Sayyed Shozib Abbas  
BSCS-4A  
35038  
112434

Assignment 2: Smart Strings

# Github Repo

# Introduction

Our task is to provide implementation of Strings using the following types of smart pointers:

1. Copied Pointers
2. Owned Pointers
3. COW with Reference Counting
4. COW with Reference Linking

Moreover, we need to profile our applications and analyze the resource utilization for each one of these strategies with relatively small, medium and very large strings.

Then with Unit Tests ensure the methods are running correctly and pointer transfer is not leaking memory or leaving any dangling pointers.

# Approach

Let me just write down every inch of detail about this assignment and how it was done.

Sorry for the late submission and consuming late days. I suspect you would be expecting an earlier submission from my side but trust me, some issues at my end due to which I couldn’t do it earlier.

Firstly, there is no such thing as copied pointers. I tried searching the internet and it says that the correct term for copied pointers is “deep copy”. Obvio I needed help, so I had to search the internet. But telling you the truth, there isn’t anything directly related to the assignment. Even if there is, I didn’t bother looking at it. Though I did find something related to the “copied pointers” part but trust me, all I noticed was the dirtiness of the code and how I couldn’t handle it *\*sighs\**.

Secondly, since I am using Visual Studio with C++, this just gets better while trying to ensure if there is any memory leak. Instead of using Unit tests, I used Visual Studio’s Performance Analyzer which provides options for checking memory leaks as well as performance.

After correcting the initially broken code (removing some delete statements), I got the program working.

Copied pointers and COW with reference counting was already implemented (partially or fully, I don’t remember now). The other parts have been written and are working correctly according to my instincts.

# How to Run

You may run the solution in visual studio directly. There are 4 applications attached to a single solution. The solution explorer should show:

1. copiedPointers
2. COWwithRFC
3. COWwithRL
4. ownedPointers

Load each project and compile a single project to see its output.

# Profiling

## Resource Utilization

Profiling each application with small (5), medium (25) and large (125) strings. The sampling number tells us which part of the application utilizes what number of resources.

The strings used are:

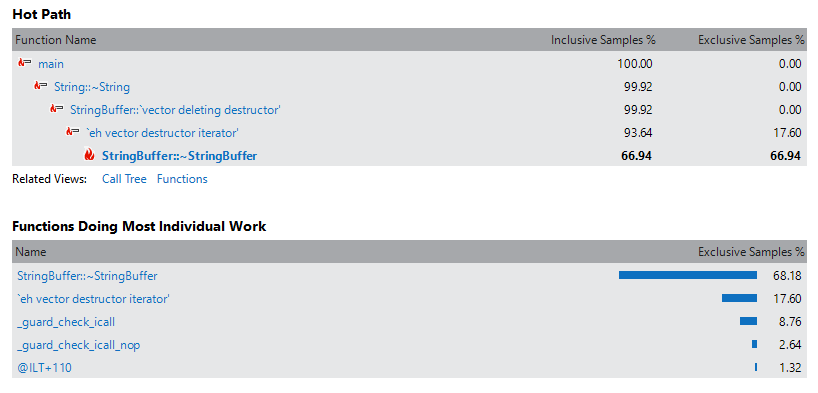
nIciq

chWvvzgI0uDn9yHDnrRrpL9O7

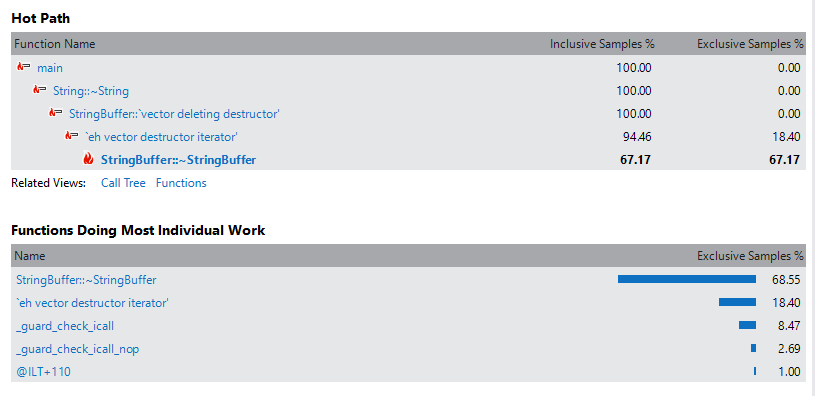
aX42bWW2Bv7Je69IKzUMGQJFgAgkGUVN7wGPsCT4m0t0YHTkPbr7XxurscpzSAuWRUGXsaIPgz3sc8txUGuB4PpoMFAt0sActpvXBreqPYnUNSi5KDDKV9iLiPkAv

### Copied Pointers

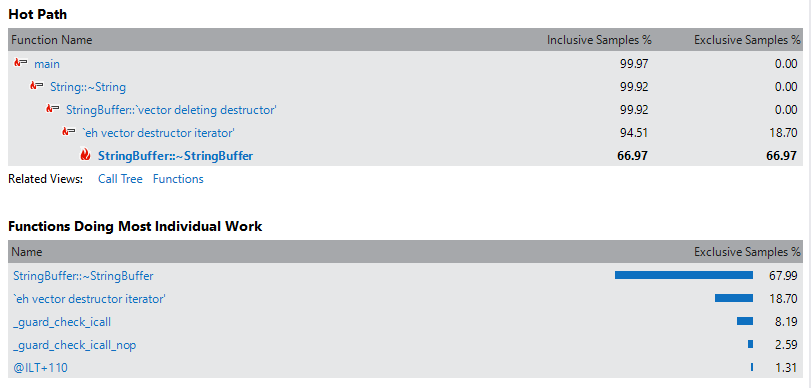
#### Small



#### Medium

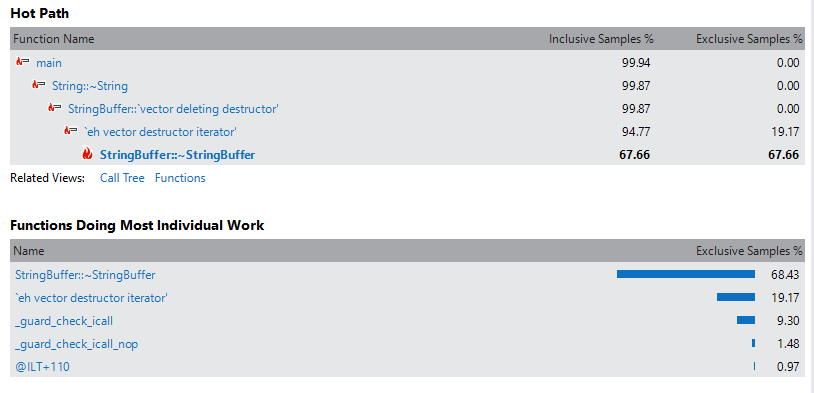


#### Large

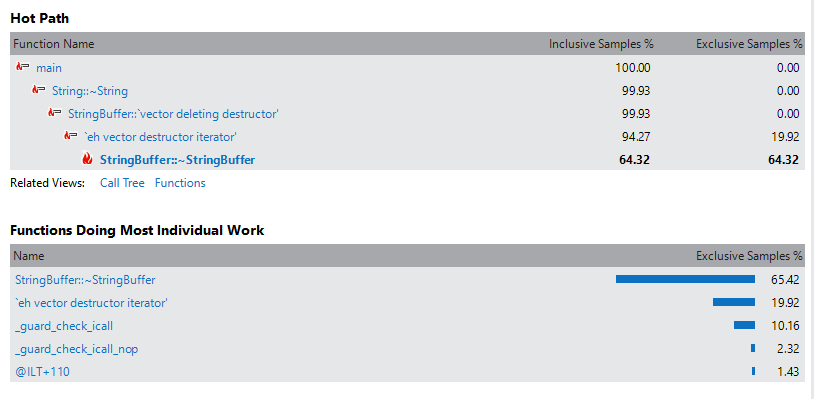


### Owned Pointers

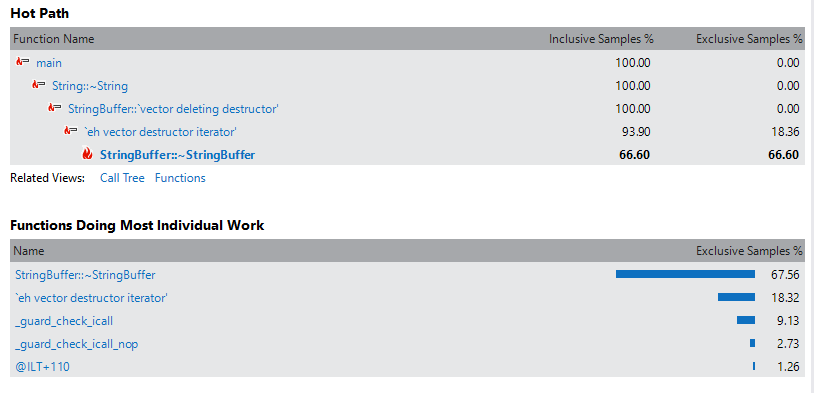
#### Small



#### Medium

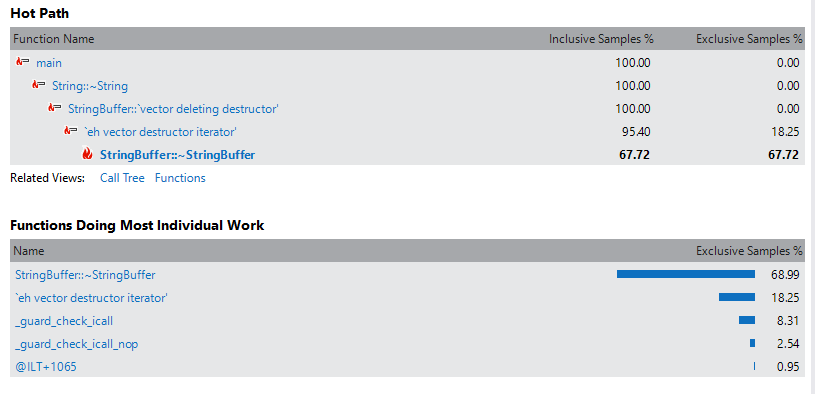


#### Large

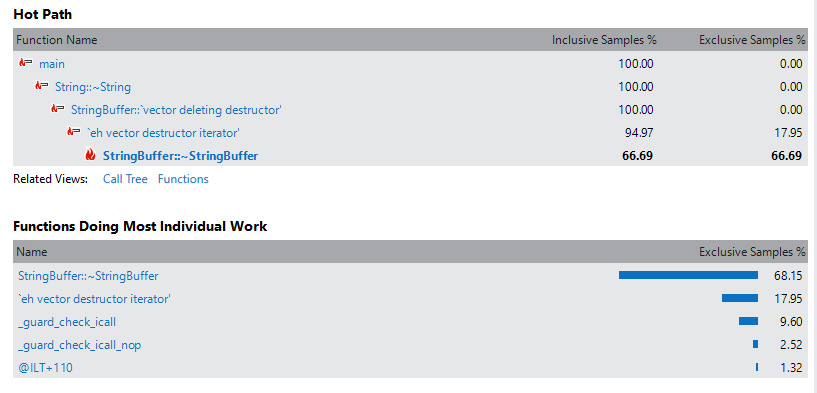


### COW with Reference Linking

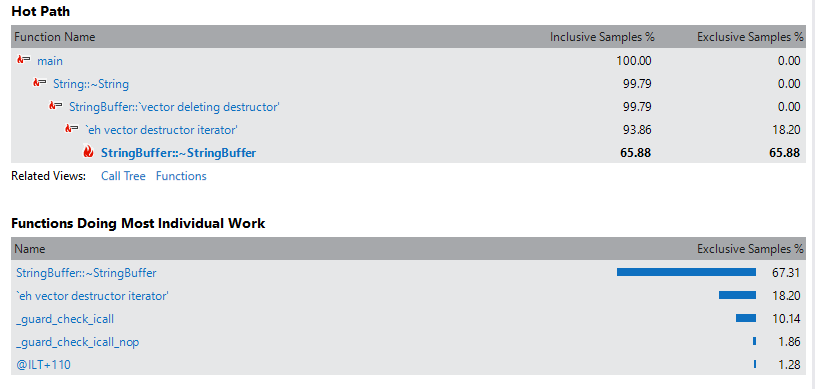
#### Small



#### Medium

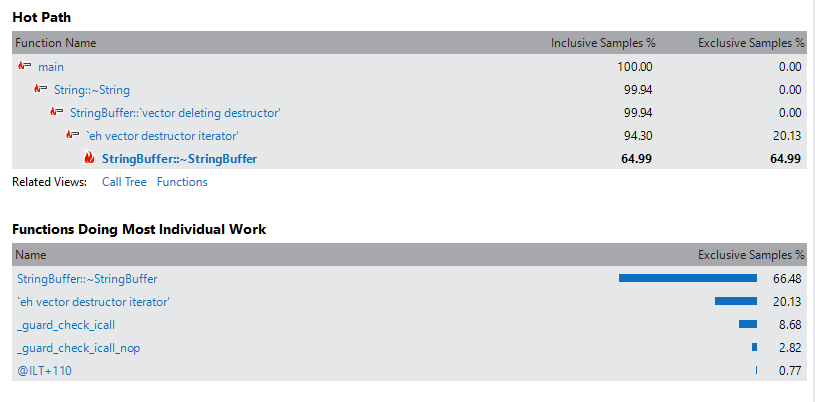


#### Large

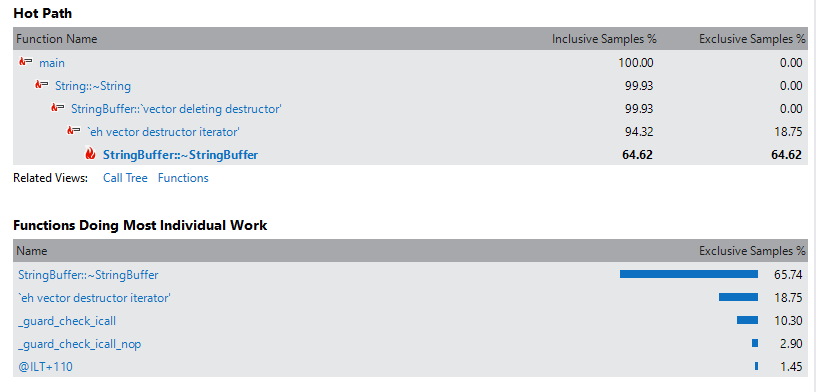


### COW with Reference Counting

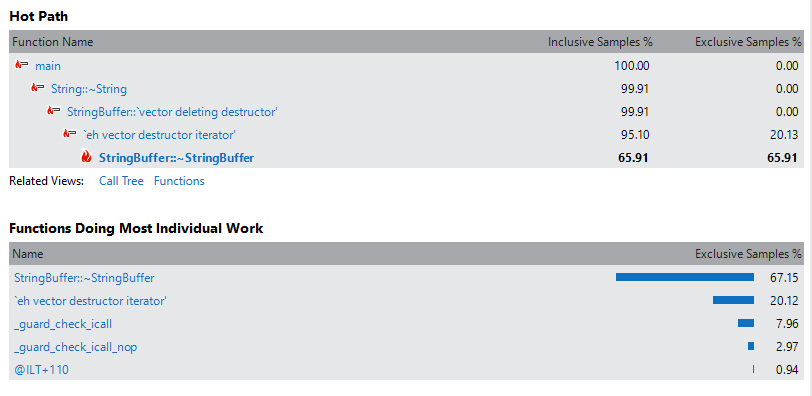
#### Small



#### Medium



#### Large



## Memory Leaks

I placed pauses on each step of the main function and took a snapshot every time the program stopped. This means there is a memory snapshot taken on every new variable created, copied, edited and destroyed.

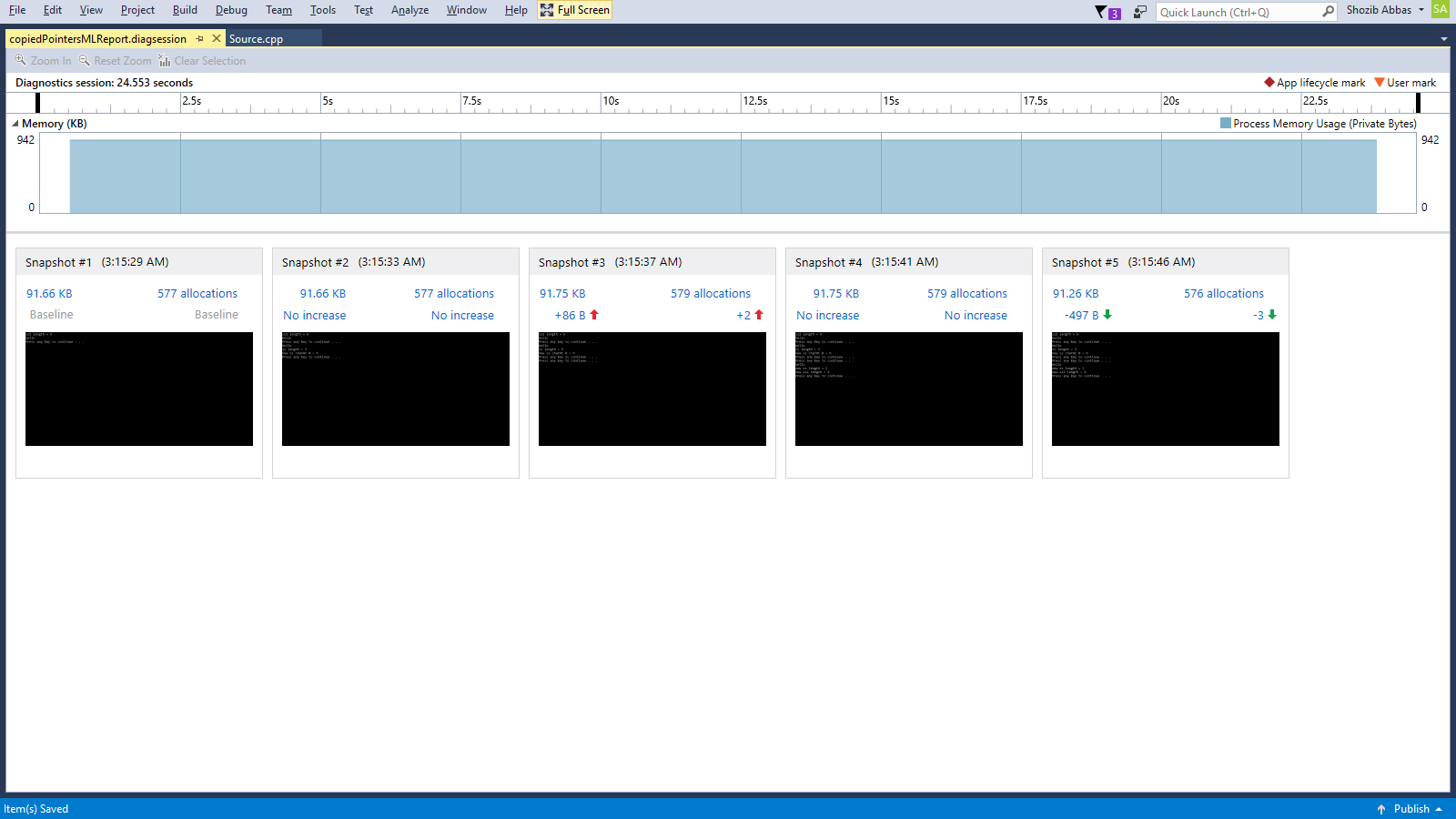
Taking a closer look, you will see snapshots showing memory increases.

Ultimately, this memory being occupied should be freed to its fullest right after the applications ends. If you see the number of allocations being less in the end than the number of allocations done between execution, that means there is no memory loss.

Moreover, the size of data that increases during execution must reduce to its fullest at the end which will prove there are no memory leaks in the application as well.

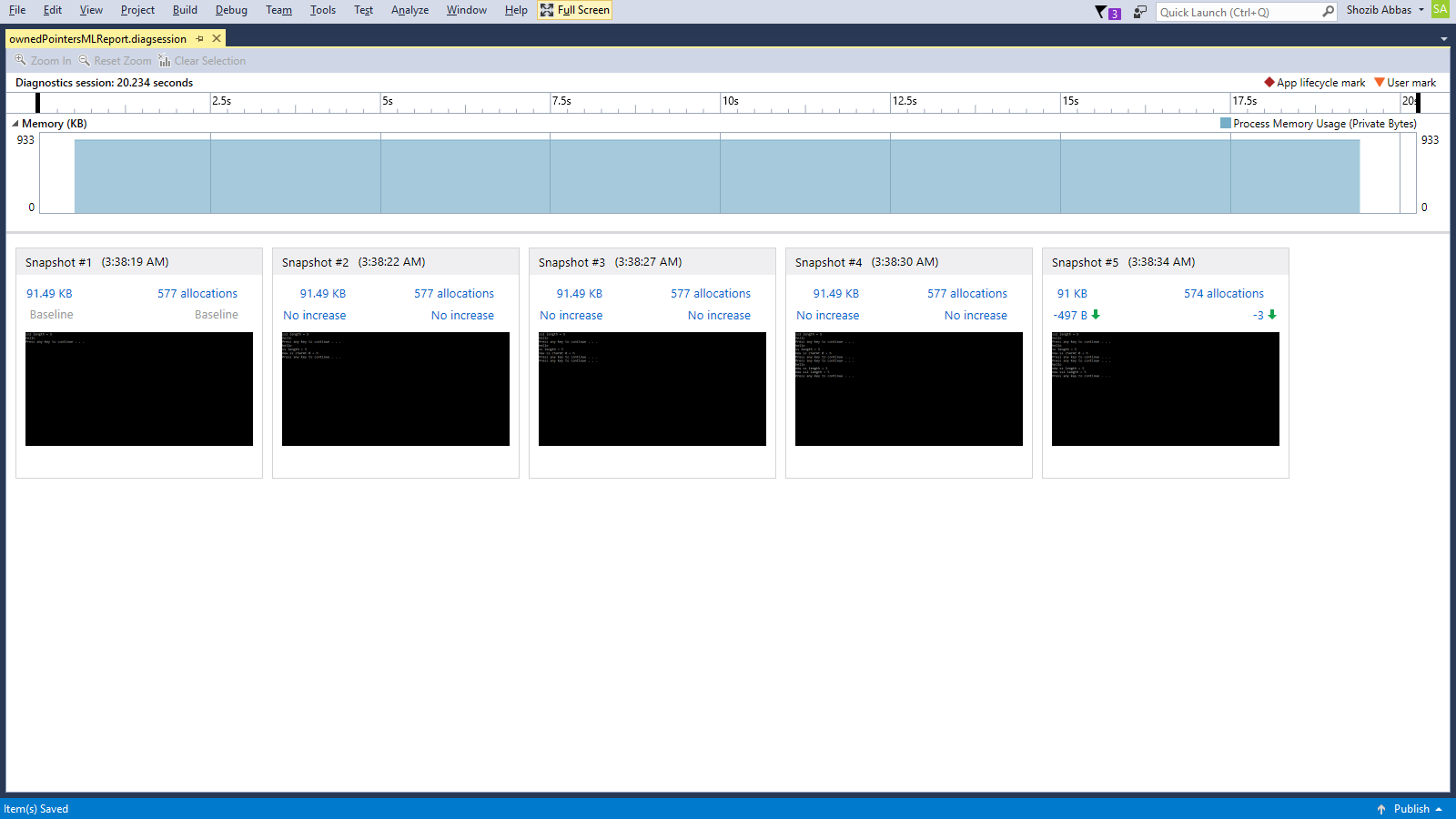
### Copied Pointers

Details of diagnostics session stored in copiedPointersMLReport.diagsession

**

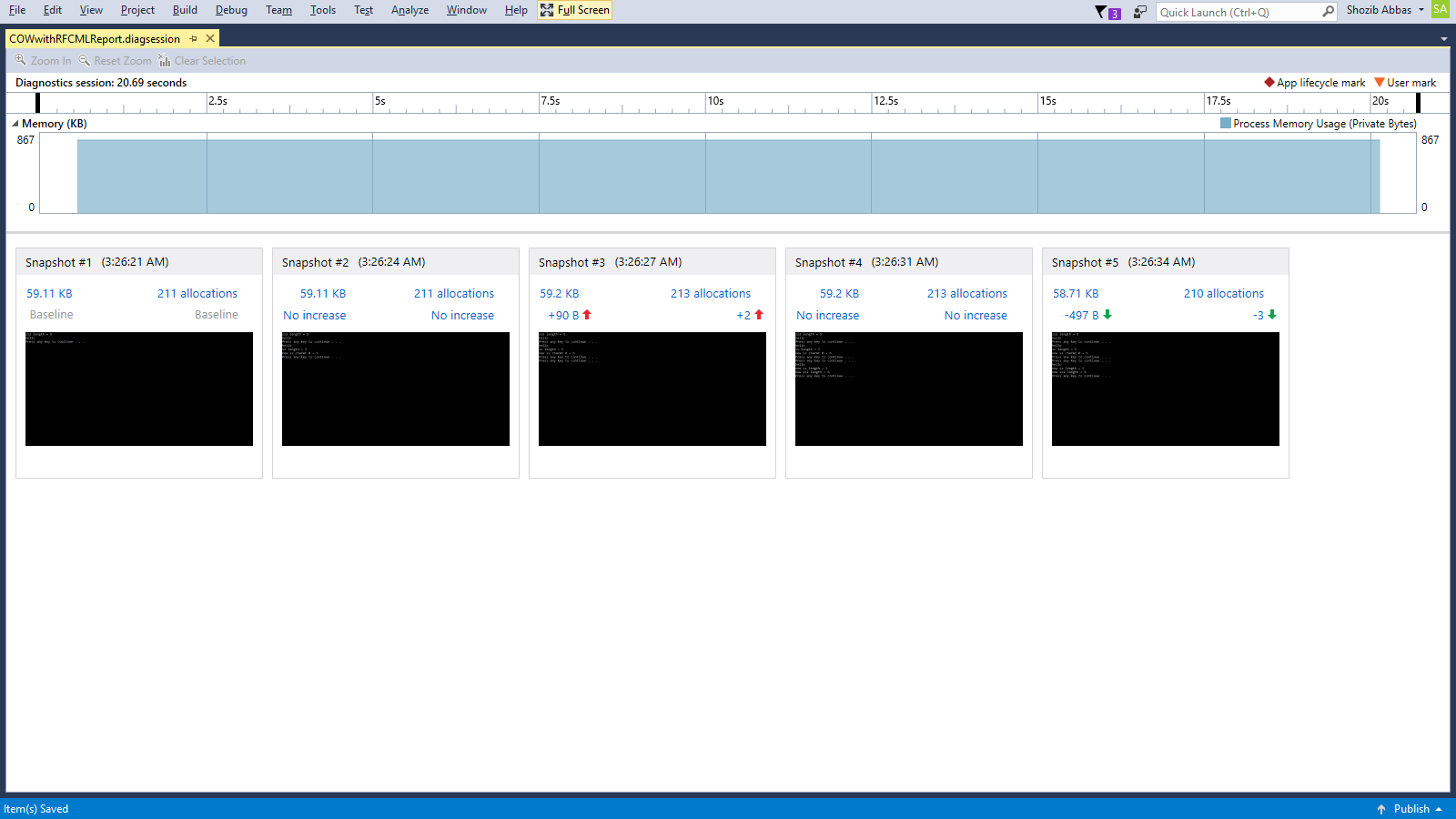
### Owned Pointers

Details of diagnostics session stored in ownedPointersMLReport.diagsession



### COW with Reference Counting

Details of diagnostics session stored in COWwithRFCMLReport.diagsession



### COW with Reference Linking

Details of diagnostics session stored in COWwithRLMLReport.diagsession

