Rule of 3 - Q&A

# Content

[Rule of 3 - Q&A 1](#_Toc531258578)

[1. Content 1](#_Toc531258579)

[2. Questions 1](#_Toc531258580)

[2.1. RuleOf3Basics 1](#_Toc531258581)

# Questions

Add your answers in the white boxes (in Dutch or English).

## RuleOf3Basics

1. Crash details

|  |
| --- |
| In which function of the Container class happens the crash? |
| In the Destructor of the container |
| Where in the Game class was this function called? (which function and where in that function, use the Call Stack window) |
| When we exit the TestFunction |
| What happens there regarding the Container objects |
| Both container objects that we create here are using the stack memory so when we exit the function, both objects are destroyed. When we try to delete the second object, we try to delete a ***dynamic array*** that already has been deleted because both pointers point to the same address. |

1. Investigate the content of the variables just before the crash happens and draw your conclusions from this test.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Compare the content of both Container objects c1 and c2   |  |  | | --- | --- | | Which data? | Where can you find it? | | m\_Size | Locals window | | m\_Capacity | Locals window | | Pointer to the dyn array: m\_pElement | Locals window | | Array elements: m\_pElement,3 | Watch window |   Do they have the same values? If not, which one are different? |
| Yes |
| What can you conclude about the dynamic array in both Container objects? |
| Both refer to the same memory location |
| Conclude: What happens by default when you create an object initializing it with another object of the same type? |
| That a shallow copy ***or member wise copy is being made. If we have pointers pointing to a dynamic array like this case, when we do this type of copy we will have two pointers pointing to the same dynamic array*** |
| Why does a crash happen when the containers c1 and c2 go out of scope? |
| ***We delete twice the same dynamic array because both pointers were pointing to the same address*** |

1. Changing an element in one of the containers.

|  |
| --- |
| What happens when you change an element in one of the 2 containers |
| Both container will change the elements |
| Why? |
| Because both containers are pointing to the same dynamic array, so both have the same content |

1. Investigate the content of the variables related to the containers c1 and c2 just before they go out of scope.

|  |
| --- |
| Which data member(s) have the same value? |
| The size and capacity of the container have the same value |
| Which data member(s) have a different value? |
| The dynamic array now points differently in each container and the have different values |
| Does changing the content of a container element, still influence the content of the elements in the other container? |
| No |

1. Crash details

|  |
| --- |
| In which function of the Container class happens the crash? |
| In the destructor of the Container |
| Where in the Game class was this function called? (which function and where in that function, use the Call Stack window) |
| When we exit the TestContainer because we are deleting the objects created in the Stack memory. When try to delete the c3 container object the crash occurs. |

1. Investigate the content of the variables just before the crash happens and draw your conclusions from this test.

|  |
| --- |
| Compare the content of both Container objects c1 and c3  Do they have the same values? If not, which one are different? |
| Yes, they have the same values |
| What can you conclude about the dynamic array in both Container objects? |
| Both containers are using the same dynamic array (pointer point to the same address) |
| Conclude: What happens by default when you assign a Container object to another one ? |
| A shallow copy is being made, so when we have pointers o dynamic objects this type of copy doesn’t work. |
| Why does a crash happen when the containers go out of scope ? |
| Because when we try to delete the c3 object, we are trying to delete the same dynamic array that was already deleted in c1 |

1. Investigation of what happens when an integer value is assigned to a Container object:   
   **c3 = 4;**

|  |
| --- |
| Using the “Step Into” Debugger button, give a list of Container functions (only mention constructor, copy constructor, assignment operator or destructor) when this statement is executed.  Write them down in order of execution.  When the constructor is called also write down the value of the capacity parameter. **Don’t mention the destructors** that are called when the 3 containers go out of scope at the closing curly brace of TestContainer. |
| 1º Constructor is called changing the capacity of the container to 4 |
| 2º The ***copy assignment operator is called with the new container created with capacity of 4*** |
| 3º Destructor |

1. Investigation of what happens when this code is executed:   
   Container c4 = c1;

|  |
| --- |
| Only mention constructor, copy constructor, assignment operator or destructor when this statement is executed.  Write them down in order of execution.  **Don’t mention the destructors** that are called when the 4 containers go out of scope at the closing curly brace of TestContainer. |
| The copy constructor is called |
|  |
|  |
|  |

1. Investigation of what happens when this code is executed:   
   c4 = c2;

|  |
| --- |
| Only mention constructor, copy constructor, assignment operator or destructor when this statement is executed.  Write them down in order of execution.  **Don’t mention the destructors** that are called when the 4 containers go out of scope at the closing curly brace of TestContainer. |
| Copy assignment operator is called |
|  |
|  |
|  |

1. Investigation of what happens when this code is executed:  
   c4 = CreateMultiplied(c1, 2);

|  |
| --- |
| Only mention constructor, copy constructor, assignment operator or destructor when this statement is executed. But also indicate why one was called.  Write them down in order of execution.  **Don’t mention the destructors** that are called when the 4 containers go out of scope at the closing curly brace of TestContainer. |
| Copy constructor |
| Constructor (capacity of 10) |
| Copy constructor |
| Construcor, capacity of 10 |
| The destructor |
| The assignment operator |
| The destructor |

1. Investigation of what happens when this code is executed:  
   AddValues(c4, 3, 20, 30);

|  |
| --- |
| Only mention constructor, copy constructor, assignment operator or destructor when this statement is executed. Write them down in order of execution.  **Don’t mention the destructors** that are called when the 4 containers go out of scope at the closing curly brace of TestContainer. |
| None |
|  |
|  |
|  |

1. Creating a **static** Texture object

|  |
| --- |
| During the creation of a static Texture object something goes wrong with as consequence that it can’t be drawn. When is the texture initialized? Why does the creation go wrong? |
| Because the SDL image inatialization hasn’t been executed yet, as the static texture is created at the start of the program, before the Core object is created |

1. Assigning a Texture object to another one

|  |
| --- |
| When assigning a Texture object to another one, you get an error. Which deleted function are you trying to call? |
| The assigment operator of the texture class |
|  |

1. When passing a Texture object by value to a function

|  |
| --- |
| What deleted function is attempted to call |
| The copy constructor |
| How can you solve this error without changing the Texture class? |
| Pass it by const reference |