Shallow and Deep Copying Solutions

A String class and Resource Management

- A class that uses the RAII idiom is responsible for managing a resource's lifetime
- Which resource does the String class manage?
  - The memory allocation used to store the characters
- What does an RAII class need to do when managing a resource's lifetime?
  - It must make sure the resource is correctly acquired before being used
  - It must make sure the resource is correctly released when it is no longer needed
  - It must make sure that any copying of the resource is handled correctly
  - It must make sure that any transfer of the resource to another object of this class is handled correctly

Shallow Copying

- What is meant by "shallow copying"?
  - In a shallow copy, all the data in the target object is overwritten by the data from the source object
- When is shallow copying dangerous?
  - Shallow copying is dangerous if the class manages a resource
  - It results in the target class and the source class sharing the same resource
  - This can result in data corruption, resource leaks and memory errors

Deep Copying

- What is meant by "deep copying"?
  - In a deep copy, the target object acquires its own version of the resource
  - It then populates it to have the same value as the resource in the source object
- Why is deep copying safe?
  - The target class and the source class have their own version of the resource
  - This avoids the problems that can arise from a shallow copy

Deep Assignment

- How does a deep assignment differ from a deep copy?
  - In a deep assignment, the target's resource is released and reacquired
  - It is then populated from the source object's resource
- When performing a deep assignment, why is it necessary to check that the source and the target are different objects?
  - If the source and the target are the same object, releasing the target's original resource will also release the source's resource
  - The source object's resource will be invalid and cannot be used to populate the target object's new resource

Rule of Three

- What is the "Rule of Three"?
  - The Rule of Three states that, if any one of destructor, copy constructor or assignment operator has to be implemented for a class to work properly, all three should be implemented
  - This does not apply to a virtual destructor in a base class which is only there to prevent derived classes being "sliced"
- Give an example where the Rule of Three would be helpful
  - Class with pointer member which allocates memory in the constructor
  - This needs a destructor which releases the allocated memory, also a copy constructor and assignment operator to perform a "deep copy"
  - Here, the Rule of Three helps us avoid memory leaks and program crashes

Rule of Three

- Implement the "rule of three" operators for the class shown below using the RAII idiom

```
class String {
  private:
    int size;
    char *buffer;
  //   ....
};
```