Unite 1 :- DATA STRUCTRURE

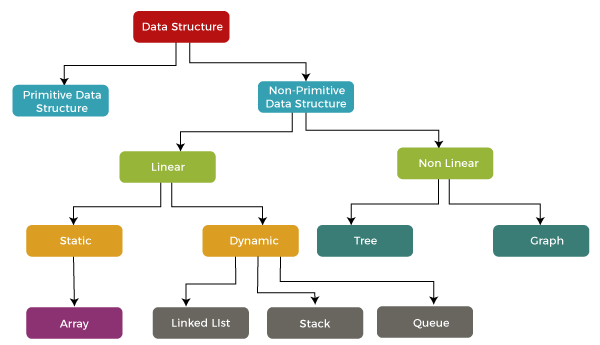
Primitive and non-primitive data structures, String manipulation and pattern matching, Storage representation of strings, Text handling, Key Word In Context (KWIC) indexing, Arrays, Storage structure for arrays, Special types of arrays – triangular and sparse.

**Q1. What is Primitive data structure and Non-primitive data structure ?**

There are two types of data structure available for the programming purpose:

1. Primitive data structure
2. Non-primitive data structure

Primitive data structure is a fundamental type of data structure that stores the data of only one type whereas the non-primitive data structure is a type of data structure which is a user-defined that stores the data of different types in a single entity.



1. **Primitive Data Structures**: These include basic types like **integer, float, character**, and **pointer**. Each of these holds a single type of value, such as:
   * Integer for whole numbers,
   * Float for decimal numbers,
   * Character for single characters,
   * Pointer for memory addresses.
2. **Non-Primitive Data Structures**: These are more complex and divided into two categories:

**Linear Data Structures**: Elements are stored sequentially in memory. Examples include:

* **Array**
* **Linked List**
* **Stack**
* **Queue**

**Non-Linear Data Structures**: Elements are not stored sequentially. Examples include:

* **Tree**: A hierarchical structure with nodes connected by edges.
* **Graph**: A collection of nodes (vertices) connected by edges, used to model complex relationships.

In a linear structure, each element is stored one after the other, ensuring order in memory allocation.

|  |  |
| --- | --- |
| **Primitive Data Structure** | **Non-Primitive Data Structure** |
| Primitive data structure is a kind of data structure that stores the data of only one type. | Non-primitive data structure is a type of data structure that can store the data of more than one type. |
| Examples of primitive data structure are integer, character, float. | Examples of non-primitive data structure are Array, Linked list, stack. |
| Primitive data structure will contain some value, i.e., it cannot be NULL. | Non-primitive data structure can consist of a NULL value. |
| The size depends on the type of the data structure. | In case of non-primitive data structure, size is not fixed. |
| It starts with a lowercase character. | It starts with an uppercase character. |
| Primitive data structure can be used to call the methods. | Non-primitive data structure cannot be used to call the methods. |

**Q2. Define String Manipulation in Data structure.**

**String Manipulation:**

String manipulation refers to various operations performed on strings (a sequence of characters). Common operations include:

* **Concatenation**: Joining two or more strings.
  + Example: "Hello" + "World" = "HelloWorld"
* **Substring**: Extracting a part of a string.
  + Example: "HelloWorld"[0:5] = "Hello"
* **Length**: Determining the number of characters in a string.
  + Example: len("HelloWorld") = 10
* **Comparison**: Comparing two strings for equality or lexicographical order.
  + Example: "abc" > "abd" = False
* **Reversal**: Reversing the order of characters in a string.
  + Example: "abc"[::-1] = "cba"
* **Replacing**: Replacing certain characters or substrings with others.
  + Example: "HelloWorld".replace("World", "Python") = "HelloPython"
* **Searching**: Finding the position of a character or substring.
  + Example: "HelloWorld".find("World") = 5

**Q3. Define Storage Representation of strings**

1. **Storage representation of strings** refers to how strings are stored in memory. Here's a brief overview:
2. **Array of Characters**:
   * Strings are typically stored as an array of characters.
   * Each character in the string occupies one memory location, and the characters are stored contiguously in memory.
   * In most languages, strings are terminated with a special character, such as a **null character ('\0')** in C, to mark the end of the string.
3. Example:
   * The string "Hello" is stored as an array: ['H', 'e', 'l', 'l', 'o', '\0'].
4. **Fixed vs. Dynamic Length**:
   * **Fixed-Length Strings**: A predefined number of memory locations are allocated for the string, even if the string length is smaller.
   * **Dynamic-Length Strings**: Memory is allocated dynamically as the string grows, allowing flexible size.
5. **Immutable Strings**:
   * In some languages like Python and Java, strings are **immutable**, meaning they cannot be changed once created. Modifying a string creates a new string in memory.
6. **Unicode and Encoding**:
   * Strings can be stored using different encodings (ASCII, UTF-8, UTF-16) based on how the characters are represented.
   * ASCII uses 1 byte per character, while UTF-8/UTF-16 use variable bytes depending on the character.

**Q4. What is Text Handling ?**

1. **Text handling** involves processing and manipulating textual data, and it includes the following operations:
2. **Storage**:
   * Text is stored as strings, typically as arrays of characters, and can use different encodings (ASCII, UTF-8, etc.).
3. **Basic Operations**:
   * **Concatenation**: Joining multiple strings.
   * **Search**: Finding substrings or characters within text.
   * **Substring Extraction**: Extracting parts of the string.
   * **Replacement**: Replacing specific characters or patterns.
   * **Splitting**: Dividing text into smaller parts based on delimiters.
4. **Formatting**:
   * Converting text to uppercase/lowercase, trimming whitespace, or aligning text (left, right, center).
5. **Pattern Matching**:
   * Finding patterns using algorithms (like KMP, Boyer-Moore) or regular expressions (regex).
6. **Validation**:
   * Checking for valid formats (e.g., email, phone numbers) using predefined rules.

**Q5. Keyword In Context Indexing.**

**Keyword in Context Indexing**

KWIC indexing is **a technique used in information retrieval to create an index of keywords extracted from document titles, along with their surrounding context**. This approach helps users search for specific words or phrases within a collection of documents, facilitating efficient and accurate retrieval.

**Formation of KWIC Index**

A KWIC index is **formed by sorting and aligning the words within an article title**. Each keyword occurrence is displayed in a list with surrounding words, providing context for the search query. This format allows for alphabetization by keyword, making it easier to locate specific terms.

**Advantages**

1. **Efficient search**: KWIC indexing enables users to search for specific words or phrases within a collection of documents, reducing the time and effort required to locate relevant information.
2. **Contextual understanding**: By including surrounding words, KWIC indexing provides context for the search query, helping users better understand the relevance and meaning of the retrieved documents.
3. **Improved accuracy**: KWIC indexing reduces the likelihood of false positives, as the context surrounding the keyword helps to disambiguate its meaning.

**Types of KWIC Indexing**

1. **KWIC (Keyword-In-Context)**: The most common format, where each keyword occurrence is displayed with surrounding words.
2. **KWOC (Keyword-Out-Of-Context)**: A less common format, where keywords are listed without their surrounding context.
3. **KWAC (Keyword-Augmented-In-Context)**: A variation that includes additional information, such as part-of-speech tags or semantic roles, to provide more nuanced context.

**Q6. What / Why / When Array use in data Structure**.

<https://www.javatpoint.com/data-structure-array>

Refer this link for better context to explain

**WHAT ?**

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

In C programming, they are the derived data types that can store the primitive type of data such as int, char, double, float, etc. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define a different variable for the marks in different subjects. Instead, we can define an array that can store the marks in each subject at the contiguous memory locations.

**WHY ?**

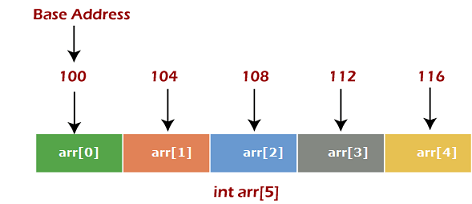
* Sorting and searching a value in an array is easier.
* Arrays are best to process multiple values quickly and easily.
* Arrays are good for storing multiple values in a single variable - In computer programming, most cases require storing a large number of data of a similar type. To store such an amount of data, we need to define a large number of variables. It would be very difficult to remember the names of all the variables while writing the programs. Instead of naming all the variables with a different name, it is better to define an array and store all the elements into it.

**Memory allocation of an array**

As stated above, all the data elements of an array are stored at contiguous locations in the main memory. The name of the array represents the base address or the address of the first element in the main memory. Each element of the array is represented by proper indexing.

We can define the indexing of an array in the below ways -

1. 0 (zero-based indexing): The first element of the array will be arr[0].
2. 1 (one-based indexing): The first element of the array will be arr[1].
3. n (n - based indexing): The first element of the array can reside at any random index number.



In the above image, we have shown the memory allocation of an array arr of size 5. The array follows a 0-based indexing approach. The base address of the array is 100 bytes. It is the address of arr[0]. Here, the size of the data type used is 4 bytes; therefore, each element will take 4 bytes in the memory.

**Basic operations**

Now, let's discuss the basic operations supported in the array -

* Traversal - This operation is used to print the elements of the array.
* Insertion - It is used to add an element at a particular index.
* Deletion - It is used to delete an element from a particular index.
* Search - It is used to search an element using the given index or by the value.
* Update - It updates an element at a particular index.

**Advantages of Array**

* Array provides the single name for the group of variables of the same type. Therefore, it is easy to remember the name of all the elements of an array.
* Traversing an array is a very simple process; we just need to increment the base address of the array in order to visit each element one by one.
* Any element in the array can be directly accessed by using the index.

**Disadvantages of Array**

* Array is homogenous. It means that the elements with similar data type can be stored in it.
* In array, there is static memory allocation that is size of an array cannot be altered.
* There will be wastage of memory if we store less number of elements than the declared size.