**Task 1 Documentation Outline**

**Title**: Task 1 - Reference Counting Basics

***What’s the Goal?***  
In Task 1, I’ve created a my\_string class to explore how reference counting works. The idea is to share memory between multiple objects and keep track of how many references exist. When no one’s referencing the memory anymore, it gets cleaned up automatically—pretty handy for avoiding memory leaks!

***How It Works****:*

* The class has a simple reference count built into it.
* Every time a new object shares the string, the reference count goes up.
* When objects go out of scope or are destroyed, the reference count goes down.
* When the count hits zero, the memory is freed.

***How to Run It****:*

1. Open the terminal in VS Code.
2. Compile and run the program:

g++ test\_string.cpp -o test\_string && ./test\_string

***What You’ll See****:*  
The program shows how the reference count changes as objects share and release memory. It also confirms that the memory is freed when no references are left.

**Task 2 Documentation Outline**

**Title**: Task 2 - Extending Reference Counting

***What’s the Goal?***  
In Task 2, I extended the my\_string class from Task 1 to fully implement **automatic reference counting**. The class now keeps track of how many objects are sharing the same string, and the destructor ensures that the memory is freed only when the reference count hits zero.

***What’s New Here?***

* The interface for my\_string has been kept the same, but the destructor now plays a key role.
* It frees the string memory only when no references are left (i.e., reference count == 0).
* Added more tests to showcase the lifecycle of shared memory and how reference counts are updated.

***How to Run It:***

1. Open the terminal in VS Code.
2. Compile and run the program:

g++ test\_string.cpp -o test\_string && ./test\_string

***What You’ll See:***  
The output shows the reference count for each string object and how it changes when objects share or release the string. For example:

* The count increases to [2] when a second object shares the string.
* It decreases back to [1] when the second object goes out of scope.

**Task 3 Documentation Outline**

**Title**: Task 3 - Showing Memory Release

***What’s the Goal?***  
Task 3 builds on Task 2. The focus here is to clearly show what happens when the reference count hits zero and memory is released. It’s like seeing the lifecycle of the shared memory in action.

***What’s New Here?***

* I’ve added more test cases to demonstrate when memory is freed.
* For example, when all references to a string go out of scope, the program shows that the memory is cleaned up.

***How to Run It****:*

1. Open the terminal in VS Code.
2. Compile and run:

g++ test\_string.cpp -o test\_string && ./test\_string

***What You’ll See****:*  
Alongside the usual reference count changes, you’ll see extra output when memory is released. For example:

* A temporary string might be created and shared.
* When no references are left, the program confirms the memory is freed.

**Task 4 Documentation Outline**

**Title**: Task 4 - Generalizing Reference Counting with Templates

***What’s the Goal?***  
In Task 4, I generalized the reference counting logic into a reusable template class called ref\_counted. This allows us to manage reference counting not just for strings but for any type of object, making the code much more versatile.

***What’s New Here?***

* Created a reusable template, ref\_counted, that manages reference counting for any type.
* Applied the template to both my\_string (from previous tasks) and a new Point class to demonstrate its flexibility.
* The memory is freed automatically when the reference count reaches zero, regardless of the type of object.

***How to Run It:***

1. Open the terminal in VS Code.
2. Compile and run the program:

g++ test\_ref\_counting.cpp -o test\_ref\_counting && ./test\_ref\_counting

***What You’ll See:***

* ***For my\_string:***
  + Reference count increases as objects share the string.
  + Copy-on-write behaviour ensures safe modifications.
  + Memory is freed when the reference count reaches zero.
* ***For Point:***
  + Reference count increases and decreases as objects are shared and destroyed.
  + Memory for the Point object is also freed when no references remain.