* taking input string from the user

→ We use getline() function to read a line of text.

→ it take cin as the first parameter, of the string variable as second.

String in details

→ String class stores the characters as a sequence of bytes with the functionality of allowing access to the single-byte characters.

String vs character Array.

i) A character array is simply an array of characters that can be terminated by a null character.

A string is a class that defines objects that be represented as a stream of characters.

ii) size of the character array has to be allocated statically, more memory cannot be allocated at run time if required. Unused allocated memory is wasted in the case of the character array.

In case of string, memory is allocated to the string dynamically, more memory can be allocated at run time on demand. Also no memory is preallocated, no memory is wasted.

iii. Implementation of character array is faster than std::string.

String are slower when compared to implementation than character array.

iv. Character arrays do not offer many inbuilt functions to manipulate strings.

→ String class defines a number of functionalities that allow manifold operation on string.

Operation on strings

1. getline() → This function is used to store a stream of character as entered by the user in the object memory.

2. Push_back() → input character at the end of the string (single ' ' comma)

ex. str.push_back('s')

3. Pop_back() → used to delete the last character from the string.

ex: str.pop_back()

* Capacity function :-

1. Capacity() → return capacity alloy to the string.

ex str.capacity().

iii. Implementation of character array is faster than std::string.
→ string are slower when compared to implementation than character array.

iv. Character arrays do not offer many inbuilt functions to manipulate strings.
→ String class defines a number of functionalities that allow manifold operation on strings.

* Operation on strings

1. getline() → This function is used to store a stream of characters as entered by the user in the object memory.

2. Push_back() → input character at the end of the string (style ' 'comm
ex. str.push_back(c);

3. Pop_back() → used to delete the last character from the string.
ex. str.pop_back();

* Capacity functions :-

1. Capacity() → returns capacity allocated to the string.
ex. str.capacity().

5. resize() → this function change the size of the string, it can be increased or decreased

ex: str.resize(5);

6. length() → this find the length of the string

ex: str.length();

7. shrink_to_fit() → This function decreases the capacity of the string and make it equal to the minimum capacity of the string.

ex: str.shrink_to_fit();

* Iterator functions: on string

8. begin() → This returns an iterator to the begin of the string.

ex: for(it = str.begin(); it != str.end(); it++)
cout << *it;
cout << endl;

Take reference from gfg.

9. end() → returns an iterator to the end of string.

10. rbegin() → returns a reverse iterator pointing at the end of the string.

ex: for(it1 = str.rbegin(); it1 != str.rend(); it1++)
cout << *it1;
cout << endl;

11. r.end()

* Manipulating functions :-

12° 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.

ex:
```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str1 = "geeksforgeeks is for geeks";
    string str2 = "geeksforgeeks rocks";
    char ch[80];
    cout << ch;

    str1.copy (ch,13,0);

    // output :-
    geeksforgeeks
```

13° swap() → This function use to swapping the two string

ex: str1.swap(str2);

Array of string in C++.

→ In C & C++, a string is a 1-dimensional array of characters & an array of strings. in C is a 2-dimensional array of characters.

→ There is 5 different way to create.

1° Using Pointers:

We create an array of string literals by creating an array of pointers

ex:
```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    const char *colour[4] = {"Blue", "Red", "Orange",
                             "Yellow"};

    for (int i=0; i< 4; i++)
        cout << colour[i] << "\n";
    return 0;
}
```
output:-
Blue
Red
Orange
Yellow,

2. Using 2D array :

→ This method is useful when the length of a strings is known or a particular memory footprint is desired. Space for strings will be allocated in a single block.

ex: #include <iostream>
int main()
{
char colour[4][10] = { "Blue", "Red", "Orge",
"Yellow" };
for (int i = 0; i < 4; i++)
cout << colour[i]
return 0;
}

→ Both the number of strings of the size of strings are fixed. The 4, again, maybe left out; of 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.

3. Using the string class :→ The STL string class may be used to create an array of mutable strings.

→ In this method, the size of the string is not fixed, & the string can be changed.

eg:
```
#include<iostream>
#include<string>
int main()
{
    std::string colour[4] = {"Blue", "Red",
                             "orange", "yellow"};
    for(int i=0; i<4; i++)
        std::cout << colour[i] << "\n";
```

Output
Blue
Red
orange
Yellow.

• → The array is of fixed size, but needn't be. Again, the 4 here may be omitted & the compiler will determine the appropriate size of the array.

→ The string are also mutable, allowing them to changed.

Lab 4

Using the vector class → The STL container vector can be used to dynamically allocate an array that can vary in size.

ex:
```
#include<iostream>
#include<vector>
#include<string>

int main()
{
  std::vector<std::string> colour {"Blue", "Red", "Orange"};

  colour.push_back("Yellow");
  for(int i=0; i<colour.size(); i++)
      std::cout<<colour[i]<<"\n";
```

//output:
Blue
Red
Orange
Yellow

* Vector are dynamic array of allow you to add of 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 :
→ The STL container the array can be used to allocate a fixed size array.

Note It may be similar to vector but size is always fixed.

ex:
```
#include<iostream>
#include<array>
#include<string>

int main()
{
    std::array<std::string, 4> colours
        {"Blue", "Red", "Orange",
         "Yellow"};
    for(int i=0; i<4; i++)
        std::cout<<colours[i]<<"\n";
    return 0;
}
//output
Blue
Red
Orange
Yellow
```

#Q Conclusion → out of all methods, Vector seems to be the best way for creating an array of string in C++

5. Using the array class :
→ The STL container, the array can be used to allocate a fixed sized array.

Note: It may be similar to vector but size is always fixed.

ex:
```
#include<iostream>
#include<array>
#include<string>

int main()
{
  std::array<std::string, 4> colours
        {"Blue", "Red", "Orange",
         "Yellow"};
for(int i=0; i<4; i++)
    std::cout<<colours[i]<<"\n";
return 0;
}
// output
Blue
Red
Orange
Yellow
```

#6 Conclusion → Out of all methods, Vector seems to be the best way for creating an array of string in C++

STL strings

→ To use strings in a program, you need to include a header called string for ex:

```
#include<string>.
```

Declare string:

```
string str = "rishab";
```

keyword

- string str(10); : it declares a string of size 10.
- string s(5, 'N'); : it declare a string of size 5 with all character 'N'.
- ~~It declares~~
- string abc(str): It declares a copy of string str.

* Taking Input

→ We use cin to input the string

```
cin >> str;
```

Using getline() function: To input the string with space we use getline() funct" instead of cin.

eg:

```
string s;
getline(cin, s);
```

Throughing output

- We use cout to throw output to the terminal.

```
cout<<str;
```

* Different functions of string

i. append(): Inserts addition character at the end of the string. time complexity O(N)

```
S1.append(S2);
```

ii. assign(): New string by replace the previous. Using = operator

iii. []: Return character at particular point

iv. clear(): Erases all the contents of the string of assign an empty string ("") of length zero. T.C O(1)

v. Compare(): S1.comare(S2)

vi. C_str(): Converts the string into C-style string (null terminating string) of returns the pointer to the C-style string. T.C O(1).

vii. empty(): Return a boolean value, true if the string is empty of false if the string is not empty

viii. erase(): Delete all substring

ex: stoi(): Returns the string converted into datatype.

```
string s = "786";
int x = stoi(s);
cout << to_string(x) + "2" << endl;
```

* Most important *

- Sorting a string: To sort a string, we need to include a header file is known as algorithm.

```
#include<algorithm>
```

ex:

```
string s = "xcmnzdsfka";
sort(s.begin(), s.end());
cout << s << endl;
```