1)
The difference between an array and a structure lies in their fundamental characteristics and how they store data:

Array:
- An array is a collection of elements of the same data type organized in a linear manner.
- Elements within an array are accessed using an index, starting from 0 up to the size of the array minus one.
- Arrays are homogeneous, meaning all elements must be of the same data type.

Structure (Struct):
- A structure, also known as a struct, is a custom data type that enables the combination of different data types under a single name.
- Unlike arrays, a struct can hold elements of different data types, making it heterogeneous.
- Accessing elements in a structure is done through dot notation, which assigns a meaningful name to each element within the structure.

2)
Data structures have wide-ranging applications across various computing scenarios, aiding in efficient data organization and manipulation. Some common areas where data structures find utility are as follows:

1. Algorithm Implementation: Data structures play a crucial role in implementing algorithms, facilitating tasks such as searching, sorting, and graph traversals.

2. Database Management Systems: Data structures are instrumental in optimizing data storage and retrieval processes within databases, enhancing overall system performance.

3. Compiler Design: Data structures are utilized during parsing and code generation stages of a compiler, ensuring effective processing of programming code.

4. Networking: Data structures assist in the efficient handling and routing of data packets within networking protocols, enhancing communication between networked devices.

5. Operating Systems: Data structures are employed in managing system resources, including process scheduling and memory management, contributing to smooth system operation.

6. Artificial Intelligence and Machine Learning: Data structures form the foundational framework for organizing and processing data in AI and ML applications, enabling complex data analysis and decision-making.

3)
Data structures can be classified into two primary types:

a. Linear Data Structures: In this category, elements are arranged in a linear sequence, and each element possesses a distinct predecessor and successor.

Examples of linear data structures include Arrays, Linked Lists, Stacks, and Queues.

b. Non-linear Data Structures: This type of data structure does not follow a linear arrangement, allowing elements to have multiple predecessors and successors.

Examples of non-linear data structures encompass Trees, Graphs, and Heaps.

4)
A linked list is a type of linear data structure where elements, known as nodes, are interconnected through pointers or references. Each node comprises two fields: one for holding the actual data and another to point to the subsequent node in the sequence. The final node typically points to null, signifying the end of the list.

Linked lists come in different variations, including:

- Singly Linked List: Each node points solely to the next node in the sequence.
- Doubly Linked List: Each node maintains references to both the next and previous nodes.
- Circular Linked List: The last node points back to the first node, creating a closed loop.

Linked lists possess dynamic characteristics, enabling efficient handling of insertions and deletions of elements when compared to arrays, as there is no need to shift elements. Nevertheless, they do incur higher memory overhead due to the inclusion of pointers.

5)
(A recursive data structure is a data arrangement that can be self-referential, meaning it contains instances of the same type within itself.

A classic illustration of a recursive data structure is the "tree." A tree comprises nodes, where each node can have its individual subtree, which is also a tree. This recursive definition permits trees to exhibit hierarchical and nested formations.

Example:

```
struct TreeNode {
   int data;
   struct TreeNode* left;
   struct TreeNode* right;
};
```

In this example, the `TreeNode` structure is self-referential as it includes `left` and `right` pointers, which point to subtrees of the same type.)

6)
Linear data structures store elements in a linear sequence with a definite order, each having a unique predecessor and successor. Examples include arrays, linked lists, stacks, and queues. Operations like traversal, search, and sorting are relatively straightforward, and memory allocation is usually more contiguous.

On the other hand, non-linear data structures do not follow a linear sequence, and elements can have multiple predecessors and successors. Examples include trees, graphs, and heaps. Operations like traversal and search may involve more complex algorithms, and memory allocation can be more scattered due to arbitrary relationships between elements.

Comparison-wise, linear data structures are simpler to implement and understand compared to non-linear structures. They have predictable and straightforward traversal patterns. Non-linear structures are better suited for representing real-world relationships like networks and hierarchies. However, non-linear structures can have more complex algorithms for operations like searching or traversing, and they generally have higher memory overhead due to additional pointers or references.