Assignment No-7

Q1] What is mean by array? Explain the use of array with example.

→ An array is ~~an~~ data structure in programming used to store multiple values of the same type in a single variable.

- These values are stored in contiguous memory locations and can be accessed using an index.

• Use of array

1] Storing Multiple Items-
Instead of declaring multiple variables, an array can hold all the items.

2] Looping Through Data-
Arrays can be iterated using loops for processing each element.

3] Implementing Data structure-
Arrays are used in implementing other data structures like stacks, queues and matrices.

Q2] What are the types of Array.

→ Array can be classified into different types based on their structure and usage. The main types of arrays are:

1] One-Dimensional Array (1D-Array)
2] Two - Dimensional Array (2-D Array).
3] Multi - Dimensional Array.

1] One- Dimensional Array (1-D Array).

A one- Dimensional Array is a linear data structure that store elements of the same type in a sequential manner.

It is essentially a list of ~~all~~ itemes where each element is accessed using a single index.

ex -

```
#include <stdio.h>
int main()
{ int numbers[5] = {10, 20, 30, 40, 50};
   printf(" First element : %d\n", numbers[0]);
   return 0;
}
```

Output- First element : 10

2] Two-Dimensional Array (2D Array)

A Two-Dimensional Array is a data structure in programming that organizes data in rows and columns, forming a table-like structure.

Each element in a 2-D array is accessed using two indices : one for the row and one for the column.

ex -

```
#include <stdio.h>
int main()
{ int matrix[2][3] ={{1,2,3}, {4,5,6}};
   printf(" Element at row 0, column 1:
          %d\n", matrix[0][1]);
   printf(" Element at row 1, column 2: %d\n",
          matrix[1][2]);
return 0;
}
```

Output- Element at row 0, column 1: 2.
Element at row 1, column 2: 6

3] Multi-Dimensional Array.

multi-Dimensional array is an extension of a two-dimensions (3D, 4D and so on) Each element in the array is accessed using multiple indices, representing the position in each dimensions.

ex

```
#include<Stdio.h>
int main()
{ int cube[2][2][2] = {{{1,2}, {3,4}},
                       {{5,6}, {7,8}}};
  printf("Element at (0,0,1): %d\n",
          cube[0][0][1]);
  printf("Element at (1,1,0): %d\n",
          cube[1][1][0]);
```

output- Element at (0,0,1): 2
Element at (1,1,0): 7

Q3] What are the different ways of initializing the elements of array?

→ There are multiple ways in which we can create and initialized the array.

① with the size and initialization list

- we can create the array by specifing the length of array as well as initializing the array immediately using initialization of array

• ex-

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

- If the array elements are initialized inside {} called as member Initialization list

• Syntax

datatype array-name [length] = {value1, ... valuen};

② Array initialization with Declaration and without length.

- At the time of creating the array if we initialized it using member initialization list then it is optimal to specify the length of array in [].
- Because the length of array is automatically calculate by Compiler.

• eg-

int Brr[] = {10, 20, 30};

• Syntax -

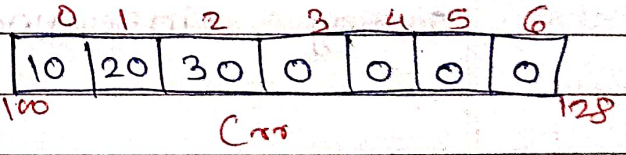
datatype array-name [] = {value1, ... valuen};

③ Initialization, Declaration of array with less numbers.

- It is also possible to provide, specify the length of array with some value but initialize less number of elements in it.

• eg-

int Crr [7] = {10, 20, 30};



- If we initialized less no. of elements in the array then all remaining member of array gets automatically initialized default value.

④ Member by Member initialization (Initialized after declaration).

• eg- int Drr[3];

```
Arr[0] = 10;
Arr[1] = 20;
Arr[2] = 30;
```

⑤ We can create const array.

When we create the array which contains const data member then we cannot change increase, decrease any value.

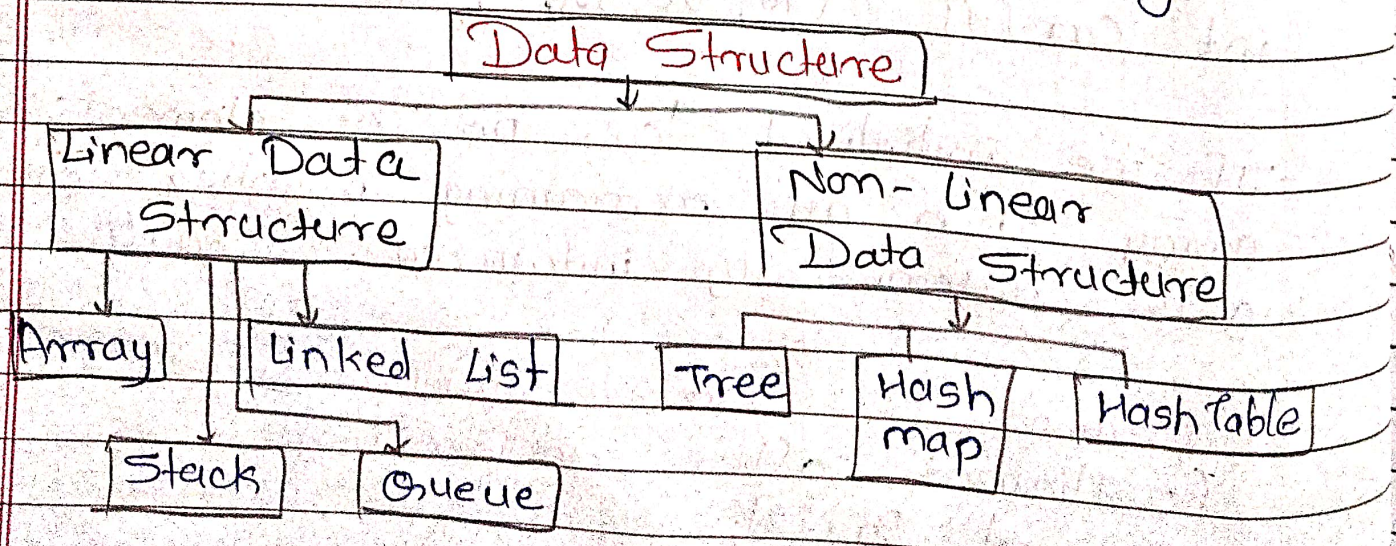
- eg-

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

4] What is mean by data structure? and what are the types of data structures?

→ Data structure is a specialized format for organizing, storing and managing data in a way that enables efficient access and modification.

- The concept of data structure is language independent.
- It provides a systematic way of handling data to perform operations like searching, sorting, inserting, updating and deleting.



• Linear Data Structure.
In this, data is organized sequentially, and elements are connected one after another.
exe -
1) Array - Collection of elements stored in contiguous memory locations.
2) Linked List - It consists of nodes where each node contains data and a reference to the next node
Types of Linked List -
Singly linked list, Doubly, Circular linked list.
3) Stack -
In this structure, elements are added and removed (pushed or popped) only from the top, making it a restricted access data structure.
The last element added to the stack is the first one to be removed.
4) Queue - In a queue, elements are added at the rear and removed from the front, ensuring that the first element added is the first one removed.

• Non- Linear Data Structure
Data elements are connected hierarchically or in a network format, not sequencially.
1) Tree - Hierarchical structure with a root node and child nodes.
2) Graph - set of nodes connected by edges.

3) Hash-based (HashTable, Hash Map)
These use a hash function to map data to a specific index in a table.

5] Explain different ways of memory allocation for an array.

- Static Memory Allocation
- Memory is allocated at compile time.
- The size of the array must be known and fixed before the program runs.
- Once allocated, the memory cannot be resized during execution.
- The array is usually stored in Stack.

- Dynamic Memory Allocation
- Memory is allocated at runtime.
- The size of the array can be determined during execution.
- The array is stored in heap, and memory management is the programmer's responsibility.

- Methods of Dynamic Allocations

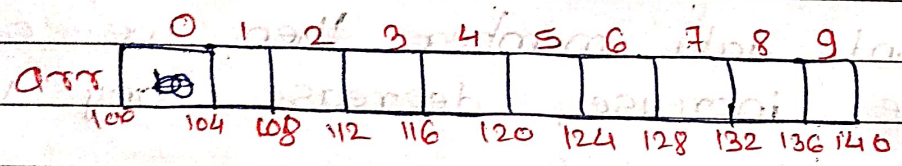
using malloc() -
The return value of malloc() is void *, which indicates the address of allocated memory
- malloc() accepts only one parameter i.e. no-of-bytes that we want to allocate.

using Calloc()
In this case we have to pass two parameters.
1st parameter is no_of_elements.
2nd parameter is size_of_each_element.

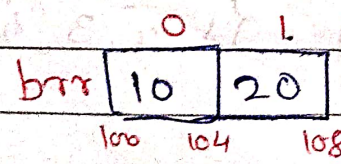
using realloc() -
This function is used to resize allocated memory size.
- It used to increase or decrease size of already allocated memory.

6] Read below statements and draw its diagrammatic representation.

a] int arr[10];
arr is a one dimensional array, which contents five elements in it. Each element is of integer data type.

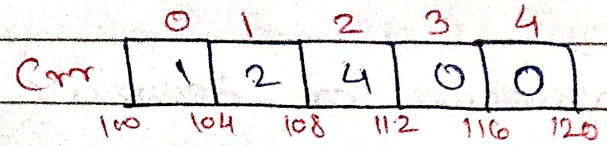


b] int brr[2] = {10, 20}
brr is a one dimensional array, which contents two elements in it. Each element is of integer data type. The elements are initialized with 10, 20.



c] float crr[5] = {1, 2, 4};
crr is one dimensional array, which contents five elements in it. Each element is of

float data type. Elements are initialized with 1, 2, 4.



d] int drr[5] = {20, 40, 60};
drr is one dimensional array, which contents five elements in it, each element is of integer data type. The elements are initialized with 20, 40, 60.

e] const int Demo[2] = {10, 20, 30, 40};
Invalid
When we create the array which contains constant data member then we cannot change increase, decrease any value.

7] Find out the problems in the following statements if any & correct it.

a] int arr[4] = {1, 2, 3, 4, 5, 6, 7}
→ The array size is declared as a 4, but there are 7 elements in the initializer list
Correction - int arr[4] = {1, 2, 3, 4}

b] int brr[]
The size of the array is not specified and there is no initializer list to infer the size.
Correction - int brr[5];
int brr[] = {1, 2, 3};

c) int arr [] = {10, 20, 30, 40};
No problem here.

d) int drr [3+2] = {7+1, 3*5, 100, 10-1}; (valid)
→ int drr [3+2] = The size of array is 5.
drr [0] = 8;
drr [1] = 15;
drr [2] = 100;
drr [3] = 9;
drr [4] = 0; // default

e) int i = 4;
int arr[i] = {23, 45, 67, 89};
→ Array cannot initialized using a variables.
~~Correction~~

8) What is mean by sizeofoperator; explain with example.
→ The sizeof operator in C is a compile-time operator that determines the size, in bytes of a data type or variable.
- It is primarily used to find the memory allocation of data types or structures, which is particularly useful for dynamic memory allocation and ensuring portability across platforms.
Syntax- sizeof(type)
ex-
#include<stdio.h>
int main()
{

```
int a;
double b;
char c;

printf("Size of int: %d bytes \n", sizeof(int));
printf("Size of double: %d bytes \n", sizeof(double));
printf("Size of char: %d bytes\n", sizeof(char));

return 0;
}
```

output-

Size of int : 4 bytes
Size of double : 8 bytes
Size of char : 1 bytes

9] Predict the output of below code snippet

```
#include<stdio.h>
int main()
{
    int arr[3] = {21, 43, 50};
    int x = 0;
    x = arr[2] + 20 + arr[0];
    x++;
    printf("%d", x);
    return 0;
}
```

[50 + 20 + 21] = [91]

91++

92

output- 92

10] #include <stdio.h>
int main()
{
float arr [4] = {98.3, 4.5, 51.5, 7.5};
int i = 0;
printf ("%f :: ", arr [i]);
i++;
printf ("%f :: ", arr [i]);
i++;
printf ("%f ", arr [i]);
return 0;
}

Output-

98.300003 :: 4.500000 :: 51.500000