Custom Sort string

↳ use custom comparator

order => "cba"

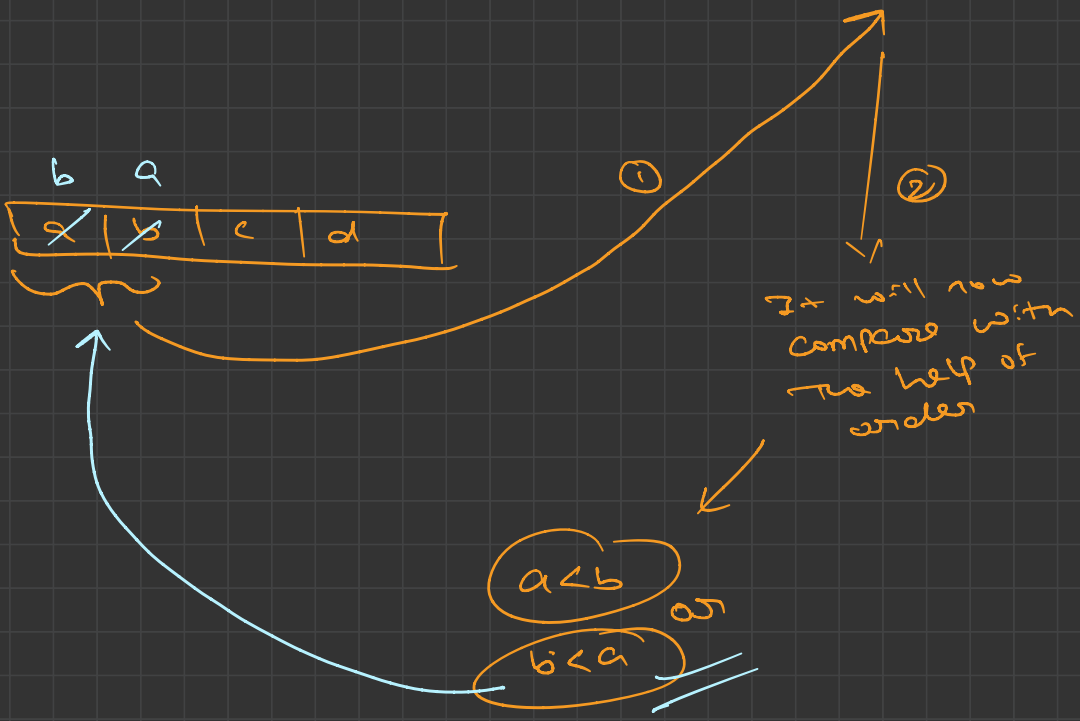
S => "abcd" => op => "cbad"

```
class solution {
public:

static string str; // global declaration
        ↳ w/o global we can't use in compare fnc()

static bool compare( char ch1, char ch2)
{
    return (str.find(ch1) < str.find(ch2));
} // this will return true if pos of ch1 in str string is less than the pos of ch2 in str string

string customSortString( string order, string s)
{
    str = order; // define global str

    sort( s.begin(), s.end(), compare);
                              ↳ custom comparator

    return s;
}

};

String Solution :: str;
```

order = "cba"

s = "abcd"

sort(s.begin(), s.end(), compare)



In the code you provided, the use of `static` has specific purposes related to the class and its behavior:

1. *Static Member Variable `str`*:
   - The `static string str;` line declares a static member variable `str` within the `Solution` class. When a member variable is declared as `static`, it means there will be only one instance of this variable shared among all instances of the class.
   - In this case, `str` is intended to store the custom order used for sorting. Making it static ensures that all instances of the `Solution` class share the same custom order. This is useful when you want to set the custom order once for the entire class and have all instances of the class use the same order.

2. *Static Member Function `compare`*:
   - The `static bool compare(char ch1, char ch2)` function is declared as static. Static member functions do not have access to non-static member variables because they are associated with the class itself, not with specific instances of the class.
   - Since `compare` uses `str`, which is also static, it can access `str` without having an instance of the class.

In summary, `static` is used here to ensure that the `str` variable is shared among all instances of the class and that the `compare` function can access `str` without needing an instance of the class. This design choice may be suitable for cases where you want a single custom order to be used by all instances of the class.

The line `string Solution::str;` is used to define and initialize the static member variable `str` outside of the class `Solution`. This is necessary because static member variables need to be defined and initialized separately from the class definition.

Here's why this line is used:

1. *Declaration vs. Definition*: In C++, when you declare a static member variable in a class, you are essentially saying, "This variable exists, but its storage is not allocated here." You need to provide a separate definition for the static member variable to allocate storage for it.

2. *Static Member Variables*: Static member variables are shared among all instances of a class. They exist independently of any specific object of the class. This means that the variable needs to exist and be allocated in memory even before you create any objects of the class.

So, `string Solution::str;` is providing the definition and allocation of memory for the `str` static member variable. It's essentially saying, "Here is the memory space for the `str` variable," and this definition is typically placed in the source file (.cpp) associated with the class implementation.

Without this line, you would encounter linker errors because the memory for the static member variable `str` would not be allocated, and the program wouldn't know where to find it. This line ensures that the variable exists in memory and can be accessed by all instances of the class.