Report on String, Vector in STL

a/ String:

1/ Definition:

Strings are objects that represent sequences of characters. 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 standard string class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

The string class is an instantiation of the basic_string class template that uses char (i.e., bytes) as its character type, with its default char_traits and allocator types (see basic_string for more info on the template).

**Note:* that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

2/ Storage for Strings in C:

In C, a string can be referred 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. For example, if str[] is an auto variable then string is stored in stack segment, if it’s a global or static variable then stored in data segment, etc.

Strings using character pointers
Using character pointer strings can be stored in two ways:
a) Read only string in a shared segment.
When string value is directly assigned to a pointer, in most of the compilers, it’s stored in a read only block (generally in data segment) that is shared among functions.
b) Dynamically allocated in heap segment.
Strings are stored like other dynamically allocated things in C and can be shared among functions.

3/ Constructor:

Constructs a string object, initializing its value depending on the constructor version used:

Name	Definition

empty string constructor (default constructor)	Constructs an empty string, with a length of zero characters.
copy constructor	Constructs a copy of str.
substring constructor	Copies the portion of str that begins at the character position pos and spans len characters (or until the end of str, if either str is too short or if len is string::npos).
from c-string	Copies the null-terminated character sequence (C-string) pointed by s.
from buffer	Copies the first n characters from the array of characters pointed by s.
fill constructor	Fills the string with n consecutive copies of character c.
range constructor	Copies the sequence of characters in the range [first,last), in the same order.
initializer list	Copies each of the characters in il, in the same order.
move constructor	Acquires the contents of str. str is left in an unspecified but valid state.

4/ Destructor:

~string();

Destroys the string object.

This deallocates all the storage capacity allocated by the string using its allocator.

5/ Functions:

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

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

6/ References:

https://www.geeksforgeeks.org/storage-for-strings-in-c/

https://www.tutorialspoint.com/cplusplus/cpp_strings.htm

http://www.cplusplus.com/reference/string/string/string/

https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/handouts/08-C++-Strings.pdf

b/ Vector:

1/ Definition:

Vector is a template class that is a perfect replacement for the good old C-style arrays. It allows the same natural syntax that is used with plain arrays but offers a series of services that free the C++ programmer from taking care of the allocated memory and help operating consistently on the contained objects.

std::vector<T> v; is a template class that will wrap an array of Ts. In this widely used notation, 'T' stands for any data type, built-in, or user-defined class. The *vector* will store the Ts in a contiguous memory area that it will handle for you, and let you access the individual Ts simply by writing *v[0]*, *v[1]*, and so on, exactly like you would do for a C-style array.

Below is definition of std::vector from <vector> header file

```
template < class T, class Alloc = allocator<T> > class vector;
```

Parameters

- T − Type of the element contained.

  T may be substituted by any other data type including user-defined type.

- Alloc − Type of allocator object.

  By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.

Function	Purpose
alloc	Allocator to use for all memory allocations of this container
count	The size of the container
value	The value to initialize elements of the container with
first, last	The range to copy the elements from
other	Another container to be used as source to initialize the elements of the container with
init	Initializer list to initialize the elements of the container with

2/ The storage of the vector

The storage of the vector is handled automatically, being expanded and contracted as needed. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using capacity() function. Extra memory can be returned to the system via a call to shrink_to_fit(). (since C++11)

Reallocations are usually costly operations in terms of performance. The reserve() function can be used to eliminate reallocations if the number of elements is known beforehand.

The complexity (efficiency) of common operations on vectors is as follows:

- Random access - constant O(1)
- Insertion or removal of elements at the end - amortized constant O(1)
- Insertion or removal of elements - linear in the distance to the end of the vector O(n)

3/ Constructor

Function	Purpose
empty container constructor (default constructor)	Constructs an empty container, with no elements.
fill constructor	Constructs a container with n elements. Each element is a copy of val.
range constructor	Constructs a container with as many elements as the range [first,last), with each element constructed from its corresponding element in that range, in the same order.
copy constructor	Constructs a container with a copy of each of the elements in x, in the same order.

4/ Destructor

~vector();

Destructs the container. The destructors of the elements are called and the used storage is deallocated. Note, that if the elements are pointers, the pointed-to objects are not destroyed.

5/ Modifiers

`clear`	clears the contents (public member function)

insert	inserts elements (public member function)
emplace (C++11)	constructs element in-place (public member function)
erase	erases elements (public member function)
push_back	adds an element to the end (public member function)
emplace_back (C++11)	constructs an element in-place at the end (public member function)
pop_back	removes the last element (public member function)
resize	changes the number of elements stored (public member function)
swap	swaps the contents (public member function)

6/ Element access

at	access specified element with bounds checking (public member function)
operator[]	access specified element (public member function)
front	access the first element (public member function)
back	access the last element (public member function)
data (C++11)	direct access to the underlying array (public member function)

7/ References

https://www.tutorialspoint.com/cpp_standard_library/vector.htm http://www.cplusplus.com/reference/vector/vector/

https://en.cppreference.com/w/cpp/container/vector

https://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c4027/C-Tutorial-A-Beginners-Guide-to-stdvector-Part-1.htm