MyString Class Project Documentation

1. Introduction

The **MyString Project** is a C++ implementation of a custom string class that mimics and extends the functionality of std::string.
It demonstrates core Object-Oriented Programming (OOP) concepts, including encapsulation, abstraction, modularity, operator and function overloading, file handling, and dynamic memory management.

Key features include:

- **Dynamic memory management** → Manual allocation and deallocation for character arrays using new/delete.
- **Operator overloading** → Custom behavior for =, +, ++, --, ==, etc.
- **Function overloading** → Multiple constructors for different input types.
- **File handling with custom objects** → Writing, reading, and verifying string objects.
- **Encapsulation** → Protecting class data members (string1, ID, etc.) while providing controlled access through public methods.
- **Abstraction** → Hiding internal implementation details and exposing only the necessary interfaces for string operations.
- **Modularity** → Clean separation of interface (MyString.h), implementation (MyString.cpp), and execution (main.cpp).
- **Constructors, Destructors, and Copy Semantics** → Proper initialization, cleanup, and deep copying of objects.
- **Static members and object tracking** → Using static variables to maintain object counts and assign unique IDs.

By combining these concepts, the project not only replicates string functionality but also demonstrates core OOP pillars (Encapsulation, Abstraction, Modularity, Function Overloading, and File Handling) in a practical and applied manner.

---

2. Project Structure

```
MyString-OOP-Project/
|
|— docs/              # Documentation
|   ├— MyString_Project_Documentation.docx
|   └— MyString_Project_Documentation.pdf
|
|— src/               # Source code
|   ├— main.cpp
|   ├— MyString.cpp
|   └— MyString.h
|
|— output/            # Screenshots of program output
|   ├— Output1_BasicConstructors.png
|   ├— Output2_StringOperations.png
|   ├— Output3_OperatorOverloading.png
|   ├— Output4_FileHandling.png
|   └— Output5_FinalExecution.png
|
|— media/             # Project demo videos
|   ├— Code_Walkthrough.mp4
|   └— Execution_Demo.mp4
|
|— README.md          # Main project overview
```

3. Features of MyString Class

3.1 Constructors and Destructor

- **Default Constructor** → Initializes string1 as nullptr.
- **Parameterized Constructor** → Deep copies a C-string into the object.
- **Copy Constructor** → Creates a deep copy of another MyString object.
- **Destructor** → Frees allocated memory and decrements the static object counter.

3.2 Core String Functions

- append(const char*) → Appends a C-string at the end.
- copy(const char*) → Replaces the current string with a new one.
- compare(const char*) → Compares lengths of strings.
- isEmpty() → Checks if the string is empty.
- count() → Returns string length.
- stringSlicing(int, int) → Returns substring within the given range.
- findOccurrence(const char*) → Finds the first occurrence of a substring.
- erase(const char*) → Removes the first occurrence of a substring.
- insertString(int, const char*) → Inserts a substring at a given index.

3.3 Character Operations

- pushBack(char) → Adds a character at the end.
- popBack() → Removes the last character.
- front() / back() → Returns the first/last character of the string.
- toUpperCase() / toLowerCase() → Case conversion.

3.4 Operator Overloading

- **Assignment Operator (=)** → Deep copy assignment.
- **Increment/Decrement (++ / --)** → Increments/decrements ASCII values of characters.
- **Comparison Operators (<, <=, >, >=, ==, !=)** → Compare string lengths or equality.
- **Concatenation (+)** → Joins two strings.

- **Stream Operators (<<, >>)** → For input/output with console and files.

3.5 File Handling

- writeMyStringsToFile() → Writes multiple objects to a file.
- readMyStringFromFile() → Reads objects from a file.
- verifyMyStringsFiles() → Verifies file contents with original data.

3.6 Static Members

- staticVariable → Tracks total number of objects created.
- objectCount() → Returns current object count.

---

4. How to Run

To test the MyString project on your machine, follow these steps:

1. Clone the repository:

```
git clone https://github.com/Muneeb-techpro/MyString-OOP-Project.git
cd MyString-OOP-Project
```

2. Compile the program:

- Ensure you have g++ installed (Linux/macOS) or MinGW/MSYS2 (Windows).

```
g++ src/main.cpp src/MyString.cpp -o mystring_app
```

3. Run the executable:

- On Linux/macOS:

```
./mystring_app
```

- On Windows:

```
mystring_app.exe
```

The program will display detailed logs demonstrating all MyString class functionalities.

---

5. Demonstrated Concepts in main.cpp

The **main program** demonstrates all features step-by-step:

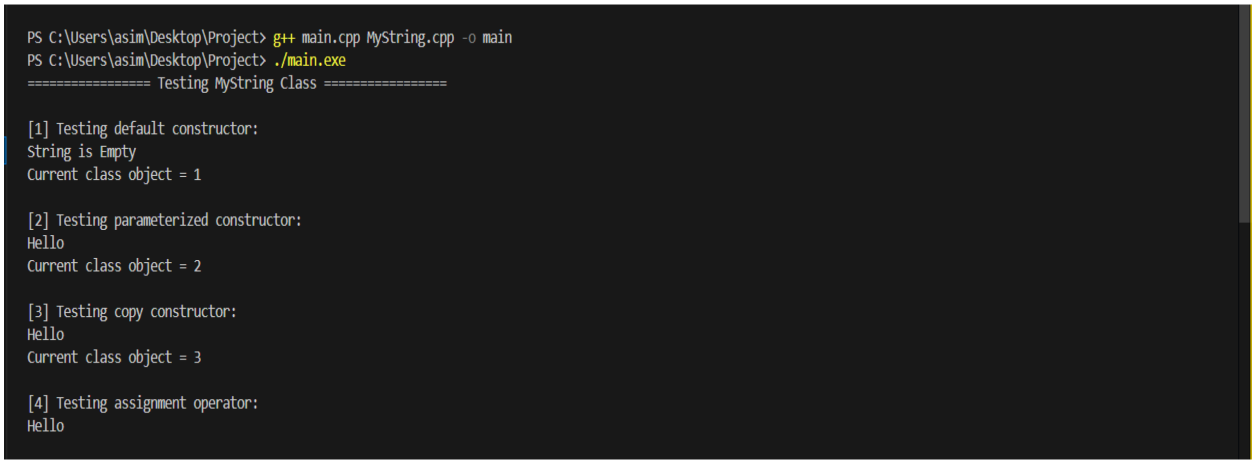
1. Object creation using different constructors.
2. Assignment operator testing.
3. String operations (append, copy, compare, slicing, insertion, erasing).
4. Character operations (pushBack, popBack, front, back).
5. Operator overloading (++, --, +, comparisons).
6. File handling (writing, reading, verifying).
7. Testing helper functions for operators.

The program outputs detailed logs to clearly track operations.

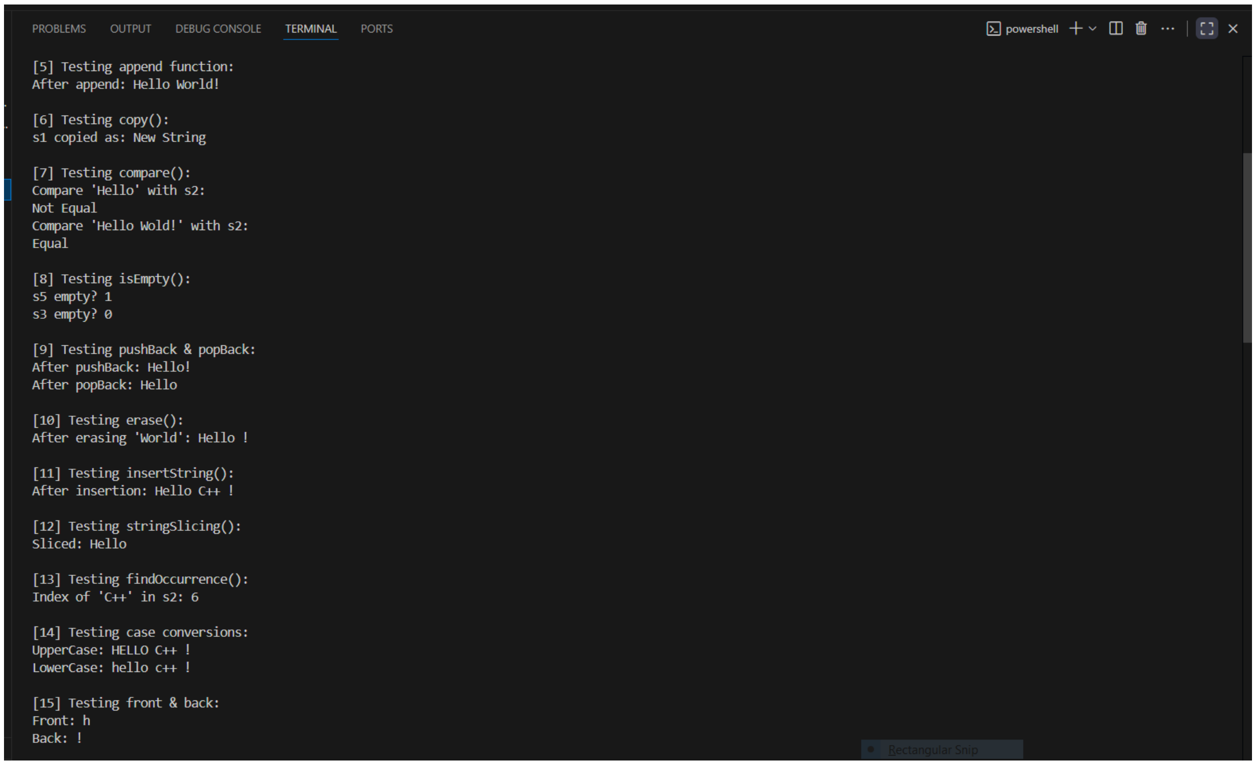
---

6. Sample Output (Highlights)

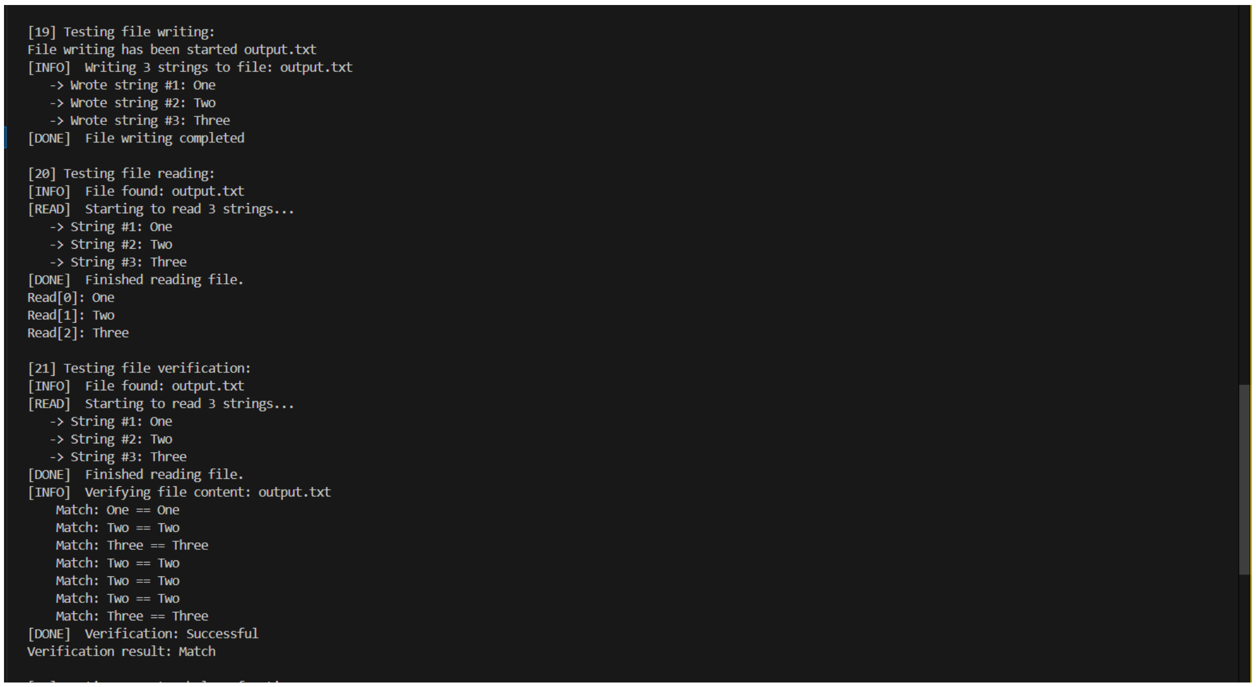
```
================= Testing MyString Class =================


[1] Testing default constructor:

String is Empty

Current class object = 1


[2] Testing parameterized constructor:

Hello

Current class object = 2


[5] Testing append function:

After append: Hello World!


[12] Testing stringSlicing():

Sliced: Hello


[19] Testing file writing:

  -> Wrote string #1: One

  -> Wrote string #2: Two

  -> Wrote string #3: Three
```



```
PS C:\Users\asim\Desktop\Project> g++ main.cpp MyString.cpp -o main
PS C:\Users\asim\Desktop\Project> ./main.exe
================ Testing MyString Class ================

[1] Testing default constructor:
String is Empty
Current class object = 1

[2] Testing parameterized constructor:
Hello
Current class object = 2

[3] Testing copy constructor:
Hello
Current class object = 3

[4] Testing assignment operator:
Hello
```



```
[5] Testing append function:
After append: Hello World!

[6] Testing copy():
s1 copied as: New String

[7] Testing compare():
Compare 'Hello' with s2:
Not Equal
Compare 'Hello Wold!' with s2:
Equal

[8] Testing isEmpty():
s5 empty? 1
s3 empty? 0

[9] Testing pushBack & popBack:
After pushBack: Hello!
After popBack: Hello

[10] Testing erase():
After erasing 'World': Hello !

[11] Testing insertString():
After insertion: Hello C++ !

[12] Testing stringSlicing():
Sliced: Hello

[13] Testing findOccurrence():
Index of 'C++' in s2: 6

[14] Testing case conversions:
UpperCase: HELLO C++ !
LowerCase: hello c++ !

[15] Testing front & back:
Front: h
Back: !
```



```
[19] Testing file writing:
File writing has been started output.txt
[INFO]  Writing 3 strings to file: output.txt
   -> Wrote string #1: One
   -> Wrote string #2: Two
   -> Wrote string #3: Three
[DONE]  File writing completed

[20] Testing file reading:
[INFO]  File found: output.txt
[READ]  Starting to read 3 strings...
   -> String #1: One
   -> String #2: Two
   -> String #3: Three
[DONE]  Finished reading file.
Read[0]: One
Read[1]: Two
Read[2]: Three

[21] Testing file verification:
[INFO]  File found: output.txt
[READ]  Starting to read 3 strings...
   -> String #1: One
   -> String #2: Two
   -> String #3: Three
[DONE]  Finished reading file.
[INFO]  Verifying file content: output.txt
    Match: One == One
    Match: Two == Two
    Match: Three == Three
    Match: Two == Two
    Match: Two == Two
    Match: Two == Two
    Match: Three == Three
[DONE]  Verification: Successful
Verification result: Match
```

7. Applications

- Custom **string manipulation library** for C++.
- Educational use for **OOP concepts** in C++.
- Understanding of **memory management** without std::string.
- File storage and retrieval of custom objects.

---

8. Future Enhancements

- Implement **move constructor & move assignment** for efficiency.
- Add **exception handling** for invalid operations.
- Expand operators to handle **character-wise lexicographical comparison**.
- Optimize string operations to avoid unnecessary copies.