is difficult to compute, we will discuss this later

OutLine

- Data structure of full matrix
- Implementation of *PA=LU*
- Visual Studio 2005: How To
- Linux machine: How to compile

Algorithm (PA = LU) [1]

Given full matrix $A \in R^{n \times n}$, ~~construct initial lower triangle matrix $L = I$~~

use permutation vector P to record permutation matrix $P^{(k)}$

verification

```
int  lu_partialPivot( matrixHandler Ah, int_matrixHandler Ph, int conti_flag )
{
    integer  m, n, LU_dim ;
    integer  i, j, k ;
    doublereal **A ;
    integer    **P ;
    integer  p ;
    doublereal ksi ; // ksi = max(|A(k:m,k)|)
    doublereal tmp ; // temperary real variable
    integer    int_tmp ; // temperary integer variable
    int isLU_succ = 0 ; // isLU_succ = 0, LU is done
                        // = k < m, means (PA)(k:m,k) = 0, A is singular

    assert( Ah ) ;  assert( Ph ) ;
    assert( COL_MAJOR == Ah->sel ) ;
    assert( COL_MAJOR == Ph->sel ) ;

    m = Ah->m ; n = Ah->n ;

    assert( 1 == Ph->n ) ;
    assert( m <= Ph->m ) ;
```

(1) we store lower triangle matrix *L* into storage of matrix *A*

(2) The reason to construct type *int_matrix* is for permutation matrix *P*,

since we can check dimension between *A* and *P*.

Algorithm (PA = LU) [2]

let $A^{(1)} := A$, $\tilde{L}^{(-1)} = L^{(0)} = I$ and $P^{(0)} = (1, 2, 3, \cdots, n)$

```
A = Ah->A ; P = Ph->A ;
for ( i = 1 ; i <= m ; i++){
    P[1][i] = i ;
}
```

```
P = 1:m ;
```

Note that *P* is a column vector

for $k = 1 : n-1$

we have compute $P^{(k-1)}A = \tilde{L}^{(k-2)}L^{(k-1)}A^{(k)}$, $P^{(k-1)} = P_{k-1}P_{k-2}\cdots P_2P_1$

$$A^{(k)} = \left(\begin{array}{ccc|cc} a_{11}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ 0 & \ddots & & & \\ \vdots & & a_{k-1,k-1}^{(k-1)} & \cdots & a_{k-1,n}^{(k-1)} \\ \hline \vdots & & 0 & a_{k,k}^{(k)} & a_{k,n}^{(k)} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{nk}^{(k)} & a_{nn}^{(k)} \end{array}\right)$$

update original matrix A

$$\tilde{L}^{(k-2)}L^{(k-1)} = \left(\begin{array}{c|c} \overbrace{W}^{k-1} & O \\ \hline M & I \end{array}\right) \Big\}k-1$$

stores in lower triangle matrix L

Algorithm (PA = LU) [3]

1 find a $p \in \{k, k+1, \cdots, n\}$ such that $\left|A_{pk}^{(k)}\right| = \max \left|A^{(k)}(k:n, k)\right|$

```
/*
 *  step 1: find a p belongs to {k,k+1,...,m} such that
 *       ksi = |A_{k}(p,k)| = max |A_{k}(k:m,k)|
 */
       ksi = 0.0 ;
       for ( i=k ; i <= m ; i++ ){
           tmp = fabs(A[k][i]) ;
           if ( tmp > ksi ){
               ksi = tmp ;
               p = i ;
           }
       }// for each row i
```

```
62 %
63 % step 1: find a p belongs to (k,k+1,...,m)
64 %   such that |A_(k)(pk)| = max |A_(k)(k:m,k)|
65 %
66   [y, rel_p ] = max( abs(A(k:m,k)) ) ;
67   p = rel_p + k - 1 ;
```

2 swap row $A^{(k)}(k, k:n)$ and row $A^{(k)}(p, k:n)$, NOT efficient

```
   for ( j=k ; j <= n ; j++ ){
       tmp     = A[j][k] ;
       A[j][k] = A[j][p] ;
       A[j][p] = tmp ;
   }
```

```
tmp_row(1:n-k+1) = A(k,k:n) ;
A(k,k:n) = A(p,k:n) ;
A(p,k:n) = tmp_row(1:n-k+1) ;
```

define permutation matrix $P_k = (1, 2, \cdots k-1, p, k+1, \cdots, p-1, k, p+1, \cdots, n)$

then after swapping rows k, p , we have $\tilde{A}^{(k)} = P_k A^{(k)}$

Algorithm (PA = LU) [4]

$$\tilde{A}^{(k)} = \begin{pmatrix} \times & \times & \times \\ 0 & \xi & d^T \\ 0 & c & B \end{pmatrix} \leftarrow k-row \qquad \left(\ |\xi| \geq \max|c| \ \right)$$

$k-col$

3 compute $P^{(k)} \leftarrow P_k P^{(k-1)}$ for $P^{(0)} = (1,2,3,\cdots,n)$

```
/*
 *  step 3: compute P_{k} = p_k * P_{k-1}
 *      by swapping P_{k-1}(k) and P_{k-1}(p)
 */
        int_tmp = P[1][k] ;
        P[1][k] = P[1][p] ;
        P[1][p] = int_tmp ;
```

```
86 %
87 % step 3: compute P_{k} = p_k * P_{k-1}
88 %  by swapping P_{k-1}(k) and P_{k-1}(p)
89 %
90   temp = P(k) ;
91   P(k) = P(p) ;
92   P(p) = temp ;
```

if $k > 1$ *then*

4 compute $\tilde{L}^{(k-1)} \leftarrow P_k \left(\tilde{L}^{(k-2)} L^{(k-1)} \right) P_k^T$ by swapping $L(k, 1:k-1)$ and $L(p, 1:k-1)$

then $P^{(k-1)} A = \tilde{L}^{(k-2)} L^{(k-1)} A^{(k)} \longrightarrow P^{(k)} A = \tilde{L}^{(k-1)} \tilde{A}^{(k)}$

endif

Algorithm (PA = LU) [5]

```
if (1 < k ){
    for ( j=1 ; j < k ; j++ ){
        tmp     = A[j][k] ;
        A[j][k] = A[j][p] ;
        A[j][p] = tmp ;
    }
}// for row != 1
```

```
103   if (1 < k )
104     tmp_row(1:k-1) = L(k,1:k-1) ;
105     L(k,1:k-1) = L(p,1:k-1) ;
106     L(p,1:k-1) = tmp_row(1:k-1) ;
107   end
```

we store lower triangle matrix into storage of matrix *A*, this code is different from code in MATLAB.

5 decompose $\tilde{A}^{(k)} = L^{(k)} A^{(k+1)}$

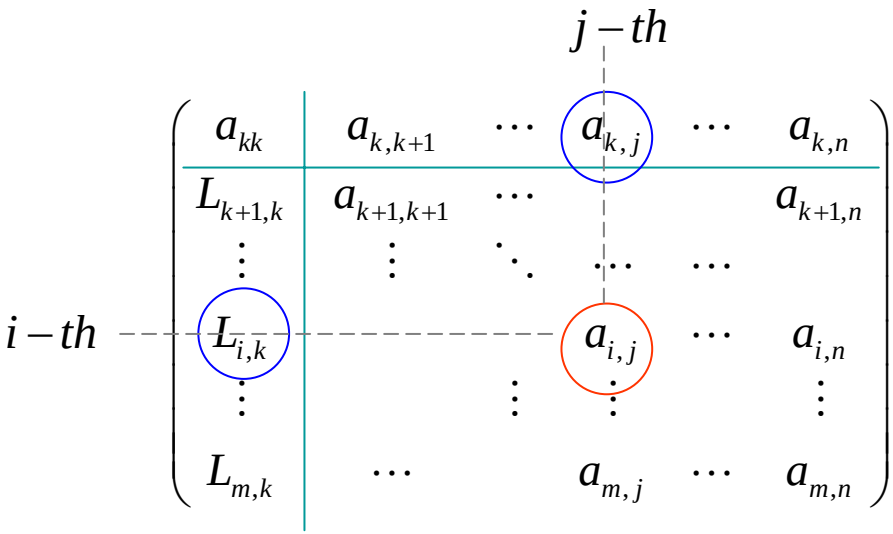
```
// update L, L(k+1:m, k) = A(k+1:m,k)/A(k,k),
// still store into L
          for ( i = k+1 ; i <= m ; i++ ){
              A[k][i] /= ksi ;
          }
// update A(k+1:m,k+1:n) =
//        A(k+1:m,k+1:n) - L(k+1:m,k) * A(k,k+1:n)
          for ( j = k+1 ; j <= n ; j++ ){
              tmp = A[j][k] ;
              for ( i = k+1 ; i <= m ; i++){
                  A[j][i] -= A[k][i] * tmp ;
              }// for each row i
          }// for each col j
```

```
L(k+1:m, k   ) = A(k+1:m,k)/A(k,k) ;
A(k+1:m, k   ) = zero_col(m-k,1) ;
A(k+1:m,k+1:n) = A(k+1:m,k+1:n) - L(k+1:m,k) * A(k,k+1:n) ;
```

$$A^{(k+1)} = \begin{pmatrix} \times & \times & \times \\ 0 & \xi & d^T \\ 0 & 0 & \tilde{B} \end{pmatrix}, \quad \tilde{B} = B - \frac{cd^T}{\xi}$$

How to compute $\tilde{B}$ efficiently?

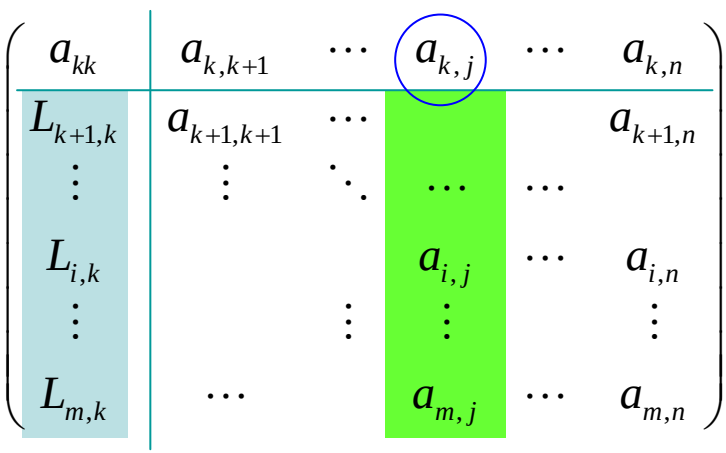
Algorithm (PA = LU) [6]

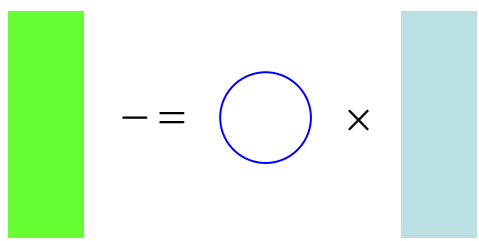


component-wise update : $a_{i,j} = a_{i,j} - L_{i,k} a_{k,j}$

Observation: we implement with column-major form, column-wise update is good

$$A(k+1:m, j) -= A_{k,j} \times L(k+1:m, k)$$





Algorithm (PA = LU) *[7]*

where

$$L^{(k)} = \left(\begin{array}{c|c|c} I & & \\ \hline 0 & 1 & \\ \hline 0 & c/\xi & I \end{array}\right) \longleftarrow k-row$$

(the partition column marked $k-col$ is the column containing 1 and c/ξ)

by updating $L(k+1:n,k) \leftarrow c/\xi$

$$A^{(k+1)} = \left(\begin{array}{c|c|c} \times & \times & \times \\ \hline 0 & \xi & d^T \\ \hline 0 & 0 & \tilde{B} \end{array}\right), \quad \tilde{B} = B - \frac{cd^T}{\xi}$$

by updating matrix $A(k+1:n,k+1:n) -= \dfrac{cd^T}{\xi}$

then $P^{(k-1)}A = \tilde{L}^{(k-2)}L^{(k-1)}A^{(k)} \longrightarrow P^{(k)}A = \tilde{L}^{(k-1)}L^{(k)}A^{(k+1)}$ (recursion is done)

endfor

Algorithm (PA = LU) [8]

$A =$

6	-2	2	4
12	-8	6	10
3	-13	9	3
-6	4	1	-18

$P = (2, 3, 1, 4)$

L

1			
0.25	1		
-0.5	0	1	
0.5	-2/11	1/11	1

U

12	-8	6	10
	-11	7.5	0.5
		4	-13
			3/11

Store *L* and *U* into original *A*

```
PA = LU : store L and U into A:
dimension of matrix is (4,4) with col-major
 12.0000   -8.0000    6.0000   10.0000
  0.2500  -11.0000    7.5000    0.5000
 -0.5000    0.0000    4.0000  -13.0000
  0.5000   -0.1818    0.0909    0.2727

permutation matrix P:
dimension of matrix is (4,1) with col-major
       2
       3
       4
       1
```

Question 1: how to write down a test driver

Question 2: how to do backward and forward

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 ;

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

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

$$A[j][i] \equiv A(i, j)$$

Remember to duplicate matrix *A* since *A* is modified when execute ***LU*** decomposition.

Test driver [2]

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

// 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, stdout ) ;

    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, stdout ) ;
// 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 ) ;
        printf("\nsupnorm(r = b - A*x ) = %6.2E\n", r_supnorm);

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

Linear solver: do backward and forward

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

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

Exercise

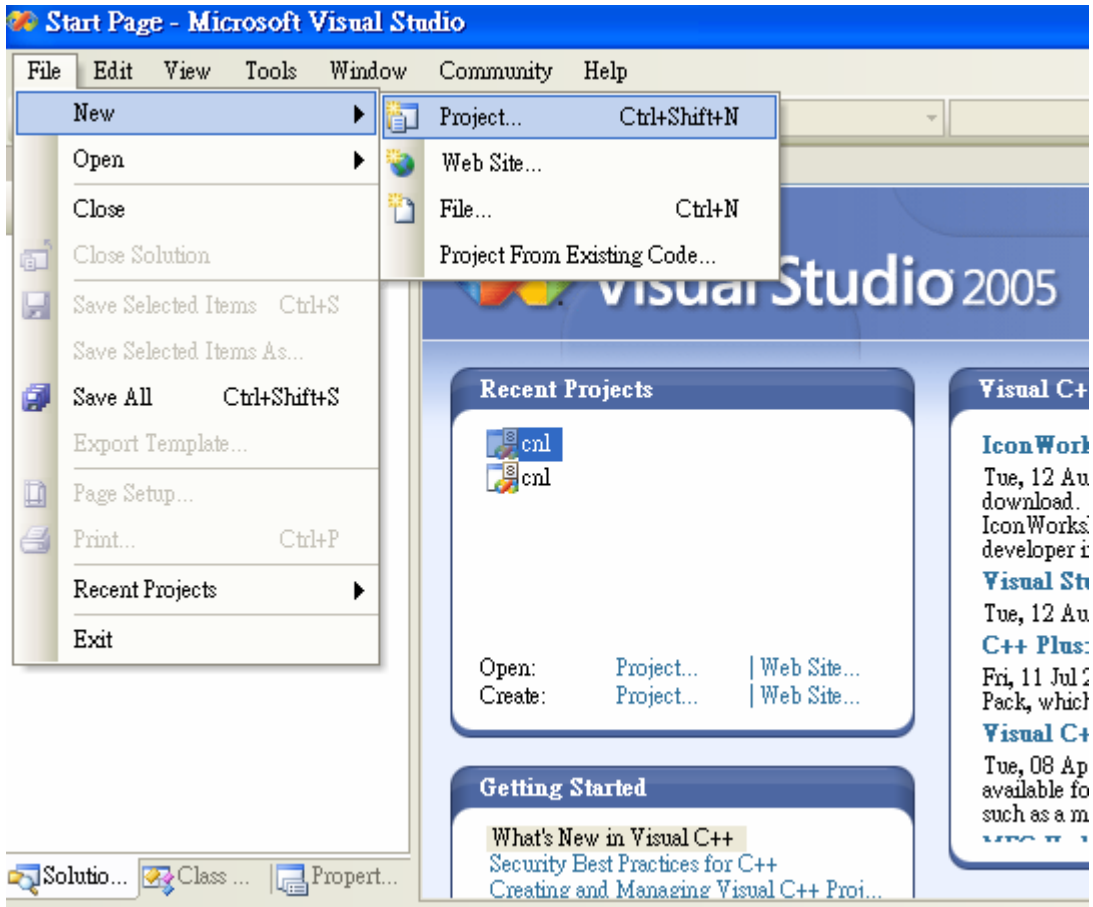
- Implement $PA = LU$.
- Implement forward and backward.
- Write a driver to test your linear solver.
- Any other implementation for full matrix?

OutLine

- Data structure of full matrix
- Implementation of *PA=LU*
- Visual Studio 2005: How To
- Linux machine: How to compile

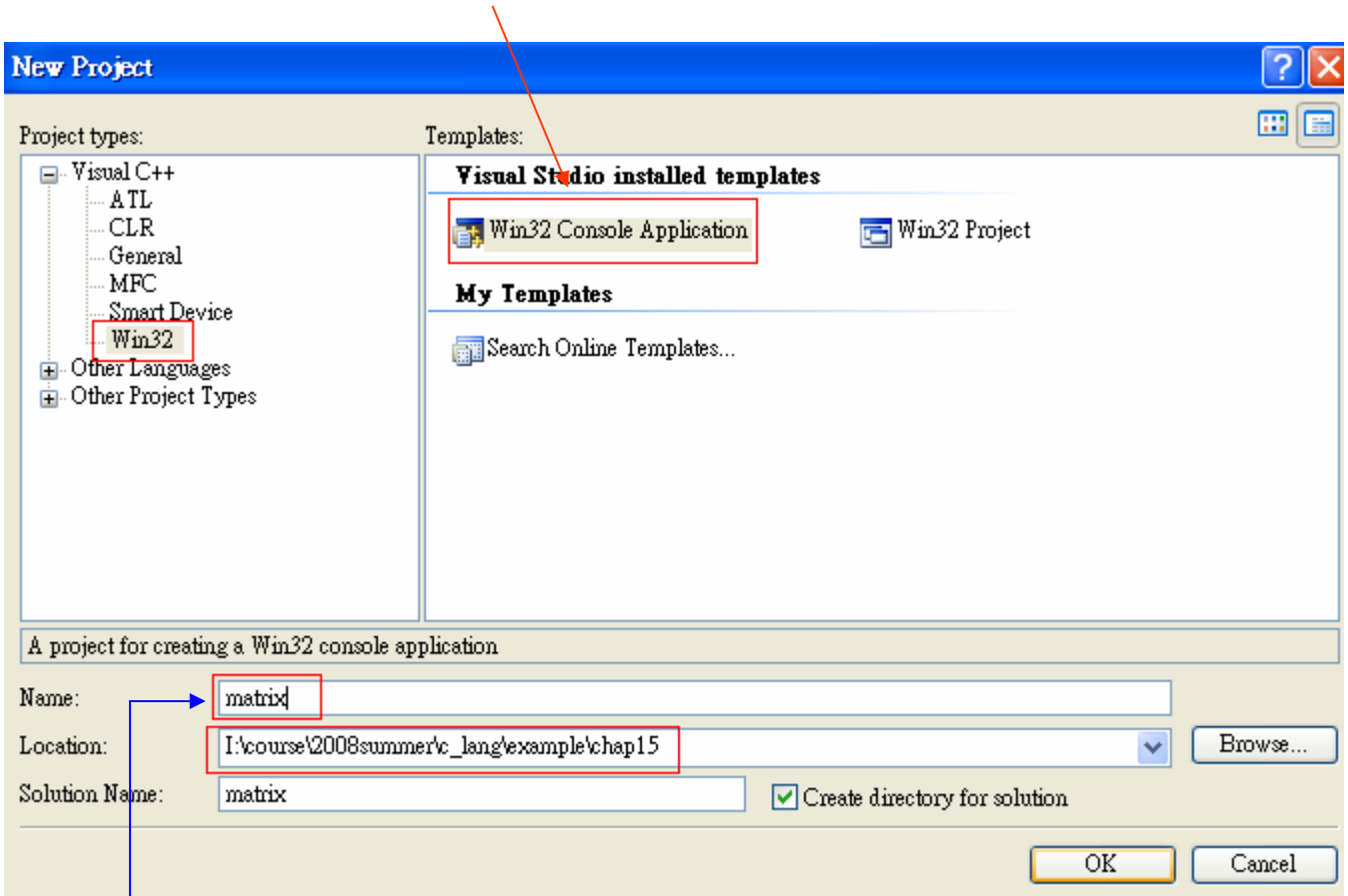
Visual Studio 2005: How To [1]

Create a new project: File → Project



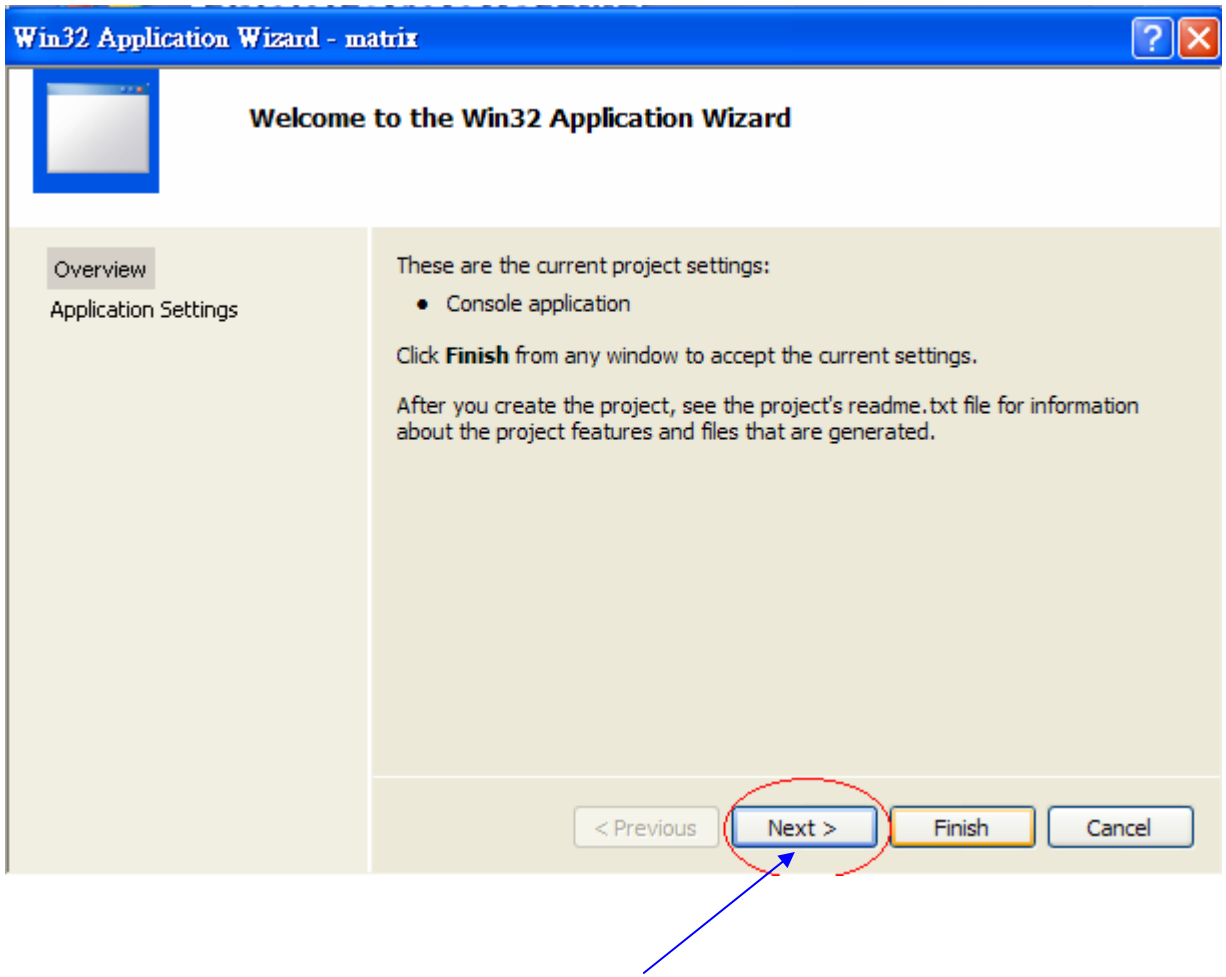
Visual Studio 2005: How To [2]

Choose Win32 console application, this is the same as what we do in VC 6.0



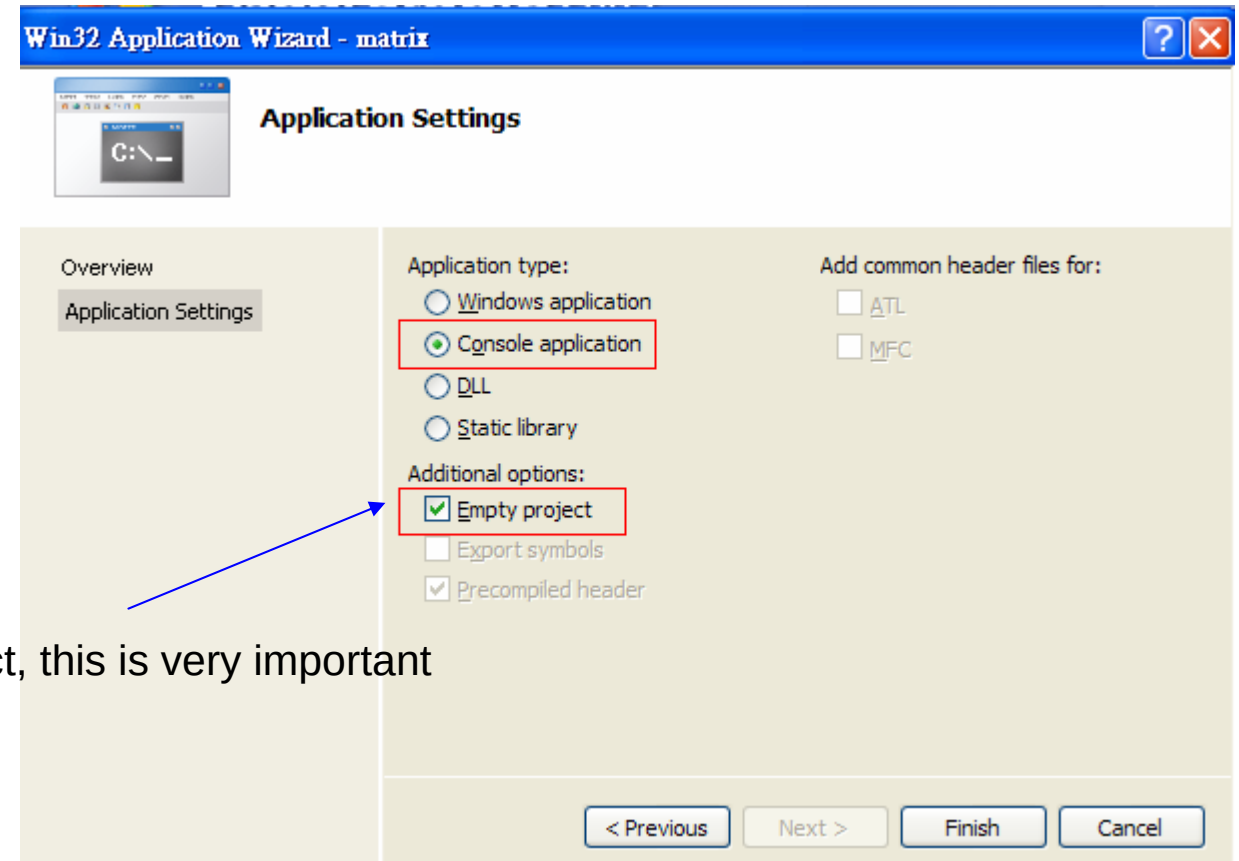
Project name: vc 2005 will create a directory, the same name as project name

Visual Studio 2005: How To [3]



We choose “next” bottom to set an ***empty*** project, DO NOT press Finish bottom.

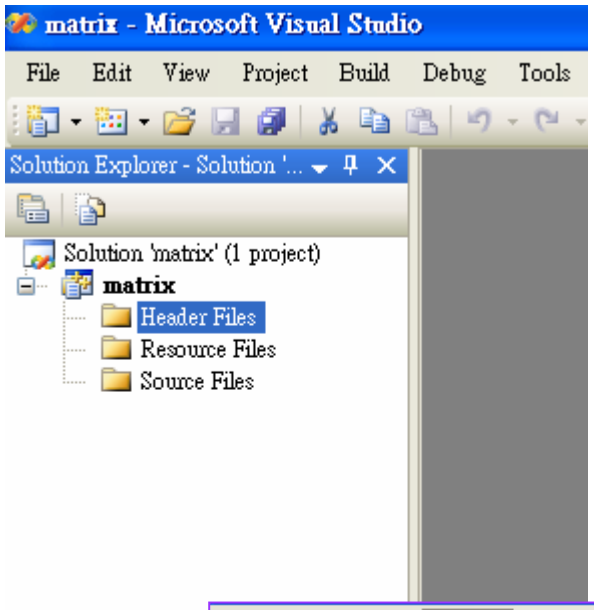
Visual Studio 2005: How To [4]



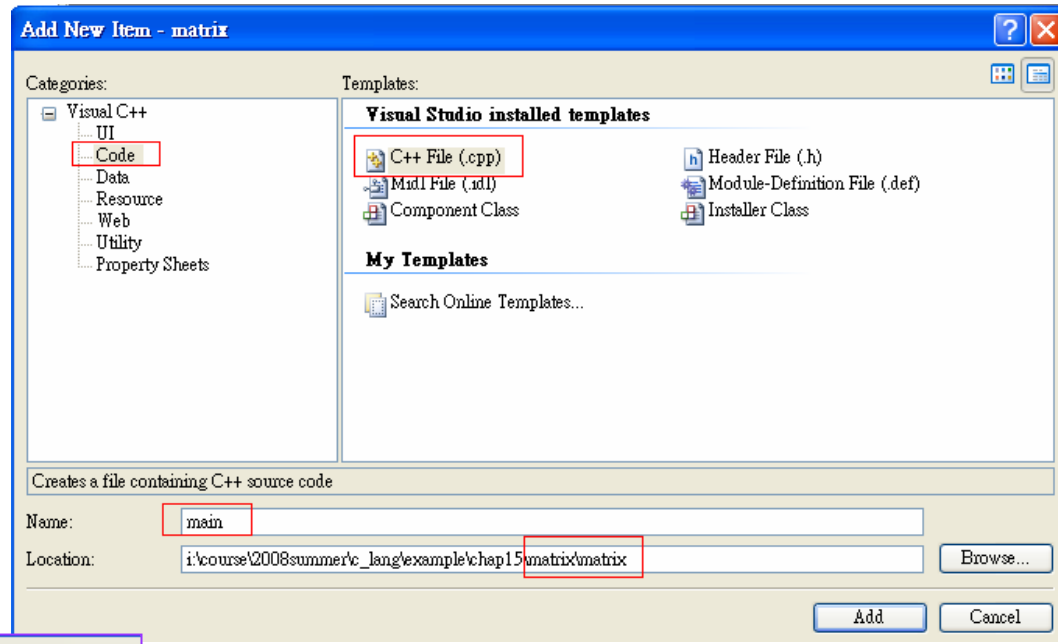
Empty project, this is very important

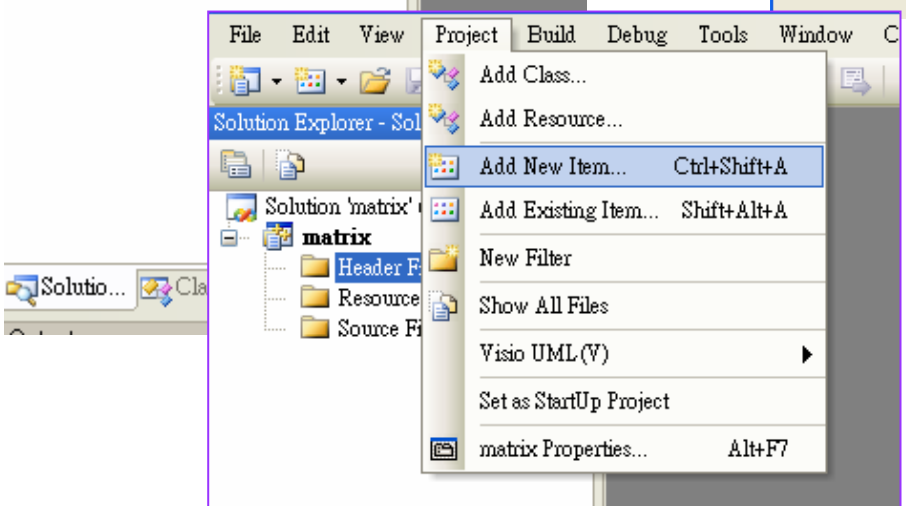
Visual Studio 2005: How To [5]

1 Empty project



3 Add “main.cpp” to this project

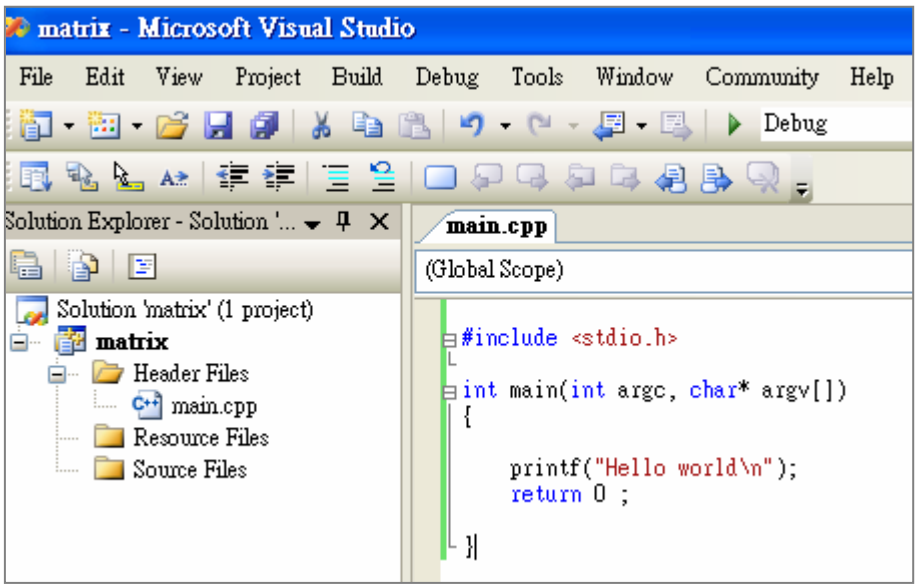




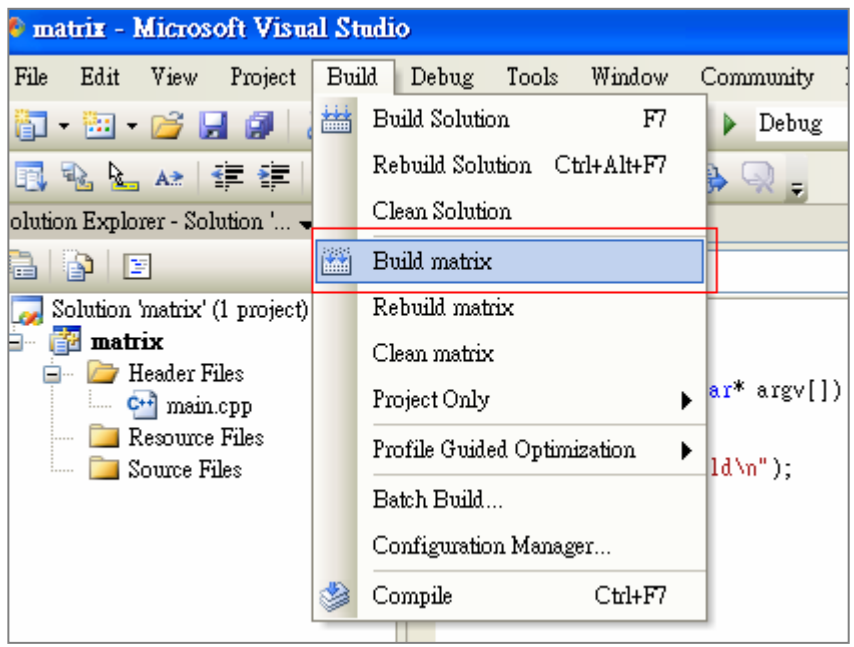
2 Add new item (source file or header file) to this project

Visual Studio 2005: How To [6]

Write source code “hello world”

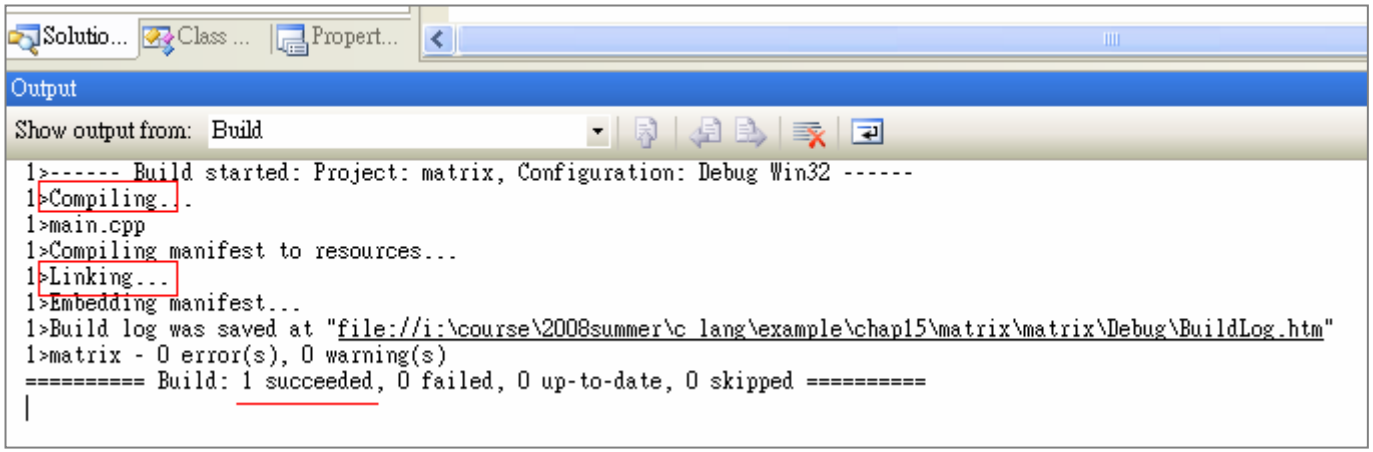


Compile: Build → Build matrix

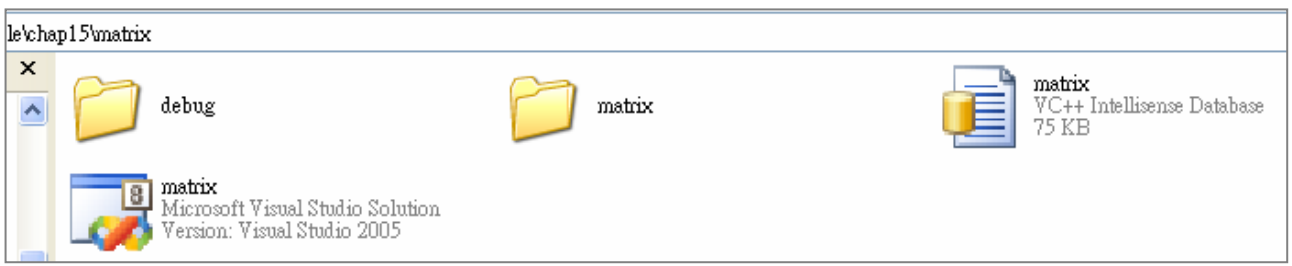


Visual Studio 2005: How To [7]

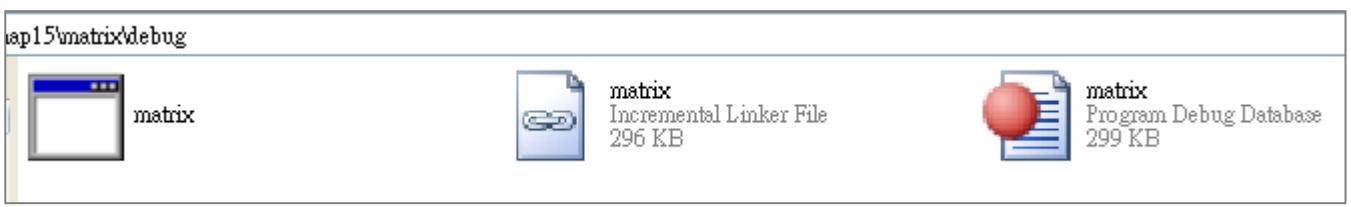
Result of compilation, including linking: success



Additional directory “debug” appears

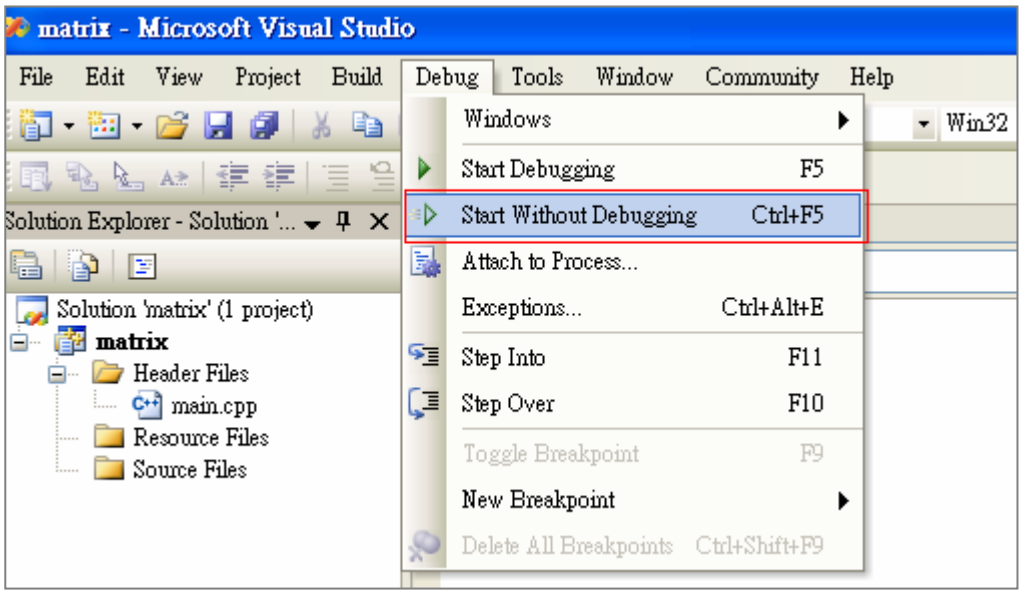


Executable file “matrix.exe” in this debug directory

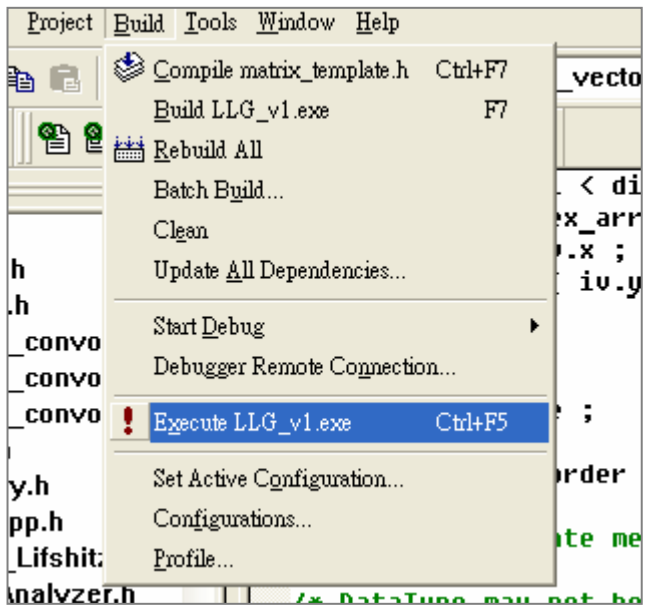


Visual Studio 2005: How To [8]

Execute matrix.exe: Debug → Start Without Debugging



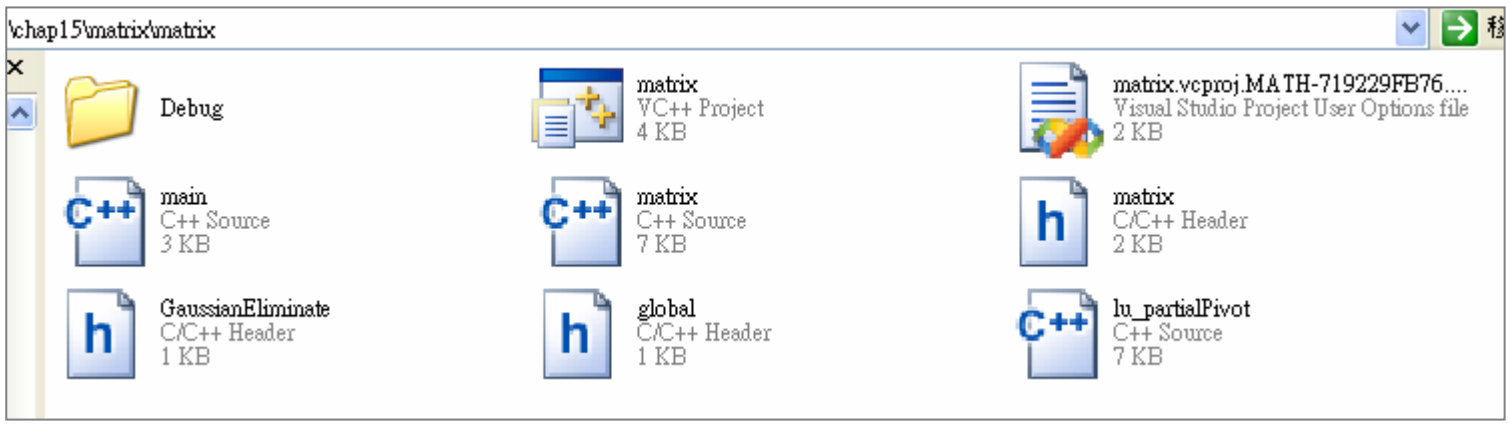
Execution in VC 6.0



Visual Studio 2005: How To [9]

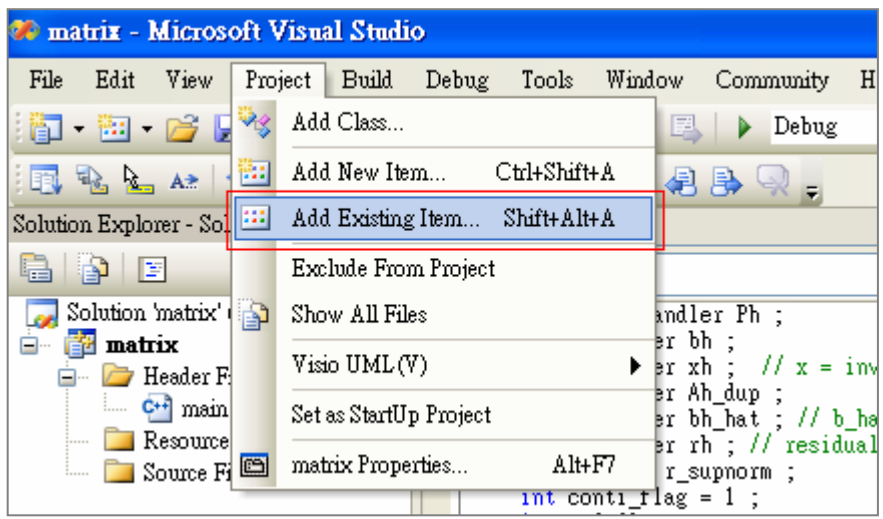
Suppose we want to add several source files into this project.

1. Copy files into directory /matrix/matrix



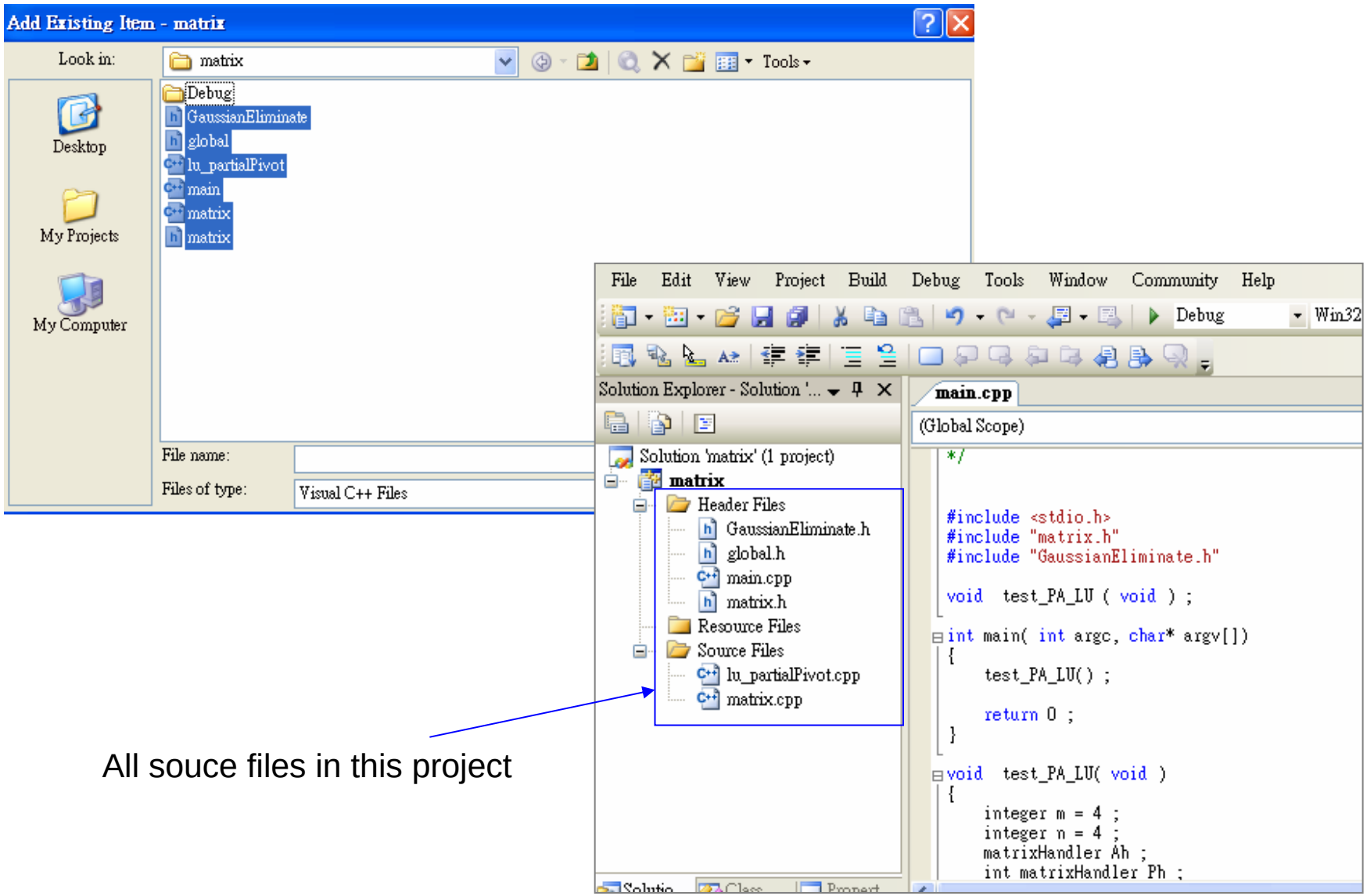
2. Add source files into the project by

   Project → Add Existing item



Visual Studio 2005: How To [10]

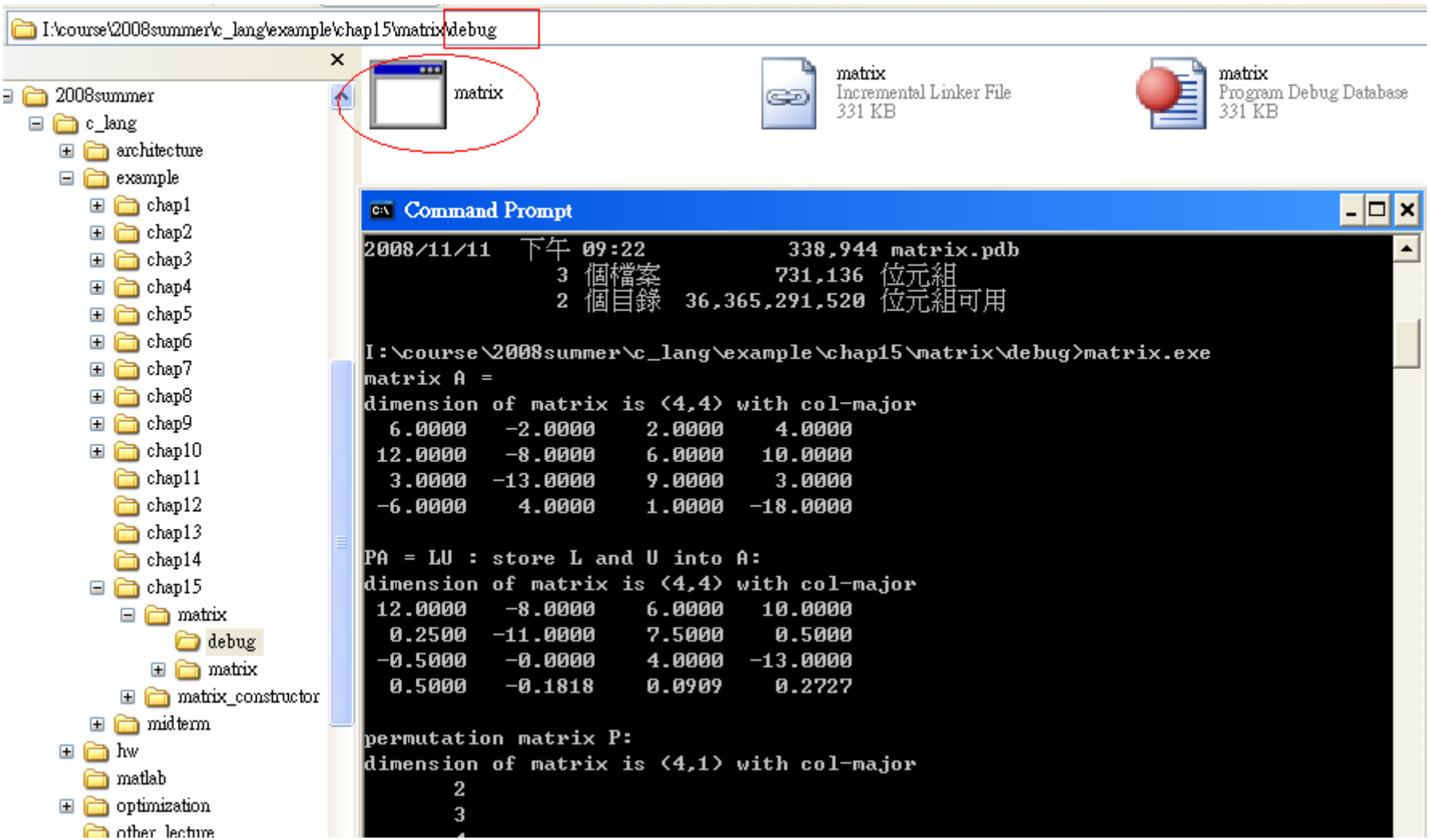
Select the files you want to add



All souce files in this project

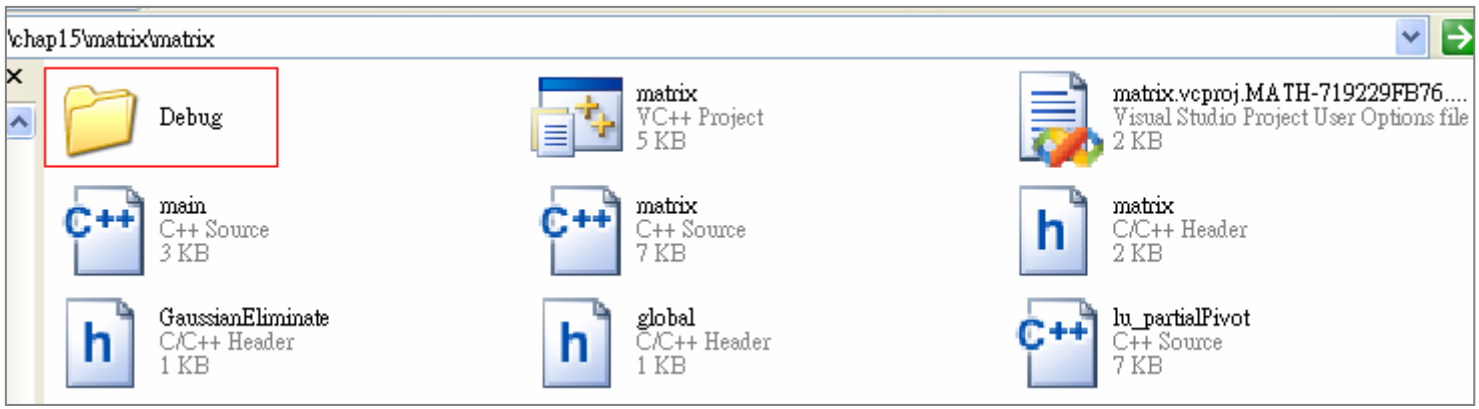
Visual Studio 2005: How To [11]

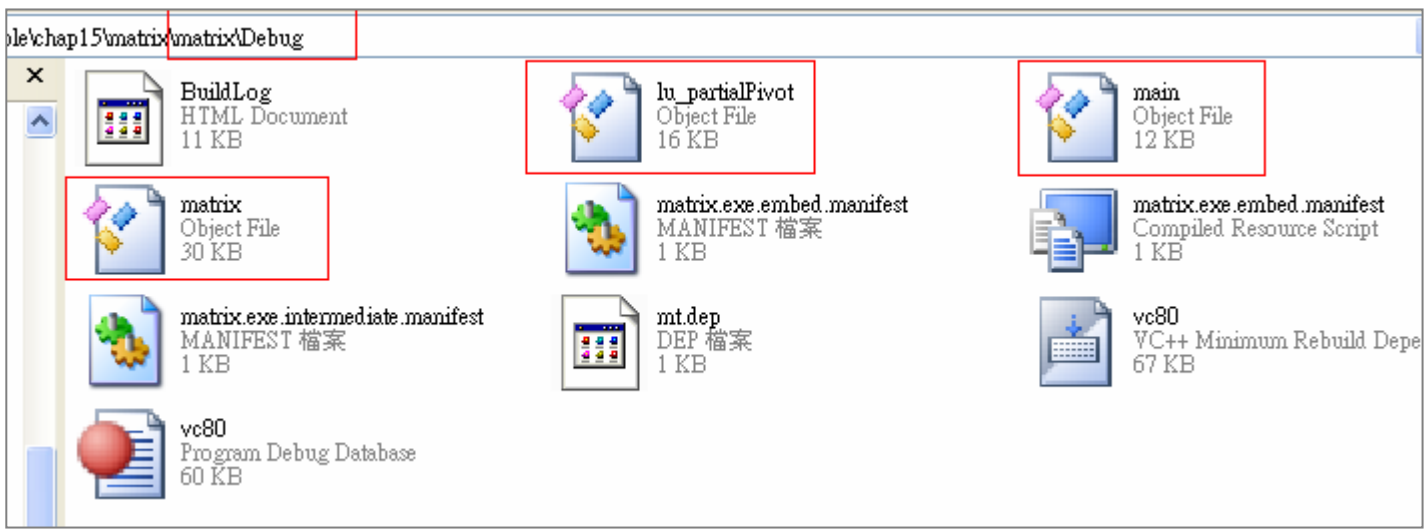
Remember that executable file is in directory /matrix/debug, you can use command window to execute matrix.exe



Visual Studio 2005: How To [12]

There is another “debug” directory in /matrix/matrix, this directory contains object files, no executable file





OutLine

- Data structure of full matrix
- Implementation of *PA=LU*
- Visual Studio 2005: How To
- Linux machine: How to compile

Linux machine: How to compile [1]

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

```
[macrold@quartet2 matrix_constructor]$ ls
Debug                lu_partialPivot.cpp  matrix_constructor.dsp  matrix_constructor.plg
GaussianEliminate.h  main.cpp             matrix_constructor.dsw
Makefile             matrix.cpp           matrix_constructor.ncb
global.h             matrix.h             matrix_constructor.opt
[macrold@quartet2 matrix_constructor]$
```

2. Remove all VC related files.

```
[macrold@quartet2 matrix_constructor]$
[macrold@quartet2 matrix_constructor]$ rm matrix_constructor.*
rm: remove regular file `matrix_constructor.dsp'? y
rm: remove regular file `matrix_constructor.dsw'? y
rm: remove regular file `matrix_constructor.ncb'? y
rm: remove regular file `matrix_constructor.opt'? y
rm: remove regular file `matrix_constructor.plg'? y
[macrold@quartet2 matrix_constructor]$ ls
Debug                Makefile  lu_partialPivot.cpp  matrix.cpp
GaussianEliminate.h  global.h  main.cpp             matrix.h
[macrold@quartet2 matrix_constructor]$
```

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

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

Linux machine: How to compile [2]

4 use command “qmake matrix_constructor.pro” to generate Makefile

Later on, we will introduce Makefile

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

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

```
[macrold@quartet2 matrix_constructor]$ 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/qt/include -o lu_partialPivot.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/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/qt/include -o matrix.o matrix.cpp
matrix.cpp(72): (col. 3) remark: LOOP WAS VECTORIZED.
matrix.cpp(34): (col. 3) remark: LOOP WAS VECTORIZED.
icpc -Wl,-rpath,/opt/qt/lib -o matrix_constructor lu_partialPivot.o main.o matrix.o   -L/opt/
qt/lib -L/usr/X11R6/lib -lqt-mt -lXext -lX11 -lm
[macrold@quartet2 matrix_constructor]$ ls
Debug              lu_partialPivot.cpp  matrix.cpp          matrix_constructor.pro
GaussianEliminate.h  lu_partialPivot.o    matrix.h
Makefile           main.cpp             matrix.o
global.h           main.o               matrix_constructor
[macrold@quartet2 matrix_constructor]$
```

Linux machine: How to compile [3]

6 Execute executable file by “./matrix_constructor”

```
[macrold@quartet2 matrix_constructor]$
[macrold@quartet2 matrix_constructor]$ ./matrix_constructor
matrix A =
dimension of matrix is (4,4) with col-major
  6.0000   -2.0000    2.0000    4.0000
 12.0000   -8.0000    6.0000   10.0000
  3.0000  -13.0000    9.0000    3.0000
 -6.0000    4.0000    1.0000  -18.0000

PA = LU : store L and U into A:
dimension of matrix is (4,4) with col-major
 12.0000   -8.0000    6.0000   10.0000
  0.2500  -11.0000    7.5000    0.5000
 -0.5000   -0.0000    4.0000  -13.0000
  0.5000   -0.1818    0.0909    0.2727

permutation matrix P:
dimension of matrix is (4,1) with col-major
       2
       3
       4
       1

right hand side vector b:
dimension of matrix is (4,1) with col-major
  5.0000
  6.0000
```