- Difficulty Level : Easy
- Last Updated : 06 Jul, 2022

Read
Discuss

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. 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 decayoccurs.	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 operations on strings.	Character arrays do not offer many inbuilt functions to manipulate 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:

- 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 : geeksforgeek
The string after push_back operation is : geeksforgeeks
The string after pop_back operation is : geeksforgeek
```

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.
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 : 15
```

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 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 at beginning of the string.

Example:

- CPP

```
// C++ Program to demonstrate the working of
// begin(), end(), rbegin(), rend()
#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;

    // Displaying string
    cout << "The string using forward iterators is : ";
    for (it = str.begin(); it != str.end(); it++)
        cout << *it;
    cout << endl;

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

    return 0;
}
```

Output

The string using forward iterators is : geeksforgeeks

The reverse string using reverse iterators is : skeegrofskeeg

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 other.

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

C++ Strings

In this tutorial, you'll learn to handle strings in C++. You'll learn to declare them, initialize them and use them for various input/output operations.

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

- Strings that are objects of string class (The Standard C++ Library string class)
- C-strings (C-style Strings)

C-strings

In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called C-strings.

C-strings are arrays of type `char` terminated with null character, that is, `\0` (ASCII value of null character is 0).

How to define a C-string?

```
char str[] = "C++";
```

In the above code, `str` is a string and it holds 4 characters.

Although, "`C++`" has 3 character, the null character `\0` is added to the end of the string automatically.

Alternative ways of defining a string

```
char str[4] = "C++";
```

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

```
char str[4] = {'C','+','+','\0'};
```

Like arrays, it is not necessary to use all the space allocated for the string. For example:

```
char str[100] = "C++";
```

Example 1: C++ String to read a word

C++ program to display a string entered by user.

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

int main()
{
    char str[100];

    cout << "Enter a string: ";
    cin >> str;
    cout << "You entered: " << str << endl;

    cout << "\nEnter another string: ";
    cin >> str;
    cout << "You entered: "<<str<<endl;

    return 0;
}
```

Output

```
Enter a string: C++
You entered: C++
```

```
Enter another string: Programming is fun.
You entered: Programming
```

Notice that, in the second example only "Programming" is displayed instead of "Programming is fun".

This is because the extraction operator >> works as `scanf()` in C and considers a space " " has a terminating character.

Example 2: C++ String to read a line of text

C++ program to read and display an entire line entered by user.

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

int main()
{
    char str[100];
    cout << "Enter a string: ";
    cin.get(str, 100);

    cout << "You entered: " << str << endl;
    return 0;
}
```

Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

To read the text containing blank space, `cin.get` function can be used. This function takes two arguments.

First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array.

In the above program, `str` is the name of the string and `100` is the maximum size of the array.

---

string Object

In C++, you can also create a string object for holding strings.

Unlike using char arrays, string objects has no fixed length, and can be extended as per your requirement.

Example 3: C++ string using string data type

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

int main()
{
    // Declaring a string object
    string str;
    cout << "Enter a string: ";
    getline(cin, str);

    cout << "You entered: " << str << endl;
    return 0;
}
```

Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

In this program, a string `str` is declared. Then the string is asked from the user.

Instead of using `cin>>` or `cin.get()` function, you can get the entered line of text using `getline()`.

`getline()` function takes the input stream as the first parameter which is `cin` and `str` as the location of the line to be stored.

C++ provides following two types of string representations –

- The C-style character string.
- The string class type introduced with Standard C++.

The C-Style Character String

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

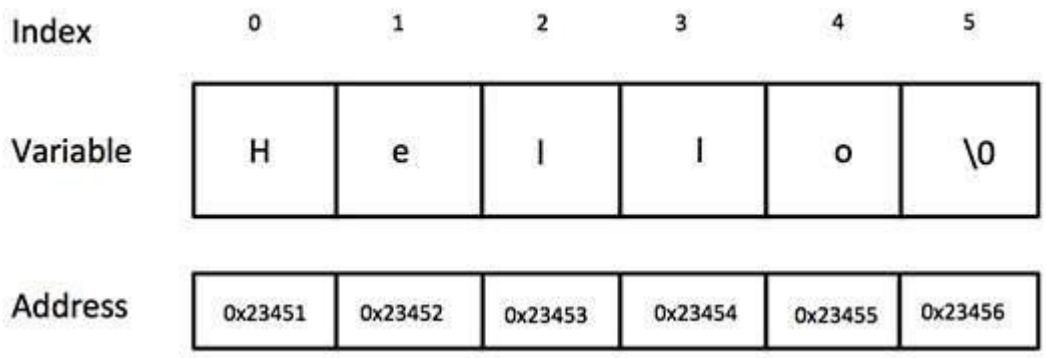
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows –

```
char greeting[] = "Hello";
```

Following is the memory presentation of above defined string in C/C++ –



Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string –

Live Demo

```
#include <iostream>

using namespace std;

int main () {

   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   cout << "Greeting message: ";
   cout << greeting << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Greeting message: Hello
```

C++ supports a wide range of functions that manipulate null-terminated strings –

Sr.No	Function & Purpose
1	strcpy(s1, s2);

	Copies string s2 into string s1.
2	strcat(s1, s2); Concatenates string s2 onto the end of string s1.
3	strlen(s1); Returns the length of string s1.
4	strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.
6	strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

Following example makes use of few of the above-mentioned functions –

Live Demo

```
#include <iostream>
#include <cstring>

using namespace std;

int main () {

   char str1[10] = "Hello";
   char str2[10] = "World";
```

```
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows –

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

The String Class in C++

The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality. Let us check the following example –

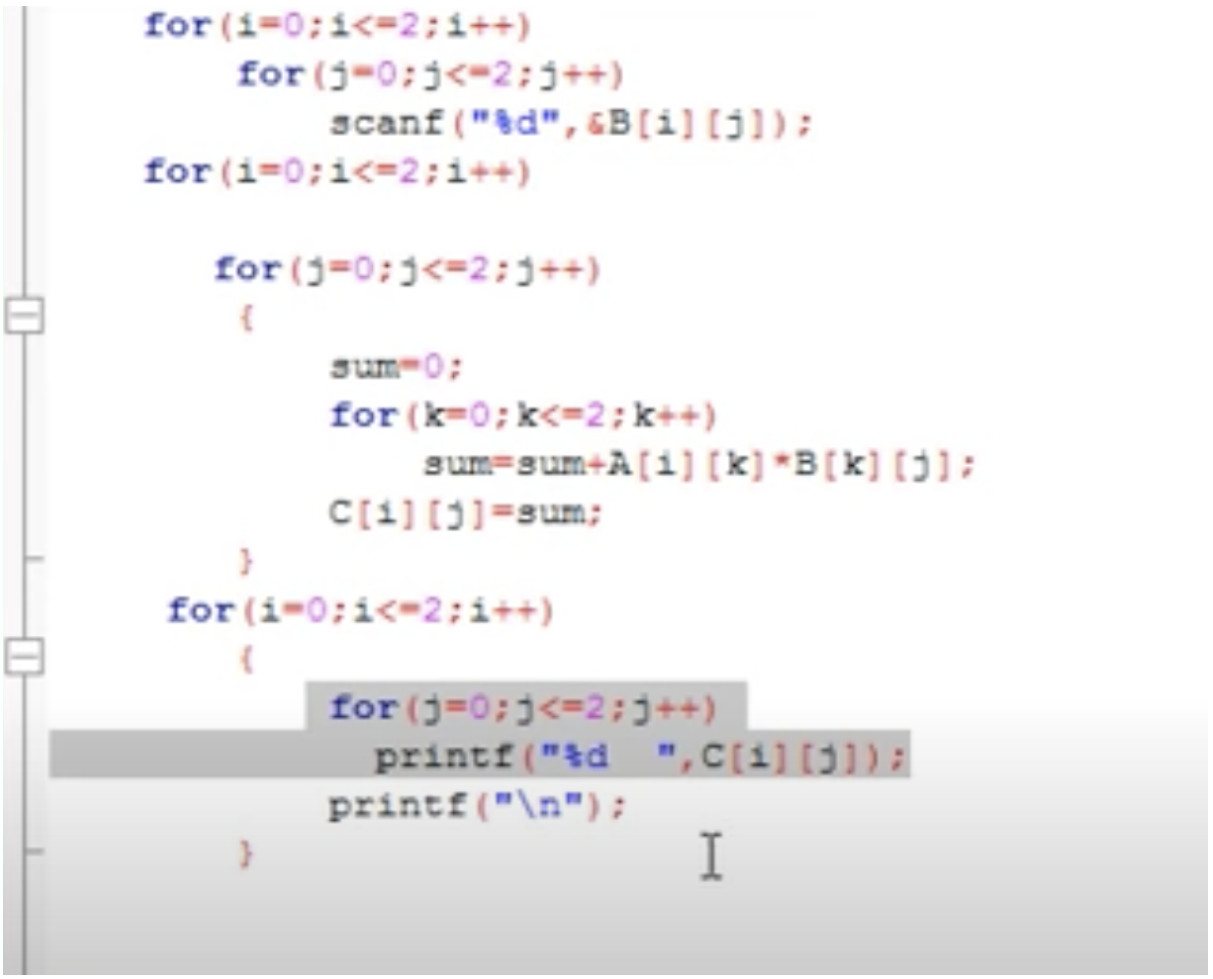
Live Demo

```
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
```

```
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows –

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```


for(i=0;i<=2;i++)
for(j=0;j<=2;j++)
scanf("%d",&B[i][j]);
for(i=0;i<=2;i++)
for(j=0;j<=2;j++)
{
sum=0;
for(k=0;k<=2;k++)
sum=sum+A[i][k]*B[k][j];
C[i][j]=sum;
}
for(i=0;i<=2;i++)
{
for(j=0;j<=2;j++)
printf("%d ",C[i][j]);
printf("\n");
}