Advanced Class Lab06 04/07

1. **(On Workstation) Long Number Factors**
   Please show 64-digit number of facotrs from 1! to 49!. The result is as follows.

```
 1 != 0000000000000000000000000000000000000000000000000000000000000001
 2 != 0000000000000000000000000000000000000000000000000000000000000002
 3 != 0000000000000000000000000000000000000000000000000000000000000006
 4 != 0000000000000000000000000000000000000000000000000000000000000024
 5 != 0000000000000000000000000000000000000000000000000000000000000120
 6 != 0000000000000000000000000000000000000000000000000000000000000720
 7 != 0000000000000000000000000000000000000000000000000000000000005040
 8 != 0000000000000000000000000000000000000000000000000000000000040320
 9 != 0000000000000000000000000000000000000000000000000000000000362880
10 != 0000000000000000000000000000000000000000000000000000000003628800
11 != 0000000000000000000000000000000000000000000000000000000039916800
12 != 0000000000000000000000000000000000000000000000000000000479001600
13 != 0000000000000000000000000000000000000000000000000000006227020800
14 != 0000000000000000000000000000000000000000000000000000087178291200
15 != 0000000000000000000000000000000000000000000000000001307674368000
16 != 0000000000000000000000000000000000000000000000000020922789888000
17 != 0000000000000000000000000000000000000000000000000355687428096000
18 != 0000000000000000000000000000000000000000000000006402373705728000
19 != 0000000000000000000000000000000000000000000000121645100408832000
20 != 0000000000000000000000000000000000000000000002432902008176640000
21 != 0000000000000000000000000000000000000000000051090942171709440000
22 != 0000000000000000000000000000000000000000001124000727777607680000
23 != 0000000000000000000000000000000000000000025852016738884976640000
24 != 0000000000000000000000000000000000000000620448401733239439360000
25 != 0000000000000000000000000000000000000015511210043330985984000000
26 != 0000000000000000000000000000000000000403291461126605635584000000
27 != 0000000000000000000000000000000000010888869450418352160768000000
28 != 0000000000000000000000000000000000304888344611713860501504000000
29 != 0000000000000000000000000000000008841761993739701954543616000000
30 != 0000000000000000000000000000000265252859812191058636308480000000
31 != 0000000000000000000000000000008222838654177922817725562880000000
32 != 0000000000000000000000000000263130836933693530167218012160000000
33 != 0000000000000000000000000008683317618811886495518194401280000000
34 != 0000000000000000000000000295232799039604140847618609643520000000
35 != 0000000000000000000000010333147966386144929666651337523200000000
36 != 0000000000000000000000371993326789901217467999448150835200000000
37 != 0000000000000000000013763753091226345046315979581580902400000000
38 != 0000000000000000000523022617466601111760007224100074291200000000
39 != 0000000000000000020397882081197443358640281739902897356800000000
40 != 0000000000000000815915283247897734345611269596115894272000000000
41 != 0000000000000033452526613163807108170062053440751665152000000000
42 != 0000000000001405006117752879898543142606244511569936384000000000
43 != 0000000000060415263063373835637355132068513997507264512000000000
44 != 0000000002658271574788448768043625811014615890319638528000000000
45 != 0000000119622220865480194561963161495657715064383733760000000000
46 != 0000005502622159812088949850305428800254892961651752960000000000
47 != 0000258623241511168180642964355153611979969197632389120000000000
48 != 0012413915592536072670862289047373375038521486354677760000000000
49 != 0608281864034267560872252163321295376887552831379210240000000000
```

2. ## Gaussian Elimination

The process of row reduction of Gaussian Elimination makes use of elementary row operations, and can be divided into two parts.

The first part (sometimes called Forward Elimination) reduces a given system to row echelon form, from which one can tell whether there are no solutions, a unique solution, or infinitely many solutions. The example is as follows.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \text{ —————— ①}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \text{ —————— ②}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \text{ —————— ③}$$

=>

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \text{ —————— ① '}$$

$$a'_{22}\, x_2 + a'_{23}\, x_3 = b'_2 \text{ —————— ② '}$$

$$a'_{32}\, x_2 + a'_{33}\, x_3 = b'_3 \text{ —————— ③ '}$$

=>

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \text{ —————— ① ''}$$

$$a'_{22} + a'_{23}\, x_3 = b'_2 \text{ —————— ② ''}$$

$$a''_{33}x_3 = b''_3 \text{ —————— ③ ''}$$

The second part (sometimes called back substitution) continues to use row operations until the solution is found; in other words, it puts the matrix into reduced row echelon form. The process of example is as follows.

$$x_3 = b''_3 \,/\, a''_{33}$$

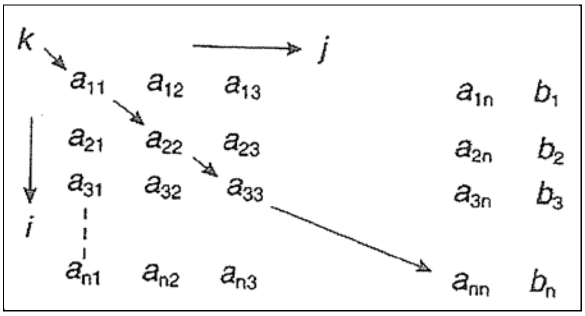
$$x_2 = (b_2' - a_{23}' x_3) \,/\, a_{22}'$$

$$x_1 = (b_1 - a_{12}\, x_2 - a_{13}\, x_3) \,/\, a_{11}$$

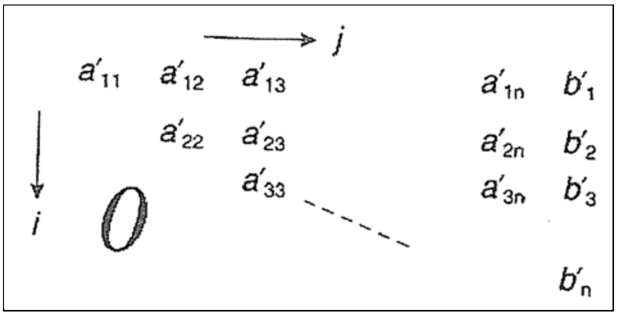
The summary of overall algorithm is as follows.

a) Forward Elimination: sweep pivot from 1st row, 1st column to (n-1) th, (n-1) th , and repeat following steps. Set a_{kk} as pivot and calculate a_{ik} / a_{kk} with i th

column , and then calculate $(i-k)^{th}\,column \times a_{ik} / a_{kk}$.



b) Forward Elimination: repeat following steps from $b_n^{'}$ to $b_1^{'}$. Set $b_i^{'}$ as initial value, and subtract $a_{ij}^{'} \times b_j^{'}$ from each element between ranges, $b_{i+1}^{'}$ and $b_n^{'}$, and then make $a_{ii}^{'}$ be divided by the result.



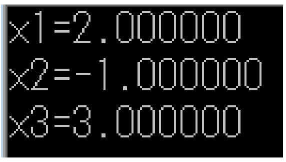
You should implement a function with the following prototype:

double* GaussElim(double **A, double *b, int row, int col)

Please implement Gauss Elimination Method to solve following linear equation in three variables.

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 4 \\ 4x_1 + x_2 - 3x_3 = -2 \\ -x_1 + 2x_2 + 2x_3 = 2 \end{cases}$$

The result is as follows.



3. Please correct following code and make the output be as follows. You just can correct red part, i.e. "->" or ".". Please do not rewrite or change other codes.

```c
#include <stdio.h>
#define N 4

static struct man {
        int id;
        char name[20];
        int age;
} person[N] = {1,"li",18, 2,"wang",23, 3,"zhang",20, 4,"sun",22};

int main()
{
        struct man *q, *p;
        int i, m=0;
        p=person;

        for (i=0; i<N; i++)
            printf("%d: %s,%d\n", (person+i)->id, (*(person+i)).name, person[i]->age);

        for (i=0; i<N; i++)
                if (m < p.age) { q = p++; m=q->age; }
        printf("Oldest: %d,%s,%d", q[0]->id, q.name, (*q)->age);

        return 0;
}
```

The result is

```
1: li,18
2: wang,23
3: zhang,20
4: sun,22
Oldest: 2,wang,23
```

4. (On Workstation) Royal hold 'em (also royal hold'em or royal holdem) is a deviation of limit Texas hold 'em played without deuces through nines, leaving only the tens, jacks, queens, kings, and aces. Please write a function that shuffles a deck of cards. The deck of cards is represented by an array of 20 elements. Each element in the array is a structure for one card, as shown below.

```
enum suit { HEART = 3, DIAMOND, CLUB, SPADE };
enum numb { ACE='A', TEN='0', JACK='J', QUEEN='Q', KING='K' };

struct Card
{
	suit cardSuit;
	numb cardNum;
};
Card deck[20];
```

The function must use a random number to ensure that each shuffle results in a different card sequence.
Please write a main program to test the function and printout shuffle results shown as below figure.

Hint: Generate a random number in the range of 0 to 19 and then exchange the current card with the card in the random position. Please notice how to exchange two variables with struct type.

5. **A complex number can be represented by real part and imaginary part, i.e. $a+bi$. Another way of encoding points in the complex plane is to use polar form $r(\cos\theta + i\sin\theta)$ or $re^{i\theta}$. Therefore, we can represent a complex by a structure shown as follows. (You can refer to Wikipedia to get more information about complex number. (http://en.wikipedia.org/wiki/Complex_number#Polar_form)**

Please construct a complex number type by using a nested structure. The format of the complex number is as follows. Please declare a struct type by following table.

<table>
<tr><th colspan="4">Complex</th></tr>
<tr><td>r</td><td>I</td><td colspan="2">polar</td></tr>
<tr><td rowspan="2">float</td><td rowspan="2">Float</td><td>mag</td><td>arg</td></tr>
<tr><td>double</td><td>double</td></tr>
</table>

Complete the following functions.

```
void C_print (const Complex z)
{
```

```
            // printout (a + bi)
}

void P_print (const Complex z)
{
            // printout (r * e^ (i sita) ◦ )
}

double convertC2P (Complex *z)
{
      // complete polar form transform
}

double convertP2C (Complex *z)
{
      // complete general form transform
}

void readInGeneral (Complex &z)
{
      // read real part and imaginary part from user
      // and then convert to polar form by convertC2P function
}

void readInPolar (Complex &z)
{
      // read magnitude part and phase part from user
      // and then convert to general form by convertP2C function
}

Complex C_add (const Complex z1, const Complex z2)
{
      // return (z1 + z2)
}

Complex C_sub (const Complex *z1, const Complex *z2)
{
      // return (z1 - z2)
}
```

```
Complex C_mul (const Complex &z1, const Complex &z2)
{
	// return (z1 * z2)
}

Complex C_div (const Complex *z1, const Complex *z2)
{
	// return (z1 / z2)
}
```

6. **Complete the following functions.**

```
Complex C_pown (Complex *z, const int n)
{
    // return (z^n)
}

void C_sort (Complex *z, const int n)
{
    // sort the complex data in z according to magnitude
}
```

You can use following code to verify your functions implemented in problem 5 and 6.

```
int main(void)
{
	Complex a[N] = {{3.0, 4.0}, {6.0, 8.0}, {2,7}, {5, 3}, {10, 10}, {9, 4}};
	Complex b[2]; // for general form input
	Complex c[2]; // for polar form input
	int i;

	// read data from user into b[i] and c[i]
	for (i=0; i<2; i++)
		readInGeneral(b[i]);

	for (i=0; i<2; i++)
		readInPolar(c[i]);
	printf("\n");

	// transform complex number a[i] into polar form
```

```
for (i=0; i<N; i++) {
        convertC2P(&a[i]);
        C_print(a[i]);
        printf(" = ");
        P_print(a[i]);
        printf("\n");
}
printf("\n");

// print out a[i] and b[i] and c[i] with general form and polar form
for (i=0; i<2; i++) {
        C_print(b[i]);
        printf(" = ");
        P_print(b[i]);
        printf("\n");
}
printf("\n");

for (i=0; i<2; i++) {
        C_print(c[i]);
        printf(" = ");
        P_print(c[i]);
        printf("\n");
}
printf("\n");

// verify add, sub, mul, div, and pow functions here
C_print(C_add(a[0], a[1]));
printf("\n");
C_print(C_sub(&a[0], &a[1]));
printf("\n");
C_print(C_mul(a[0], a[1]));
printf("\n");
C_print(C_div(&a[0], &a[1]));
printf("\n");
C_print(C_pown(&a[0], 5));
printf("\n");
P_print(C_pown(&a[0], 5));
printf("\n\n");
```

```
        // verify sort function here
        C_sort(a, N);
        for (i=0; i<N; i++) {
                C_print(a[i]);
                printf("\n");
        }
        printf("\n");

        getch();
        return 1;
}
```

The example result is as follows.

```
Please input two parameters (a b for a+bi): 3 4
Please input two parameters (a b for a+bi): 5 6
Please input parameters (r s for r<e^s): 7 35
Please input parameters (r s for r<e^s): 8 60

3.00+4.00i = 5.00*e^(i53.13)
6.00+8.00i = 10.00*e^(i53.13)
2.00+7.00i = 7.28*e^(i74.05)
5.00+3.00i = 5.83*e^(i30.96)
10.00+10.00i = 14.14*e^(i45.00)
9.00+4.00i = 9.85*e^(i23.96)

3.00+4.00i = 5.00*e^(i53.13)
5.00+6.00i = 7.81*e^(i50.19)

5.73+4.02i = 7.00*e^(i35.00)
4.00+6.93i = 8.00*e^(i60.00)

9.00+12.00i
-3.00+-4.00i
-14.00+48.00i
0.50+0.00i
-272.36+-3113.11i
3125.00*e^(i-95.00)

10.00+10.00i
6.00+8.00i
9.00+4.00i
2.00+7.00i
5.00+3.00i
3.00+4.00i
```