Page No.
Date : / /

ASSIGNMENT - 4

1) Draw diagramatic layout after reading statements

```
int roll = 1;
const int No = 10;
const char CH = 'A';
const int arr [6] = {10, 20, 30, 40, 50, 60};
```

1) int roll | 1 | 100 104 — sizeof(roll) = 4 bytes

Read : 'Roll' is a variable of type int, initialized with value '1'

2) const int No | 10 | 200 204 — sizeof(No) = 4 bytes

Read : No. is a constant variable of type integer initialized with value '10'.

3) const char CH | A | 300 301 — sizeof(CH) = 1 byte

Read : 'CH' is a variable of character data type, initialised with value 'A'.

4) const int arr

0	1	2	3	4	5
10	20	30	40	50	60
400	404	408	412	416	420

sizeof(arr) = 48 bytes

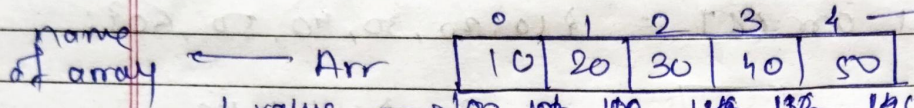
Read : arr is one-dimensional array, containing 6 element of int data type initialized with 10, 20, 30, 40, 50, 60.

2) Explain one dimensional array with example & diagramatic layout :

→ - One dimensional array contains series of elements of same datatype

- 1st element = 0th element index
last element = (n-1)th element index [n = length of array]
- memory of all elements gets stored is a sequential manner

int Arr[5] = {10, 20, 30, 40, 50};



One dimensional array has 1 row but multiple column.

3] Explain concept of 2D Array, examples, & layout.
→ - We can create 2D array as Arr[a][b]
Read:
Arr is 2D array which contains a 1D arrays in it, each 1D array contains 3 elements, each element is of type integer.

a) member initialization list: int Arr[3][3] = {1,2,3,4,5,6,7,8,9};

Columns[3]

Rows[3]	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

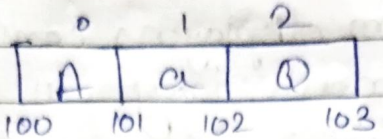
b) Member by member initialization:

Arr[0][0] = 1	Arr[1][0] = 4	Arr[2][0] = 7
Arr[0][1] = 2	Arr[1][1] = 5	Arr[2][1] = 8
Arr[0][2] = 3	Arr[1][2] = 6	Arr[2][2] = 9

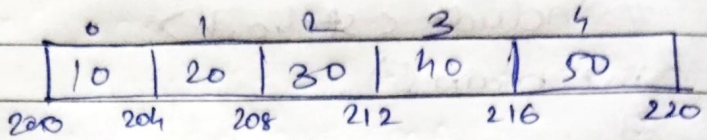
5) Program to create array of all Primitive data type, draw diagramatic layout.

→
```
#include <stdio.h>
int main()
{
    Char Arr[3] = {'A', 'a', '0'};
    int Brr[5] = {10, 20, 30, 40, 50};
    float Crr[3] = {10.1, 20.1, 30.1};
    double Drr[2] = {10.123, 20.567};
    return 0;
}
```

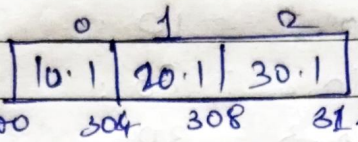
Char Arr



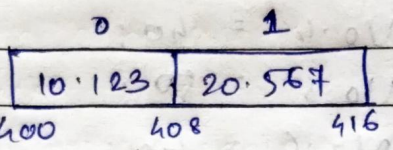
int Brr



float Crr



double Drr



6] Explain different approaches (ways) to create array in C / C++

1] With size & initialization list:

- By specifying length of array & initializing using member initializ^n list:

int Arr [5] = {1, 2, 3, 4, 5};

Syntax:

Datatype Arrayname [length] = {V1, V2 ... Vn};

2] Array initialization with length & declarat^n:

- If we initialize array using member initializ^n list then it is optional to specify length of array in [].

because, array is automatically calculated by compiler

int Brr [] = {1, 2, 3};

Syntax:

Datatype Arrayname [] = {V1, V2 ... Vn};

3] Declaration & initializ^n with less members:

Matrix

Rows [

Read:

8]

→

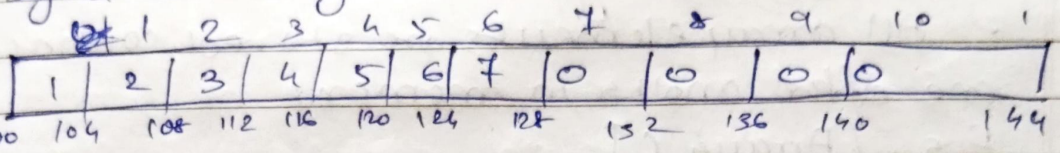
2]

3]

- Possible to specify length of array with some value but initialize with less no. of elements in it.

int Cr [11] = {1, 2, 3, 4, 5, 6, 7};

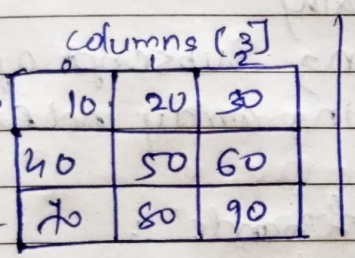
Syntax: Layout



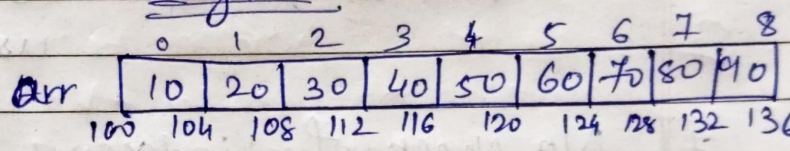
7] Draw diagrammatic layout & matrix of below array

int arr[3][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90};

Matrix



Layout:



size of (arr) = 36

Read:
- It is a 2 dimensional array, have 3 columns & 3 rows.
- Arr is a 2D array which contains 3 1D arrays in it, each 1D array contains 3 elements, each element is of type integer.

8] Note down properties of Array in C/C++;

→ 1] Fixed size:
a) Is of fixed size
b) Length must be known at: compiling time
c) Size of array - can't change at runtime.

2] Homogeneous Collection:
Can store only 1 type of data elements

3] Array indexing:
a) Starts from: 0 (zero)
b) ends at - (n-1) ... [n = length of array = no. of elements in array]

4] Dimensions of Array:
1D, 2D, 3D ... multi-D [D = Dimensional]

5] Continuous Storage:
- All array elements are stored continuously one after another in memory
E.g. [Array4.C]

6] Random Access:
- Provides random access to elements
- We can get any element of array at any index in a constant time complexity
* - Access Array - using Subscript operator using array index

7] No check of 'index out of bounds':
- C & C++ - no provision for index out of bounds
- Can compile but produces unexpected output at runtime

8] Length of array in C:
a) Length = no. of elements in array
b) Specified at - time of declaration
c) No in-built function for cal^n of length
methods to find out:
A) Using size of operator
B) Pointer Arithmetic
C) Using loops

A) Using size of operator:
```
int lengthOfArray = sizeof(Arr) / sizeof(Arr[2]);
printf("length of Array is: %d \n", lengthOfArray);
```

9] Explain concept of length of Array & size of Array

→ 1) [Length of Array = no. of all element in an array]

indexing = start = 0

ends : (n-1) ... n = length of array

int Arr[5] = {1, 2, 3, 4, 5};

Here length of Arr[5] = 5 (elements)

2) [size of Array = Summation of size of all data elements.]

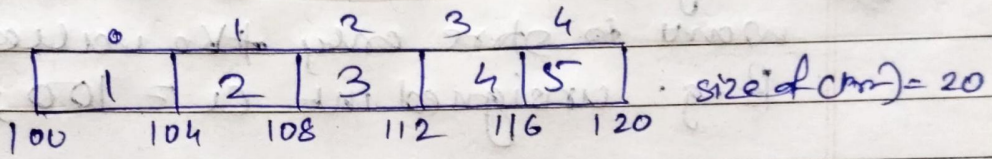
int Arr[5] = {1, 2, 3, 4, 5};

Here datatype is integer, with size 4 bytes

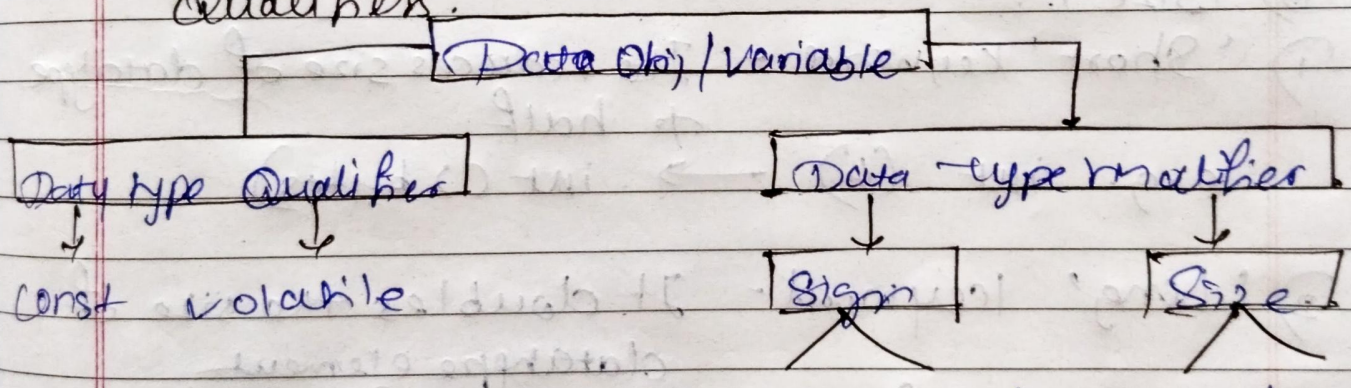
∴ size of (Arr) = ~~4~~ 5(4)

= ~~4~~ 20 bytes

= ~~24~~



10] write notes on Data type modifiers & Data type Qualifiers.



* Data - type Qualifier

A) const ← Value can't be changed

B) volatile -

Date : / /

0-4

* Data type modifier :
1) Modifier which affects the sign
2) Modifier which affects the size.

A) [Sign]
1) [Signed] : Creation of variable w/o specifying signed/unsigned, that variable is by default considered as signed variable.
- Value stored - +ve, -ve or zero.

Eg. signed int i = 10;
int j = -30;
int k = 0;

2) [Unsigned] : written before variable name, if we want to store only the values in it [+ve / zero]
E.g. unsigned int a = 100;

* Applicable to all Primitive Data types.

B) [Size] :
1) 'Short' keyword - It's reduces size of datatype to half.
e.g. int (4) → int (2 bytes)

2) 'Long' keyword - It doubles the size of datatype element
eg. int (4) → int (8 bytes)

Eg. चाय - half glass, normal, jumbo
₹8, ₹10, ₹15

Note : Sizes are based on Compiler, MPL OS used.

Q.4] Program to display content of below array:
const int Arr[6] = {10, 20, 30, 40, 50, 60};

```
#include <stdio.h>
int main()
{
    const int Arr[6] = {10, 20, 30, 40, 50, 60};
    for (int i = 0 ; i < 6 ; i++)
    {
      printf("Arr [%d] = %d \n", i, Arr[i]);
    }
    return 0;
}
```

Output:

Arr [0] = 10
Arr [1] = 20
Arr [2] = 30
Arr [3] = 40
Arr [4] = 50
Arr [5] = 60.