FUNDAMENTALS OF PROGRAMMING WITH C#

ARRAYS

Liu Fan
isslf@nus.edu.sg

Total:31

Objectives

- Create and initialize array to store multiple values
- Access and modify values of single and multi-dimensional arrays

Agenda

- Arrays
  - One-Dimensional
  - N-Dimensional
  - Declaration, Storage and Use
- Sorting Example

Indexed Variables

- The use of indexed variables is useful in some problem. For example, if the Daily Sales is to be stored for one full month, a single variable, say DS can be used with an index as: DS_i
  where i indicates the date.
  In this case DS_1 will depict the sales for Day 1, DS_2 for Day 2, DS_3 for day 3 and so on.
- The reason for writing them as suffixes rather than part of the name i.e., DS_2 instead of DS2 is because the former gives us the facility of manipulating the suffix independent of the name.
- This is useful when a list of values pertaining to a single fact is to be stored.
- The indexed variables are stored as arrays in computer.

Indexed Variables

- The merit of using indices rather than separate variables lies in the fact that we stick to a single variable and only change the index as required. This gives us the required flexibility.

- Arrays comprises of the variable name and index. The index (or suffix) is actually enclosed within simple parenthesis.
  - Eg.,

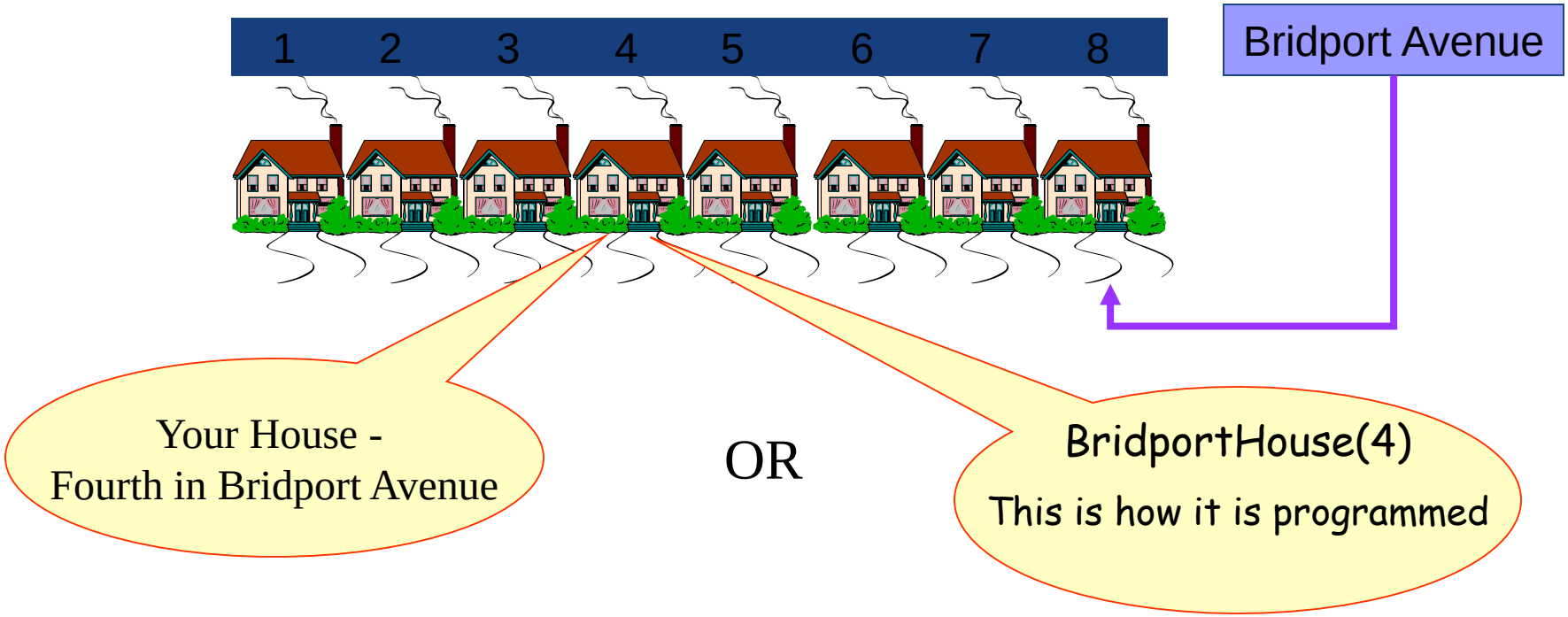
    Sales[Month] may contain the sales for different months Jan, Feb etc.

    Marks[Student,Subject] may contain the Marks for
    students 1, 2, 3 … N in various subjects A, B, C, D.
    In this case, it is a 2-D array with N rows and 4 columns

- In C#, the array indices are specified at the end of the variable name by enclosing them between a pair of square brackets.

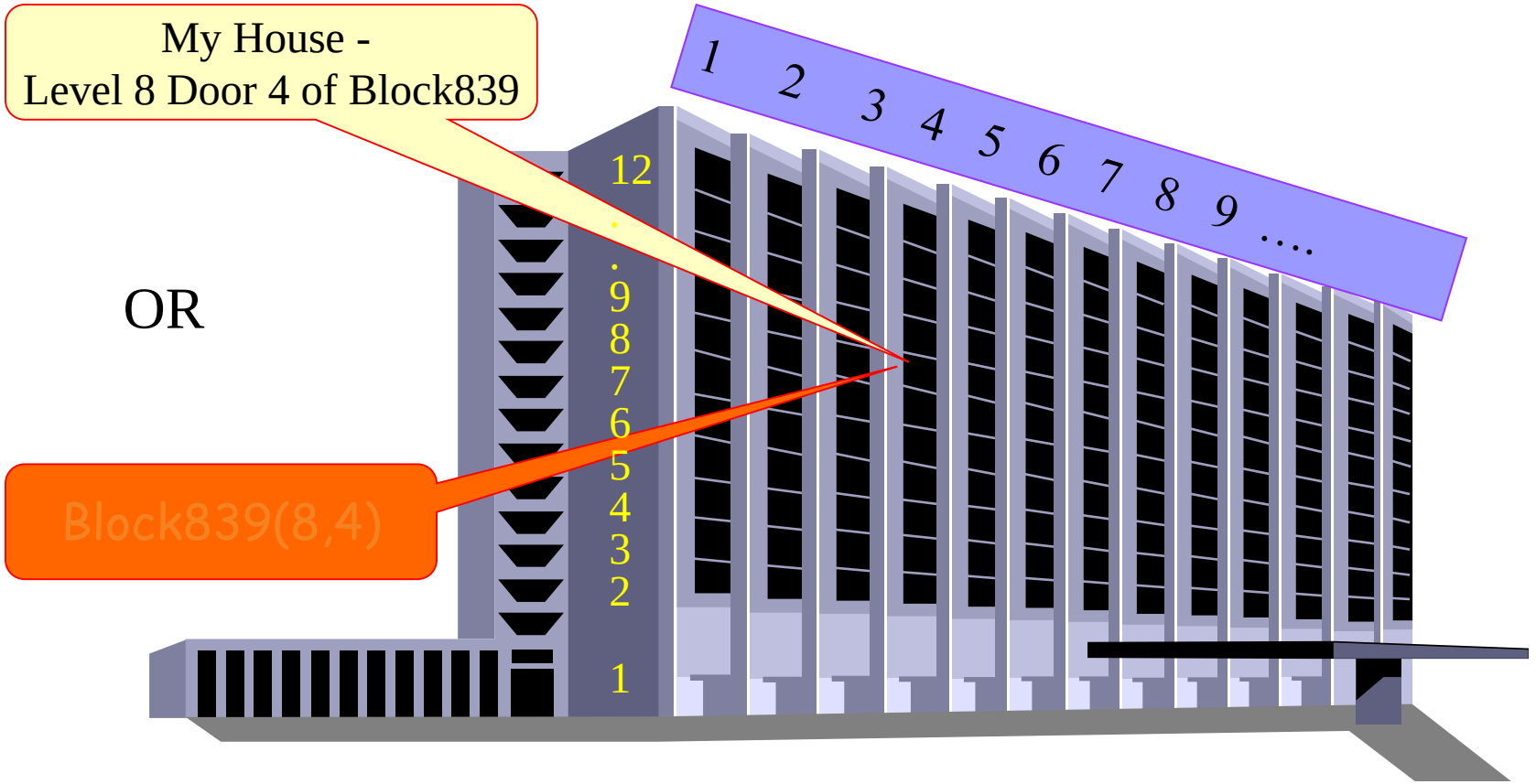
One Dimensional Array

- Suppose that you own the Fourth row house in Bridport Avenue then this is how it is represented:



Two Dimensional Array

- Your Instructor can't afford a house so he stays in the HDB level 8 house 4 of Block839

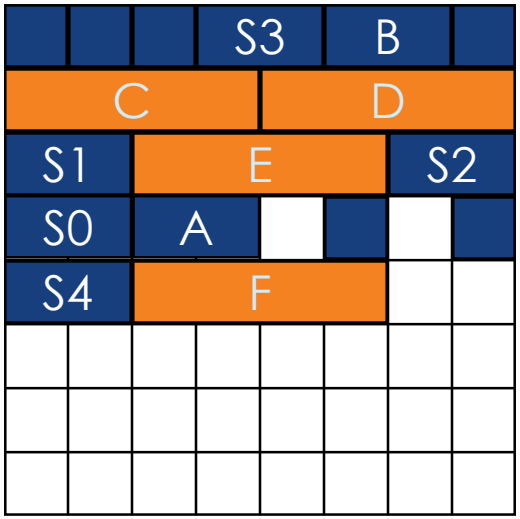


How computer keeps the arrays in Memory

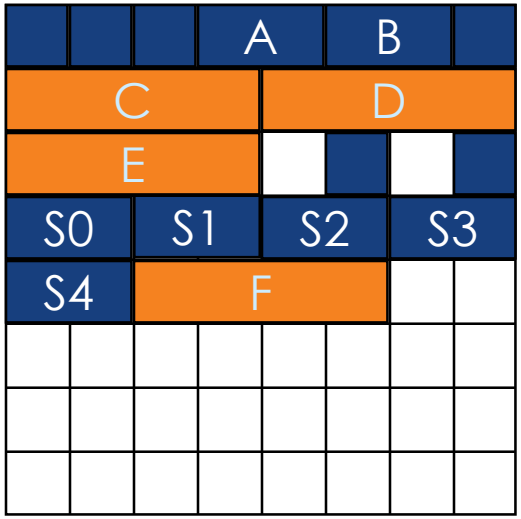
- Just as any other variable, arrays are allocated space when an array variable is declared.
- In a non-indexed variable, the memory is allocated based on the data type - for simplicity assume that a variable occupies one word of memory
- When an array is declared adjacent memory location are contiguously allocated one word for each element in the array.
- Hence it is necessary for programmers to declare the size of the array even at the start.
- For two dimensional arrays depending on the language, either a row wise allocation or column wise allocation is made.
- Arrays can have thousands of memory locations, but are addressed by a single variable name but with different identifying number. Based on this number (index), the computer locates the memory word using the start and offset.

Allocation for Arrays

- Non indexed variables storage
  - A,B are integers
  - C,D,E,F are double
  - S0, S1,S2,S3,S4 are integers



- Indexed variables storage
  - A,B are integers
  - C,D,E are double
  - S[] is an array of integers of size 5



Not Used Double String Integer

Each square is Two bytes

Declaring Arrays

- Just as with other variables, we need to declare arrays before we use them.
- The type of data stored in each array element is determined by the keyword used before the variable name as with any other variable declaration. Only addition is that the datatype is suffixed open/close square brackets
- There would be a open/close parenthesis for each index suffix.
- The size of the array is determined by:
  - the instantiation process, OR
  - data value supplied during the declaration.

Array Declaration Syntax

Method 1:

```
datatype[] VariableName;
        // declares variableName as an Array of given datatype
VariableName = new datatype[N];
    // instantiates the array to be having size N (int) with suffixes ranging 0 to (n-1)
```

Method 2:

```
datatype[] VariableName = new datatype[N];
        // declares variableName as an Array of given datatype and instantiates
           the array to be having size N with suffixes ranging 0 to (n-1)
```

Method 3:

```
datatype[] VariableName = new datatype[] {data1, data2, data3};
  // declares variableName as an Array of given datatype and instantiates the array
        to be having size 3 (determined by data provided) with suffixes ranging 0 to 2.
        Data are provided in braces at right of instantiation; dataN should be of same type.
```

Array Declaration Syntax

Method 1:

```
int[] A;
int[] Sales;
int[] C,D;
double[] E;
string[] EmployeeName;


A = new int[5];
Sales = new int[12];
C = new int[10];
D = new int[10];
EmployeeName = new  string[25];
E = new double[5];
```

Declarations

Instantiations

Array Declaration Syntax

Method 2:

```
int[] A = new int[5];
int[] Sales = new int[12];
int[] C = new int[10],D = new int[10];
double[] E = new double[5];
string[] EmployeeName = new string[25];
```

Caution:
Either initiate all array elements or don't initiate at all. Int[] A = new int[5]{11,2,4,5} will produce error

Declarations
&
Instantiation
performed in a single statement

Array Declaration Syntax

Method 3:

Declarations, Instantiation & Initialisation of Values performed in a single statement

```
int[] A = new int[5] {12,3,8,45,2};

double[] E = new double[5] {10.0,5.3,6.9,0.0,2};

string[] EmpName = new string[3]
                        {“Chris", "John", "Sabina"};
```

Index ->	0	1	2	3	4
Array					
A	12	3	8	45	2
E	10.0	5.3	6.9	0.0	2.0
EmpName	Chris	John	Sabina	n/a	n/a

Using an Array: Example

- Arrays can be used just like other variables, as long as you include their subscript.
- Example

```
int[] Marks = new int[3];
string[] StudentName = new string[3]
                          {“Chris", "John", "Sabina"};
int i;
double ClassAvg;
Marks[0] = 35;
Marks[1] = 82;
Marks[2] = 67;
ClassAvg = (Marks[0] + Marks[1] + Marks[2])/3.0;
for(i=0; i<=2; i++)
  Console.WriteLine("{0}\t{1}\t{2}",i, StudentName[i],Marks[i]);
Console.WriteLine("Average Marks for Class is {0}",ClassAvg);
```

Length

- Array in C# is treated as an Object
- Hence Array has some operations and properties
- To access the property of the Array, just use the array name; brackets and indices are not to be suffixed.
- Important Properties of Arrays are
  - Length:
    - Returns the size of the array
    - Most important property that is extensively used in programs, particularly in for loop construct as given below:

```
for (int j=0; j < arr.Length; j++)
{
   … …
}
```

Sorting Example

- A simple naïve sort algorithm:
  Sort a list of numbers (in the range 0 to 100) in ascending order.
  - Read in the list of numbers into an array, say A_i until -999 is entered
  - Define another array, say B_i where the sorted values are to be stored.
  - Go through entire array A and find the smallest number and place it as the first element in array B. Make this number in array A as BIG say 999999.
  - Repeat the above step to determine the next smallest number and place it as second element. This is repeated until all the elements are placed at their appropriate location in array B.
  - The array B, which is in sorted order is printed out.

Note:

Please note that the above is a very naïve algorithm and should not be taken as a standard algorithm. The algorithm is used here due to its simplicity and ease of understanding so that the concept of nested loops can be adequately demonstrated. More Efficient Algorithms will be formally introduced later.

Sorting

- What a sort program does:

- We had seen Flow Charting a naïve algorithm. Please revise the algorithm and develop a C# program.

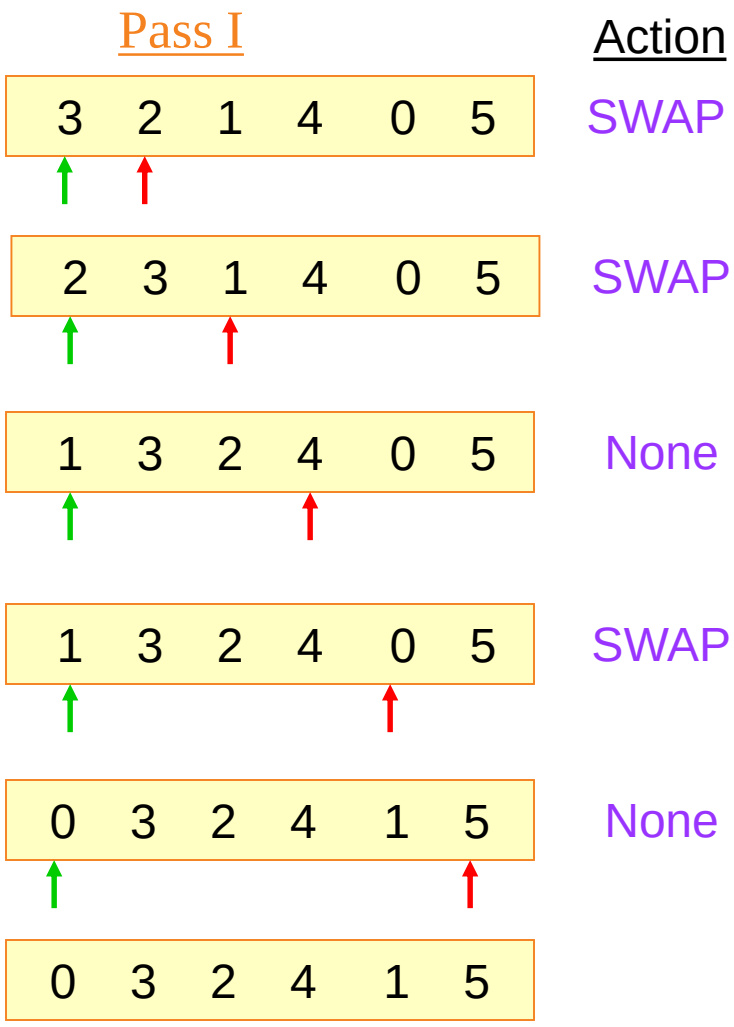
- In the previous illustration we had a lay person approach, let's now consider a more structured sort methodology as given in literature. This again is a straight forward algorithm and we restrict to this algorithm and discussion on more efficient algorithms are beyond the scope of the current module.

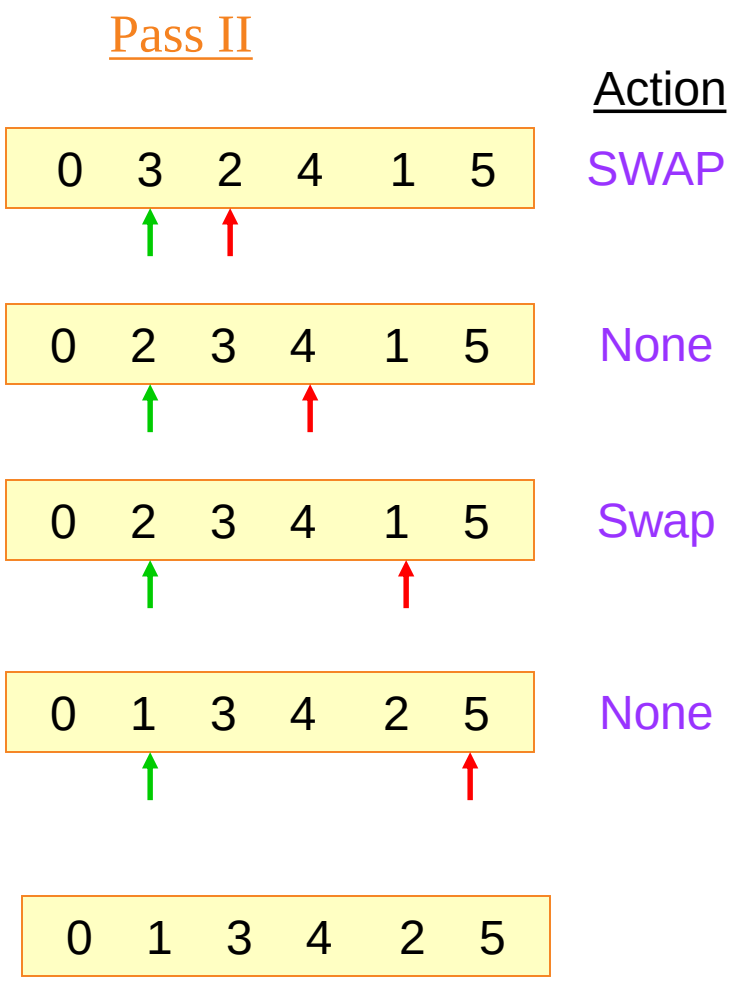
Simplified Selection Sort Algorithm The Steps

- Selection sort “selects” the smallest element and places it in the first element by swapping the elements. Repeats this for other elements until the array is fully sorted.

    - Start with Unsorted numbers.
    - Assume the first number is the smallest.
    - Compare it to each of the numbers remaining in the array.
    - If any numbers are smaller than the first number, swap them with the first position.
    - Now you know the smallest number is at the beginning of the array.
    - Repeat this process (for second, third etc. location) ignoring elements you’ve already put in the proper place.

Sort in Action!

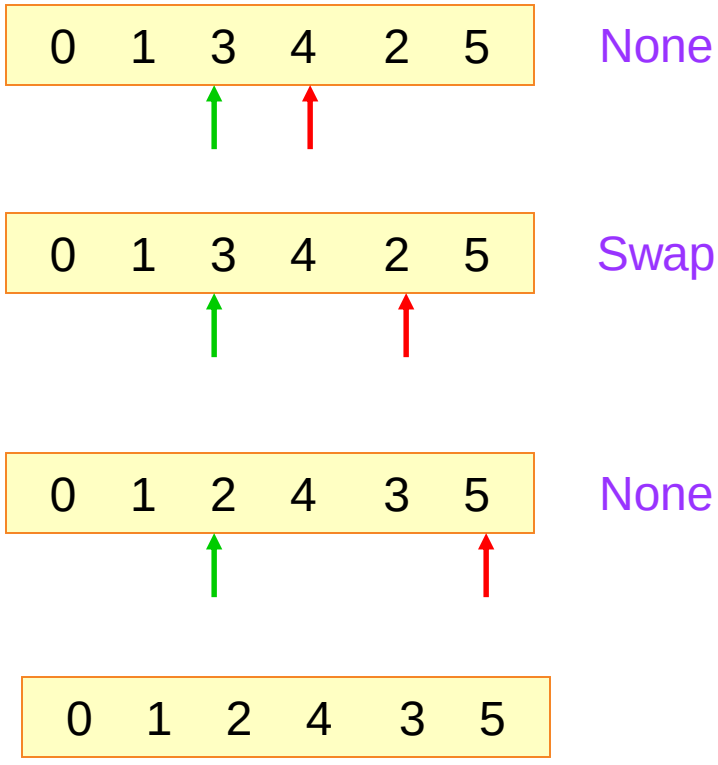




Sort in Action!

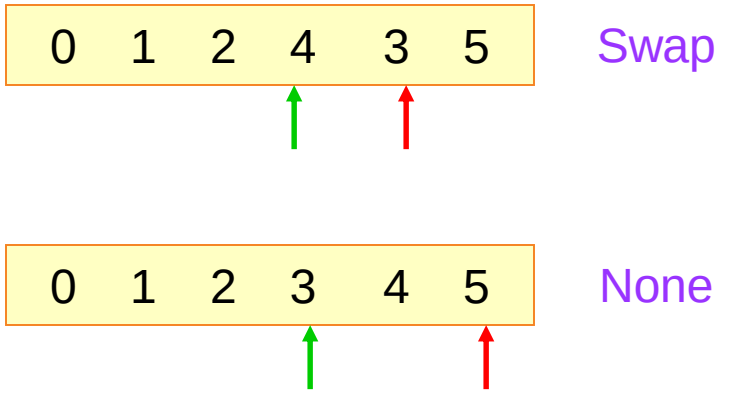
Pass III

Action



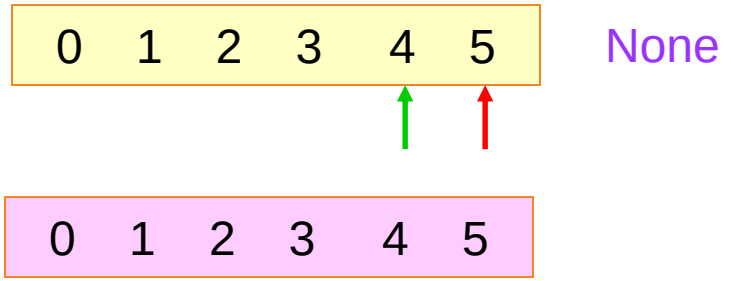
Pass IV

Action



Pass V

Action



Sort in Pseudo Code

- Pseudo Code:

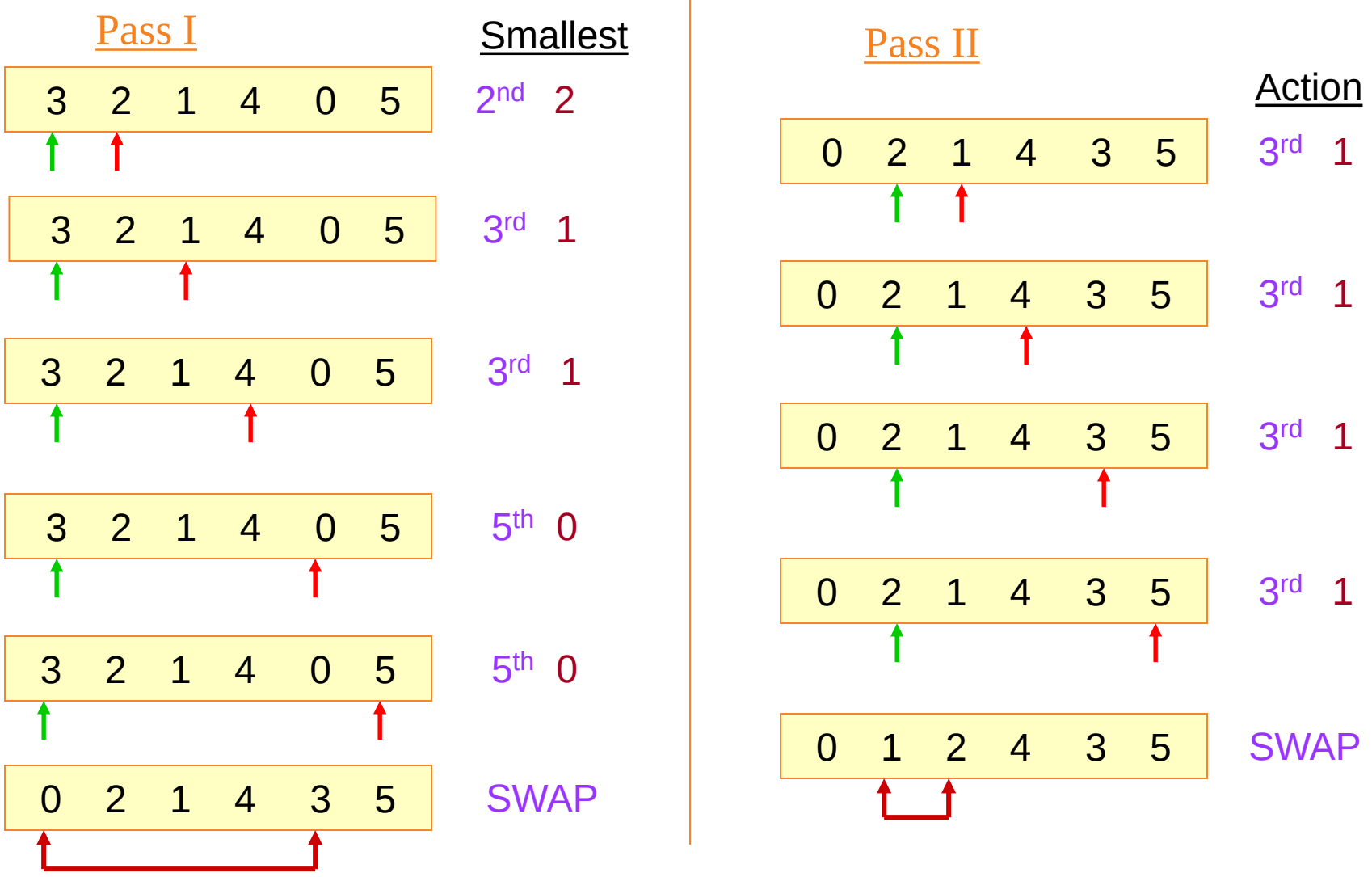
```
for i = the beginning of the array to the end
    'Look for the smallest element and put it in the
     i'th element (so the i'th element is the first)
   for j = i to the array end
          'Starting at i means we skip those elements
                                  already in place.
     if (i'th element > j'th element) then
        swap the j'th and i'th element
     endif
   next j
next i
```

Refined Selection Sort Algorithm The Steps

- Refined selection sort “selects” the smallest element and places the location (index) of the smallest number in a temporary variable; and at the completion of a Pass, swaps the smallest number so identified with the first element. Repeats this for other elements until the array is fully sorted.

  - Start with Unsorted numbers.
  - Assume the first number is the smallest.
  - Compare it to each of the numbers remaining in the array.
  - If any numbers are smaller than the first number, write the location of the smallest number in a temporary variable and keep track of the smallest value.
  - At the end of the pass, swap the smallest number with the first element.
  - Now you know the smallest number is at the beginning of the array.
  - Repeat this process (for second, third etc. positions) ignoring elements you’ve already put in the proper place.

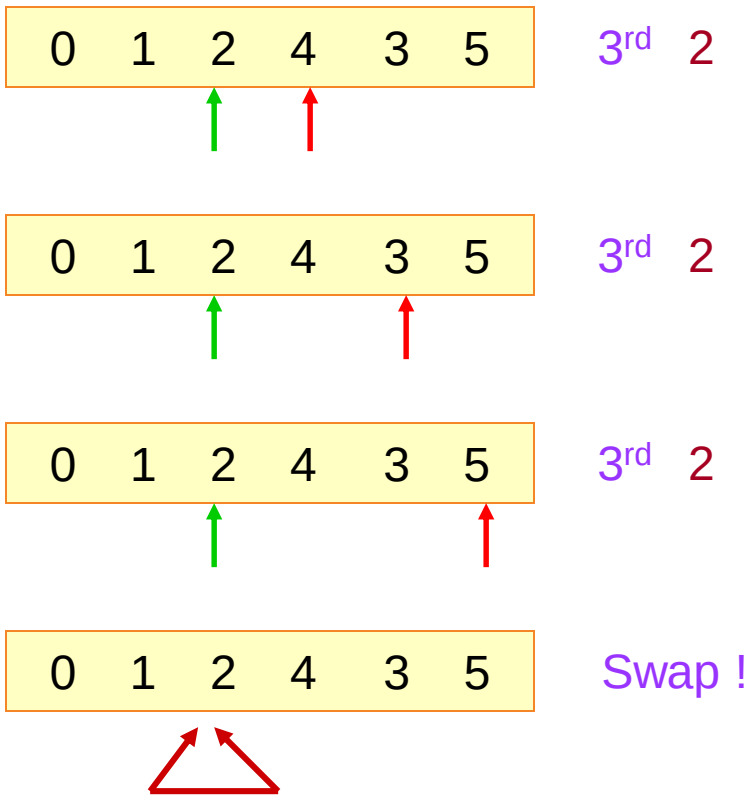
Sort in Action!



Sort in Action!

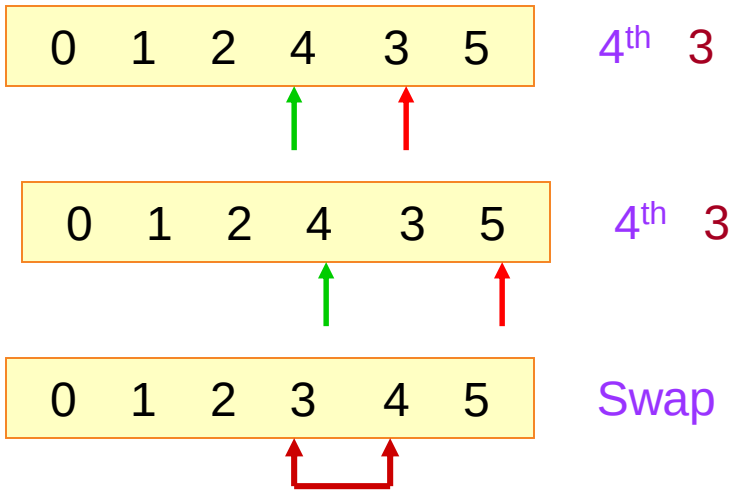
Pass III

Action



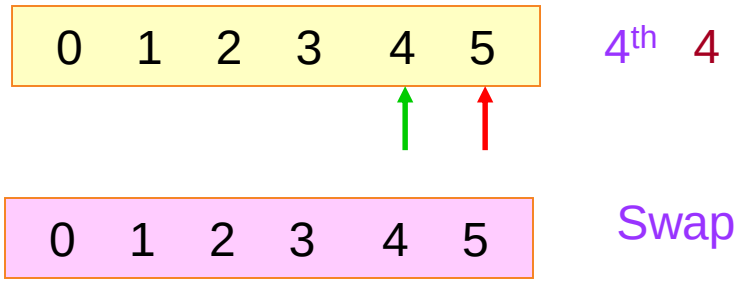
Pass IV

Action



Pass V

Action



N-Dimensional Arrays

- N-dimensional arrays are those that have more than one suffixes / indexes
  - Eg:

    M_{ij} OR X_{klm}
- There are two ways of declaring Multi-dimensional arrays in C#
  - Multidimensional Arrays
    - These are matrices (rectangular array) having fixed length & breadth.
    - These have multiple indices in square bracket separated by comma
    - Eg:

      M[i, j] OR X[k, l, m]
  - Jagged Arrays *(we would not discuss this in this module)*
    - These are like arrays within array
    - So it is possible to have different sizes of second index for each index of first element. That is for each row the number of columns can vary.
    - These have multiple indices represented as separate square bracket suffix after the array name. Eg.

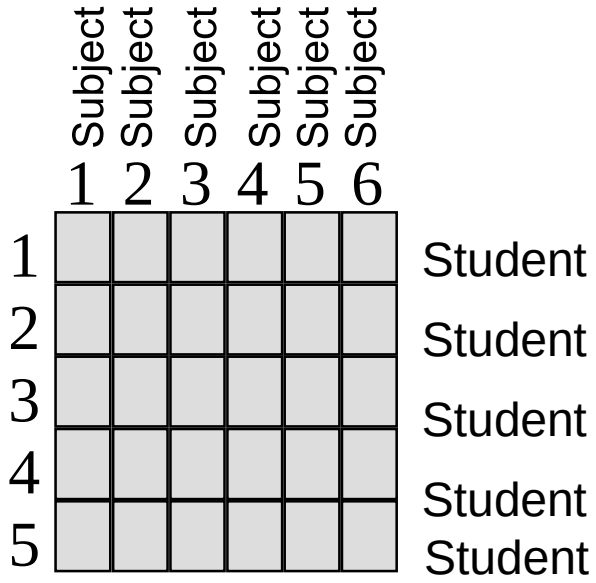
      M[i] [j] OR X[k] [l] [m]

Declaring a Multi-Dimensional Array

- The syntax is similar to a one dimensional array, but there is an additional term added
- These declaration will give us 5 rows and 6 columns to keep data in. Each row might be a student, each column a Subject. The student Marks are stored in this array in double data type.

```
double[,] Marks;
Marks= new double[5,6];
```

```
double[,] Marks = new double[5,6];
```



Multi-Dimensional Array : Initialising

- Method 1
  - Declaring/instantiating in one statement and initialising separately.

```
int[,] Marks = new int[3,2];
Marks[0,0] = 35;     Marks[0,1] = 82;
Marks[1,0] = 67;     Marks[1,1] = 45;
Marks[2,0] = 62;     Marks[2,1] = 77;
```

- Method 2
  - Declaring, instantiating and initialising in one statement

```
int[,] Marks = new int[,] { {35,82}, {67,45}, {62,77} };
```

Multi-Dimensional Array: Example

```
int[,] Marks = new int[,] { {35,82}, {67,45}, {62,77} };
string[] StudentName = new string[]{“Chris", "John  ", "Sabina"};
int i=0,j=0 ;
double ClassAvg=0, ClassSum=0;

for(i=0; i < 3; i++)
{
  Console.Write("{0} \t {1} ",i, StudentName[i]);
  for (j=0; j < 2; j++)
  {
        ClassSum = ClassSum + Marks[i,j];
        Console.Write("\t {0}",Marks[i,j]);
  }
  Console.WriteLine();
}

Console.WriteLine();
ClassAvg = ClassSum / (i*j);
Console.WriteLine("Average Marks for Class is {0}",ClassAvg);
```

Multi-Dimensional Array: Example

Output:

```
0       Chris       35      82
1       John         67      45
2       Sabina       62      77

Average Marks for Class is 61.3333333333333
```

Summary

- Array has a special declaration syntax
- Array elements can be accessed and modified by using the index and square bracket [ ]
- For multi-dimensional array, we write the multiple indexes separated with comma
- For multidimensional array, we use GetLength(0) to get the length of the first dimension, GetLength(1) for the length of the second dimension and so on.