std::string class in C++

Difficulty Level : Easy • Last Updated : 17 Feb, 2023

C++ has in its definition a way to represent a **sequence of characters as an object of the class**. This class is called std:: string. The string class stores the characters as a sequence of bytes with the functionality of allowing **access to the single-byte character**.

String vs Character Array

String	Char Array
A string is a **class that defines objects** that be represented as a stream of characters.	A character array is simply an **array of characters** that can be terminated by a null character.
In the case of strings, memory is **allocated dynamically**. More memory can be allocated at run time on demand. As no memory is preallocated, **no memory is wasted**.	The size of the character array has to be **allocated statically**, more memory cannot be allocated at run time if required. Unused allocated **memory is also wasted**
As strings are represented as objects, **no array decay** occurs.	There is a **threat of** array decay in the case of the character array.
Strings are slower when compared to implementation than character array.	Implementation of **character array is faster** than std:: string.
String class defines **a number of functionalities** that allow manifold	Character arrays **do not offer** many **inbuilt functions** to manipulate strings.

String	Char Array
operations on strings.	

Operations on Strings

1) Input Functions

Function	Definition
getline()	This function is used to store a stream of characters as entered by the user in the object memory.
push_back()	This function is used to input a character at the end of the string.
pop_back()	Introduced from C++11 (for strings), this function is used to delete the last character from the string.

Example:

AD

CPP

```
// C++ Program to demonstrate the working of
// getline(), push_back() and pop_back()
#include <iostream>
#include <string> // for string class
using namespace std;

// Driver Code
int main()
{
    // Declaring string
    string str;

    // Taking string input using getline()
    getline(cin, str);

    // Displaying string
    cout << "The initial string is : ";
    cout << str << endl;

    // Inserting a character
    str.push_back('s');

    // Displaying string
    cout << "The string after push_back operation is : ";
    cout << str << endl;

    // Deleting a character
    str.pop_back();

    // Displaying string
    cout << "The string after pop_back operation is : ";
    cout << str << endl;

    return 0;
}
```

Output

```
The initial string is :
The string after push_back operation is : s
The string after pop_back operation is :
```

Time Complexity: O(1)

Space Complexity: O(n) where n is the size of the string

2) Capacity Functions

Function	Definition
capacity()	This function returns the capacity allocated to the string, which can be equal to or more than the size of the string. Additional space is allocated so that when the new characters are added to the string, the operations can be done efficiently.
resize()	This function changes the size of the string, the size can be increased or decreased.
length()	This function finds the length of the string.

Function	Definition
shrink_to_fit()	This function decreases the capacity of the string and makes it equal to the minimum capacity of the string. This operation is useful to save additional memory if we are sure that no further addition of characters has to be made.

Example:

CPP

```cpp
// C++ Program to demonstrate the working of
// capacity(), resize() and shrink_to_fit()
#include <iostream>
#include <string> // for string class
using namespace std;

// Driver Code
int main()
{
    // Initializing string
    string str = "geeksforgeeks is for geeks";

    // Displaying string
    cout << "The initial string is : ";
    cout << str << endl;

    // Resizing string using resize()
    str.resize(13);

    // Displaying string
    cout << "The string after resize operation is : ";
    cout << str << endl;

    // Displaying capacity of string
    cout << "The capacity of string is : ";
    cout << str.capacity() << endl;

    // Displaying length of the string
    cout << "The length of the string is :" << str.length()
         << endl;

    // Decreasing the capacity of string
    // using shrink_to_fit()
    str.shrink_to_fit();

    // Displaying string
    cout << "The new capacity after shrinking is : ";
    cout << str.capacity() << endl;

    return 0;
}
```

Output

```
The initial string is : geeksforgeeks is for geeks
The string after resize operation is : geeksforgeeks
The capacity of string is : 26
The length of the string is :13
The new capacity after shrinking is : 13
```

Time Complexity: O(1)

Space Complexity: O(n) where n is the size of the string

3) Iterator Functions

Function	Definition
begin()	This function returns an iterator to the beginning of the string.
end()	This function returns an iterator to the next to the end of the string.
rbegin()	This function returns a reverse iterator pointing at the end of the string.
rend()	This function returns a reverse iterator pointing to the previous of beginning of the string.
cbegin()	This function returns a constant iterator pointing to the beginning of the string, it cannot be used to modify the contents it points-to.
cend()	This function returns a constant iterator pointing to the next of end of the string, it cannot be used to modify the contents it points-to.
crbegin()	This function returns a constant reverse iterator pointing to the end of the string, it cannot be used to modify the contents it points-to.
crend()	This function returns a constant reverse iterator pointing to the previous of beginning of the string, it cannot be used to modify the contents it points-to.

Algorithm:

1. *Declare a string*
2. *Try to iterate the string using all types of iterators*
3. *Try modification of the element of the string.*
4. *Display all the iterations.*

Example:

CPP

```
// C++ Program to demonstrate the working of
// begin(), end(), rbegin(), rend(), cbegin(), cend(), crbegin(), crend()
#include <iostream>
#include <string> // for string class
using namespace std;

// Driver Code
int main()
{
    // Initializing string`
    string str = "geeksforgeeks";

    // Declaring iterator
    std::string::iterator it;

    // Declaring reverse iterator
    std::string::reverse_iterator it1;
    cout<<"Str:"<<str<<"\n";
    // Displaying string
    cout << "The string using forward iterators is : ";
    for (it = str.begin(); it != str.end(); it++){
        if(it == str.begin()) *it='G';
        cout << *it;
    }
    cout << endl;

      str = "geeksforgeeks";
    // Displaying reverse string
    cout << "The reverse string using reverse iterators is "
            ": ";
    for (it1 = str.rbegin(); it1 != str.rend(); it1++){
        if(it1 == str.rbegin()) *it1='S';
        cout << *it1;
    }
    cout << endl;

  str = "geeksforgeeks";
  //Displaying String
  cout<<"The string using constant forward iterator is :";
  for(auto it2 = str.cbegin(); it2!=str.cend(); it2++){
        //if(it2 == str.cbegin()) *it2='G';
        //here modification is NOT Possible
        //error: assignment of read-only location
        //As it is a pointer to the const content, but we can inc/dec-rement the itera
        cout<<*it2;
    }
    cout<<"\n";

    str = "geeksforgeeks";
    //Displaying String in reverse
    cout<<"The reverse string using constant reverse iterator is :";
    for(auto it3 = str.crbegin(); it3!=str.crend(); it3++){
        //if(it2 == str.cbegin()) *it2='S';
        //here modification is NOT Possible
        //error: assignment of read-only location
        //As it is a pointer to the const content, but we can inc/dec-rement the itera
        cout<<*it3;
    }
    cout<<"\n";

      return 0;
}

//Code modified by Balakrishnan R (rbkraj000)
```

Output

```
Str:geeksforgeeks
The string using forward iterators is : Geeksforgeeks
The reverse string using reverse iterators is : Skeegrofskeeg
The string using constant forward iterator is :geeksforgeeks
The reverse string using constant reverse iterator is :skeegrofskeeg
```

Time Complexity: O(1)

Space Complexity: O(n) where n is the size of the string

4) Manipulating Functions:

Function	Definition
copy(“char array”, len, pos)	This function copies the substring in the target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied, and starting position in the string to start copying.
swap()	This function swaps one string with another

Example:

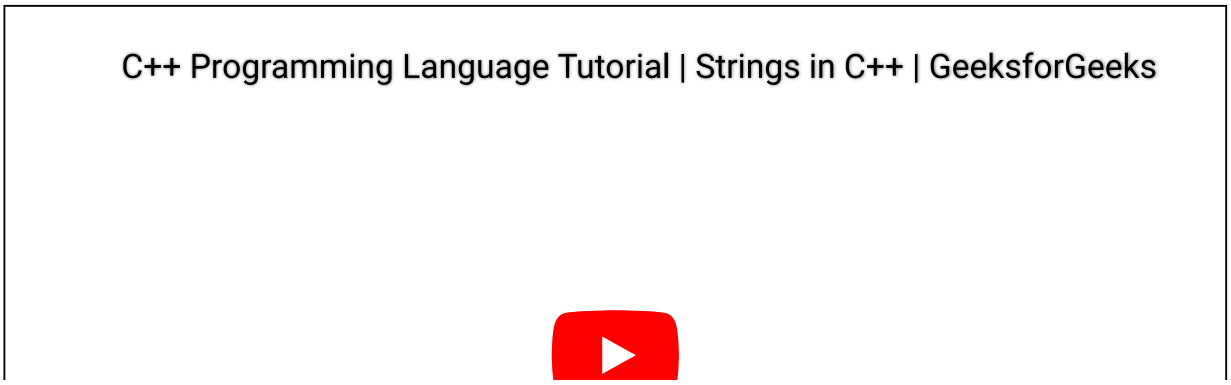
CPP

```cpp
// C++ Program to demonstrate the working of
// copy() and swap()
#include <iostream>
#include <string> // for string class
using namespace std;

// Driver Code
int main()
{
    // Initializing 1st string
    string str1 = "geeksforgeeks is for geeks";

    // Declaring 2nd string
    string str2 = "geeksforgeeks rocks";

    // Declaring character array
    char ch[80];

    // using copy() to copy elements into char array
    // copies "geeksforgeeks"
    str1.copy(ch, 13, 0);

    // Displaying char array
    cout << "The new copied character array is : ";
    cout << ch << endl;

    // Displaying strings before swapping
    cout << "The 1st string before swapping is : ";
    cout << str1 << endl;
    cout << "The 2nd string before swapping is : ";
    cout << str2 << endl;

    // using swap() to swap string content
    str1.swap(str2);

    // Displaying strings after swapping
    cout << "The 1st string after swapping is : ";
    cout << str1 << endl;
    cout << "The 2nd string after swapping is : ";
    cout << str2 << endl;

    return 0;
}
```

Output

```
The new copied character array is : geeksforgeeks
The 1st string before swapping is : geeksforgeeks is for geeks
The 2nd string before swapping is : geeksforgeeks rocks
The 1st string after swapping is : geeksforgeeks rocks
The 2nd string after swapping is : geeksforgeeks is for geeks
```

Must Read: C++ String Class and its Applications


C++ Programming Language Tutorial | Strings in C++ | GeeksforGeeks

480

Related Articles

1. Behavior of virtual function in the derived class from the base class and abstract class
2. How to convert a class to another class type in C++?
3. Base Class Pointer Pointing to Derived Class Object in C++
4. Difference between Base class and Derived class in C++
5. Can a C++ class have an object of self type?
6. Hiding of all Overloaded Methods with Same Name in Base Class in C++
7. Why is the Size of an Empty Class Not Zero in C++?
8. Simulating final Class in C++
9. What happens when more restrictive access is given to a derived class method in C++?

Raw String Literal in C++

Difficulty Level : Easy • Last Updated : 27 Jan, 2023

A Literal is a constant variable whose value does not change during the lifetime of the program. Whereas, a raw string literal is a string in which the escape characters like ' **\n**, **\t,** or **\"** ' of C++ are not processed. Hence, a raw string literal that starts with **R"(and ends in)".**

The syntax for Raw string Literal:

```
R "delimiter( raw_characters )delimiter" // delimiter is the end of logical
entity
```

Here, delimiter is optional and it can be a character except the backslash{ / }, whitespaces{ }, and parentheses { () }.

These raw string literals allow a series of characters by writing precisely its contents like raw character sequence**.**

AD

Example:

Ordinary String Literal

```
"\\\\n"
```

Raw String Literal

```
   \/-- Delimiter
R"(\\n)"
      /\-- Delimiter
```

Difference between an Ordinary String Literal and a Raw String Literal:

Ordinary String Literal	Raw String Literal
It does not need anything to be defined.	It needs a defined line{ parentheses ()} to start with the prefix **R.**
It does not allow/include nested characters.	It allows/includes nested character implementation.
It does not ignore any special meaning of character and implements their special characteristic.	It ignores all the special characters like **\n** and **\t** and treats them like normal text.

Example of Raw String Literal:

CPP

```cpp
// C++ program to demonstrate working of raw string literal
#include <iostream>
using namespace std;

// Driver Code
int main()
{
    // A Normal string
    string string1 = "Geeks.\nFor.\nGeeks.\n";

    // A Raw string
    string string2 = R"(Geeks.\nFor.\nGeeks.\n)";

    cout << string1 << endl;

    cout << string2 << endl;

    return 0;
}
```

Output

```
Geeks.
For.
Geeks.

Geeks.\nFor.\nGeeks.\n
```

Time Complexity: O(n)

Space complexity: O(n)

This article is contributed by **MAZHAR IMAM KHAN**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

144

Related Articles

1. How to Print String Literal and Qstring With Qdebug in C++?
2. ASCII NULL, ASCII 0 ('0') and Numeric literal 0
3. Integer literal in C/C++ (Prefixes and Suffixes)
4. Multi-Character Literal in C/C++
5. std::string::length, std::string::capacity, std::string::size in C++ STL
6. std::string::replace , std::string::replace_if in C++
7. std::string::replace_copy(), std::string::replace_copy_if in C++
8. std::string::append vs std::string::push_back() vs Operator += in C++
9. std::string::remove_copy(), std::string::remove_copy_if() in C++
10. Check if a string can be formed from another string using given constraints

Array of Strings in C++ – 5 Different Ways to Create

Difficulty Level : Easy • Last Updated : 11 Mar, 2023

In C++, a string is usually just an array of (or a reference/points to) characters that ends with the NULL character '**\0**'. A string is a 1-dimensional array of characters and an array of strings is a 2-dimensional array of characters where each row contains some string.

Below are the 5 different ways to create an Array of Strings in C++:

1. Using **Pointers**
2. Using **2-D Array**
3. Using the **String Class**
4. Using the **Vector Class**
5. Using the **Array Class**

1. Using Pointers

Pointers are the symbolic representation of an address. In simple words, a pointer is something that stores the address of a variable in it. In this method, an array of string literals is created by an array of pointers in which each pointer points to a particular string.

Example:

C++

```
// C++ program to demonstrate
// array of strings using
// pointers character array
#include <iostream>

// Driver code
int main()
{
  // Initialize array of pointer
  const char* colour[4]
      = { "Blue", "Red", "Orange", "Yellow" };

  // Printing Strings stored in 2D array
  for (int i = 0; i < 4; i++)
      std::cout << colour[i] << "\n";

  return 0;
}
```

Output

```
Blue
Red
Orange
Yellow
```

Explanation:

- The number of strings is fixed, but needn't be. The 4 may be omitted, and the compiler will compute the correct size.
- These strings are constants and their contents cannot be changed. Because string literals (literally, the quoted strings) exist in a read-only area of memory, we must specify "const" here to prevent unwanted accesses that may crash the program.

2. Using a 2D array

A 2-D array is the simplest form of a multidimensional array in which it stores the data in a tabular form. This method is useful when the length of all strings is known and a particular memory footprint is desired. Space for strings will be allocated in a single block

Example:

C++

```cpp
// C++ program to demonstrate
// array of strings using
// 2D character array
#include <iostream>

// Driver code
int main()
{
  // Initialize 2D array
  char colour[4][10]
      = { "Blue", "Red", "Orange", "Yellow" };

  // Printing Strings stored in 2D array
  for (int i = 0; i < 4; i++)
      std::cout << colour[i] << "\n";

  return 0;
}
```

Output

```
Blue
Red
Orange
Yellow
```

Explanation:

- Both the number of strings and the size of the strings are fixed. The 4, again, may be left out, and the appropriate size will be computed by the compiler. The second dimension, however, must be given (in this case, 10), so that the compiler can choose an appropriate memory layout.
- Each string can be modified but will take up the full space given by the second dimension. Each will be laid out next to each other in memory, and can't change size.
- Sometimes, control over the memory footprint is desirable, and this will allocate a region of memory with a fixed, regular layout.

3. Using the String class

The STL string or string class may be used to create an array of mutable strings. In this method, the size of the string is not fixed, and the strings can be changed which somehow makes it dynamic in nature nevertheless **std::string** can be used to create a string array using in-built functions.

Example:

C++

```
// C++ program to demonstrate
// array of strings using
// string class
#include <iostream>
#include <string>

// Driver code
int main()
{
  // Initialize String Array
  std::string colour[4]
      = { "Blue", "Red", "Orange", "Yellow" };

  // Print Strings
  for (int i = 0; i < 4; i++)
      std::cout << colour[i] << "\n";
}
```

Output

```
Blue
Red
Orange
Yellow
```

Explanation:

The array is of fixed size, but needn't be. Again, the 4 here may be omitted, and the compiler will determine the appropriate size of the array. The strings are also mutable, allowing them to be changed.

4. Using the vector class

A **vector** is a dynamic array that doubles its size whenever a new character is added that exceeds its limit. The STL container vector can be used to dynamically allocate an array that can vary in size.

This is only usable in C++, as C does not have classes. Note that the initializer-list syntax here requires a compiler that supports the 2011 C++ standard, and though it is quite likely your compiler does, it is something to be aware of.

Example:

C++

```
// C++ program to demonstrate
// array of strings using
// vector class
#include <iostream>
#include <string>
#include <vector>

// Driver code
int main()
{
  // Declaring Vector of String type
  // Values can be added here using
  // initializer-list
  // syntax
  std::vector<std::string> colour{"Blue", "Red",
                                  "Orange"};

  // Strings can be added at any time
  // with push_back
  colour.push_back("Yellow");

  // Print Strings stored in Vector
  for (int i = 0; i < colour.size(); i++)
      std::cout << colour[i] << "\n";
}
```

Output

```
Blue
Red
Orange
Yellow
```

Explanation:

- Vectors are dynamic arrays and allow you to add and remove items at any time.
- Any type or class may be used in vectors, but a given vector can only hold one type.

5. Using the Array Class

An array is a homogeneous mixture of data that is stored continuously in the memory space. The STL container array can be used to allocate a fixed-size array. It may be used very similarly to a vector, but the size is always fixed.

Example:

C++

```
// C++ program to demonstrate
// array of string using STL array
#include <array>
#include <iostream>
#include <string>

// Driver code
int main()
{
  // Initialize array
  std::array<std::string, 4> colour{"Blue", "Red",
                                    "Orange", "Yellow"};

  // Printing Strings stored in array
  for (int i = 0; i < 4; i++)
      std::cout << colour[i] << "\n";

  return 0;
}
```

Output

```
Blue
Red
Orange
Yellow
```

These are by no means the only ways to make a collection of strings. C++ offers several container classes, each of which has various tradeoffs and features, and all of them exist to fill requirements that you will have in your projects. Explore and have fun!

Conclusion: Out of all the methods, Vector seems to be the best way for creating an array of Strings in C++.

This article is contributed by **Kartik Ahuja**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.s.

Tokenizing a string in C++

Difficulty Level : Medium • Last Updated : 02 Jan, 2023

Tokenizing a string denotes splitting a string with respect to some delimiter(s). There are many ways to tokenize a string. In this article four of them are explained:

Using stringstream

A **stringstream** associates a string object with a stream allowing you to read from the string as if it were a stream.

Below is the C++ implementation :

C++

```
// Tokenizing a string using stringstream
#include <bits/stdc++.h>

using namespace std;

int main()
{

    string line = "GeeksForGeeks is a must try";

    // Vector of string to save tokens
    vector <string> tokens;

    // stringstream class check1
    stringstream check1(line);

    string intermediate;

    // Tokenizing w.r.t. space ' '
    while(getline(check1, intermediate, ' '))
    {
        tokens.push_back(intermediate);
    }

    // Printing the token vector
    for(int i = 0; i < tokens.size(); i++)
        cout << tokens[i] << '\n';
}
```

Output

AD

```
GeeksForGeeks
is
a
must
try
```

Time Complexity: O(n) where n is the length of string.
Auxiliary Space: O(n-d) where n is the length of string and d is the number of delimiters.

Using strtok()

```
// Splits str[] according to given delimiters.
// and returns next token. It needs to be called
// in a loop to get all tokens. It returns NULL
// when there are no more tokens.
char * strtok(char str[], const char *delims);
```

Below is the C++ implementation :

C++

```
// C/C++ program for splitting a string
// using strtok()
#include <stdio.h>
#include <string.h>
```

```
int main()
{
    char str[] = "Geeks-for-Geeks";

    // Returns first token
    char *token = strtok(str, "-");

    // Keep printing tokens while one of the
    // delimiters present in str[].
    while (token != NULL)
    {
        printf("%s\n", token);
        token = strtok(NULL, "-");
    }

    return 0;
}
```

Output

```
Geeks
for
Geeks
```

Time Complexity: O(n) where n is the length of string.

Auxiliary Space: O(1).

Another Example of strtok() :

C

```
// C code to demonstrate working of
// strtok
#include <string.h>
#include <stdio.h>

// Driver function
int main()
{
 // Declaration of string
    char gfg[100] = " Geeks - for - geeks - Contribute";

    // Declaration of delimiter
    const char s[4] = "-";
    char* tok;

    // Use of strtok
    // get first token
    tok = strtok(gfg, s);

    // Checks for delimiter
    while (tok != 0) {
        printf(" %s\n", tok);

        // Use of strtok
        // go through other tokens
        tok = strtok(0, s);
    }

    return (0);
}
```

Output

```
Geeks
for
geeks
Contribute
```

Time Complexity: O(n) where n is the length of string.

Auxiliary Space: O(1).

Using strtok_r()

Just like strtok() function in C, **strtok_r()** does the same task of parsing a string into a sequence of tokens. strtok_r() is a reentrant version of strtok().

There are two ways we can call strtok_r()

```
// The third argument saveptr is a pointer to a char *
// variable that is used internally by strtok_r() in
// order to maintain context between successive calls
// that parse the same string.
char *strtok_r(char *str, const char *delim, char **saveptr);
```

Below is a simple C++ program to show the use of strtok_r() :

C++

```
// C/C++ program to demonstrate working of strtok_r()
// by splitting string based on space character.
#include<stdio.h>
#include<string.h>

int main()
{
    char str[] = "Geeks for Geeks";
    char *token;
    char *rest = str;

    while ((token = strtok_r(rest, " ", &rest)))
        printf("%s\n", token);

    return(0);
}
```

Output

```
Geeks
for
Geeks
```

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

Using std::sregex_token_iterator

In this method the tokenization is done on the basis of regex matches. Better for use cases when multiple delimiters are needed.

Below is a simple C++ program to show the use of std::sregex_token_iterator:

C++

```
// CPP program for above approach
#include <iostream>
#include <regex>
#include <string>
#include <vector>

/**
 * @brief Tokenize the given vector
   according to the regex
 * and remove the empty tokens.
 *
 * @param str
 * @param re
 * @return std::vector<std::string>
 */
std::vector<std::string> tokenize(
                     const std::string str,
                          const std::regex re)
{
    std::sregex_token_iterator it{ str.begin(),
                             str.end(), re, -1 };
    std::vector<std::string> tokenized{ it, {} };

    // Additional check to remove empty strings
    tokenized.erase(
        std::remove_if(tokenized.begin(),
                            tokenized.end(),
                       [](std::string const& s) {
                           return s.size() == 0;
                       }),
        tokenized.end());

    return tokenized;
}

// Driver Code
int main()
{
    const std::string str = "Break string
                   a,spaces,and,commas";
    const std::regex re(R"([\s|,]+)");

    // Function Call
    const std::vector<std::string> tokenized =
                           tokenize(str, re);

    for (std::string token : tokenized)
        std::cout << token << std::endl;
    return 0;
}
```

Output

```
Break
string
a
spaces
and
commas
```

Time Complexity: O(n * d) where n is the length of string and d is the number of delimiters.
Auxiliary Space: O(n)

75

Related Articles

1. Count characters of a string which when removed individually makes the string equal to another string
2. Generate string by incrementing character of given string by number present at corresponding index of second string

strrchr() in C++

C++ strrchr() function finds the location of the **last occurrence** of the **specified** character in the given string and returns the pointer to it. It returns the NULL pointer if the character is not found.

It is a standard library function of C which is inherited by C++ so it only works on C-style strings (i.e. array of characters). It is defined inside **<cstring>** and **<string.h>** header files.

Syntax:

```
char *strrchr(const char *str, int chr);
```

Parameter:

- **str:** specifies the pointer to the null-terminated string in which the search is to be performed.
- **chr:** specifies the character to be searched.

Return Value:

- The function returns a pointer to the last location of **chr** in the string if the **chr** is found.
- If **chr** is not found, it returns a **NULL point**

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

Example:

C++

```
// C++ program to demonstrate working strchr()
#include <cstring>
#include <iostream>
using namespace std;

int main()
{
    char str[] = "This is a string";
    char* ch = strrchr(str, 'i');
    cout << "Index of last occurence of i: "
         << ch - str + 1;
    return 0;
}
```

Output

```
9
```

Time Complexity: O(n),

Space Complexity: O(1),

where **n** is the length of the string.

Practical Application of strrchr() function in C++

Since it returns the entire string after the last occurrence of a particular character, it can be used to **extract the suffix of a string**. For e.g to know the entire leading zeroes in a denomination when we know the first number.

Example:

C++

```
// C++ code to demonstrate the application of
// strrchr()
#include <cstring>
#include <iostream>
using namespace std;
```

```
int main()
{

    // initializing the denomination
    char denom[] = "Rs 10000000";

    // Printing original string
    cout << "The original string is : " << denom;

    // initializing the initial number
    char first = '1';
    char* entire;

    // Use of strrchr()
    // returns entire number
    entire = strrchr(denom, first);

    cout << "\nThe denomination value is : " << entire;

    return 0;
}
```

Output

```
The original string is : Rs 10000000
The denomination value is : 10000000
```

Time Complexity: O(N), as time complexity for function strrhcr() is O(N) where N is the length of given String .

Auxiliary Space: O(1), since we are not using any extra space.

This article is contributed by **Ayush Saxena** and **Vaishnavi Tripathi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

36

Related Articles

1. C++ Error - Does not name a type
2. Execution Policy of STL Algorithms in Modern C++
3. C++ Program To Print Pyramid Patterns

stringstream in C++ and its Applications

Difficulty Level : Medium • Last Updated : 28 Mar, 2023

A stringstream associates a string object with a stream allowing you to read from the string as if it were a stream (like cin). To use stringstream, we need to include ***sstream*** header file. The stringstream class is extremely useful in parsing input.

Basic methods are:

1. **clear()-** To clear the stream.
2. **str()-** To get and set string object whose content is present in the stream.
3. **operator <<-** Add a string to the stringstream object.
4. **operator >>-** Read something from the stringstream object.

Examples:

1. Count the number of words in a string

Examples:

Input: *Asipu Pawan Kumar*
Output: *3*

Input: *Geeks For Geeks Ide*

Output: *4*

Below is the C++ program to implement the above approach-

C++

```
// C++ program to count words in
// a string using stringstream.
#include <iostream>
#include <sstream>
#include<string>
using namespace std;

int countWords(string str)
{
    // Breaking input into word
    // using string stream

    // Used for breaking words
    stringstream s(str);

    // To store individual words
    string word;

    int count = 0;
    while (s >> word)
        count++;
    return count;
}

// Driver code
int main()
{
    string s = "geeks for geeks geeks "
               "contribution placements";
    cout << " Number of words are: " << countWords(s);
    return 0;
}
```

Output

```
Number of words are: 6
```

Time complexity: O(n*log(n)).

Auxiliary space: O(n).

2. Print frequencies of individual words in a string

Examples:

Input: *Geeks For Geeks Quiz Geeks Quiz Practice Practice*

Output: *For -> 1*

Geeks -> 3

Practice -> 2

Quiz -> 2

Input: *Word String Frequency String*

Output: *Frequency -> 1*

String -> 2

Word -> 1

Below is the C++ program to implement the above approach-

C++

```
// C++ program to demonstrate use
// of stringstream to count
// frequencies of words.
#include <bits/stdc++.h>
#include <iostream>
#include <sstream>
#include<string>
using namespace std;

void printFrequency(string st)
{
    // Each word it mapped to
    // it's frequency
    map<string, int>FW;

    // Used for breaking words
    stringstream ss(st);

    // To store individual words
    string Word;

    while (ss >> Word)
        FW[Word]++;

    map<string, int>::iterator m;
    for (m = FW.begin(); m != FW.end(); m++)
        cout << m->first << "-> "
             << m->second << "\n";
}

// Driver code
```

```
int main()
{
    string s = "Geeks For Geeks Ide";
    printFrequency(s);
    return 0;
}
```

Output

```
For-> 1
Geeks-> 2
Ide-> 1
```

Time complexity: O(n*log(n)).

Auxiliary space: O(n).

3. Convert Integer to string

Since, the insertion and extraction operators of string stream work with different data types. So that's why it works well with integers.

We will insert an integer into the string stream and after extracting that into a string, that integer value will become a string.

Code-

C++

```
// C++ program to demonstrate the
// use of a stringstream to
// convert int to string
#include <iostream>
#include <sstream>
using namespace std;

// Driver code
int main()
{
    int val=123;

    // object from the class stringstream
    stringstream geek;

    // inserting integer val in geek stream
    geek << val;

    // The object has the value 123
    // and stream it to the string x
    string x;
    geek >> x;
    // Now the string x holds the
    // value 123
    cout<<x+"4"<<endl;
    return 0;
}
```

Output-

```
1234
```

Time complexity: O(n),n is the length of the integer

Auxiliary space: O(n)

Removing spaces from a string using Stringstream
Converting Strings to Numbers in C/C++

This article is contributed by **ASIPU PAWAN KUMAR**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

346

Related Articles

1. StringStream in C++ for Decimal to Hexadecimal and back
2. Removing spaces from a string using Stringstream
3. Finding number of days between two dates using StringStream
4. Find words which are greater than given length k using stringstream
5. C++ String Class and its Applications
6. strchr() function in C++ and its applications
7. C++ string class and its applications
8. MakeFile in C++ and its applications
9. Different Types of Queues and its Applications