std::string

. Topics:
    . std::string represents a collection of characters
    . Basic Declaration and Initialization: Shows how to declare and initialize std::string.
    . Using Constructors: Demonstrates different ways to create std::string using constructors.
    . Accessing Characters: Accessing individual characters using [] and at().
    . Modifying Strings: Modifying characters in the string.
    . Appending and Concatenation: Appending and concatenating strings.
    . Substrings: Extracting substrings.
    . Finding Substrings: Finding substrings within a string.
    . Comparison: Comparing strings using compare().
    . Inserting and Erasing: Inserting and erasing parts of a string.
    . Length and Capacity: Getting the length and capacity of a string.
    . Iterating over a string: Iterating over a string using a range-based for loop.
    . Clearing a string: Clearing the contents of as string.

std::string

```cpp
// Declaring a std::string and initializing it with a literal
std::string str1 = "Hello, World!";
fmt::println("str1: {}", str1);


// Using Constructors
// Creating a std::string using various constructors
std::string str2(str1); // Copy constructor
std::string str3(str1, 7, 5); // Substring constructor: start at index 7, length 5
std::string str4(10, 'A'); // Fill constructor: 10 characters of 'A'
fmt::println("str2: {}", str2);
fmt::println("str3: {}", str3);
fmt::println("str4: {}", str4);

// Accessing individual characters using the [] operator and at() method
fmt::println("First character of str1: {}", str1[0]);
fmt::println("Second character of str1 using at(): {}", str1.at(1));

// Modifying characters in the string
str1[0] = 'h';
str1.at(7) = 'w';
fmt::println("Modified str1: {}", str1);
```

std::string

```cpp
// Appending to a string using operator+= and append()
str1 += " How are you?";
str2.append(" Goodbye!");
fmt::println("Appended str1: {}", str1);
fmt::println("Appended str2: {}", str2);

// Concatenating strings using operator+
std::string str5 = str3 + " Everyone!";
fmt::println("Concatenated str5: {}", str5);

// Extracting a substring using substr()
std::string subStr = str1.substr(7, 5); // Start at index 7, length 5
fmt::println("Substring of str1: {}", subStr);

// Finding a substring within a string using find()
size_t pos = str1.find("World");
if (pos != std::string::npos) {
    fmt::println("'World' found at position: {}", pos);
} else {
    fmt::println("'World' not found");
}
```

std::string

```cpp
// Comparing strings using compare()
if (str1.compare(str2) == 0) {
    fmt::println("str1 is equal to str2");
} else {
    fmt::println("str1 is not equal to str2");
}

// Inserting a substring into a string
str1.insert(5, ", Beautiful");
fmt::println("After insertion: {}", str1);

// Erasing a part of the string
str1.erase(5, 11); // Erase 11 characters starting from index 5
fmt::println("After erasing: {}", str1);
```

std::string

```cpp
// Getting the length and capacity of a string
fmt::println("Length of str1: {}", str1.length());
fmt::println("Capacity of str1: {}", str1.capacity());

// Using range-based for loop to iterate over a string
fmt::print("Characters in str1: ");
for (const auto& ch : str1) {
    fmt::print("{} ", ch);
}
fmt::print("\n");

// Clearing the contents of a string
str1.clear();
fmt::println("Cleared str1, new length: {}", str1.length());
```

std::vector vs std::array vs std::string

Feature	`std::vector<T>`	`std::array<T, N>`	`std::string`
Memory Layout	Dynamic (heap)	Fixed size (stack or embedded)	Dynamic (heap)
Resizable	Yes (automatic resizing)	No (fixed size)	Yes (automatic resizing)
Element Type	Generic (`T` can be any type)	Generic (`T` can be any type)	Only `char` or `wchar_t` (text data)
Insertion Complexity	O(1) at end, O(n) elsewhere	N/A (fixed size)	O(1) at end, O(n) elsewhere
Access Time	O(1) random access	O(1) random access	O(1) random access
Memory Efficiency	Efficient, but may over-allocate	Most efficient (fixed size)	Efficient but may over-allocate
Best for	Dynamic arrays of any type	Fixed-size arrays	Handling and manipulating text
Supports Move Semantics	Yes	Yes	Yes
Thread Safety	Not thread-safe	Not thread-safe	Not thread-safe
Contiguous Memory	Yes	Yes	Yes
Iterator Invalidations	Invalidation on reallocation or insertion	N/A	Invalidation on reallocation or insertion
Default Initialization	Elements are default-initialized	Must be initialized with values	N/A (initialized to empty string by default)
Special Functions	`resize`, `push_back`, `pop_back`	N/A (fixed size, no resizing)	`append`, `substr`, `find`, `replace`
Best for String Use	Not ideal (stores generic types)	Not applicable	Specifically designed for strings