MODULE-I

➢ Review Of Basic Data Structures

1. **Array**
2. **linked list and its variants**
3. **Stack**
4. **Queue**
5. **Trees**

Variable

- ✓ Placeholder/Structure to store a basic unit of data.

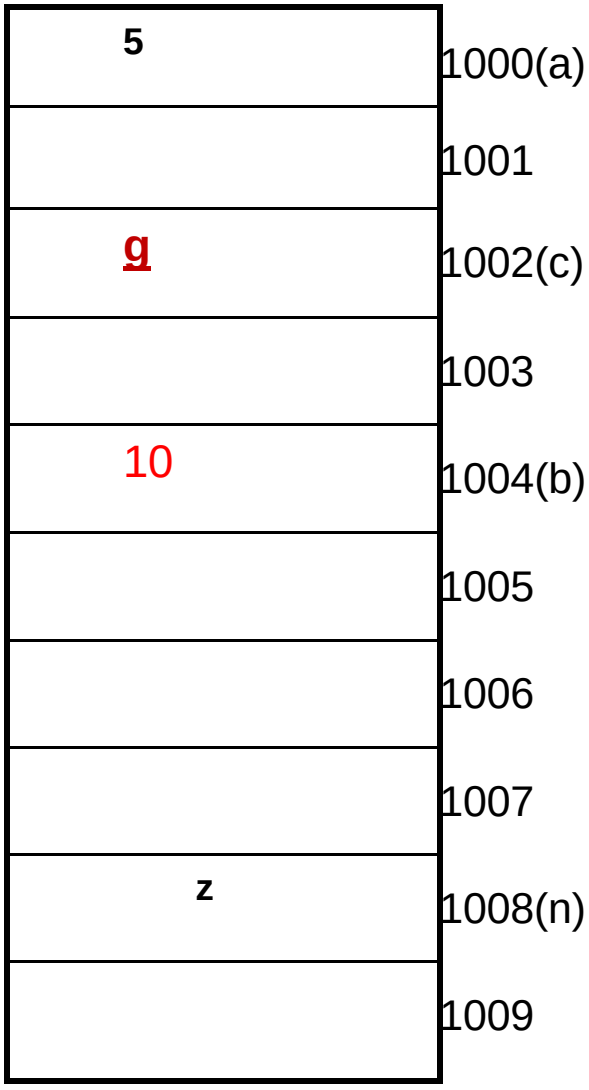
  Example

  - 1 integer variable stores single integer data.
  - 1 float variable stores single float data.

- ✓ Question is:
  - In a computer, where is that value assigned to a variable stored?
    - In memory, possibly Random Access Memory (RAM)
  - But where exactly in memory?
    - On some physical address in memory because,
      - » Memory is divided in physical locations and,
      - » Each location has a particular address associated with it.

EXAMPLE

- **int a = 5;**
- **int b=10**
- **char c='g'**
- **char n = 'z';**



✓ Sometimes in our program there might be a need to store multiple data of similar type.

E.g. Roll numbers of 5 students.

- ✓ 2 solutions:
    1. Create different variables each having a different name.

        int rollnum1, rollnum2, rollnum3, rollnum4, rollnum5

    2. Create a collection of variables referred by a common name.

        int rollnum[5] ⟶ This looks much better solution

Array Data Structure

- An array is a Finite, Ordered collection of Homogeneous data elements stored at contiguous/continuous memory locations.
- The idea is to store multiple items of the same type together.

Properties:

- Finite:
    - ✓ Contains only a limited (finite) number of elements.
- Ordered:
    - ✓ All elements are stored one by one in continuous / contiguous locations of computer memory.
- Homogeneous:
    - ✓ All the elements of an array are of the same data type.

Example

- An array of integers to store the roll numbers of all students in a class.
- An array of strings to store the names of all villagers in a village

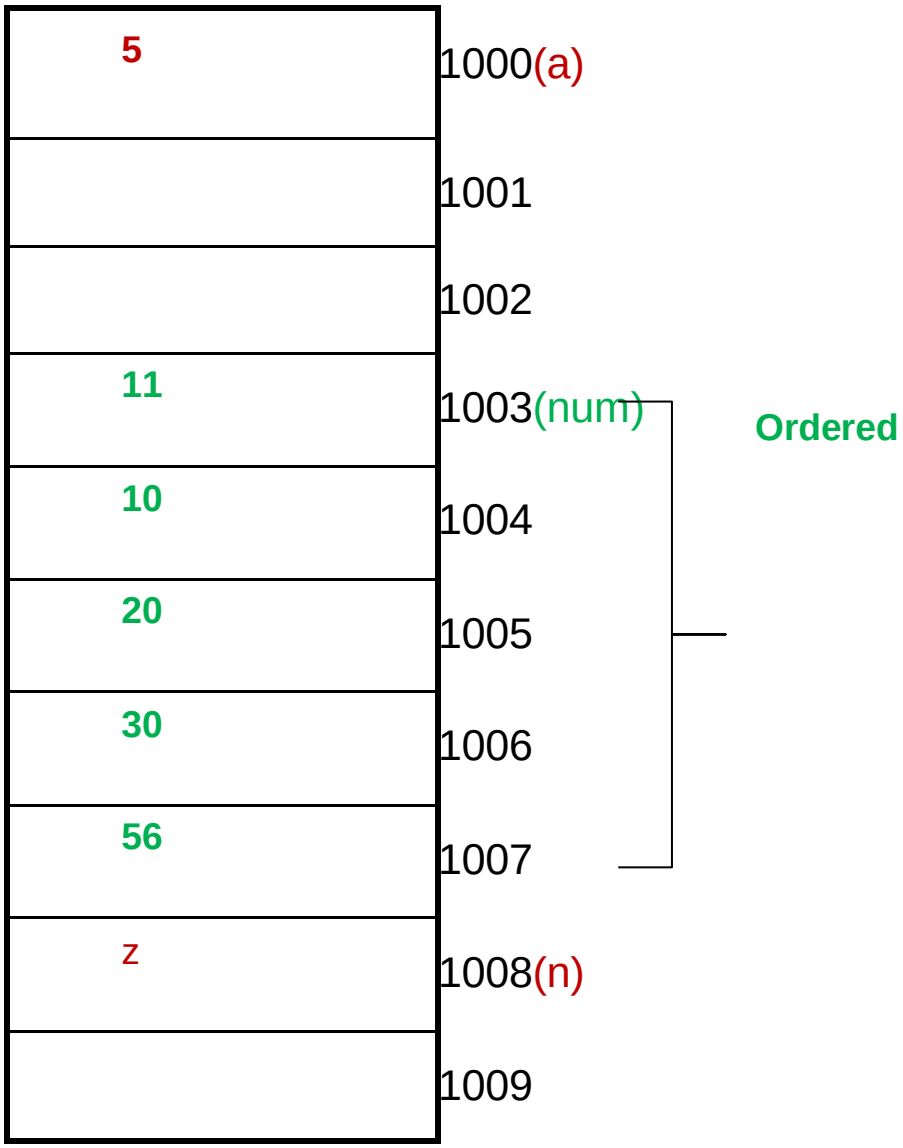
- int num[5] = {11, 10, 20,30,56};

Finite **same data type (Homogeneous)**

int a=5

char n='z'

other variables



Terminologies:

Size:

- Number of elements in an array.
- Also called:
  - Length, Dimension.
- Example:
  - int A[5]
    - » Size is 5.

Type:

- Kind of data type it is meant for.
- Example:
  - int A[5]
    - » Type is integer.

Base / Base Address (M):

- Address of the memory location where the first element of the array is located.
- Denoted by symbol M.

Index (i):

- Subscript by which the elements in an array can be referenced / accessed.
- Each element can be uniquely identified by their index in the array.
- Always an integer value.

Lower Bound (L):

- Starting index of an array.
- Denoted by symbol L.

Upper Bound (U):

- Ending index of an array.
- Denoted by symbol U.

Range of Indices:

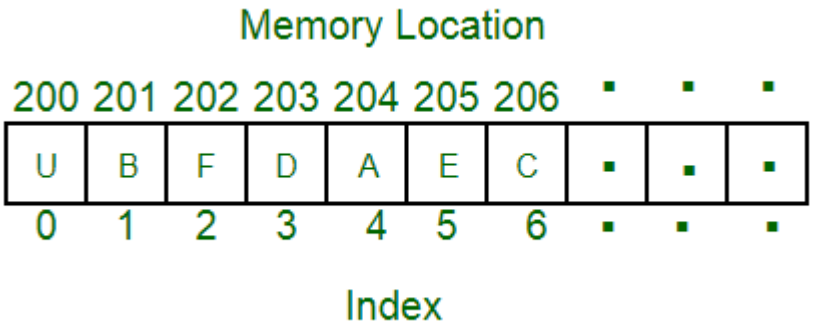
- Depends on the programming language in which the array is created.
  - In C,
    - » Index always starts from 0 upto Size-1.

Position (P):

- Relative rank of an element in the array.
- Does not depend on the programming language.
- Example:
  - A_1: First element in the array.
  - A_5: Fifth element in the array.

Word Size (W):

- Size of one element in the array.
- Denoted by symbol W.



Formulas / Equations:

1. **Size = U – L + 1**

   U: Upper Bound of the array.

   L: Lower Bound of the array.

2. **Index or i = L + P – 1**

   L: Lower Bound of the array.

   P: Position of element in the array.

3. **Address = M + (i – L) * W**

   M: Base address of the array.

   i: Index of the element.

   L: Lower bound of the array.

   W: Word size of the array.

   Also called: Indexing Formula

EXAMPLE

SIZE

Array A [0...4]

L = 0

U = 4

Size=U-L+1=4-0+1=5

INDEX

Position (P)	Index (i)
1	0
2	1
3	2
4	3
5	4

Index i = L + P – 1

Index A(1) = 0+1-1=0

Index A(2) = 0+2-1=1

ADDRESS

Array A[0...4]

L = 0, U = 4, M=1000, W=2

M is the Base Address

i - index

L –lower bound

W-Word Size

Address = M + (i – L) * W

Address (A[0]) = 1000 + (0-0)*2=1000

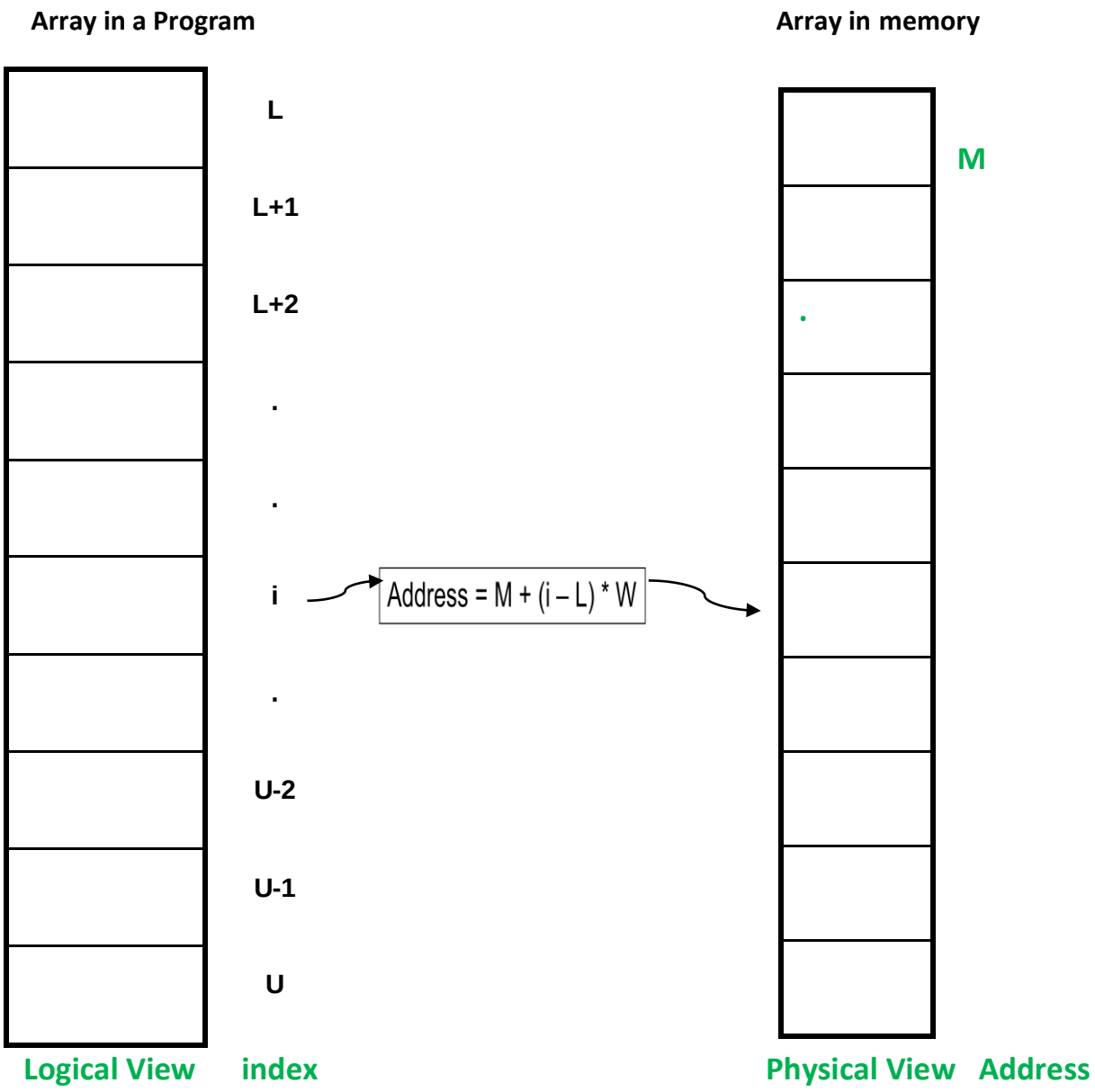
i=0,L=0

Address (A[1]) = 1000 + (1-0)*2=1000+2=1002

Will Word Size (W) matter?

YES

ARRAY (INDEXING FORMULA)



Array-Basic Operations

- Following are the basic operations supported by an array.

1. **Traverse** − print all the array elements one by one.
2. **Insertion** − Adds an element at the given index.
3. **Deletion** − Deletes an element at the given index.
4. **Search** − Searches an element using the given index or by the value.

Traverse Operation

- This operation is to traverse through the elements of an array.
- Visit each and every element of the array

Algorithm: TraverseArray

Input:

- Array A with elements.

Output:

- According to Process ().

Data Structure:

- Array A[L...U]

Algorithm: TraverseArray

Steps:

```
i = L

While i <= U, do

        Process (A[i])
```

i = i + 1

End While

Stop

Insertion Operation

- Insert operation is to insert one or more data elements into an array.
- Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Algorithm:

– InsertArray.

Input:

– Array A with elements.
– KEY: Element to be inserted.
– LOCATION: Index where KEY is to be inserted.

Output:

– On Success, Element with value KEY inserted at index LOCATION.
– On Failure, appropriate message to be displayed.

Data Structure:

– Array A[L...U]

Deletion Operation

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Algorithm:

DeleteArray.

Input:

- Array A with elements.
- KEY: Element to be deleted.

Output:

- On Success, Element with value KEY deleted from the array.
- On Failure, appropriate message to be displayed.

Data Structure:

- Array A[L...U]

Search Operation

- You can perform a search for an array element based on its value or its index.
- Search for an element and tell whether it is present in the array or not

Algorithm:

- SearchArray.

Input:

- Array A with elements.
- KEY: Element to be searched.

Output:

- On Success, appropriate message and return Index/Location of KEY in array A.
- On Failure, appropriate message and return NULL.

Data Structure:

- Array A[L...U]