String in Data Structure

A string is a sequence of characters. The following facts make string an interesting data structure.

- Small set of elements. Unlike normal array, strings typically have smaller set of items. For example, lowercase English alphabet has only 26 characters. ASCII has only 256 characters.
- Strings are immutable in programming languages like Java, Python, JavaScript and C#.
- Many String Problems can optimized using the fact that the character set size is small. For example sorting can be done faster, counting frequencies of items is faster and many interesting interview questions are based on this.

Introduction to Strings - Data Structure and Algorithm

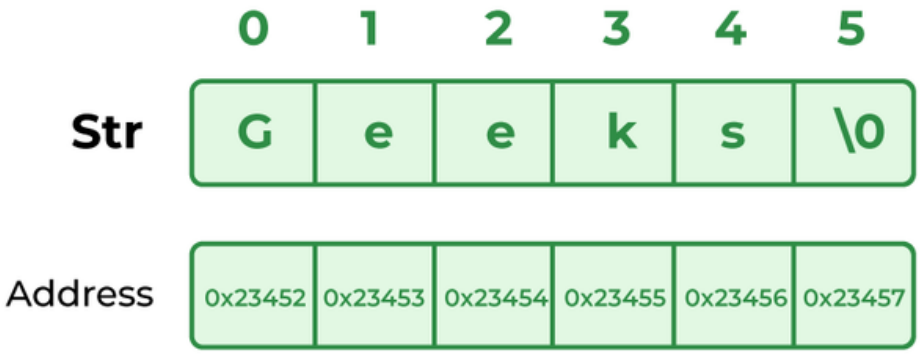
Strings are sequences of characters. The differences between a character array and a string are, a string is terminated with a special character **'\0'** and strings are typically immutable in most of the programming languages like Java, Python and JavaScript. Below are some examples of strings:

"It's", "good" ,"for", "you", "Memory"

How Strings are represented in Memory?

In C, a string can be referred to either using a character pointer or as a character array. When strings are declared as character arrays, they are stored like other types of arrays in C. String literals (assigned to pointers) are immutable in C and C++.

In C++, strings created using string class are mutable and internally represented as arrays. In Python, Java and JavaScript, strings characters are stored at contiguous locations (like arrays).



How to Declare Strings in various languages?

- **C:** Strings are declared as character arrays or pointers and must end with a null character (\0) to indicate termination.
- **C++:** Supports both C-style character arrays and the std::string class, which provides built-in functions for string manipulation.
- **Java:** Strings are immutable objects of the String class, meaning their values cannot be modified once assigned.
- **Python:** Strings are dynamic and can be declared using single, double, or triple quotes, making them flexible for multi-line text handling.
- **JavaScript:** Strings are primitive data types and can be defined using single, double, or template literals (backticks), allowing for interpolation.
- **C#:** Uses the string keyword, which represents an immutable sequence of characters, similar to Java.
- There is no character type on Python and JavaScript and a single character is also considered as a string.

Below is the representation of strings in various languages:

```
// C++ program to demonstrate String
// using Standard String representation

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

int main()
{

    // Declare and initialize the string
    string str1 = "Welcome Back";

    // Initialization by raw string
```

```
    string str2("A Computer Science Portal");


    // Print string
    cout << str1 << endl << str2;


    return 0;
}
```

Are Strings Mutable in Different Languages?

- In C/C++, string literals (assigned to pointers) are immutable.
- In C++, string objects are mutable.
- In Python, Java and JavaScript, strings are immutable.

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


int main() {
    const char* str = "Hello, world!";
    str[0] = 'h';  // Error : Assignment to read only
    cout << str;
    return 0;
}
```

General Operations performed on String

Here we are providing you with some must-know concepts of string:

Length of string: The length of a string refers to the total number of characters present in it, including letters, digits, spaces, and special characters. It is a fundamental property of strings in any programming language and is often used in various operations such as validation, manipulation, and comparison.

Search a character: Searching for a character in a string means finding the position where a specific character appears. If the character is present multiple times, you might need to find its first occurrence, last occurrence, or all occurrences.

- **Check for substring:** Checking for a substring means determining whether a smaller sequence of characters exists within a larger string. A substring is a continuous part of a string, and checking for its presence is a common operation in text processing, search algorithms, and data validation.
- **Insert a character :** Inserting a character into a string means adding a new character at a specific position while maintaining the original order of other characters. Since strings are immutable in many programming languages, inserting a character usually involves creating a new modified string with the desired character placed at the specified position.
- **Deleting a character :** Deleting a character from a string means removing a specific character at a given position while keeping the remaining characters intact. Since strings are immutable in many programming languages, this operation usually involves creating a new string without the specified character.
- **Check for same strings:** Checking if two strings are the same means comparing them character by character to determine if they are identical in terms of length, order, and content. If every character in one string matches exactly with the corresponding character in another string, they are considered the same.
- **String Concatenation:** String concatenation is the process of joining two or more strings together to form a single string. This is useful in text processing, formatting messages, constructing file paths, or dynamically creating content.
- **Reverse a string :** Reversing a string means arranging its characters in the opposite order while keeping their original positions intact in the reversed sequence. This operation is commonly used in text manipulation, data encryption, and algorithm challenges.
- **Rotate a string**: Rotating a string means shifting its characters to the left or right by a specified number of positions while maintaining the order of the remaining characters. The characters that move past the boundary wrap around to the other side.
- **Check for palindrome:** Checking for a palindrome means determining whether a string reads the same forward and backward. A palindrome remains unchanged when reversed, making it a useful concept in text processing, algorithms, and number theory.

Introduction to Strings - Data Structure and Algorithm

Is string a linear data structure?

Yes, string is a linear data structure.

Where are strings used?

It is used to store the sequence of characters.

Is string a data type?

A string is generally considered a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding.

Why is text called string?

Text are also called string because it consists of sequence of characters like string.

What are characters in a string?

Each digit in a string is a character and character is a single visual object used to represent text, numbers, or symbols.

Given a character **ch** and a string **s**, the task is to find the **index** of the **first occurrence** of the character in the string. If the character is **not present** in the string, return **-1**.

Examples:

Input: *s = "geeksforgeeks", ch = 'k'*
Output: *3*
Explanation: *The character 'k' is present at index **3** and **11** in "geeksforgeeks", but it first appears at index 3.*

Input: *s = "geeksforgeeks", ch = 'z'*
Output: *-1*
Explanation: *The character 'z' is not present in "geeksforgeeks".*

Approach - By traversing the string - O(n) Time and O(1) Space

*The idea is to traverse the input string **s** and for each character, check if it is equal to the character we are searching for. If we find a match, return the **index** of the current character.*

*If we reach the **end** of the string without finding any occurrence of the character, return **-1**.*

// Java program to search a character in a string

```
class GfG {

    // function to find the first occurrence of ch in s
    static int findChar(String s, char ch) {
        int n = s.length();
        for (int i = 0; i < n; i++) {
```

```
        // If the current character is equal to ch,
        // return the current index
        if (s.charAt(i) == ch)
            return i;
    }

    // If we did not find any occurrence of ch,
    // return -1
    return -1;
  }

  public static void main(String[] args) {
    String s = "geeksforgeeks";
    char ch = 'k';

    System.out.println(findChar(s, ch));
  }
}
```

Output

3

Approach - By Using in-built library functions - O(n) Time and O(1) Space

We can also use inbuilt library functions to search for a character in a string. This makes the search simple and easier to implement.

```
public class GfG{
  public static int findCharacterIndex(String s, char ch) {
    int idx = s.indexOf(ch);
    return (idx != -1) ? idx : -1;
  }

  public static void main(String[] args) {
```

```
        String s = "geeksforgeeks";
        char ch = 'k';

        int index = findCharacterIndex(s, ch);
        System.out.println(index);
    }
}
```

Output

3

Length of a String

Given a string s**,** the task is to find the length of the string.

Examples:

Input*: s = "abc"*
Output*: 3*

Input: s *= "GeeksforGeeks"*
Output: *13*

Input: s *= ""*
Output: *0*

Using In-built methods

Every programming language offers a built-in method as well to find the length

```
/*package whatever //do not write package name here */

import java.io.*;

class GfG {
    public static void main(String[] args)
    {
        String s = "gfg";
        System.out.println(s.length());
    }
}
```

Output

3

Programming Language	In-Built method to find the length of the string
C	strlen()
C++	size()
Java	length()
Python	len()
JavaScript	length
C#	length()

Writing your Method

The most traditional method of finding the length of the string is by traversing each character through the loop.

- Using a counter, traverse each character of the string with the help of Loop.
- Return the counter value as the length of the string.

```
public class GfG {


    // Method to calculate length of a string
    static int getLength(String s) {
        int cnt = 0;
        for (char c : s.toCharArray()) {
            cnt++;
        }
        return cnt;
    }
```

```
    public static void main(String[] args) {
        String s = "GeeksforGeeks";
        System.out.println(getLength(s));
    }
}
```

Output

13

Time Complexity: O(n), where n is the length of the string.
Auxiliary space: O(1)

Check if a string is substring of another

Given two strings **txt** and **pat**, the task is to find if **pat** is a substring of **txt**. If yes, return the index of the first occurrence, else return **-1**.

Examples :

Input: *txt = "geeksforgeeks", pat = "eks"*
Output: *2*
Explanation: *String "eks" is present at index 2 and 9, so 2 is the smallest index.*

Input: *txt = "geeksforgeeks", pat = "xyz"*
Output: *-1.*
Explanation: *There is no occurrence of "xyz" in "geeksforgeeks"*

Using nested loops - O(m*n) Time and O(1) Space

The basic idea is to iterate through a loop in the string **txt** and for every index in the string **txt**, check whether we can match **pat** starting from this index. This can be done by running a nested loop for traversing the string **pat** and checking for each character of **pat** matches with **txt** or not. If all character of **pat** matches then return the starting index of such substring.

```
// Java program to check if a string is substring of other
// using nested loops

class GfG {

    // Function to find if pat is a substring of txt
    static int findSubstring(String txt, String pat) {
```

```
    int n = txt.length();
    int m = pat.length();

    // Iterate through txt
    for (int i = 0; i <= n - m; i++) {

        // Check for substring match
        int j;
        for (j = 0; j < m; j++) {

            // Mismatch found
            if (txt.charAt(i + j) != pat.charAt(j)) {
                break;
            }
        }

        // If we completed the inner loop, we found a match
        if (j == m) {

            // Return starting index
            return i;
        }
    }

    // No match found
    return -1;
}

public static void main(String[] args) {
    String txt = "geeksforgeeks";
    String pat = "eks";
```

```
        System.out.println(findSubstring(txt, pat));


    }
}
```

Output

2

Time complexity: O(m * n) where **m** and **n** are lengths of **txt** and **pat** respectively.
Auxiliary Space: O(1), as no extra space is required.

Using in-built library functions

This approach uses a built-in function to quickly check if **pattern** is part of **text** or not. This makes the process simple and efficient.

```
// Java program to check if a string is substring of other

// using in-built functions


class GfG {
    public static void main(String[] args) {
        String txt = "geeksforgeeks";
        String pat = "eks";


        // If pat is found, returns the index of first

        // occurrence of pat. Otherwise, returns -1

        int idx = txt.indexOf(pat);


        if (idx != -1)

            System.out.println(idx);

        else

            System.out.println(-1);

    }
}
```

Output

2

Note: The time complexity of in-built functions can differ across different languages.