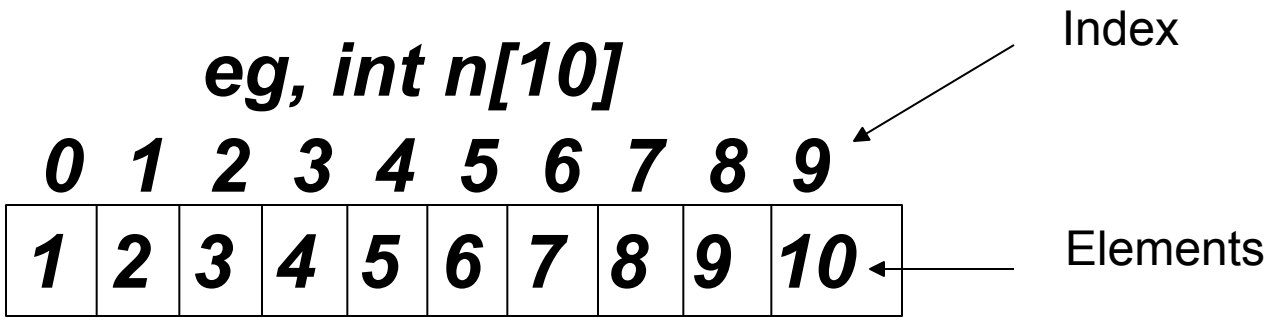
Data Structures

Lecture 03: Arrays

Data Structures

Array

- A ***finite ordered*** set of ***homogeneous*** elements
- Finite - specific numbers of elements
- Ordered - arranged one after another
- Homogeneous - similar types
  - All integers, all floating points etc.



Array

- Array can be initialized only during declaration time

  eg, int n[10] = {1, 2, 3 …9};

  eg, int n[ ] = {1, 2, 3 …9};

- There is no bound checking concept for arrays in C (any no. of values can be entered)

Array Operations

- Two basic operations are defined
- Extraction – accepts an array a, an index i, and returns an element of the array
- Storing – accepts an array a, an index i, and an element x (a[i] = x)
- Operations in arrays are vary fast

Limits in Array

- Lower bound - smallest array's index i.e 0
- Upper bound - highest array's index
- Range - Upper bound – Lower bound + 1
- Lower and upper bound never changes during program execution

1 D Array

- Only one subscription is needed to specify a parameter element of the array
- eg. datatype variable[expression]

eg, int n[10] ← ***Only one subscript i.e, 10***

- Size in bytes = size of array * size of (base type)
- eg, float var[5]
- Size of var = 5 * 4 bytes = 20 bytes

Implementation in memory

- Address of an element n[k] = B + W*k
- B – Base address
- W – size of datatype
- k- array index
- Eg. n[5]
  - Let B = 2000, W = 2 bytes and k = 5
  - So, address of n[5] is 2000 + 2*5 = 2010

Traversing and Merging

- Traversing
  - Accessing all the elements of the array from first element up to the last one by one

- Merging
  - Adding elements of one array (a) into another array (b)
  - Merging can be done in the new array (c)

Arrays as Parameter

- Arrays can be passed to function in two ways;
  - Passing as the array
    - Passing the whole array to the function makes another copy of the same array, this causes unwanted duplication
  - Passing as pointer to the array
    - Since an array variable is a pointer, array parameters are passed by reference, i.e, base address of the array is passed

Example

- Passing by reference
  - Float avg(float a[], int size)

```
{
  int i;
  float sum = 0;
  for(i=0; i<size; i++)
            sum+=a[i];
  return (sum/size);
}
```

Function call:

```
  average = avg(a, size);
```

Array as an ADT

ADT:

- A useful tool for specifying the ***logical properties*** of a data type.
- Type defined for the ***data*** that contains values and set of operations

  An ADT consists of
- **Data definition** (Data Holder)
  - Pre – conditions
  - Post – conditions (Situation of the data holder after calculation)
- **Operator definition**

Array as an ADT

<ADT definition> Array

Dataholder[items]	//Holds data
Index	//serial number of items

<process>Process name:<returns none>**Store** (Array, I, element)
Pre-condition: the index should be within the range
Method:
Post-condition:Array[i]=element
<end process>

<process>Process name:<returns item>**Extract** (Array, i)
Pre-condition:the index should be within the range
Post-condition: Extract=Array[i]
<end process>

<end definition>

2-D Array

- Defined as arrays of array
- The component type of an array can be another array
- eg, int a[3][4];
  - Array containing 3 elements, and each of these elements is itself an array containing 4 elements

2-D Array

- int a[3][4]

	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1			a[1][2]	
Row 2				

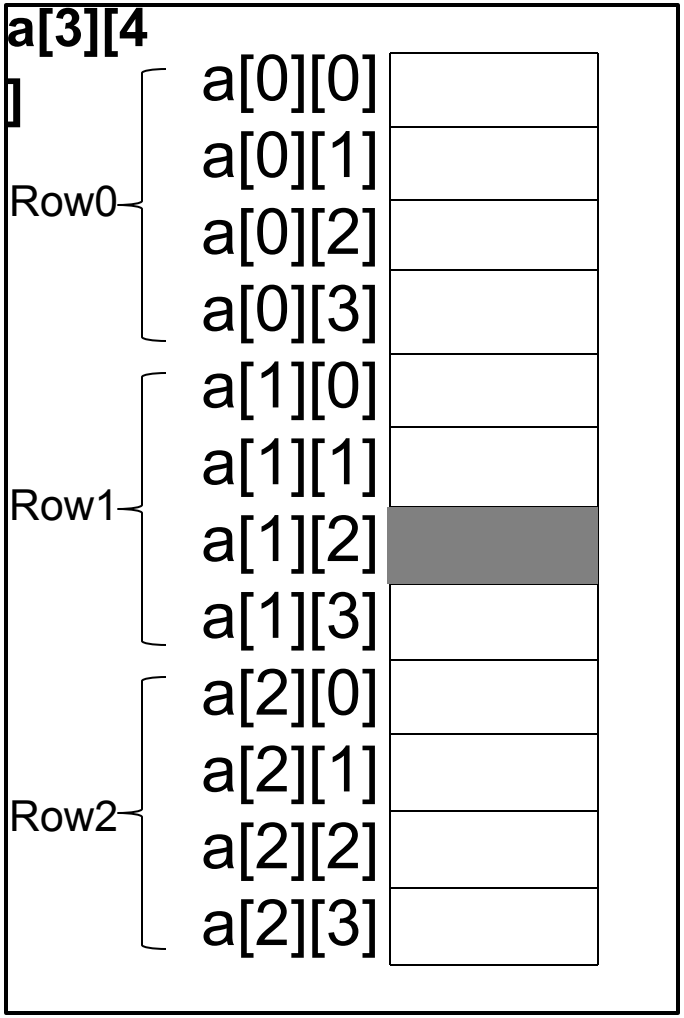
- It is only a logical data structure that is useful in programming and problem solving

Representation of 2-D Array

- **Row-major representation**
  - First row of the array occupies first set of memory locations, second occupies second and so on.

    eg arr[r][c]
  - Address of arr[i][j] is given by **base(arr)+(i*c+j)*esize**

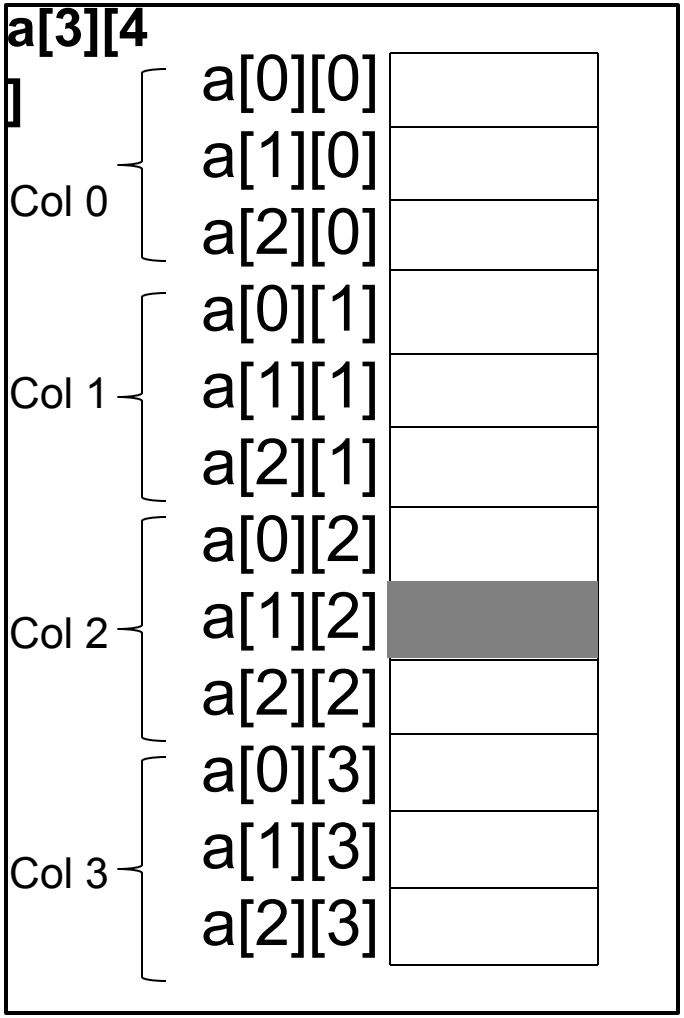
    i.e base(arr)+(1*4+2)*esize



Representation of 2-D Array

- **Column-major representation**
  - For column major representation, memory allocation is done column by column, ie, frist the element of the complete first column is stored, then elements of second and so on. **eg arr[r][c]**
  - Address of arr[i][j] is given by **base(arr)+(j*r+i)*esize**

    i.e base(arr)+(2*3+1)*esize



Class Assignment

1. Write a program to sum two 1 dimensional arrays and store the sum of the corresponding elements into third array and print all the three arrays
2. Write a program to find the smallest and largest number in an array of size 10
3. Implement the above 2 programs using function call

The End