K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Course: Data Structure

(Course Code : ENCS205)

UNIT-1: Foundations of Data Structures

School of Engineering & Technology
K.R. Mangalam University

SESSION 3:
Static & Dynamic Implementations

Recapitulation (Previous Session)

Quiz

Q1: What is an Abstract Data Type (ADT)?

a) A specific implementation of a data structure.

b) A mathematical model for data types where only the operations are defined.

c) A type of data structure that uses pointers extensively.

d) An algorithm for sorting data.

Q2: Which of the following is NOT an example of an ADT?

a) Stack

b) Linked list

c) Queue

d) Integer

Recapitulation (Previous Session Cont..)

Q3: Which of the following is an example of a basic operation in an ADT?

a) Sorting

b) Searching

c) Adding an element

d) Multiplying two numbers

Q4: Which ADT operation is used to remove an element from the structure?

a) Pop

b) Push

c) Peek

d) Insert

Recapitulation (Previous Session)

Quiz (Answers)

A1: b) A mathematical model for data types where only the operations are defined.

A2: d) Integer

A3: c) Adding an element

A4: a) Pop

Static & Dynamic Implementation

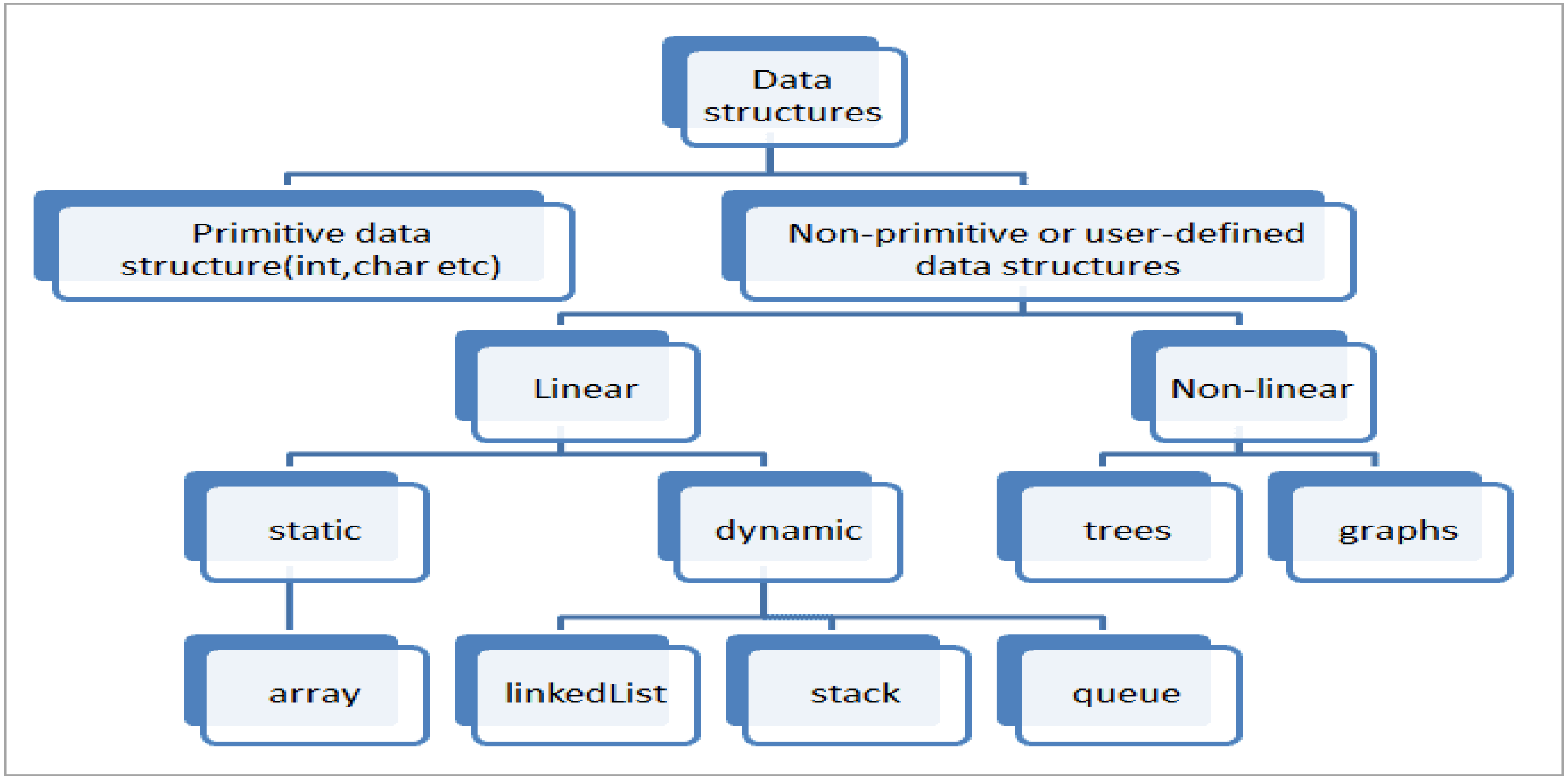
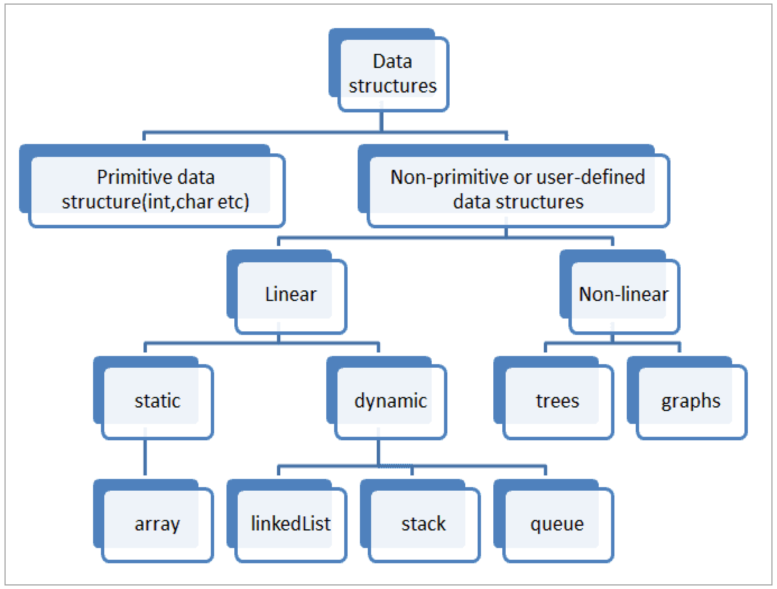

Data structures
Primitive data structure(int,char etc)
Non-primitive or user-defined data structures
Linear
Non-linear
static
dynamic
trees
graphs
array
linkedList
stack
queue

Fig. 1. Classification of Linear Non-primitive data structure

Static Data Structure

Arrays:

- Fixed memory size
- Memory allocated at compile time.
- Easy to access elements.
- Example: Arrays


Data structures
Primitive data structure(int,char etc)
Non-primitive or user-defined data structures
Linear
Non-linear
static
dynamic
trees
graphs
array
linkedList
stack
queue

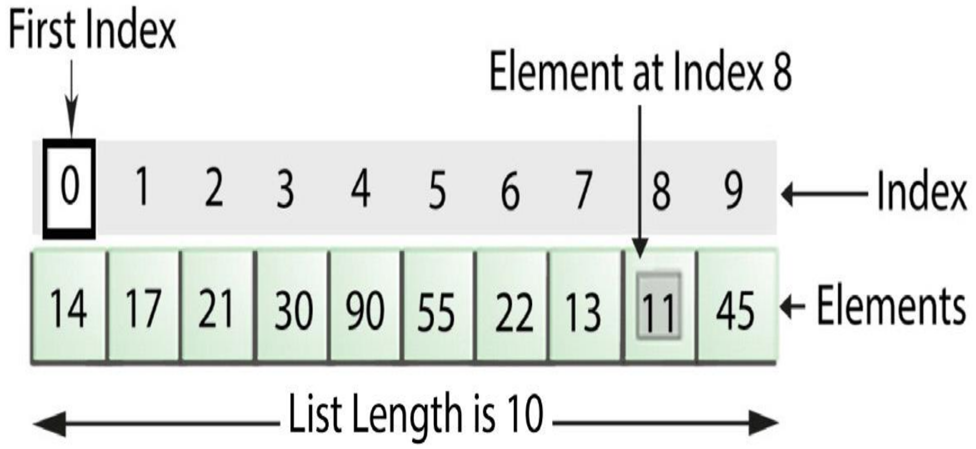


Fig. 2. Example of Array

Static Data Structure-Real world situation

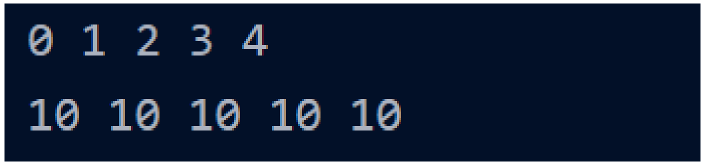
Scenario Description: Consider a home automation system embedded in a smart home device, such as a thermostat, security system, or light controller. These devices often operate with limited hardware resources and need to be highly reliable and fast. The software embedded in these devices controls various aspects of a home, such as adjusting temperatures, managing security sensors, and controlling lighting based on predefined settings.

Justification for Using Static Data Structures:
Predictability: Static data structures have a fixed size and memory allocation, which ensures predictability in memory usage. This is crucial in embedded systems where memory resources are limited and must be allocated precisely to avoid running out of space during operation.
Performance: Static data structures typically allow faster access to data because their memory location is fixed, making them quicker to access and manipulate. This speed is beneficial in real-time systems where delay can affect the functionality of the device.
Resource Constraints: Embedded systems, especially those in smart home devices, often operate with strict power and processing limitations. Static data structures, due to their fixed size, help in optimizing the use of these limited resources.

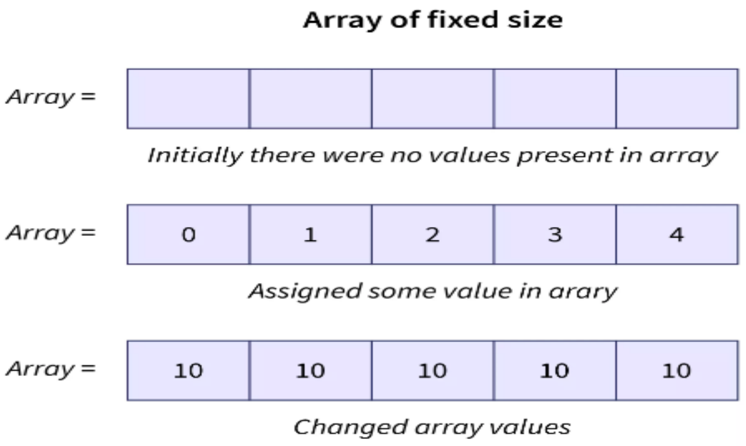
Example of Static Data Structure Implementation

```
#include <iostream>
using namespace std;

int main()
{
     int array[5];

    for(int i=0;i<5;i++)
    {
      array[i] = i;
      cout<<array[i]<<" ";
    }
    cout<<endl; // changing line

    for(int i=0; i<5; i++)
    {
      array[i] = 10;
      cout<<array[i]<<" ";
    }
    return 0; }
```

Output:



```
0 1 2 3 4
10 10 10 10 10
```


Array of fixed size
Array =
Initially there were no values present in array
Array =
0
1
2
3
4
Assigned some value in arary
Array =
10
10
10
10
10
Changed array values

Advantages of Static Data Structures

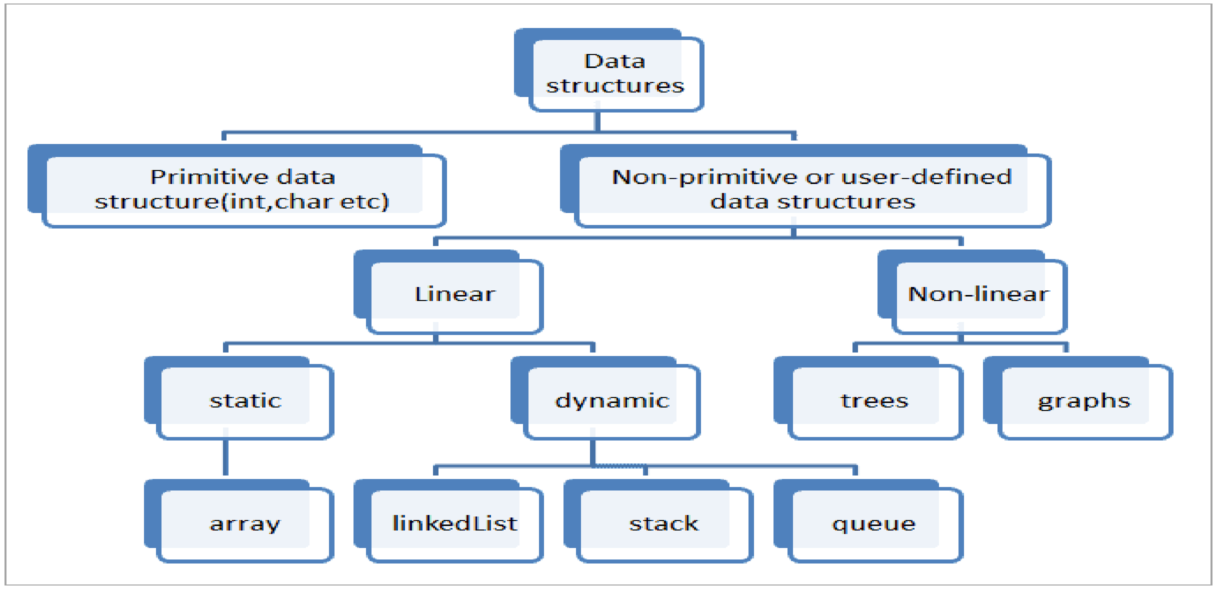
- Easy to handle as compiler handles all the allocation and deallocation processes.
- Memory allocated in contiguous form so no need to maintain the structure of the data structure.
- No problem in overflow or underflow conditions while inserting or deleting any element.

Disadvantages of Static Data Structures

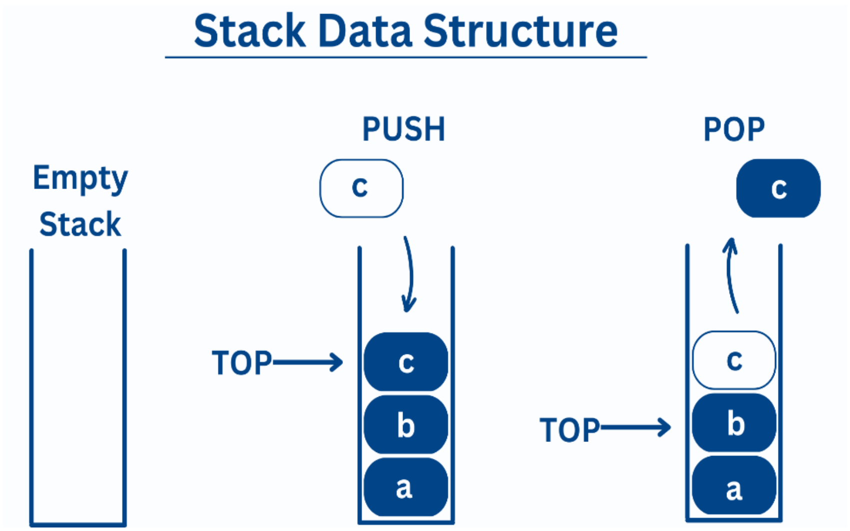
- Users have to estimate maximum required space.
- Insertion of new element between two elements, if there is any vacant space present between them otherwise insertion will take a lot of time.
- Element deletion create vacant space between two elements and covering up that space is costly concerning time.

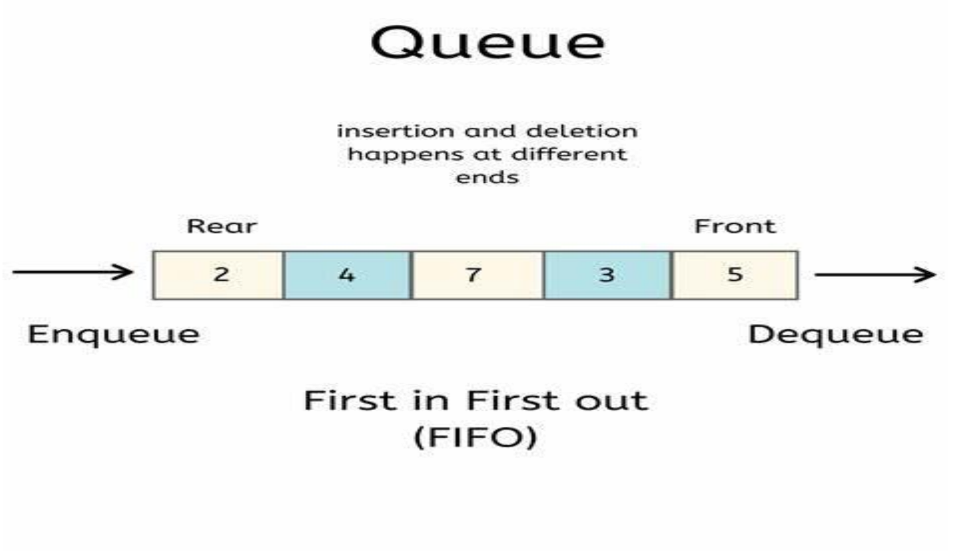
Dynamic Data Structure

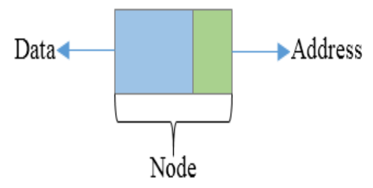
- Size is not fixed
- Memory allocated at run time.
- Randomly updated during runtime
- Example: Queues, Stack, Linked List.


Data structures
Primitive data structure(int,char etc)
Non-primitive or user-defined data structures
Linear
Non-linear
static
dynamic
trees
graphs
array
linkedList
stack
queue

Examples of Dynamic Data Structures




Queue
insertion and deletion happens at different ends
Rear
Front
2
4
7
3
5
Enqueue
Dequeue
First in First out (FIFO)



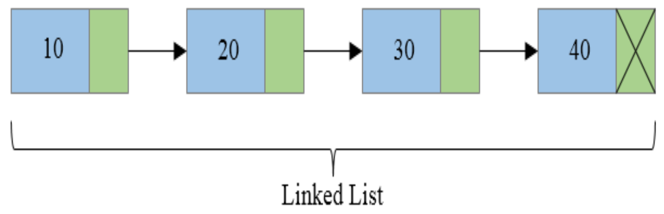


Fig. 3. Examples of Dynamic Data Structures

Dynamic Data Structure-Real world Situation

Scenario: Online Retail Platform's Shopping Cart:

Imagine an online retail platform where users can browse and add various items to their shopping cart. Users can add as many items as they want, remove items, and adjust quantities before finalizing their purchase.

Dynamic data structures are ideal for managing a shopping cart on an online retail platform due to their flexibility in handling data that can change in size frequently.

Key Features and Advantages:

1. **Flexibility in Size:** Users can add any number of items to their shopping cart, so the data structure needs to be able to expand and contract as items are added and removed.
2. **Ease of Modification:** Items can be dynamically added or removed, and the quantities adjusted, which is facilitated smoothly by dynamic data structures like linked lists or dynamic arrays.
3. **Efficient Memory Usage:** Only the necessary amount of memory is used, as it adjusts based on the cart's current contents rather than allocating a fixed amount of memory that might not be used.

Disadvantages of Dynamic Data Structures

- Makes allocated memory non-contiguous which decreases the performance.
- Allocated memory only deallocates when the program ends or when the user deallocates it manually, if user forgets, memory leak problem occurs.
- Size not fixed,so overflow or underflow problems.

Dynamic Data Structure-implementation

```
#include <iostream>

int main() {
    // Declare a pointer for the dynamic array
    int* arr;

    // Determine the size of the array
    int size = 5;

    // Allocate memory for the array
    arr = new int[size];

    // Initialize the array elements
    for (int i = 0; i < size; i++) {
        arr[i] = i * 10; // Assign values
    }

    // Access and print elements of the array
    std::cout << "Array elements:";
    for (int i = 0; i < size; i++) {
        std::cout << " " << arr[i];
    }
    std::cout << std::endl;

    // Deallocate the memory
    delete[] arr;

    return 0;
}
```

In C++, the new keyword is used for dynamic memory allocation, which allows you to allocate memory on the heap during the runtime of your program.

Differences between Static and Dynamic Data Structures

Feature	Static Data Structure	Dynamic Data Structure
Size	Fixed	Dynamic
Memory allocation	Compile-time	Run-time
Memory deallocation	When data structure goes out of scope or program ends	When program ends or user manually deallocated using free() or delete() in C and C++ respectively.
Memory Leak	No memory leak	Memory leak problem occurs.
Data access	Easy to access data with index number.	Difficult to access data as memory is non-contiguous.

Table 1: Differences between Static and Dynamic Data Structures

Situation- Based Questions

Q1 You're designing a system for a video game where the size of the player's inventory can change frequently during gameplay. Which type of data structure would you select to represent the inventory, and how would you manage its size?

Q2 You're tasked with implementing a high-performance caching system for a web application. Would you prefer a static or dynamic data structure for the cache, considering the varying workload and memory constraints?

Q3 You're developing software for a manufacturing plant where sensor data needs to be processed in real-time. How would you select between static and dynamic data structures to handle the incoming sensor data efficiently?

Situation- Based Questions (Answers)

A1: We would choose a dynamic data structure like a dynamic array or a linked list to represent the player's inventory. To manage its size, we would implement resizing mechanisms within the data structure to dynamically allocate more memory as the inventory grows and deallocate memory if the inventory size decreases. This ensures that the inventory can expand or shrink as needed during gameplay.

A2: We would prefer a dynamic data structure for the cache due to its ability to resize dynamically, accommodating varying workloads and memory constraints. This ensures optimal use of resources and allows the caching system to adapt to changing demands efficiently.

A3: For handling real-time sensor data in a manufacturing plant, We would choose static data structures due to their predictable memory allocation and faster access times. This ensures efficient processing of the incoming data without the overhead of resizing operations associated with dynamic data structures.

THANK YOU