**String**

In C++, a string is a dynamic sequence of characters. std::string simplifies manipulation with built-in functions, offering efficiency and safety.

**Using namespace std**; use it to avoid making use of std:: library again and again

**String initialization in c++**

**Dynamic way:**

        string n;

        getline(cin,n);

        cout<<n;

**Static way :**

        char str[10]= {'C', '+', '+', '\0'};

The \0 is the null character in C and C++. It serves as the string terminator, indicating the end of a string. When used in a character array, it marks the end of the string data.

**String operations**

Use <string> header to work with below operations in c++

**strcpy:** Copy string data.

**strcat**: Concatenate two strings.

**strlen:** Get string length.

**strcmp:** Compare two strings.

**strchr:** Locate character in string.

**NOTE :[ In string Index start from 0 , Length start from 1]**

Example : khadeer  length is 7 , Index of r is 6

**Libraries and Operations**

**<iostream>**

Basic Input/Output: std::cin, std::cout

String Output: std::cout << "Hello";

**<string>**

String Declaration:  string myString = "Hello";

String Concatenation: myString = myString + " World";

String Length: myString.length()

**<cstring>**

String Copying: strcpy(str1, str2);

String Concatenation: strcat(dest, source);

String Length: strlen(str):

**<sstream>**

**istringstream (Input String Stream):**

**Purpose:** Used for reading from strings.

**Example:**

#include <iostream>

#include <sstream

using namespace std;

int main()

{

    string str = "123";

    int numericVar;

    istringstream iss(str);

    iss >> numericVar;

    cout << "Original String: " << str << endl;

    cout << "Converted Numeric Value: " <<numericVar << endl;

    return 0;

}

**Output :** int: 123, Float: 45.6, String: hello

**std::ostringstream (Output String Stream):**

Purpose: Used for writing to strings.

Example:

#include <iostream>

#include <sstream>

int main()

{

    std::ostringstream oss;

    int intValue = 123;

    float floatValue = 45.6;

    std::string stringValue = "hello";

    oss << "Int: " << intValue << ", Float: " << floatValue << ", String: " << stringValue;

    std::string outputStr = oss.str();

    std::cout << "Concatenated String: " << outputStr << std::endl;

    return 0;

}

**OUTPUT :**

**Concatenated String:** Int: 123, Float: 45.6, String: hello

**std::stringstream (Input/Output String Stream):**

**Purpose:** Supports both reading and writing operations on strings.

**Example:**

#include <iostream>

#include <sstream>

using namespace std;

int main() {

    stringstream ss;

    int intValue = 123;

    float floatValue = 45.6;

    string stringValue = "hello";

    ss << "Int: " << intValue << ", Float: " << floatValue << ", String: " << stringValue;

    int newIntValue;

    float newFloatValue;

    string newStringValue;

    ss >> newIntValue >> newFloatValue >> newStringValue;

    cout << "Read values - Int: " << newIntValue << ", Float: " << newFloatValue << ", String: " << newStringValue << endl;

    return 0;

}

**BASIC ALL PROGRAMS ON STRINGS IN C++**

1. <https://leetcode.com/problems/add-strings/>

**DESCRIPTION:**This C++ code implements the "Two Sum" problem using a hash map to efficiently find pairs of numbers that sum up to the target value. Here's a breakdown of the code:

1. **Initialization:**
   * Create an unordered map num\_map to store elements of the input array nums, where the keys are the array elements, and the values are their corresponding indices.
2. **Main Loop:**
   * Use a for loop to iterate through each element of the array nums.
   * Inside the loop:
     + Calculate the complement, which is the difference between the target value and the current element (complement = target - nums[i]).
     + Check if the complement is present in the num\_map. If it is, a pair of indices with the desired sum is found. Return the indices as a vector {num\_map[complement], i}.
     + If the complement is not in the map, add the current element and its index to the num\_map.
3. **Result:**
   * If no such pair is found, return an empty vector {}.

**CODE:**

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

        unordered\_map<int, int> num\_map;

        for (int i = 0; i < nums.size(); i++) {

            int complement = target - nums[i];

            if (num\_map.find(complement) != num\_map.end()) {

                return {num\_map[complement], i};

            }

            num\_map[nums[i]] = i;

        }

        return {};

    }

};

1. <https://leetcode.com/problems/longest-common-prefix/>
2. **Check Empty Input:**
   * If the input vector strs is empty, return an empty string as there is no common prefix.
3. **Prefix Comparison Loop:**
   * Iterate through each character position i of the first string in the array (strs[0]).
   * For each character position, compare the character c from the first string with the corresponding characters in the rest of the strings (strs[j]) starting from the second string.
   * If any of the following conditions are met:
     + The index i is beyond the length of the current string strs[j].
     + The character at position i in the current string strs[j] is different from the character c.
   * Return the substring of the first string (strs[0]) up to the index i. This substring is the longest common prefix found so far.
4. **Result:**
   * If the loop completes without returning, the entire first string (strs[0]) is the longest common prefix among all strings in the array.

#include <vector>

#include <string>

class Solution {

public:

    std::string longestCommonPrefix(std::vector<std::string>& strs) {

        if (strs.empty()) return "";

        for (int i = 0; i < strs[0].length(); i++) {

            char c = strs[0][i];

            for (int j = 1; j < strs.size(); j++) {

                if (i >= strs[j].length() || strs[j][i] != c) {

                    return strs[0].substr(0, i);

                }

            }

        }

        return strs[0];

    }

};

1. <https://leetcode.com/problems/valid-palindrome-ii/>
2. **Initialization:**
   * Initialize two pointers i and j to the beginning and end of the string s.
   * Use a while loop that continues as long as i is less than j.
3. **Palindrome Check Loop:**
   * Inside the loop, compare characters at positions i and j in the string s.
   * If the characters are not equal, return the result of two palindrome checks:
     + Check if the substring from i + 1 to j is a palindrome.
     + Check if the substring from i to j - 1 is a palindrome.
   * If both checks fail, return false.
4. **Move Pointers:**
   * If characters at positions i and j are equal, increment i and decrement j.
5. **Result:**
   * If the loop completes without returning false, the string is a valid palindrome after at most one deletion.
6. **Helper Function isPalindrome:**
   * A helper function to check if a given substring is a palindrome. It uses two pointers (i and j) to compare characters from the start and end of the substring, returning true if it's a palindrome and false otherwise.

#include <string>

class Solution {

public:

    bool validPalindrome(std::string s) {

        int i = 0, j = s.length() - 1;

        while (i < j) {

            if (s[i] != s[j]) {

                return isPalindrome(s, i + 1, j) || isPalindrome(s, i, j - 1);

            }

            i++;

            j--;

        }

        return true;

    }

private:

    bool isPalindrome(const std::string& s, int i, int j) {

        while (i < j) {

            if (s[i] != s[j]) return false;

            i++;

            j--;

        }

        return true;

    }

};

1. <https://leetcode.com/problems/roman-to-integer/>
2. **Roman Values Map:**
   * Initialize an unordered map roman\_values to store the integer values corresponding to each Roman numeral character.
3. **Initialization:**
   * Initialize result to store the final integer value and i as a pointer to iterate through the string s.
4. **Conversion Loop:**
   * Use a while loop that continues as long as i is less than the length of the string s.
   * Inside the loop:
     + Check if the current character at position i is a valid Roman numeral character.
     + If there is a next character (i + 1 < s.length()) and the value of the current character is less than the value of the next character:
       - Add the difference between the values of the next and current characters to the result.
       - Increment i by 2 to skip the next character.
     + Otherwise, add the value of the current character to the result and increment i by 1.
5. **Result:**
   * After the loop, the variable result contains the integer value corresponding to the input Roman numeral string.

#include <unordered\_map>

#include <string>

class Solution {

public:

    int romanToInt(std::string s) {

        std::unordered\_map<char, int> roman\_values = {

            {'I', 1},

            {'V', 5},

            {'X', 10},

            {'L', 50},

            {'C', 100},

            {'D', 500},

            {'M', 1000}

        };

        int result = 0, i = 0;

        while (i < s.length()) {

            if (i + 1 < s.length() && roman\_values[s[i]] < roman\_values[s[i + 1]]) {

                result += roman\_values[s[i + 1]] - roman\_values[s[i]];

                i += 2;

            } else {

                result += roman\_values[s[i]];

                i++;

            }

        }

        return result;

    }

};

1. <https://leetcode.com/problems/implement-strstr/>

This C++ code is an implementation of the Knuth-Morris-Pratt (KMP) algorithm to find the index of the first occurrence of a substring (needle) within a string (haystack).

Here's an explanation of how the algorithm works:

1. **Check Empty Needle:**
   * If the needle is an empty string, return 0, as an empty needle is considered to be present at the beginning of any string.
2. **Compute Prefix Function:**
   * Call the computePrefixFunction function to compute the prefix function (pi) for the needle. The prefix function helps in efficiently skipping unnecessary comparisons.
3. **Main Loop:**
   * Use a for loop to iterate through each character of the haystack.
   * Inside the loop:
     + While the current characters in the haystack and needle do not match and j is greater than 0, update j using the prefix function (failure function).
     + If the characters match, increment j.
     + If j becomes equal to the length of the needle, a match is found. Return the starting index of the match in the haystack (i - needle.size() + 1).
4. **Compute Prefix Function Function:**
   * The computePrefixFunction function calculates the prefix function (pi) for the needle using a while loop and updates the pi array accordingly.
5. **Result:**
   * If no match is found, return -1.

class Solution {

public:

    int strStr(string haystack, string needle) {

        if (needle.empty()) return 0;

        std::vector<int> pi = computePrefixFunction(needle);

        for (int i = 0, j = 0; i < haystack.size(); i++) {

            while (j > 0 && haystack[i] != needle[j]) {

                j = pi[j - 1];

            }

            if (haystack[i] == needle[j]) {

                j++;

            }

            if (j == needle.size()) {

                return i - needle.size() + 1;

            }

        }

        return -1;

    }

private:

    std::vector<int> computePrefixFunction(const std::string& needle) {

        std::vector<int> pi(needle.size());

        pi[0] = 0;

        for (int i = 1; i < needle.size(); i++) {

            int j = pi[i - 1];

            while (j > 0 && needle[i] != needle[j]) {

                j = pi[j - 1];

            }

            if (needle[i] == needle[j]) {

                j++;

            }

            pi[i] = j;

        }

        return pi;

    }

};