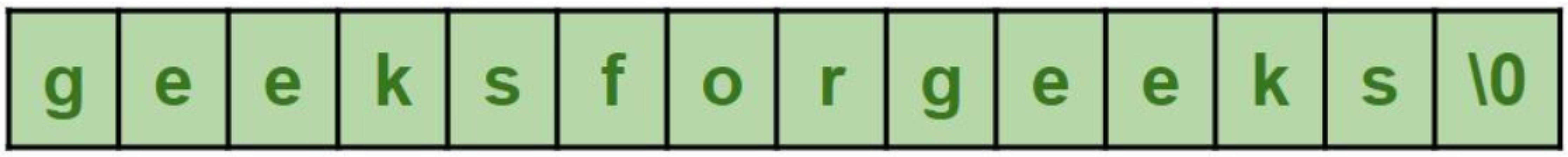
String in C++

C-style string: In C, strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'. In C, the string is actually represented as an array of characters terminated by a null string. Therefore the size of the character array is always one more than that of the number of characters in the actual string. This thing continues to be supported in C++ too. The C++ compiler automatically sets "\0" at the end of the string, during initialization of the array. Here is an example representation of C-Style strings.

```
char str[] = "geeksforgeeks";
```

This will be stored as shown in contiguous memory locations.



C++ Standard String representation and String Class: In C++, one can directly store the collection of characters or text in a string variable, surrounded by double-quotes. C++ provides string class, which supports various operations like copying strings, concatenating strings etc. These strings are also stored in contiguous memory locations and hence any letter can be accessed in constant time.

Initializing string in C++:

```
string str = "geeksforgeeks";
```

- *string*: This refers to the String class, present in *std namespace* and is a part of *iostream*.
- *str*: This refers to the string variable.
- *"geeksforgeeks"*: This refers to the string value.

Example:

```cpp

// C++ program to illustrate
// strings in c++
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string str = "geeksforgeeks";

    // Displaying the string
    cout << str;
    return 0;
}

```

Run

Output:

```
geeksforgeeks
```

The advantages of C++ string class over the C-style strings are:

- We do not have to worry about the size. One can append both characters and other strings to the existing string and the compiler perform the internal operation automatically.
- Can assign a string later: Unlike the C-style string, one can create an empty string initially and then assign value to it later. For eg.,

```cpp

// C++ program to illustrate
// strings in c++
#include <iostream>
using namespace std;

// Main Method
```

```
int main()
{
    // Declaring string
    string str;

    // Initialising the value
    str = "gfg";

    // Displaying the string
    cout << str;
    return 0;
}

```

Run

Output:

```
gfg
```

- Support of operators like +, <, >, ==, <= and >=. The "+" operator is used to append strings and the other operators are used to compare strings lexicographically. In C-style string, strcmp() function was used instead.

- Richer Library: C++ string library has many built-in functions. Two of the important functions of C++ string class are, begin() and end(), that returns iterators which allow random access of elements in a string. This helps to utilise all the algorithmic functions of the algorithm class like sort(), find() etc.

- Can be converted to C-style strings if needed. There is a function c_str(), that takes a C++ string object and converts it into C-style strings.

Program 1: This program illustrates a few important functions of the string class.

```

// C++ program to illustrate
// functions of string class
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string str = "gfg";

    // Calculating the length of
    // the string using length()
    cout << str.length() << endl;

    // Appending string using '+'
    // operator
    str = str + "xyz";

    // Displaying the string
    cout << str << endl;

    // Returns a substring within the range
    cout << str.substr(1, 3) << endl;

    // Finding for a substring
    // within a string
    cout << str.find("fg");
```

Run

Output:

```
3
gfgxyz
fgx
1
```

Working:

1.
   ```
   string str="gfg";
   ```
   Initializing the string with the value "gfg".

2.
   ```
   cout << str.length() << endl;
   ```
   In this, the length() function has been used to find the length of the string 'str'. The length of "gfg" is 3.

3.
   ```
   str = str + "xyz";
   ```
   In this, the '+' operator has been used to append the string "xyz" with the existing string "gfg". This modifies str to "gfgxyz"

4.
   ```
   cout << str << endl;
   ```
   This prints the value of str.

5.
   ```
   cout << str.substr(1, 3) << endl;
   ```
   The **substr()** is a predefined function used for string handling.
   Syntax:
   ```
   string substr(size_t pos, size_t len) const;
   ```
   Parameters:
   - pos: Position of the first character to be copied.
   - len: Length of the sub-string.
   - size_t: It is an unsigned integral type.

   Return value: It returns a string object.
   When the following line is executed then 1 point to the position of the string i.e., 'f' and 3 refers to the length upto which the substring must constitute i.e., till 'x'. Therefore the function returns "fgx"

6.
   ```
   cout << str.find("fg");
   ```
   The find(sub_string) function searches the string for the first occurrence of the substring specified in arguments. It returns the position of the first occurrence of substring. Here "fg" occurs for the first time at position 1.

Program 2: This program gives more insight into the find() function. When the passed string into the find() function is not present in the given string, then a constant called "string::npos" is returned. In this particular program, the passed string is present, hence the first occurrence position is returned.

```

// C++ program to illustrate
// find() function of string class
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string str = "geeksforgeeks";

    // Finding string "eek"
    int res = str.find("eek");

    // Checking for the presence
```

```
    if (res == string::npos)
        cout << "Not Present \n";
    else
        cout << "First Occurrence"
             << " at index " << res;
    return 0;
}

```

Run

Output:

```
First Occurrence at index 1
```

Program 3: This program gives more insight into the find() function. When the passed string into the find() function is not present in the given string, then a constant called "string::npos" is returned.

```

// C++ program to illustrate
// find() function of string class
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string str = "geeksforgeeks";

    // Finding string "eek"
    int res = str.find("xyz");

    // Checking for the presence
    if (res == string::npos)
        cout << "Not Present \n";
    else
        cout << "First Occurrence"
             << " at index " << res;
    return 0;
}

```

Run

Output:

```
Not Present
```

Program 4: The following program illustrates various comparison operators. The comparison operators make the comparison in a lexicographic manner. In the following program "abc" is lexicographically smaller than "xyz".

```

// C++ program to illustrate
// find() function of string class
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string s1 = "abc";
    string s2 = "bcd";

    // Checks for equality
    if (s1 == s2)
        cout << "Same";

    // Checks whether s1 is Smaller
    // than s2 or not
    // than s2 or not
    else if (s1 < s2)
        cout << "Smaller";

    // Checks whether s1 is greater
    // than s2 or not
    else
        cout << "Greater";

    return 0;
}
```

Run

Output:

```
Smaller
```

Note: The comparing operators compare the values in the strings one by one or character by character and not there references.

Program 5: The following program illustrates various comparison operators. The comparison operators make the comparison in a lexicographic manner. In the following program both the strings are same.

```

// C++ program to illustrate
// find() function of string class
#include <iostream>
using namespace std;

// Main Method
int main()
{
    // Initialisation of string
    // in C++
    string s1 = "abc";
    string s2 = "abc";

    // Checks for equality
    if (s1 == s2)
        cout << "Same";

    // Checks whether s1 is Smaller
    // than s2 or not
    else if (s1 < s2)
        cout << "Smaller";

    // Checks whether s1 is greater
    // than s2 or not
    else
        cout << "Greater";

    return 0;
}
```

Run

Output:

```
Same
```

Program 6: This program shows how to take input from the console.

```

// C++ program to illustrate
// reading of a string
#include <iostream>
using namespace std;

// Main Method
int main()
{
    string name;
    cout << "Enter your name";

    // This reads a string in console
    cin >> name;

    cout << endl;
    cout << "Your name is: " << name;
    return 0;
}

```

Run

Input:

```
Sandeep
```

Output:

```
Enter your name

Your name is: Sandeep
```

Program 7: In the previous program if the user enters the name with space in between them, then only the first part of the string gets printed.

```

// C++ program to illustrate
// reading of a string
#include <iostream>
using namespace std;

// Main Method
int main()
{
    string name;
    cout << "Enter your name";

    // This reads a string in console
    cin >> name;

    cout << endl;
    cout << "Your name is: " << name;
    return 0;
}

```

Run

Input:

```
Sandeep Jain
```

Output:

```
Enter your name

Your name is: Sandeep
```

Program 8: Using the getline() function, one can print the string with spaces. The getline() function reads the string until an enter or new line is encountered. This occurs by default, but if you want to specify a delimiter, you can also pass that as an argument.

```

// C++ program to illustrate
// reading of a string
#include <iostream>
using namespace std;

// Main Method
int main()
{
    string name;
    cout << "Enter your name";

    // This reads a string in console
```

```cpp
    getline(cin, name);

    cout << endl;
    cout << "Your name is: " << name;
    return 0;
}

```

Run

Input:

```
Sandeep Jain
```

Output:

```
Enter your name

Your name is: Sandeep Jain
```

Program 9: Using the getline() function, one can print the string with spaces. The getline() function reads the string until an enter or new line is encountered. This occurs by default, but if you want to specify a delimiter, you can also pass that as an argument.

```cpp

// C++ program to illustrate
// reading of a string
#include <iostream>
using namespace std;

// Main Method
int main()
{
    string name;
    cout << "Enter your name";

    // This reads a string in console
    getline(cin, name, '$');

    cout << endl;
    cout << "Your name is: " << name;
    return 0;
}

```

Run

Input:

```
Sandeep Jain$
```

Output:

```
Enter your name

Your name is: Sandeep Jain
```

Program 10: This program shows various ways of travesing a given string.

```cpp

// C++ program to illustrate
// reading of a string
#include <iostream>
using namespace std;

// Main Method
int main()
{
    string str = "gfg";

    // Simplest way to traverse the string
    // using the for loop and accessing
    // each characters as we do in an array
    for (int i = 0; i < str.length(); i++)
```

```
        cout << str[i];
    cout << endl;

    // Using foreach loop to print each
    // characters
    for (char x : str)
        cout << x;
    cout << endl;

    // Using iterators to traverse through the
    // string
    for (auto it = str.begin(); it != str.end(); it++)
        cout << (*it);

    return 0;
```

Run

Output:

```
gfg
gfg
gfg
```

Working:

1. The first way of traversing the string is similar to the way we traverse the array, by accessing the index position of each character in a string.
2. The second way is to use a for each loop to traverse through the array by accessing each character directly.
3. The third way is to traverse using iterators. We can use iterators by calling the begin() and end function(), that points to the start and end position of the given string.
   Here begin() points to the first character 'g' and end() points to the character next to the last character 'g'. 'it' refers to the iterator that is used to traverse the string.

Note: It is always recommended to use C++ string class instead of C-style string. Whenever you pass a C++ string to a function, it is always a good practice to pass it as a reference. This avoids the unnecessary copying of one object to another.

− Sample Problem - string: Pattern Searching in String

Problem 1: This is a basic Pattern search problem where a text and a word will be given. The task is to find in what all positions the word will be present in the text.

Example:

```
Input: text = "geeks for geeks"
       word = "geeks"
Output: 0 10
Explanation: The word "geeks" can be found in two positions 0 & 10.

Input: arr1[] = "Welcome to the world of geeks"
       word = "world"
Output: 15
Explanation: The word "world" can be found at index position 15.
```

Approach: This involves using the find() function repeatedly looking up the word to find the given pattern until the whole word is searched.

Step 1: Look for the pattern in the text using find() and store it in an integer found.

Step 2: Start the while loop

Step 3: Print the found index

Step 4: Use the find() function to look for the pattern, starting the search from (found+1)

Step 5: Repeat Step 3 to Step 4 until (found != string:npos)

```

// C++ code to search for a pattern
// in a given string
#include <bits/stdc++.h>
using namespace std;

// Function to find the occurrences
void printOccurrence(
    string text, string pattern)
{
    // Storing the first occurrence
    int found = text.find(pattern);

    // Traverse through out the string
    while (found != string::npos) {
        // Display the index of
        // occurrence
        cout << found << endl;

        // Finding next occurrence from
        // found+1
        found = text.find(
            pattern, found + 1);
    }
}

// Driver Method
int main()
{
    // The text
```

Run

Output:

```
0
10
```

Sample Problem - string: Returning Floating Point Of A Number

Problem: Given a string of length n and we have to find the number after the decimal point in that string.

Example:

```
Input : string num = 56.789
pos=0
Output: 789 // Here pos is 2

Initially:
string num = 23.342
int pos=0

After:
string num = 23.342
int pos=2

Result:
string num = 23.342
num.substr(2+1) // 2 is pos here
```

The solution lies within the string itself as we have to iterate the string till the decimal point is found and as soon as this appears, we fetch the position. After getting the position of the decimal, we just print the substring of that string only by adding 1 to the position because we don't want to print the decimal in our output.

Below is the implementation of the above approach:

```

// C++ program to return floating point of a number
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find the decimal
string return_floating(string num)
{

    int pos = num.find(".");
    if (pos == string::npos)
        return " ";
    // Returning the desired number after the deciaml
    else
        return num.substr(pos + 1);
}

int main()
{
    string num = "23.342";

    cout << return_floating(num);
    return 0;
}

```

Run

Output:

```
342
```

Time Complexity: O(n) as we are linearly searching the decimal.

Sample Problem - string: Finding one extra character in a string

Problem: Two strings are given of lengths n and n+1. The second string contains all the character of the first string, but there is one extra character. The task is to find the extra character in the second string.

Examples :

```
Input: string str1 = "abcd";

       string str2 = "cbdae";

Output: e

Explanation: str2 contains all the element

of str1 with one extra alphabet 'e'.



Input: string str1 = "kxml";

       string str2 = "klxml";

Output: l

Explanation: str2 contains all the element

of str1 with one extra alphabet 'l'.
```

Approach: This involves creating an empty array of size 26(corresponding to the 26 letters of the English alphabet) and incrementing every corresponding value of the array by 1 for each letter in str2. Now start traversing through str1 and for every letter of str1 decrement the corresponding array values by 1. This will reset the value of the array common to str1 and str2, except the extra letter's value.

Time Complexity: As we are using only the loops for the whole operation, the time complexity of the solution would be O(n).

Space Complexity: We have just created a new array of size 26, therefore the space complexity of this solution would be O(1).

```

// C++ program to find extra
// character in one string
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the extra character
char findcharater(
    string str1, string str2)
{
    // Creating an empty array of
    // 26 size
    int arr[26] = { 0 };

    // Increment the value of arr[]
    // for every corresponding letter
    // in str2
    for (int i = 0; i < 26; i++) {
        arr[str2[i] - 'a']++;
    }

    // Decrement the value of arr[]
    // for every corresponding letter
    // in str1
    for (int i = 0; i < str1.length(); i++) {
        arr[str1[i] - 'a']--;
    }

    // Displaying the extra letter
    for (int i = 0; i < 26; i++) {
```

Run

Output:

```
e
```

Sample Program - string: Pangram Checking

Problem: Given a string check if it is Pangram or not. A pangram is a sentence containing every letter in the English alphabet.

Examples:

```
Input: "The quick brown fox jumps over

the lazy dog"

Output: The quick brown fox jumps over

the lazy dog is a Pangram

Explanation: The statement contains all

the characters from 'a' to 'z'



Input: "abcdefghijklmnopqrstuvwxy"

Output: abcdefghijklmnopqrstuvwxy is not

a Pangram

Explanation: The statement contains all

the characters from 'a' to 'z' except 'z'
```

Approach: This includes creating an empty array of size 26 corresponding to all the letters of the English alphabet which will check whether all the alphabets are contained in the given string or not. We iterate through all the characters of our string and whenever we see a character we mark it in the array by incrementing the place value. Lowercase and Uppercase are considered the same. So 'A' and 'a' are marked in index 0 and similarly 'Z' and 'z' are marked in index 25.

After iterating through all the characters we check whether all the characters are marked or not. If not then return false as this is not a pangram else return true.

Time Complexity: As only a single loop will be used to iterate through the string to check for pangram, therefore the time complexity would be O(n), where n is the length of the given string. The other loop that must be used to check through the updated array would be constant.

Space Complexity: As only a new array of size 26 is created so, it occupies O(26) space which is a constant.

```

// A C++ Program to check if the given
// string is a pangram or not
#include <bits/stdc++.h>
using namespace std;

// Returns true if the string is pangram
// else false
bool checkPangram(string str)
{
    // Empty array of size 26
    int arr[26] = { 0 };

    // Iterating through the string
    for (int i = 0; i < str.length(); i++) {
        // if block to manage uppercase
        if ('A' <= (str[i]) && str[i] <= 'Z') {
            arr[str[i] - 'A']++;
        }
        // if block to manage lowercase
        else if ('a' <= str[i] && str[i] <= 'z') {
            arr[str[i] - 'a']++;
        }
    }

    // Checking for the presence of values
    for (int i = 0; i < 26; i++) {
        if (arr[i] == 0)
            return false;
```

Run

Output:

```
The quick brown fox jumps over the lazy dog is a Pangram
```

Sample Problem - string: Check whether two strings are anagram

Problem: Given two strings, check whether two strings are an anagram of each other. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

```
Input: string1 = "abcd"
       string2 = "bcad"
Output: Yes

Input: string1 = "listen"
       string2 = "silent"
Output: Yes
```

Approach: This involves counting of characters in the string. We assume that the set of possible characters in both strings is small. In the following implementation, it is assumed that the characters are stored using 8 bit and there can be 256 possible characters.

Step 1: Create count arrays of size 256 for both strings. Initialize all values in count arrays as 0.
Step 2: Iterate through every character of both strings and increment the count of character in the corresponding count arrays.
Step 3: Compare count arrays. If both count arrays are same, then return true.

```

// Program to check if both strings
// are anagram of each other
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to check for anagrams
bool areAnagram(
    string s1, string s2,
    int n1, int n2)
{
    // Checking for equal lengths
    if (n1 != n2)
        return false;
    else {
        // Creating an unordered_map
        unordered_map<char, int> map1, map2;

        // Keeping count of characters
        for (int i = 0; i < n1; i++) {
            map1[s1[i]]++;
            map2[s2[i]]++;
        }

        // Checking the occurences
        for (auto x : map1) {
            if (x.second != map2[x.first])
                return false;
        }
        return true;
```

Run

Output:

```
1
```

__builtin_popcount() in C++

The **__builtin_popcount()** is a built-in function of GCC compiler, which takes an integer as an argument and it counts how many set bits it has in its binary representation.

Syntax:

```
__builtin_popcount(int)
```

Parameter: The method takes an integer as the argument.

Return Value: The method returns the number of set bits of the passed argument.

Example:

```
Input: 5

Output: 2

Explanation:
5's binary representation: 000...0101

The first 28 bits are 0 and the last 4 bit represents 5.

As we can see that there are 2 set-bits or 1's in the

representation.



Input: 7

Output: 3

Explanation:
5's binary representation: 000...00111
```

```
The first 28 bits are 0 and the last 4 bit represents 7.

As we can see that there are 3 set-bits or 1's in the

representation.



Input: 8

Output: 3

Explanation:
8's binary representation: 000...01000

The first 28 bits are 0 and the last 4 bit represents 8.

As we can that there are 1 set-bit or 1's in the

representation.
```

Program 1: Find the set-bits of 5 & 8

```

// CPP code to illustrate
// __builtin_popcount() function
#include <iostream>
using namespace std;

// Drivers Method
int main()
{

    // Sample integer
    int n = 5;

    // Counting the set-bits of 5
    cout << __builtin_popcount(n);

    // Counting the set-bits of 8
    cout << __builtin_popcount(8);
    return 0;
}

```

Run

Output:

```
21
```

Note: The __builtin_popcount() takes unsigned integer as the argument. So when a negative integer is passed into the function, the function first converts the negative number into an unsigned integer and then it counts the set-bits of this unsigned representation. This occurs in a 2's complement form, where the bit patterns of the negative number are copied exactly into the positive number. This results in small negative integers being converted into large positive integers.

Example:

```
Input: -1

Output: 32

Explanation:
1's binary representation: 000...0001

when 2's complement is done we get all the 32 bits as 1
```

Program 2: Find the set-bits of -1

```

// CPP code to illustrate
// __builtin_popcount() function
#include <iostream>
using namespace std;

// Drivers Method
int main()
{

    // Sample integer
    int n = -1;

    // Counting the set-bits of -1
    cout << __builtin_popcount(n);
    return 0;
}

```

Run

Output:

```
32
```

Note: In addition to __builtin_popcount(), there are two other functions, namely **__builtin_popcountl()** and **__builtin_popcountll()** that are used to count the set bits of **long integer and long long integer** respectively.