

Funções Vetores Matrizes

Além dos tipos elementares (float, double, char, etc.), é possível também passar um vetor ou uma matriz como parâmetro de funções.

Quando um vetor é passado como parâmetro, o que é passado na verdade é o endereço ou localização do primeiro elemento do vetor. Desta forma, é possível que se tenha acesso a todos os elementos do vetor dentro da função, pois os elementos são contíguos na memória.

O mesmo não ocorre com parâmetros dos tipos elementares. Neste caso, quando uma variável é passada como parâmetro, o que vai é o valor da variável. Dizemos que a passagem de parâmetros é “por valor”. Por isso, quando alteramos o valor de um parâmetro do tipo elementar, a mudança só vale dentro da função.

Escreva uma função **zera (a, n)** que zera os **n** primeiros elementos do vetor **a** de inteiros.

```
/* Função zera (a, n) que zera os n primeiros elementos
   do vetor a de inteiros */

void zera (int a[], int n) {
    int i = 0;
    while (i < n) a[i++] = 0;
    /* não precisa retornar nada */
}
```

Alguns exemplos de chamada da função **zera**:

```
/* exemplo de programa principal */
int main() {
    int x[100], y[30], z[50];
    int k = 20;
    /* zerar todo o vetor x */
    zera (x,100);
    /* zerar os 30 primeiros de x */
    zera (x, 30);
    /* zerar os 200 primeiros de x - vai dar erro porque x só tem 100 elementos */
    zera (x, 200);
    /* zerar todo o vetor y */
    zera (y,30);
    /* zerar os k primeiros de z */
    zera (z, k);
}
```

Escreva uma função **conta (a, n, x)** que devolve como resultado, o número de elementos iguais a **x** que aparecem no vetor **a** de **n** elementos.

```
/* Função conta (a, n, x) que devolve como resultado, o número
   de elementos iguais a x que aparecem no vetor a de n elementos. */

int conta (int a[], int n, int x) {
    int i = 0,
        cc = 0;
    while (i < n) {
```

```
        if (a[i] == x) cc++;
        i++;
    }
    return cc;
}
```

Abaixo, alguns exemplos de chamadas da função conta:

```
#include <stdio.h>
int main() {
    int vet[200];
    int n, k;
    ...
    ...
    /* atribui a k o número de nulos de x */
    k = conta (vet, 200, 0);
    ...
    ...
    /* imprime o número de -1s nos 50 primeiros elementos de vet */
    printf("\nnúmero de elementos iguais a -1 = %5d", conta (vet, 50, -1));
    ...
    ...
    /* imprime quantas vezes cada número de 0 a 9 aparece nos n primeiros */
    for (k = 0; k < 10; k++)
        printf("\n%5d aparece %5d vezes", k, conta (vet, n, k));
    ...
    ...
    /* Verifica quantas vezes cada elemento do vetor se repete */
    for (k = 0; k < n; k++)
        printf("\nvect[%3d] = %5d aparece %5d vezes", k, vet[k],
            conta (vet, n, vet[k]));
    ...
    ...
}
```

Escreva uma função **trocavet (char a[], char b[], int n)** que troca o conteúdo dos 2 vetores **a** e **b** de **n** elementos.

```
/* Função trocavet (char a[], char b[], int n) que troca o conteúdo
de 2 vetores a e b de n elementos. */

void trocavet (char a[], char b[], int n) {
    int i = 0;
    char aux;

    while (i < n) {
        aux = a[i];
        a[i] = b[i];
        b[i] = aux;
        i++;
    }
}
```

O que será impresso pelo programa abaixo? Veja mais abaixo.

```
/* Exemplo de uso da função troca */
#include <stdio.h>
```

```
int main() {
    char x[5] = {"abcde"},
        y[5] = {"edcba"};
    int k;

    trocavet(x, y, 5);
    /* imprime x depois da troca */
    for (k = 0; k < 5; k++)
        printf("\nx[%2d] = %1c", k, x[k]);

    /* imprime y depois da troca */
    for (k = 0; k < 5; k++)
        printf("\ny[%2d] = %1c", k, y[k]);
}
```

```
x[ 0] = e
x[ 1] = d
x[ 2] = c
x[ 3] = b
x[ 4] = a
y[ 0] = a
y[ 1] = b
y[ 2] = c
y[ 3] = d
y[ 4] = e
```

Escreva uma função **busca** (**double a[]**, **int n**, **double x**), que procura **x** no vetor **a** de **n** elementos, devolvendo como resultado o índice do elemento que é igual a **x** ou **-1** caso não encontre, Embora possa existir mais de um elemento igual a **x**, devolva o índice do primeiro encontrado.

```
/* Função busca (int a[], int n, int x), que procura
   x no vetor a de n elementos, devolvendo como resultado o índice do
   elemento que é igual a x ou -1 caso não encontre. */

int busca (int a[], int n, int x) {
    int i = 0;

    while (i < n) {
        if (a[i] == x) return i; /* encontrou */
        i++;
    }
    return -1; /* não encontrou */
}
```

Abaixo um exemplo de chamada e que será impresso pelo programa:

```
/* Gerar nummax números aleatórios entre 0 e 29 verificar se são primos */
#include <stdio.h>
#include <stdlib.h>
#define nummax 20
int main() {
    int primos[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
    int n, k;
    for (k = 0; k < nummax; k++) {
        n = rand () % 30; /* n estará entre 0 e 29 */
        if (busca (primos, 10, n) >= 0)
            printf("\n%5d e um numero primo", n);
    }
}
```

```
        else printf("\n%5d nao e numero primo", n);  
    }  
}
```

```
11 e um numero primo  
17 e um numero primo  
4 nao e numero primo  
10 nao e numero primo  
29 e um numero primo  
4 nao e numero primo  
18 nao e numero primo  
18 nao e numero primo  
22 nao e numero primo  
14 nao e numero primo  
5 e um numero primo  
5 e um numero primo  
1 nao e numero primo  
27 nao e numero primo  
1 nao e numero primo  
11 e um numero primo  
25 nao e numero primo  
2 e um numero primo  
27 nao e numero primo  
6 nao e numero primo
```

Escreva uma função comp (char a[], char b[], int n) que compara os 2 vetores a e b de n elementos, devolvendo:

n - se **a[i] = b[i]** para $0 \leq i < n$

k - se **a[k] ≠ b[k]** e **a[i] = b[i]** para $0 \leq i < k$

```
/* Função comp (char a[], char b[], int n) que compara os 2 vetores a e b  
de n elementos, devolvendo:
```

```
    n - se a[i] = b[i] para 0 <= i < n
```

```
    k - se a[k] ≠ b[k] e a[i] = b[i] para 0 ≤ i < k
```

```
*/
```

```
int comp (char a[], char b[], int n) {  
    int i = 0;  
    while (i < n) {  
        if (a[i] != b[i]) break;  
        i++;  
    }  
    return i; /* note que se todos forem iguais i sai do while valendo n */  
}
```

O que será impresso pelo programa abaixo?

```
#include <stdio.h>  
int main() {  
    char a[] = {0,1,2,3,4,5,6,7,8,9},  
          b[] = {0,1,2,3,4,5,6,6,6,6};  
  
    printf("\n*****numero de elementos iguais = %5d", comp (a, b, 5));  
    printf("\n*****numero de elementos iguais = %5d", comp (a, b, 10));  
}
```

```
*****numero de elementos iguais = 5  
*****numero de elementos iguais = 7
```

Escreva uma função `int semprepeticao(int a[], int n, int b[])` que recebe um vetor `a` de `n` elementos e devolve um vetor `b`, onde `b` contém os mesmos elementos de `a` sem elementos repetidos. O valor da função é o número de elementos de `b`.

Idem `int semprepeticao(int a[], int n)` eliminando as repetições e devolvendo no próprio `a`.

Escreva uma função `void uniao(int a[], int n, int b[], int m, int c[], int *p)` que recebe os vetores `a` e `b` de `n` e `m` elementos respectivamente e devolve o vetor `c` de `p` elementos contendo o conjunto união entre `a` e `b`, ou seja, os elementos que estão em `a` ou em `b`. Note que `*p` é parâmetro de saída.

```
/* Função uniao (int a[], int n, int b[], int m, int c[], int *p) que  
   recebe os vetores a e b de n e m elementos e devolve o vetor c de p  
   elementos contendo o conjunto uniao entre a e b, ou seja,  
   os elementos que estão em a ou em b. */  
int uniao (int a[], int n, int b[], int m, int c[], int *p) {  
    int i, j;  
    /* move a para c */  
    for (i = 0; i < n; i++) c[i] = a[i];  
    *p = n;  
    /* verifica os que estão em b e não em a e os acrescenta a c */  
    for (i = 0; i < m; i++) {  
        /* procura b[i] em c[0] até c[*p - 1] */  
        for (j = 0; j < *p; j++)  
            if (b[i] == c[j]) break; // este b[i] já está  
        /* verifica se não encontrou b[i] em c  
           se não encontrou acrescenta b[i] a c e incrementa *p */  
        if (j == *p) {c[*p] = b[i]; (*p)++;}  
    }  
}
```

O que será impresso no programa abaixo?

```
#include <stdio.h>  
int main() {  
    int a[] = {0,1,2,3,4,5,6,7,8,9},  
        b[] = {5,6,7,8,9,10,11,12},  
        c[20];  
    int nc, i;  
  
    uniao (a, 10, b, 8, c, &nc);  
    printf("\n*****uniao de a com b tem %d elementos *****", nc);  
    for (i = 0; i < nc; i++)  
        printf("\nc[%2d] = %5d", i, c[i]);  
}
```

```
*****uniao de a com b tem 13 elementos *****  
c[ 0] = 0  
c[ 1] = 1
```

```
c[ 2] = 2
c[ 3] = 3
c[ 4] = 4
c[ 5] = 5
c[ 6] = 6
c[ 7] = 7
c[ 8] = 8
c[ 9] = 9
c[10] = 10
c[11] = 11
c[12] = 12
```

Escreva a função void intersecção(int a[], int n, int b[], int m, int c[], int *p) que devolve em **c** a intersecção de **a** com **b**.

Escreva a função void diferença(int a[], int n, int b[], int m, int c[], int *p) que devolve em **c** a diferença de **a** com **b**.

Matrizes e Funções

Matrizes podem ser usadas como parâmetros de funções.

Os elementos das matrizes estão dispostos contiguamente na memória, linha a linha, ou seja, variando primeiro os últimos índices.

Para a matriz a[10][10]:

```
a[0][0] a[0][1] a[0][2] . . . a[0][9]
a[1][0] a[1][1] a[1][2] . . . a[1][9]
:
:
:
a[9][0] a[9][1] a[9][2] . . . a[9][9]
```

Para a matriz b[30][40]:

```
b[0][0] b[0][1] a[0][2] . . . b[0][39]
b[1][0] b[1][1] b[1][2] . . . b[1][39]
:
:
:
b[29][0] b[29][1] b[29][2] . . . b[29][39]
```

Para a matriz c[3][4][5]:

```
c[0][0][0] c[0][0][1] c[0][0][2] c[0][0][3] c[0][0][4]
c[0][1][0] c[0][1][1] c[0][1][2] c[0][1][3] c[0][1][4]
c[0][2][0] c[0][2][1] c[0][2][2] c[0][2][3] c[0][2][4]
c[0][3][0] c[0][3][1] c[0][3][2] c[0][3][3] c[0][3][4]
c[1][0][0] c[1][0][1] c[1][0][2] c[1][0][3] c[1][0][4]
c[1][1][0] c[1][1][1] c[1][1][2] c[1][1][3] c[1][1][4]
c[1][2][0] c[1][2][1] c[1][2][2] c[1][2][3] c[1][2][4]
c[1][3][0] c[1][3][1] c[1][3][2] c[1][3][3] c[1][3][4]
c[2][0][0] c[2][0][1] c[2][0][2] c[2][0][3] c[2][0][4]
c[2][1][0] c[2][1][1] c[2][1][2] c[2][1][3] c[2][1][4]
```

`c[2][2][0]` `c[2][2][1]` `c[2][2][2]` `c[2][2][3]` `c[2][2][4]`
`c[2][3][0]` `c[2][3][1]` `c[2][3][2]` `c[2][3][3]` `c[2][3][4]`

A memória é linear, assim, do ponto de vista de armazenamento na memória, não há diferença entre uma matriz e um vetor, pois os elementos estão contíguos na memória. No caso do vetor, o índice, determina o deslocamento do elemento a partir do início do vetor. Por exemplo, considere o vetor `int x[5]` que estará disposto na memória assim:

<code>x[0]</code>	<code>x[1]</code>	<code>x[2]</code>	<code>x[3]</code>	<code>x[4]</code>
-------------------	-------------------	-------------------	-------------------	-------------------

Observe que `x[i]` estará na posição (início de `x`) + `i`.

No caso de matrizes, o deslocamento em relação ao início, é uma função dos vários índices. Esta função, nada mais é que a transformação dos vários índices num só índice, ou seja, a linearização dos vários índices. Vejamos alguns exemplos nas matrizes declaradas acima.

- 1) `a[1][2]` é o 12º elemento (o primeiro é o 0º), ou $(10 \cdot 1 + 2)^\circ$. Note que temos que contar quantos elementos anteriores existem.
- 2) `a[2][3]` é o 23º elemento, ou $(2 \cdot 10 + 3)^\circ$
- 3) `a[9][5]` é o 95º elemento, ou $(9 \cdot 10 + 5)^\circ$
- 4) `b[29][2]` é o $1162^\circ = (29 \cdot 40 + 2)^\circ$
- 5) `c[2][1][3]` é o $48^\circ = (2 \cdot 4.5 + 1.5 + 3)^\circ$
- 6) `c[1][3][1]` é o $36^\circ = (1 \cdot 4.5 + 3.5 + 1)^\circ$

Portanto, para 2 índices, se temos uma matriz `mat` com o primeiro e segundo índices de valor máximo `d1` e `d2` (por exemplo `int mat[d1][d2]`) o elemento `mat[i][j]` estará na posição (início de `mat`) + `i.d2 + j`.

Para 3 índices, a matriz `mat` com o primeiro, segundo e terceiro índices de valores máximos `d1`, `d2` e `d3` (por exemplo `int mat[d1][d2][d3]`) o elemento `mat[i][j][k]` estará na posição (início de `mat`) + `i.d2.d3 + j.d3 + k`.

Para `n` índices, a matriz `mat` com índices de valores máximos `d1`, `d2`, ..., `dn` (`int mat[d1][d2]...[dn]`) o elemento `mat[i1][i2]...[in]` estará na posição:

$$\begin{aligned} &(\text{início de } \mathbf{mat}) + \\ &\mathbf{i1} \cdot \mathbf{d2} \cdot \mathbf{d3} \cdot \dots \cdot \mathbf{dn} + \\ &\mathbf{i2} \cdot \mathbf{d3} \cdot \mathbf{d4} \cdot \dots \cdot \mathbf{dn} + \\ &\mathbf{i3} \cdot \mathbf{d4} \cdot \mathbf{d5} \cdot \dots \cdot \mathbf{dn} + \\ &\dots \\ &\mathbf{i} \cdot (\mathbf{n}-1) \cdot \mathbf{dn} + \\ &\mathbf{in} \end{aligned}$$

Então, para acessar um elemento, é necessário conhecer-se as dimensões de todos os índices, com exceção do primeiro. Por isso, na declaração de uma matriz como parâmetro formal, é necessário declarar-se todos os índices máximos a partir do segundo.

Exemplos:

```
int funcx(double a[][100], int n);  
double funcy (int x[][200][10], y[], z[][23][30]);
```

A matriz passada como parâmetro deve ter os mesmos índices da matriz que recebe este parâmetro.

O exemplo abaixo está errado:

```
double f(int x[][8])  
...  
main() {  
    int a[10][10];  
    ...  
    f(a);  
    ...  
}
```

Também é possível que uma função tenha como parâmetro formal um vetor, e seja chamada com um parâmetro do tipo matriz.

```
void zera_vet_mat(int x[], int n) {  
    /* zera vetor x com n elementos, mas recebe matriz */  
    int i;  
    for (i=0; i<n; i++) x[i] = 0;  
}
```

```
int main() {  
    int a[10][10];  
    zera_vet_mat(a, 100);  
    ...  
}
```

Outro exemplo: função que zera o elemento [i,j] de uma matriz, mas recebe um vetor como parâmetro. Neste caso é necessário também passar a quantidade de elementos de cada linha da matriz.

```
void zera_elem(int x[], int i, int j, int ncol) {  
    /* zera vetor x com n elementos, mas recebe matriz como parâmetro */  
    x[i * ncol + j] = 0;  
}
```

```
int main() {  
    int a[10][10];  
    int n,m;  
  
    /* zera a[3][4] */  
    zera_elem(a, 3, 4, 10);  
  
    /* zera toda a matriz */  
    for (n = 0, n < 9, n++)  
        for (m = 0, m < 9, m++) zera_elem(a, n, m, 10);  
  
    ...  
}
```


Outra maneira de entender a necessidade de declarar os índices de uma matriz quando esta matriz é parâmetro de função

Outra forma de entender melhor a necessidade de declarar na função os índices a partir do segundo através de um exemplo com duas dimensões:

Suponha uma função F que recebe com parâmetro uma matriz de no máximo 10 linhas por 20 colunas e também 2 outros parâmetros inteiros n e m que representam as linhas e colunas que serão realmente usadas.

```
void F(int A[][20], int n, int m) {  
...  
}
```

Os elementos da matriz estão dispostos na memória linha a linha.

Suponha a chamada abaixo da função F:

```
int X[10][20];  
...  
...  
...  
F(X,7,8); /* usando apenas 7 linhas e 8 colunas de X */  
...  
...
```

Será usado apenas o trecho indicado de **X**.

F precisa saber quantos elementos tem que pular por linha da matriz para acessar os elementos corretos.

Outros exemplos de funções usando matrizes como parâmetros

Escreva uma função `int simetrica (double a[][maxcol], int n)` que verifica se a matriz **a** **n****x****n** é uma matriz simétrica, isto é, se **a**[**i**][**j**] é igual a **a**[**j**][**i**]. Devolve 1 se for simétrica e o caso contrário.

```
#define maxlin 10  
#define maxcol 10  
  
/* Escreva uma função simetrica (double a[maxlin][], int n) que verifica
```

```
    se a matriz a nxn é uma matriz simétrica. */
int simetrica (double a[][maxcol], int n) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (a[i][j] != a[j][i]) return 0; // não é

    return 1; /* é simétrica */
}
```

Veja abaixo exemplos de chamada e o que será impresso:

```
/* Exemplo de chamada */
#include <stdio.h>
int main() {
    double a[maxlin][maxcol],
           b[maxlin][maxcol];

    int n = 2,
        m = 3;

    /* preenche a (simetrica) */
    a[0][0] = 1; a[1][1] = 2;
    a[0][1] = a[1][0] = 3;

    if (simetrica(a,n)) printf ("\n a e simetrica");
    else printf ("\n a nao e simetrica");

    // preenche b (não simetrica)
    b[0][0] = 1; b[1][1] = 2; b[2][2] = 3;
    b[0][1] = b[1][0] = 3;
    b[0][2] = b[2][0] = 4;
    b[1][2] = 5; b[2][1] = 6; // não é mesmo

    if (simetrica(b, m)) printf ("\n b e simetrica");
    else printf ("\n b nao e simetrica");
}

a e simetrica
b nao e simetrica
```

Observe que na função **simetrica**, fazemos comparações desnecessárias, pois comparamos **a[i][j]** com **a[j][i]** e depois **a[j][i]** com **a[i][j]**, pois **i** e **j** variam de **0** a **n-1**. Bastaria comparar apenas quando **i>j** ou quando **i<j**, ou seja, percorrendo o triângulo inferior ou o triângulo superior.

Idem, percorrendo o triângulo inferior:

```
#define maxlin 10
#define maxcol 10
/* Escreva uma função simetrica (double a[maxlin][], int n) que verifica
   se a matriz a nxn é uma matriz simétrica, percorrendo apenas a parte
   inferior da matriz em relação à diagonal principal.
*/
```

```
int simetrica (double a[][maxcol], int n) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
            if (a[i][j] != a[j][i]) return 0; /* não é */

    return 1; /* é simétrica */
}
```

Escreva agora a mesma função, percorrendo o triângulo superior.

Escreva uma função **void somamat(double a[][mc], b[][mc], c[][mc], int n)** que recebe as matrizes **a** e **b** e devolve a matriz **c**, soma de **a** com **b**. **a**, **b** e **c** tem dimensão **nxn**.

Idem, uma função **void multmatvet(double a[][mc], double b[], double c[], int n, int m)** que recebe a matriz **a** de **nxm** elementos, o vetor **b** de **m** e devolve o vetor **c** de **n** elementos que é a multiplicação de **a** por **b**.

```
#define maxlin 100
#define maxcol 100

/* Função multmatvet (double a[][maxcol], double b[], double c[],
    int n, int m)
    que recebe a matriz a de n x m elementos, o vetor b de m e devolve
    o vetor c de n elementos que é a multiplicação de a por b.
*/
int multmatvet (double a[][maxcol], double b[], double c[],
    int n, int m) {
    int i, j;

    for (i = 0; i < n; i++) {
        /* multiplica a linha i da matriz pelo vetor */
        c[i] = 0;
        for (j = 0; j < m; j++)
            c[i] = c[i] + a[i][j] * b[j];
    }

    return 0; /* não precisa retornar nada */
}
```

Idem, uma função **void multmatmat(double a[][mc], double b[][mc], double c[][mc], int n, int m, int p)** que recebe a matriz **a** de **nxm** elementos, a matriz **b** de **m x p** elementos e devolve a matriz **c** de **n x p** elementos, que é a multiplicação de **a** por **b**.

Escreva uma função **ind_min_col(double a[][maxc], int n, int col)** que receba a matriz **a** de **nxn** elementos e devolva como resultado o índice do menor elemento da coluna **col** desta matriz.

```
#define maxlin 100
#define maxcol 100

/* Função ind_min_col (double a[][maxc], int n, int col)
```

```
que receba a matriz a de n x n elementos e devolva como resultado
o índice do menor elemento da coluna col desta matriz.
*/
int ind_min_col (double a[][maxcol], int n, int col) {

    int lin;          /* índice para as linhas */
    int indmin = 0;  /* índice do mínimo na coluna. */
                    /* começa supondo que é o primeiro da coluna */

    /* varre todos os elementos desta linha a partir do segundo e
    verificando se é menor que o mínimo corrente
    se for troca o índice do mínimo corrente */
    for (lin = 1; lin < n; lin++)
        if (a[lin][col] < a[indmin][col]) indmin = lin;
    return indmin;
}
```

Índice inicial de vetores e matrizes

O índice inicial de vetores em C é sempre 0:

```
int x[10] declara elementos x[0], x[1], ... x[9].
```

Idem, para o índice inicial de matrizes de 2 ou mais dimensões:

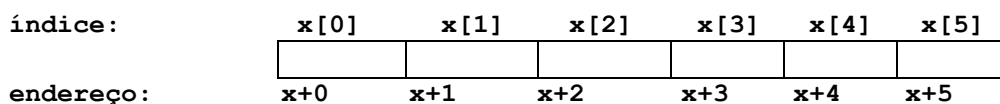
```
int y[3][2] declara y[0][0], y[0][1], y[1][0], y[1][1], y[2][0], y[2][1]
```

Em outras linguagens de programação isso não ocorre:

Fortran – índice sempre começa do 1.

Pascal – declara-se o índice inicial e final.

A vantagem de se começar o índice do zero como já vimos, é que no caso de vetores é exatamente o valor a ser incrementado para se chegar ao endereço do elemento. Não é necessário subtrair 1 para se chegar ao elemento.

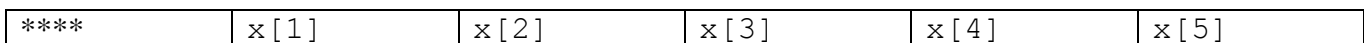


No caso de matrizes é exatamente o valor a ser multiplicado pelas várias dimensões.

A desvantagem é que quando vamos resolver problemas de matemática, principalmente de álgebra linear envolvendo vetores e matrizes é usual que os índices comecem em 1.

Uma forma de contornarmos esse problema é declarar o vetor com um elemento a mais e não usar o elemento de índice 0.

Exemplo – vetor x com 5 elementos: **int x[6]**



No caso de matrizes é mais drástico. Declarar a matriz com uma linha e uma coluna a mais e não usar a linha 0 e coluna 0.

Exemplo – matriz y com 3 linhas e 4 colunas: **int y[4][5]**

*****	*****	*****	*****	*****
*****	y[1][1]	y[1][2]	y[1][3]	y[1][4]
*****	y[2][1]	y[2][2]	y[2][3]	y[2][4]
*****	y[3][1]	y[3][2]	y[3][3]	y[3][4]