Module 17: The Rule of Three

Learning Objectives

1. Explain when to use the Rule of Three.

Process Skills

1. Information processing. Extract structural patterns from sample code.
2. Critical thinking. Use prior knowledge to interpret and explain related concepts.

Please fill in the roles for each member of your team. Take a look at the description of each role to see its responsibilities. If there are only three people in the group, please assign the same person to the **Presenter** and **Reflector** role. It is a good idea to select roles that you have not recently taken.

Team name: _____ Date: _____

Role	Team Member Name
Manager. Keeps track of time and makes sure everyone contributes appropriately.	
Presenter. Talks to the facilitator and other teams.	
Reflector. Considers how the team could work and learn more effectively.	
Recorder. Records all answers and questions and makes the necessary submission.	

For virtual activities: Once you select your roles, change your Zoom name using the format and example below.

Format: *Group X: First name, Last name initial / Role*

Example: *Group 1: Paul I / Presenter*

Model 1. Shallow copy (12 min) Start time: _____

The C++ compiler provides default destructors, copy constructors, and copy assignment operators to classes that don't explicitly define them.

The default destructor does not perform any actions. The default copy constructor and copy assignment operator will copy all member variable values to the associated object. We can perform all statements below on a Duration object by only defining a constructor, accessors, and mutators on the Duration class.

Duration dur1(900, 1000); // call to constructor taking in start time and end time
Duration copy(dur1); // call to default copy constructor
Duration dur3 = dur1; // call to default copy assignment operator

There are cases when default destructors, copy constructors, and copy assignment operators will not work as intended as shown in the code below.

```
// duration.h
class Duration {
 public:
  Duration() : Duration(800, 900) {}
  Duration(int start_time, int end_time)
      : start_time_(start_time), end_time_(end_time) {}

  int StartTime() const {
    return start_time_;
  }
  int EndTime() const {
    return end_time_;
  }

  void SetStartTime(int start_time) { start_time_ = start_time; }
  void SetEndTime(int end_time) { end_time_ = end_time; }

 private:
  int start_time_;
  int end_time_;
};
```

```
// volunteer.h
#include <iostream>
#include <memory>
#include "duration.h"
```

```
class Volunteer {
 public:
  Volunteer(const std::string& name, double hours_worked)
      : name_(name), hours_worked_(hours_worked) {}

  Volunteer(const std::string& name, double hours_worked,
            int start_time, int end_time)
      : Volunteer(name, hours_worked) {
    schedule_ = std::make_shared<Duration>(start_time, end_time);
  }

  const std::string& Name() const {
    return name_;
  }
  double HoursWorked() const {
    return hours_worked_;
  }


  std::shared_ptr<Duration> Schedule() const {
    return schedule_;
  }

  void AssignSchedule(int start_time, int end_time) {
    schedule_ = std::make_shared<Duration>(start_time, end_time);
  }

  void DisplaySchedule() {
   std::cout << schedule_->StartTime() << " - "
             << schedule_->EndTime() << "\n";
  }

 private:
  std::string name_;
  double hours_worked_;
  std::shared_ptr<Duration> schedule_;
};
```

```
// main.cc
#include <iostream>
#include <memory>
#include "volunteer.h"
int main() {
 Volunteer paul("Paul", 5, 800, 1000);
 std::cout << "Paul's schedule: ";
 paul.DisplaySchedule();
```

```
 Volunteer jc(paul); // use copy constructor to create new
                     // Volunteer with the same values as paul
 std::cout << "JC's schedule: ";
 jc.DisplaySchedule();
 Volunteer doina = paul; // use copy assignment operator to copy
                         // values to the doina object
 std::cout << "Doina's schedule: ";
 doina.DisplaySchedule();

 jc.Schedule()->SetStartTime(600);
 std::cout << "\n==After modifying JC's schedule==\n";
 std::cout << "Paul's schedule: ";
 paul.DisplaySchedule();
 std::cout << "JC's schedule: ";
 jc.DisplaySchedule();
 std::cout << "Doina's schedule: ";
 doina.DisplaySchedule();
 return 0;
}
```

Screen output:
Paul's schedule: 800 - 1000
JC's schedule: 800 - 1000
Doina's schedule: 800 - 1000

==After modifying JC's schedule==
Paul's schedule: 600 - 1000
JC's schedule: 600 - 1000
Doina's schedule: 600 - 1000

Symbol Table before `return 0;`

Variable Name	Scope	Type	Memory address	Value
paul	main()	Volunteer	0x7ffeeed5cb90	paul: Volunteer name_=Paul hours_worked_=5 schedule_: std::shared_ptr<Duration> use_count=3 pointer=0x1fe4ec0*
jc	main()	Volunteer	0x7ffeeed5cb58	jc: Volunteer name_=Paul hours_worked_=5 schedule_: std::shared_ptr<Duration> use_count=3 pointer=0x1fe4ec0*
doina	main()	Volunteer	0x7ffeeed5cb20	doina: Volunteer name_=Paul hours_worked_=5 schedule_: std::shared_ptr<Duration> use_count=3 pointer=0x1fe4ec0*

**Note: assume 0x1fe4ec0 is the address of the Duration object in the heap*

1. Review the Duration class. Take note that it does not provide a destructor, copy assignment operator, or copy constructor. However, we can still create a Duration object using the statement below. Where does it get its copy constructor? Place a check (✓) beside your answer.

   ```
   Duration meeting(930, 1030);
   Duration next_meeting(meeting);
   ```

   a. The compiler will show a compilation error because it is missing a destructor
   b. The compiler generates its copy constructor. ✓
   c. The compiler will show a compilation error because it is missing a copy constructor.
   d. The compiler will show a compilation error because it is missing a copy assignment operator.

2. Review the Volunteer class' default constructor and analyze the instantiation of the volunteer object, michael below. What is the value of michael's schedule_ member variable? Place a check (✓) beside your answer.

   ```
   Volunteer michael("Michael", 5);
   ```

   a. 0x7ffeeed5cb20
   b. 0x7ffeeed5cb58
   c. 0x7ffeeed5cb90
   d. nullptr ✓

3. The jc object was instantiated using the copy constructor, taking paul as the argument. What value was assigned to its name_ member variable? Place a check (✓) beside your answer.

   a. "paul" ✓
   b. "jc"
   c. "doina"
   d. "Anonymous"

4. The paul object was assigned to the doina variable using the copy assignment operator. What value was assigned to its hours_worked_ member variable? Place a check (✓) beside your answer.

   a. 8
   b. 5 ✓
   c. 10
   d. 13

5. Recall that the default copy constructor and copy assignment operators will copy all member variable values to the associated object. What address will jc and doina's schedule_ std::shared_ptr point to? Place a check (✓) beside your answer.

   a. 0x7ffeeed5cb90
   b. 0x7ffeeed5cb58
   c. 0x7ffeeed5cb20
   d. 0x1fe4ec0 ✓

6. Review the screen output. Why do the default copy constructor and copy assignment operator not work as intended in this example? Place a check (✓) beside your answer.

   a. Copying a std::shared_ptr results in a memory leak.
   b. Copying a std::shared_ptr results in copying the address of the object it points to. It does not create its own copy of the underlying object (shallow copy). ✓
   c. There are no issues with using the default copy constructor and assignment operator.

Model 2. Rule of Three (18 min) Start time: _____

We can implement a copy assignment operator to define how complex member variables should be copied. Implementing a copy assignment operator means we probably need to define a copy constructor because we will perform the same complex operations to copy member variables. We will most likely need to implement a destructor as well to make sure any resources are properly deallocated. The *Rule of Three* says that when we implement one of these three components for our class, we will probably need to create all three of them. The code below shows an application of the Rule of Three to make sure member variables are managed properly. The Rule of Three addresses other issues outside the scope of this course.

```
// refined volunteer.h
#include <iostream>
#include <memory>
#include "duration.h"

class Volunteer {
 public:
  Volunteer(const std::string& name, double hours_worked)
      : name_(name), hours_worked_(hours_worked) {}

  Volunteer(const std::string& name, double hours_worked,
            int start_time, int end_time)
      : Volunteer(name, hours_worked) {
    schedule_ = std::make_shared<Duration>(start_time, end_time);
  }

  ~Volunteer() {
    // If member variables are built from classes that implement
    // RAII, we don't need to clean them up in the destructor. They
    // will clean up after themselves. std::shared_ptr applies
    // RAII, so we don't need to write code to manage it.
    // We can just use the compiler-provided default destructor
    // and remove this destructor.
  }

  Volunteer(const Volunteer& other)
      : name_(other.Name()), hours_worked_(other.HoursWorked()) {
    schedule_ =
        std::make_shared<Duration>(other.Schedule()->StartTime(),
                                   other.Schedule()->EndTime());
  }
```

```
  Volunteer& operator=(const Volunteer& other) {
    name_ = other.Name();
    hours_worked_ = other.HoursWorked();
    schedule_ =
        std::make_shared<Duration>(other.Schedule()->StartTime(),
                                   other.Schedule()->EndTime());
    return *this;
  }
  const std::string& Name() const {
    return name_;
  }
  double HoursWorked() const {
    return hours_worked_;
  }

  std::shared_ptr<Duration> Schedule() const {
    return schedule_;
  }

  void AssignSchedule(int start_time, int end_time) {
    schedule_ = std::make_shared<Duration>(start_time, end_time);
  }

  void DisplaySchedule() {
    std::cout << schedule_->StartTime() << " - "
              << schedule_->EndTime() << "\n";
  }

 private:
  std::string name_;
  double hours_worked_;
  std::shared_ptr<Duration> schedule_;
};
```

Screen output (no changes in main.cc):

```
Paul's schedule: 800 - 1000
JC's schedule: 800 - 1000
Doina's schedule: 800 - 1000

==After modifying JC's schedule==
Paul's schedule: 800 - 1000
JC's schedule: 600 - 1000
Doina's schedule: 800 - 1000
```

7. Using the modified Volunteer class, what value is assigned to its schedule_ member variable when we create a Volunteer object using its copy constructor? In the example below, the volunteer liz was created using the copy constructor with jo as the argument. Place a check (✓) beside your answer.

   ```
   Volunteer jo("Jo", 2, 1300, 1500);
   Volunteer liz(jo);
   ```

   a. The copied object's (liz) schedule_ will be nullptr.
   b. The copied object's (liz) schedule_ will contain the same value as the source object's (jo) schedule_ member variable.
   c. The copied object's (liz) schedule_ will be the address of a Duration object in the heap different from the one pointed to by the jo object. ✓

8. After calling the copy constructor, what will be the start and end time of the Duration object pointed to by liz's schedule_ member variable? Place a check (✓) beside your answer.

   a. Same as the jo object (1300 and 1500) ✓
   b. start_time_ and end_time_ will be 0.
   c. There is a syntax error in the code because it does not provide a start and end time for the object.

9. Using the modified Volunteer class, what value is assigned to its schedule_ member variable when we create a Volunteer object using the copy assignment operator? In the example below, the volunteer jo was assigned to sandra using the copy assignment operator. Place a check (✓) beside your answer.

   ```
   Volunteer jo("Jo", 2, 1300, 1500);
   Volunteer sandra = jo;
   ```

   a. The copied object's (sandra schedule_ will be nullptr.
   b. The copied object's (sandra) schedule_ will contain the same value as the source object's (jo) schedule_ member variable.
   c. The copied object's (sandra) schedule_ will be the address of a Duration object in the heap different from the one pointed to by the jo object. ✓

10. After calling the copy assignment operator, what will be the start and end time of the Duration object pointed to by liz's schedule_ member variable? Place a check (✓) beside your answer.

    a. Same as the jo object (1300 and 1500) ✓
    b. start_time_ and end_time_ will be 0.

c. There is a syntax error in the code because it does not provide a start and end time for the object.

11. How do the copy assignment operator and copy constructor solve the shallow copying issue in question 6? Place a check (✓) beside your answer.

a. They do not solve the shallow copying issue.
b. The shallow copying issue can be solved by only implementing a copy constructor that instantiates a new schedule object and assigns it to the schedule_ member variable.
c. The copy assignment operator and copy constructor instantiate a new Duration object with the same values from the source object, then assign it to the schedule_ member variable. ✓

12. As a group, explain in your own words the rule of three.

The rule of three says that we need to create a destructor, copy constructor, and copy assignment operator if we decide to create at least one of these three.

Reflector questions

1. What was the most useful thing your team learned during this session?

2. What prior discussion helped you understand the Rule of Three?

3. If you used your group's explanation of the Rule of Three to describe it to a friend, how well do you think they would understand it? Use a scale of 1 - 5 where 1 means clearly understand, and 5 means do not understand at all.