Implementing the Rule of Three in a Person Class

Instructor: [Insert Instructor Name]

May 20, 2024

Objective

This assignment aims to deepen your understanding of dynamic memory management in C++ by implementing the Rule of Three. You will manage resources properly through constructors, destructors, and copy constructors and assignment operators.

Instructions

Task 1: Define the Person Class

Create a `Person` class with the following properties and methods in a header file (`Person.h`):

Private Members:

- `char* name`: Pointer to a dynamically allocated array of characters (C-string).
- `int age`: Integer representing the person's age.

Public Members:

- `Default Constructor`: Initializes `name` to `nullptr` and `age` to `0`.
- `Parameterized Constructor`: Takes a `const char*` for the name and an `int` for the age.
- `Destructor`: Frees the allocated memory.
- `Copy Constructor`: Creates a deep copy of another `Person` object.
- `Copy Assignment Operator`: Assigns one `Person` object to another with deep copying.
- `Method to Print the Person's Information`: Prints the name and age.
- `Getters for Name and Age`: Returns the name and age.

Task 2: Implement the Methods

Create the header file (`Person.h`) and corresponding implementation file (`Person.cpp`) to define the methods declared in the header file.

Task 3: Test Your Class

We have provided sample main file (`main.cpp`) to test the class.

Header File: `Person.h`

```
#pragma once

#include <iostream>
#include <cstring>

class Person {
private:
    char* name;
    int age;

public:
    // Default Constructor
    Person();

    // Parameterized Constructor
    Person(const char* name, int age);

    // Destructor
    ~Person();

    // Copy Constructor
    Person(const Person& other);

    // Copy Assignment Operator
    Person& operator=(const Person& other);

    // Method to Print Person's Information
    void print() const;

    // Getters for Name and Age
    const char* getName() const;
    int getAge() const;
};
```

Main File Example

Here is an example of how you can test your `Person` class:

```
#include "Person.h"

int main() {
    // Test default constructor
    Person person1;
    person1.print(); // Should print: Name: No Name, Age: 0

    // Test parameterized constructor
    Person person2("John Doe", 30);
    person2.print(); // Should print: Name: John Doe, Age: 30

    // Test copy constructor
    Person person3 = person2;
    person3.print(); // Should print: Name: John Doe, Age: 30

    // Modify the copied object
    person3 = Person("Jane Doe", 25);
    std::cout << "After modifying the copied object:" << std::endl;
    person3.print(); // Should print: Name: Jane Doe, Age: 25
    person2.print(); // Should still print: Name: John Doe, Age: 30

    // Test copy assignment operator
    Person person4;
    person4 = person2;
    person4.print(); // Should print: Name: John Doe, Age: 30

    // Modify the assigned object
    person4 = Person("Alice", 40);
    std::cout << "After modifying the assigned object:" << std::endl;
    person4.print(); // Should print: Name: Alice, Age: 40
    person2.print(); // Should still print: Name: John Doe, Age: 30

    return 0;
}
```

Assignment Submission

- **Source Code**: Submit `Person.h`, `Person.cpp`, and `main.cpp`.
- **Documentation**: Include comments in your code to explain each part of your implementation.
- **Report**: Write a short report (1-2 pages) explaining:

  - The design choices you made.
  - How you implemented the Rule of Three.
  - The results of your tests, including edge cases.

Grading Criteria

- **Correctness**: Implementation correctly manages dynamic memory and adheres to the Rule of Three.
- **Code Quality**: Code is well-organized, readable, and includes appropriate comments.
- **Testing**: Main function thoroughly tests all constructors and assignment operators, including edge cases.
- **Documentation**: Report clearly explains the design, implementation, and test results.

Conclusion

This assignment will help you solidify your understanding of dynamic memory management and the importance of the Rule of Three in C++. It is essential for writing robust and efficient C++ programs that manage resources properly.