BASIC CONCEPTS OF DATA REPRESENTATION

Basic Concepts of Data Representation in Abstract Data Types (ADT)

In computer science, **data representation** refers to the way information is stored and organized in a computer's memory. Efficient representation is essential to solving problems quickly and effectively. The abstraction of data and their operations is key to the development of **abstract data types (ADTs)**, which simplify complex operations.

ADTs focus on the **logical structure** of the data rather than its physical layout in memory. They help in designing programs by providing a clean separation between the *interface* (what operations the data structure supports) and *implementation* (how the data structure is implemented in memory).

Fundamental and Derived Data Types

In computer science, data types are broadly categorized into **fundamental (primitive) data types** and **derived data types**.

Fundamental Data Types

These are the most basic data types that are directly supported by a programming language. They serve as the building blocks for other types of data structures and

represent the simplest form of data, such as integers, floating-point numbers, and characters.

1. **Integer**:
    - **Definition**: Integers are whole numbers without a fractional component.
    - **Representation**: They are typically stored in binary form, with a fixed number of bits depending on the system architecture (e.g., 32-bit or 64-bit).
    - **Size**: The size of integers may vary between platforms (e.g., 4 bytes or 8 bytes), and their range is dependent on this size.
2. **Floating-Point**:
    - **Definition**: Floating-point numbers are used to represent real numbers with fractional components.
    - **Representation**: They are typically represented in memory using the IEEE 754 standard. A floating-point number consists of three components: sign, exponent, and mantissa.
3. **Character**:
    - **Definition**: A character represents a single textual element, such as a letter, digit, or punctuation mark.
    - **Representation**: Characters are typically stored as 8-bit ASCII codes or 16-bit Unicode values to accommodate international characters.

BCANPM

4. **Boolean**:
    - **Definition**: Boolean data types represent truth values, true or false.
    - **Representation**: In memory, they are often represented by a single bit, with 0 representing false and 1 representing true.

Derived Data Types

Derived data types are constructed from fundamental data types and are used to represent more complex data structures.

1. **Array**:
    - **Definition**: An array is a collection of elements, all of the same data type, stored in contiguous memory locations. Arrays allow random access to elements via an index.
    - **Representation**: The address of an element in an array is calculated using the base address of the array and the size of each element. Arrays are typically stored in row-major order in memory.
2. **Structure**:
    - **Definition**: A structure (or struct) is a collection of different data types grouped together. It allows you to store variables of different types under a single name.

BCANPM

- **Representation**: Structures are stored as a block of memory, where each field in the structure has a fixed offset relative to the start of the structure.

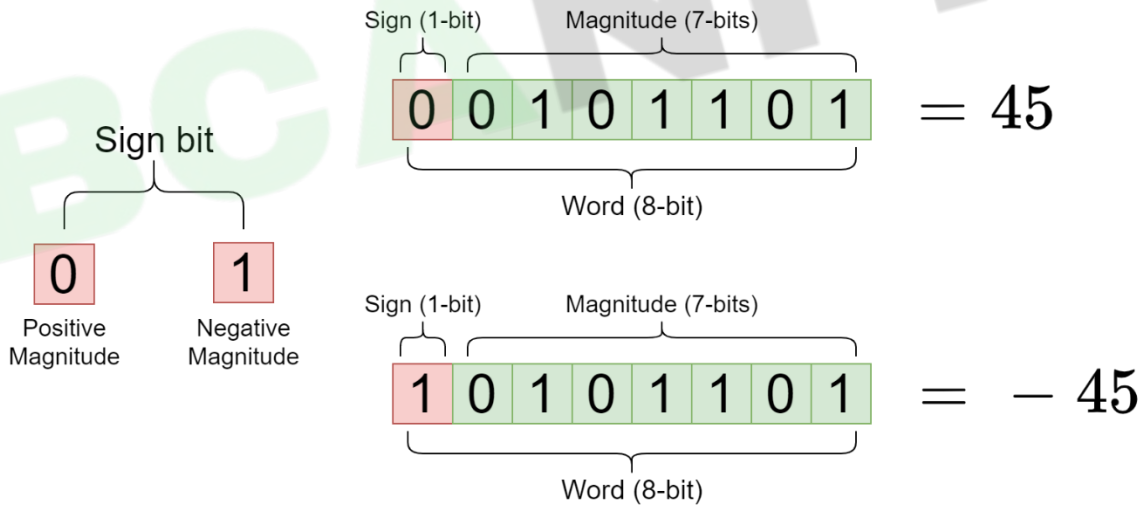
3. **Union**:
   - **Definition**: A union is similar to a structure but with a key difference: all members share the same memory location. A union can hold only one value at a time.
   - **Representation**: Since all members of the union occupy the same memory, only the largest member determines the size of the union.
4. **Pointer**:
   - **Definition**: A pointer is a variable that stores the memory address of another variable. Pointers allow dynamic memory allocation and efficient manipulation of arrays and structures.
   - **Representation**: In memory, a pointer is typically stored as an integer representing the memory address of the target variable.
5. **Linked List**:
   - **Definition**: A linked list is a sequence of elements, where each element contains data and a reference (or pointer) to the next element in the sequence.
   - **Representation**: In memory, each element (called a node) is represented by a block containing the data and a pointer to the next node.

BCANPM

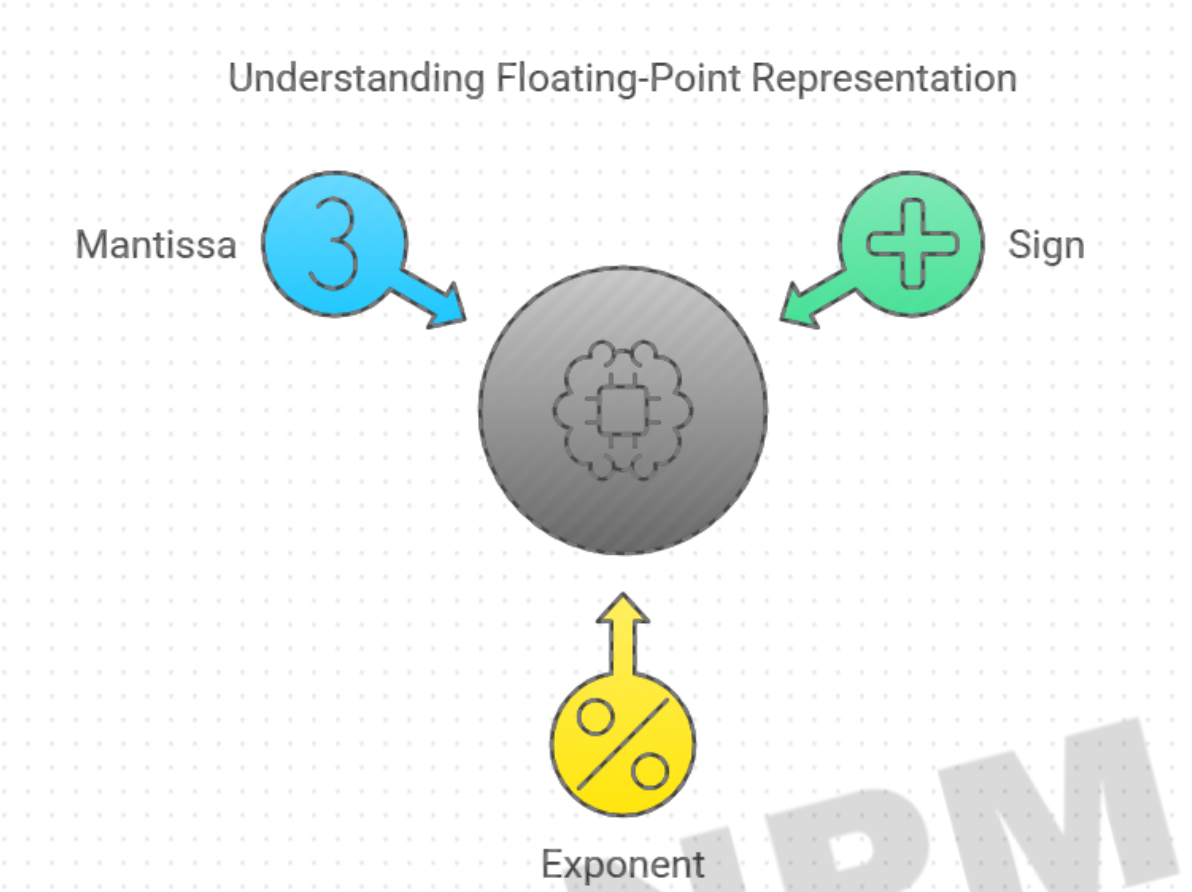
Primitive Data Structures

Primitive data structures are the most basic types of data structures and include integers, floats, characters, and pointers. These are typically built into the language itself and directly represent data in memory.

1. **Integer Representation**:
    - Integers can be stored in various formats, such as signed and unsigned.
    - Signed integers reserve a bit for indicating the sign (positive or negative), whereas unsigned integers represent only non-negative numbers.



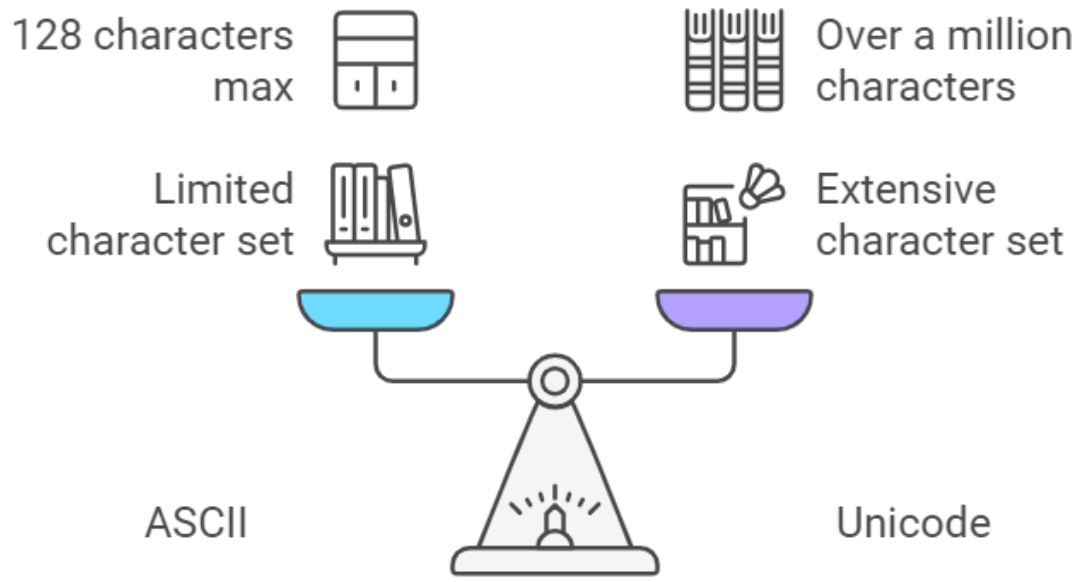
2. **Floating-Point Representation**:
    - Floating-point numbers are divided into three parts: the sign bit, exponent, and mantissa.



3. **Character Representation**:

   - Characters are stored as numerical codes in a system like ASCII or Unicode. ASCII uses 7 bits, while Unicode uses up to 32 bits to accommodate different languages and symbols.

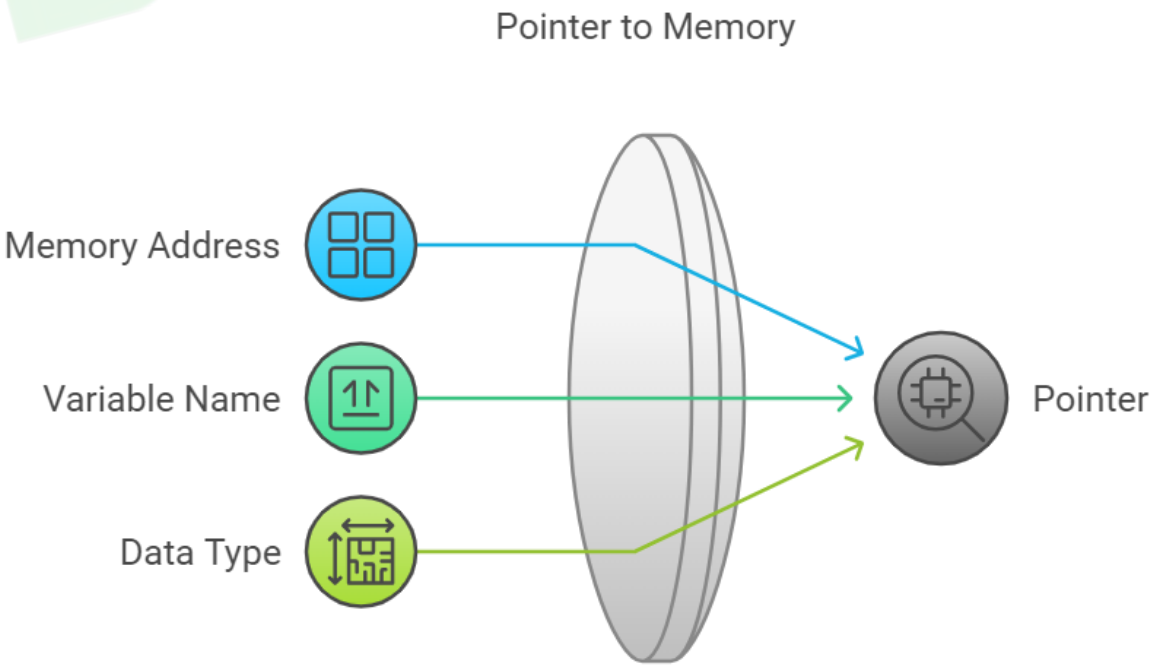
BCANPM



Comparing ASCII and Unicode character encoding systems.

4. **Pointer Representation**:

   - Pointers store the memory addresses of variables. These addresses are typically represented in hexadecimal form. Pointer arithmetic is used to navigate arrays and other structures efficiently.



BCA NPM

Representation of Data

Data can be stored and manipulated in a variety of ways in memory. Each type of data has a specific way in which it is encoded and stored.

1. Binary Representation

Data in computers is stored in binary, meaning it is represented using only two states: 0 and 1. Each 0 or 1 is referred to as a bit, and groups of 8 bits form a byte. Binary representation is the foundation of all data storage and manipulation in digital systems.

- **Integer Representation**: Integers are typically stored in a fixed-size binary format. For example, in a 32-bit system, a signed integer might be represented as a 32-bit binary number using two's complement notation to represent both positive and negative values.
- **Floating-Point Representation**: Floating-point numbers use a different format, dividing the bits between the sign, exponent, and mantissa, as per the IEEE 754 standard.
- **Character Representation**: Characters are represented using standardized encoding schemes like ASCII or Unicode.

2. Data Representation in Memory

Data is stored in **memory locations** that are addressed sequentially. Each memory location has a unique address, and different data types occupy different amounts of

BCANPM

memory. For example, an integer might take up 4 bytes, while a character might take only 1 byte.

- **Contiguous Memory Allocation**: In arrays, data is stored in contiguous memory locations, meaning that each element is stored next to its neighbor. This allows for efficient access using indices.
- **Non-Contiguous Memory Allocation**: In linked lists and trees, data elements are stored in non-contiguous memory locations, with each element containing a reference to the next element.

Abstract Data Types (ADT)

An **Abstract Data Type (ADT)** is a data structure defined by its behavior from the user's point of view, rather than its implementation. ADTs provide a way to abstract away the details of how data is stored and manipulated, focusing instead on what operations are supported and how they behave.

1. List ADT:

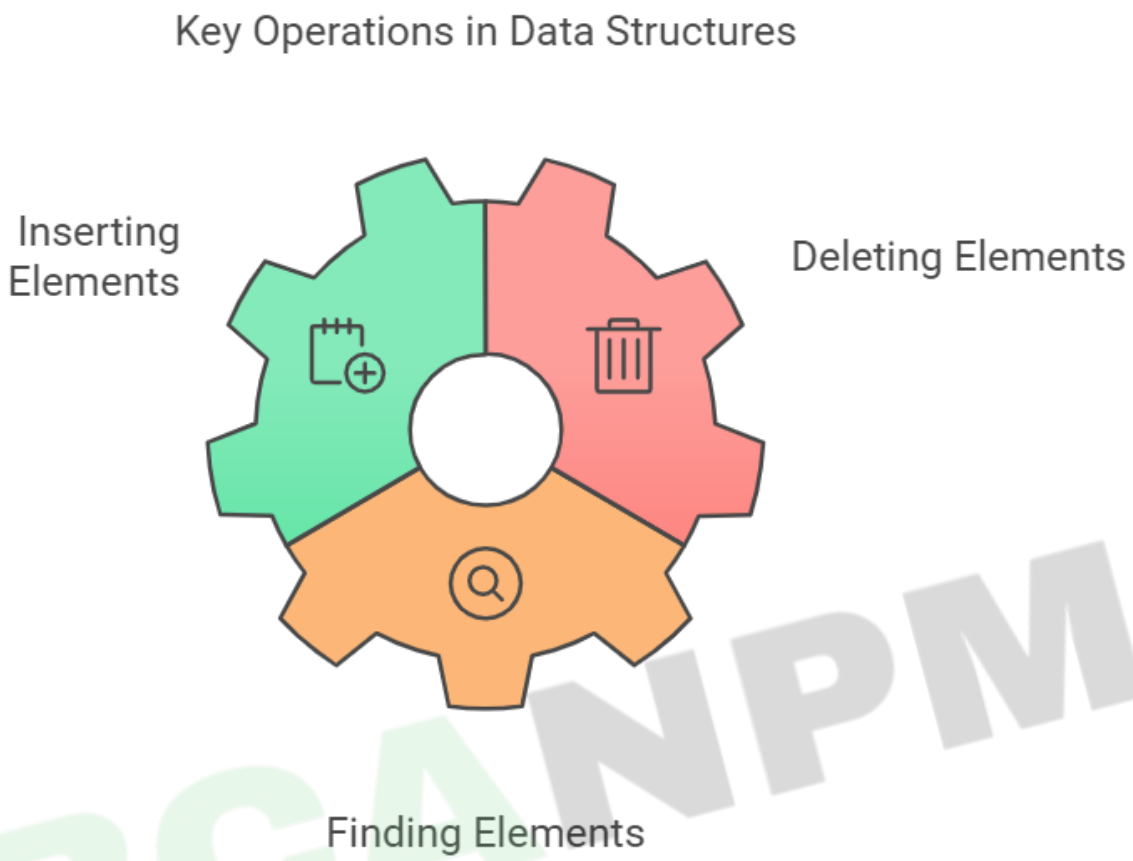
- A list is an ordered collection of elements where duplicates are allowed. Common operations include insertion, deletion, and searching for elements.

Logical Representation:

- Insert(element): Adds an element at a specific position.
- Delete(element): Removes an element.

BCANPM

- Find(element): Searches for an element in the list.



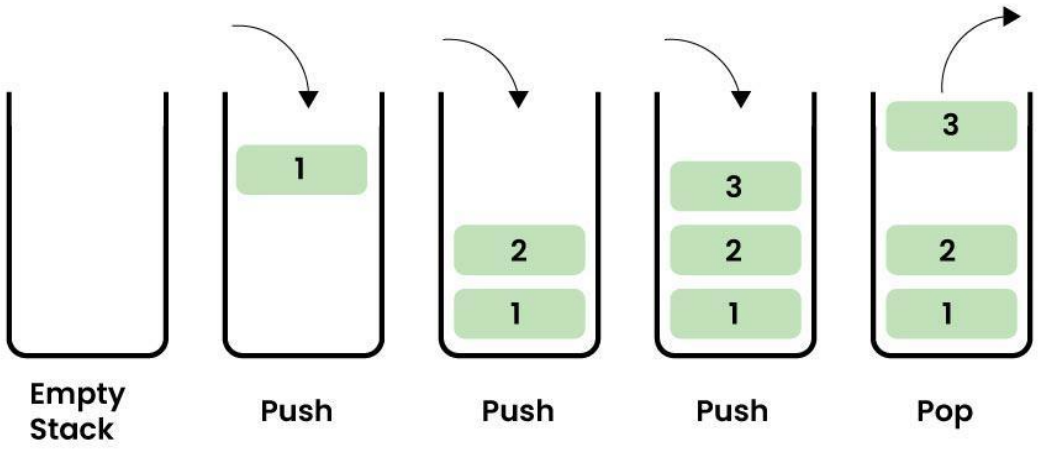
BCANPM

2. Stack ADT:

- A stack follows the **Last In, First Out (LIFO)** principle. Only the top element can be accessed or removed at any given time.

Logical Representation:

- Push(element): Adds an element to the top of the stack.
- Pop(): Removes and returns the top element of the stack.
- Peek(): Returns the top element without removing it.

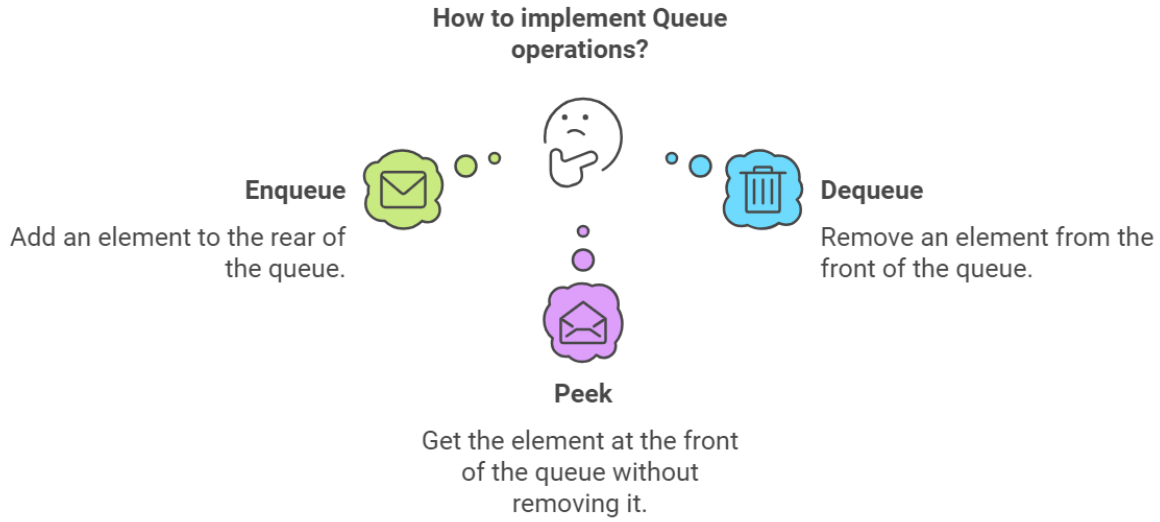


3. Queue ADT:

- A queue follows the **First In, First Out (FIFO)** principle, where elements are inserted at the rear and removed from the front.

Logical Representation:

- Enqueue(element): Adds an element to the rear.
- Dequeue(): Removes and returns the front element.
- Peek(): Returns the front element without removing it.



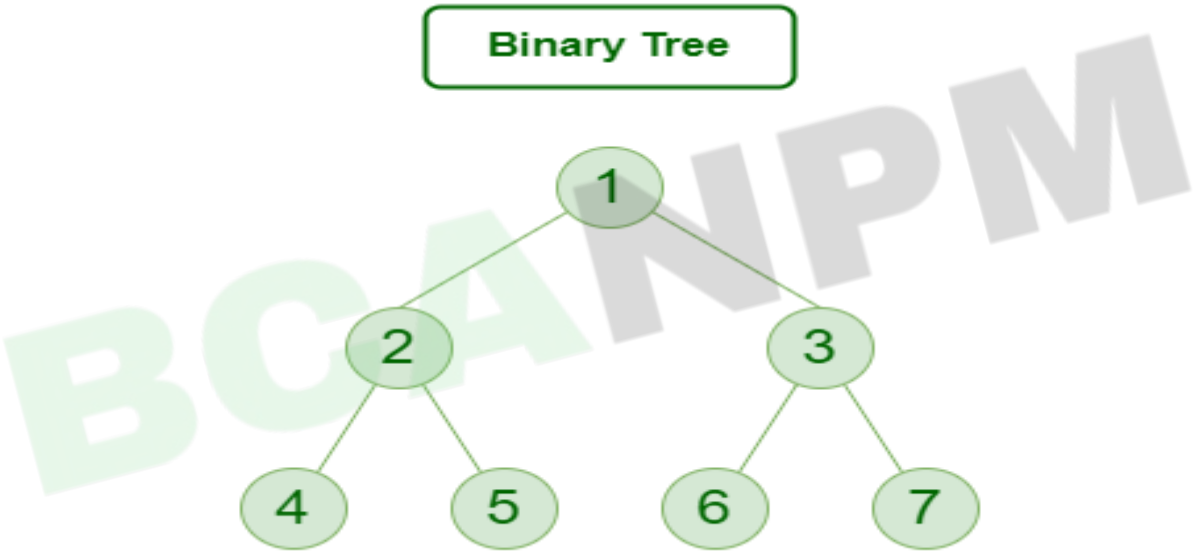
BCANPM

4. Tree ADT:

- A tree is a hierarchical structure where each node contains data and references to its child nodes.

Logical Representation:

- Insert(element): Adds an element to the appropriate position.
- Delete(element): Removes an element.
- Find(element): Searches for an element in the tree.



BCANPM

Conclusion

The basic concepts of data representation and abstract data types are foundational to understanding how computers store and manipulate information. Fundamental data types like integers, floating points, and characters form the basis of more complex derived data types, such as arrays, linked lists, and trees. Through abstract data types, these structures are

further simplified to focus on the operations they support rather than their implementation details. By mastering these concepts, developers can build efficient and optimized programs.

BCANPM