## Experiment No: 12 Date:

## Aim: to study class string and stream processing in c++

## Theory:

1. String Declaration and Initialization

In C++, the `std::string` class provides a versatile way to handle strings. Strings can be declared and initialized using various constructors, including assignment from literals or other strings.

2. String Operations

`std::string` supports numerous operations like concatenation (`+`), substring extraction (`substr`), and length retrieval (`length()`). These operations make string manipulation in C++ convenient and efficient.

3. String Input and Output

C++ allows the use of `>>` and `<<` operators for input and output of strings. This simplifies interaction with the console or other I/O streams, enhancing the user experience in programs that involve string handling.

4. String Comparison

String comparison in C++ is facilitated by relational operators (`==`, `!=`, `<`, `>`, `<=`, `>=`) and member functions (`compare`). These mechanisms allow developers to compare strings based on lexicographical order or specific criteria.

5. String Modification

The `std::string` class provides methods for modifying strings, such as `append`, `insert`, and `erase`. These functions enable developers to alter the content of a string dynamically, supporting various string manipulation scenarios.

6. String Searching

Searching within strings is simplified with functions like `find` and `rfind`. These methods locate the position of a substring within the string, allowing for efficient extraction or replacement of specific segments.

7. String Iteration

C++ supports the use of iterators (`begin()`, `end()`) to traverse the characters of a string. This provides a flexible way to process each character in the string, facilitating custom string manipulation algorithms.

8. String Capacity

The `capacity()` and `reserve()` methods in `std::string` allow developers to manage the memory allocated for a string. This can be useful for optimizing performance in scenarios where string size is known in advance.

9. String Conversion

C++ provides methods for converting strings to numeric types (`stoi`, `stod`) and vice versa (`to_string`). These functions simplify the handling of numeric values in string form and aid in parsing user inputs.

10. String Memory Management

The `std::string` class dynamically manages memory for the stored string, eliminating the need for manual memory allocation and deallocation. This automatic memory management enhances code safety and reduces the risk of memory-related errors.

11. String Literals

String literals in C++ are sequences of characters enclosed in double quotes. They can be directly assigned to `std::string` objects, providing a convenient way to initialize and work with strings in code.

[A] Write a C++ program to create string & perform the following:
i. String assignment & concatenation
ii. Compare strings
iii. Find substrings & characters in a string
iv. Swapping strings

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

int main() {
   using namespace std;

   string str1 = "Hello, ";
   string str2 = "World!";
   string result;

   result = str1 + str2;
   cout << "Concatenated String: " << result << endl;

   if (str1 == str2) {
      cout << "Strings are equal." << endl;
   } else {
      cout << "Strings are not equal." << endl;
   }

   string mainString = "This is a sample string";
   string substring = "sample";
   char searchChar = 'a';

   size_t found = mainString.find(substring);
   if (found != string::npos) {
      cout << "Substring found at position: " << found << endl;
   } else {
      cout << "Substring not found." << endl;
   }

   found = mainString.find(searchChar);
   if (found != string::npos) {
      cout << "Character found at position: " << found << endl;
   } else {
      cout << "Character not found." << endl;
   }

   string temp = str1;
   str1 = str2;
   str2 = temp;

   cout << "After swapping:" << endl;
   cout << "str1: " << str1 << endl;
   cout << "str2: " << str2 << endl;

   return 0;
}
```

**Output:**

```
Concatenated String: Hello, World!
Strings are not equal.
Substring found at position: 10
Character found at position: 8
After swapping:
str1: World!
str2: Hello,
```

**Conclusion:** All the concepts and techniques used in strings were understood and implemented in the codes above.

**Nitesh Naik**

**(Subject Faculty)**