

# Introduction to Programming

# Two dimensional arrays

- ▶ Table, matrix: rows, columns
- ▶ Definition:
  - **type** name[row size][column size];
- ▶ Example:

```
int a[10] [10];  
float b[20] [20];  
char c[30] [30];
```
- ▶ Initialization:

```
int a[2][3]={1,3,5,0,77,-12};  
float b[50][50]={0};
```
- ▶ Reference:  
name [row\_index] [column\_index]; Example: a[2][3];

# Two dimensional array

**Example:** Print out the following two dimensional array elements with two decimal precision!

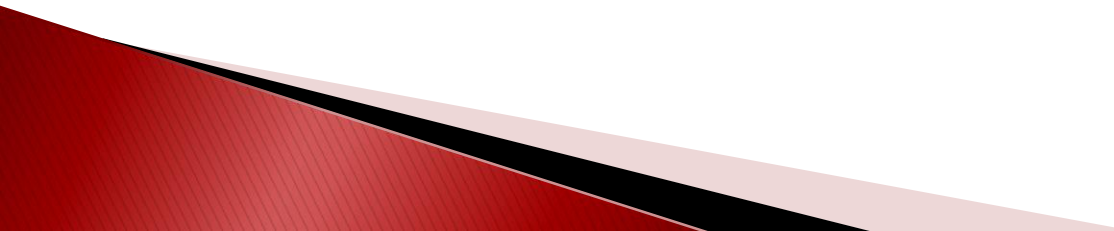
```
float a[ 3 ][ 4 ] = { 2.5, 0.0, 5.5, 4.9 ,  
-4.2, 1.0, 0.2, 2.6 , 2.7, -1.0, -5.1, 0.4 };
```

**Help:** the declaration of the array can be given according to the example above, use the **float** type print "%.2f " with two decimal precision!

# Solution

```
int i, j;
float a[ 3 ][ 4 ] = {2.5, 0.0, 5.5, 4.9,
                    -4.2, 1.0, 0.2, 2.6, 2.7, -1.0, -5.1, 0.4};

for(i=0; i<3; i++)
{
    for(j=0; j<4; j++)
        printf("%.2f\t", a[i][j]);
    printf("\n");
}
```



# Exercise

- ▶ Write a program which inputs from the keyboard a matrix elements – which contains integers – into a two dimensional array then displays the elements on the screen (standard output) in a matrix form.

# Solution

```
int i, j, row, column;
printf("row: "); scanf("%d",&row);
printf("column: "); scanf("%d",&column);
int a[row][column];
printf("Give the matrix elements!\n");
    for(i=0; i<row; i++)
        for(j=0; j<column; j++)
            {
                printf("a[%d][%d]=", i, j);
                scanf("%d",&a[i][j]);
            }
for(i=0; i<row; i++) {
    for(j=0; j<column; j++)
        printf("%d\t",a[i][j]);
    printf("\n"); }
```

# Exercise

What is the result after the execution of this code?

```
double m[4][3]={9, 7, 0.5, 3.14, 2, 5.0, 4, 6.5,  
                0, 8, 5.95, 7.2};  
printf("%.2lf %.4lf \n", m[1][1], m[3][1]);
```

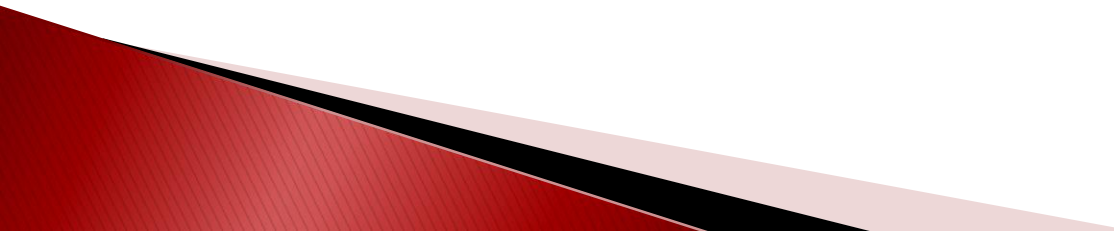
# Exercise

What is the result after the execution of this code?

```
double m[3][5]={5.5, 7.9, 5, 7, 34, 10, 11, -5,  
                0, 6.2, 0,15, 5, 5, 55.5};  
printf("%.3lf %lf \n", m[2][1], m[1][3] );
```



# Sorting algorithms

- ▶ Selection sort
    - Minimum selection
    - Maximum selection
  - ▶ Bubble sort
  - ▶ Insertion sort
  - ▶ QSORT function
- 

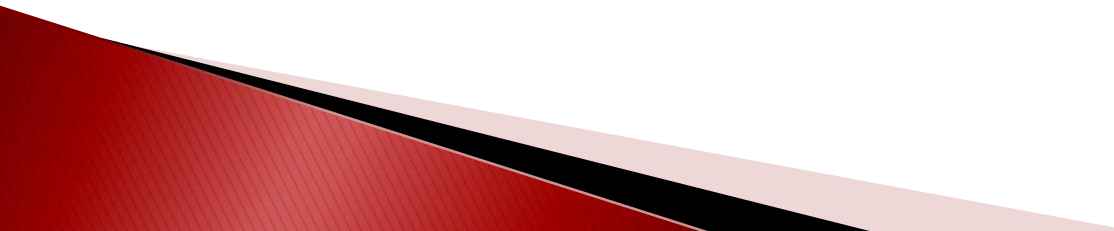
# Exchange the elements

**Swap (exchange) the two element (a, b) – using auxiliar variable (tmp)**

```
tmp = a
```

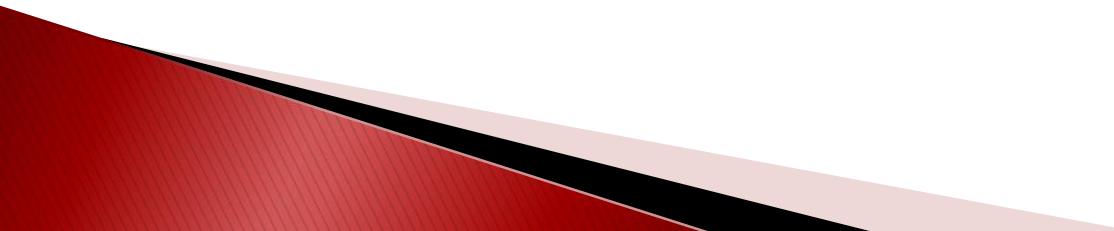
```
a = b
```

```
b = tmp
```



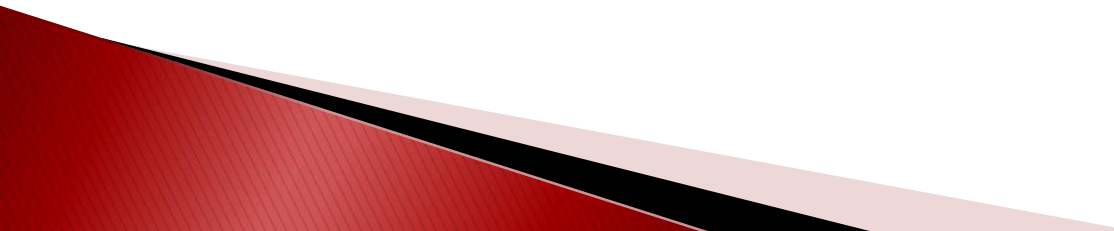
# SWAP – exchange two array variable values

```
void swap(int a[], int i, int j)
{
    int tmp;
    tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
}
```



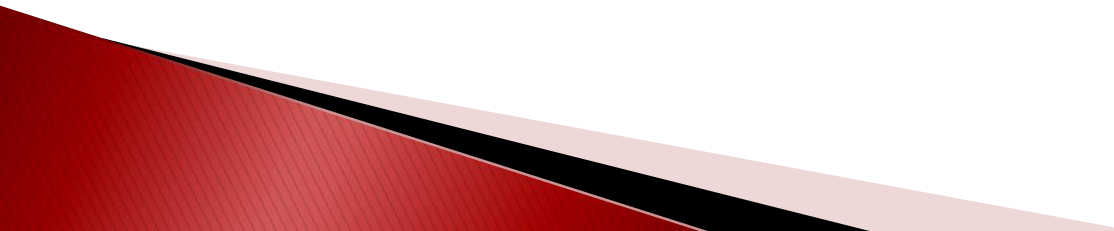
# Exercise

```
double a=-2536.234, b=-547.4, k;  
    k=a;  
    a=b;  
    b=k;  
printf("a= %lf b= %.2lf", a, b);
```



# Exercise

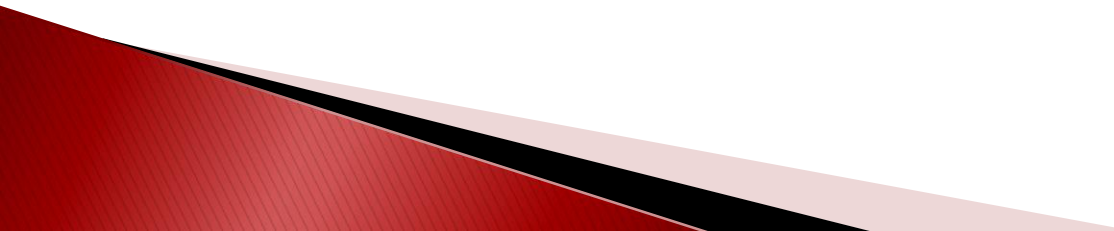
```
double a=5.12, b=42.23;  
    a=a-b;  
    b=b+a;  
    a=b-a;  
printf("a= %lf b= %.3lf", a, b);
```



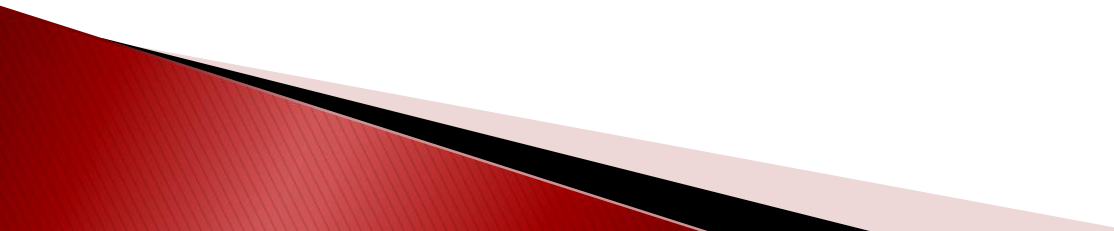
# Exercise

```
int a=12, b=26;  
    a^=b;  
    b^=a;  
    a^=b;  
printf("a= %X b= %o", a, b);
```

# Sorting algorithms

- ▶ Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order.
  - ▶ Most common orders are in **numerical** or **lexicographical** order.
  - ▶ A sequence of values is said to be in **increasing order**, if the successive element is greater than the previous one.  
For example, 1, 3, 4, 6, 8, 9 are in increasing order.
  - ▶ A sequence of values is said to be in **decreasing order**, if the successive element is less than the current one.  
For example, 9, 8, 6, 4, 3, 1 are in decreasing order.
- 

# Selection Sort – Minimum

- ▶ Selection sorting is conceptually the simplest sorting algorithm.
  - ▶ This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then finds the second smallest element and exchanges it with the element in the second position, and continues in this way until the entire array is sorted.
  - ▶ Sorting means **increasing order**.
- 



## Selection Sort.

comparisons

8 5 7 1 9 3

$(n - 1)$  first smallest

1 5 7 8 9 3

$(n - 2)$  second smallest

1 3 7 8 9 5

$(n - 3)$  third smallest

1 3 5 8 9 7

2

1 3 5 7 9 8

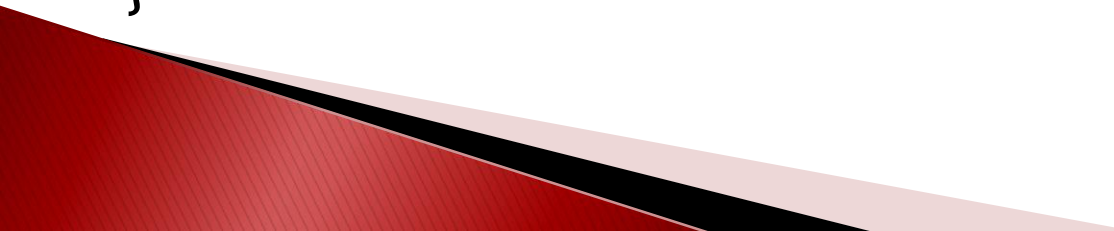
1

1 3 5 7 8 9

0

# Selection Sort – Minimum

```
void minimum_selection( int a[], int n )
{
    int i, j, min;
    for ( i = 0; i < n - 1; i++ )
    {
        min = i;
        for ( j = i + 1; j < n; j++ )
            if ( a[ j ] < a[ min ] )
                min = j;
        swap( a, i, min );
    }
}
```



# Bubble sort

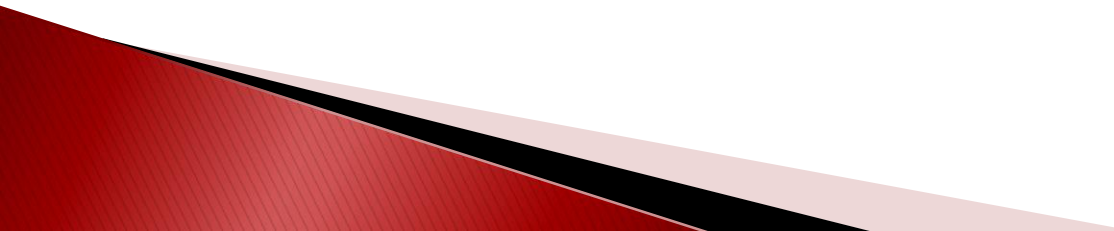
- ▶ This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared, and the elements are swapped if they are not in order.

# First pass

<div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div>									
54	26	93	17	77	31	44	55	20	Exchange
26	54	93	17	77	31	44	55	20	No Exchange
26	54	93	17	77	31	44	55	20	Exchange
26	54	17	93	77	31	44	55	20	Exchange
26	54	17	77	93	31	44	55	20	Exchange
26	54	17	77	31	93	44	55	20	Exchange
26	54	17	77	31	44	93	55	20	Exchange
26	54	17	77	31	44	55	93	20	Exchange
26	54	17	77	31	44	55	20	93	93 in place after first pass

# Bubble sort

```
void bubble_sort( int a[], int n )
{
    int i, j;
    for ( i = n - 1; i > 0; i-- )
        for ( j = 0; j < i; j++ )
            if ( a[ j ] > a[ j + 1 ] )
                swap( a, j, j + 1 );
}
```



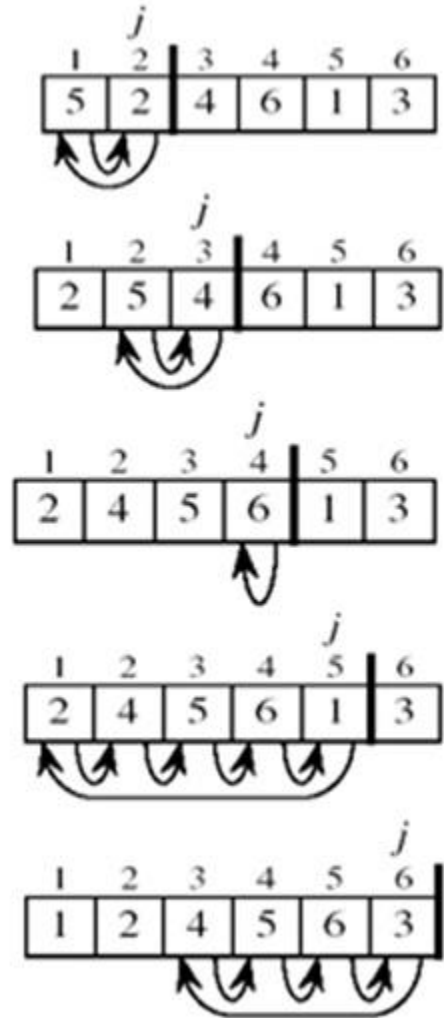
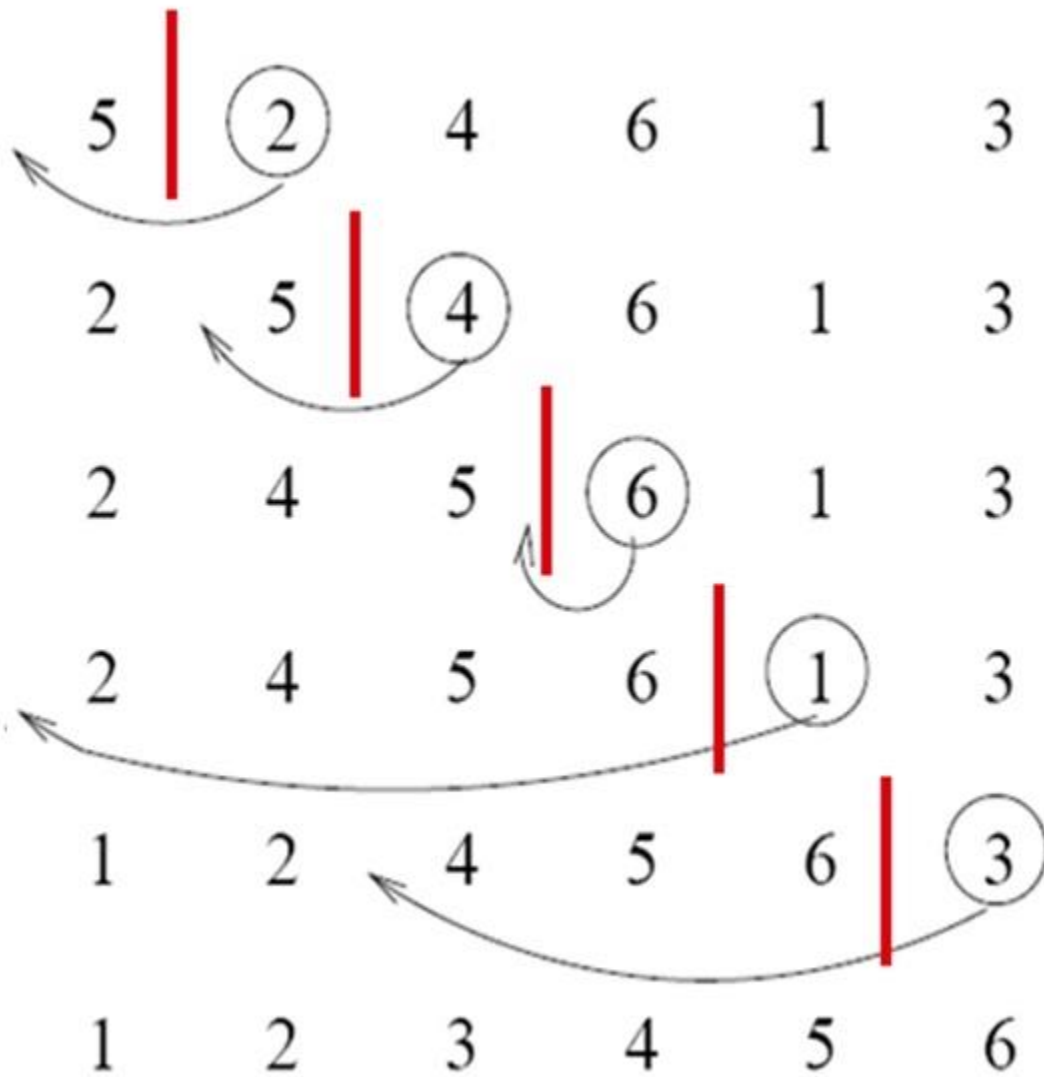
# Insertion sort



- ▶ Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.
- ▶ This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted.
- ▶ For example, the lower part of an array is maintained to be sorted.
- ▶ An element which is to be '**insert**'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there.
- ▶ Hence the name, **insertion sort**.
- ▶ The array is searched sequentially, and unsorted items are moved and inserted into the sorted sub-list (in the same array).

# Insertion sort – Steps

1. The first element is already sorted.
2. Pick the next element.
3. Compare with all the elements in the sorted sub-list.
4. Shift all the elements in the sorted sub-list that are greater than the value to be sorted.
5. Insert the value.
6. Repeat until the array is sorted.



**Done !**



# Insertion sort

```
void insertion_sort( int a[], int n )
{
    int i, j, key;
    for ( i = 1; i < n; i++ )
    {
        key = a[ i ];
        for ( j = i - 1; j >= 0 && a[ j ] > key; j-- )
            a[ j + 1 ] = a[ j ];
        a[ j + 1 ] = key;
    }
}
```