Module 9: Constructors and Overloading

Learning Objectives

1. Write code to control object construction.
2. Write code that uses constructor and method overloading to promote readability.

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. Default constructors (3 min) Start time: _______

```cpp
// volunteer.h
#include <iostream>

class Volunteer {
 public:
  std::string Name() const { return name_; }
  void SetName(const std::string &name) { name_ = name; }
  std::string Branch() const { return branch_; }
  void SetBranch(const std::string &branch) { branch_ = branch; }

  void RequestFood(const std::string  &name, int quantity) const {
    std::cout << std::to_string(quantity) << " x "
              << name << "requested.\n";
  }
  void RequestFood(const Food &f) const {
    std::cout << f.DisplayName() << "requested.\n";
  }

 private:
  std::string name_;
  std::string branch_;
};
```

```cpp
// main.cc
#include <iostream>
int main() {
   Volunteer tuffy;
   return 0;
}
```

1. Looking at the C++ code, do you see any constructors for the Volunteer class?

No

2. If a class is written without a default constructor, the compiler adds one for you. This default constructor does not perform any actions except instantiate the object. Why do you think it is valid to instantiate a Volunteer object called tuffy as shown in the main function? Place a check (✓) beside your answer.

   a. The Volunteer class does not provide any constructors, so the compiler-provided default constructor is called. ✓
   b. The Volunteer class contains code for the default constructor, so that is called.
   c. It is actually invalid because C++ classes need to have a default constructor for them to compile correctly.

3. Analyze the Food class below. Write code to instantiate a Food object using its default constructor.

```
// volunteer.h
#include <iostream>

class Food {
 public:
  std::string Name() const { return name_; }
  void SetName(const std::string &name) { name_ = name; }
  int Quantity() const { return quantity_; }
  void SetQuantity(int quantity) { quantity_ = quantity; }

  std::string DisplayName() const {
    return std::to_string(quantity_) + " x " + name_;
  }

 private:
  std::string name_;
  int quantity_;
};
```

```
Food my_food;
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 2. Member initializer list (6 min) Start time: ______

```
// volunteer.h
#include <iostream>

class Volunteer {
 public:
  Volunteer() {
    name_ = "";
    branch_ = "CSU Fullerton";
  }
  Volunteer(): name_(""), branch_("CSU Fullerton") { }
  Volunteer(const std::string &name, const std::string &branch)
      : name_(name), branch_(branch) { }

  std::string Name() const { return name_; }
  void SetName(const std::string &name) { name_ = name; }
  std::string Branch() const { return branch_; }
  void SetBranch(const std::string &branch) { branch_ = branch; }

  void RequestFood(const std::string  &name, int quantity) const {
    std::cout << std::to_string(quantity) << " x "
              << name << "requested.\n";
  }
  void RequestFood(const Food &f) const {
    std::cout << f.DisplayName() << "requested.\n";
  }

 private:
  std::string name_;
  std::string branch_;
};
```

```
// main.cc
#include <iostream>
int main() {
   Volunteer tuffy;
   Volunteer tuffy2("titan", "CSULB");
   return 0;
}
```

4. Analyze Volunteer's default constructor. The *member initializer list* is written as a comma-separated list of member variable names with their corresponding initial values enclosed in ( ). Complete the table below to indicate the initial values of each member variable according to the member initializer list. Write your answers under the Value column.

Member variable	**Value**
name_	""
branch_	"CSU Fullerton"

5. The Volunteer class provides another constructor that takes in name and branch parameters. What do you think it means when we use the parameter's names in the initializer list? Place a check (✓) beside your answer.

   a. The first parameter's value is assigned to the first member variable.
   b. The value of each parameter is assigned to the corresponding member variable. ✓
   c. The member variable's value does not change.

6. Create a constructor for the Food class that takes in name and quantity as parameters and uses them to initialize the corresponding member variables. Write code for the constructor only.

```
Food(const std::string &name, int quantity): name_(name), quantity_(quantity) { }
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 3. Copy Constructors (6 min) Start time: ______

```
class Volunteer {
 public:
  Volunteer(): name_(""), branch_("CSU Fullerton") { }
  Volunteer(const std::string &name, const std::string &branch)
      : name_(name), branch_(branch) { }
 Volunteer(const Volunteer &other)
      : name_(other.Name()), branch_(other.Branch()) { }

  std::string Name() const { return name_; }
  void SetName(const std::string &name) { name_ = name; }
  std::string Branch() const { return branch_; }
  void SetBranch(const std::string &branch) { branch_ = branch; }

  void RequestFood(const std::string  &name, int quantity) const {
    std::cout << std::to_string(quantity) << " x "
              << name << "requested.\n";
  }
  void RequestFood(const Food &f) const {
    std::cout << f.DisplayName() << "requested.\n";
  }

 private:
  std::string name_;
  std::string branch_;
};
```

```
// main.cc
#include <iostream>
int main() {
   Volunteer tuffy;
   Volunteer tuffy_clone(tuffy);
   return 0;
}
```

7. This time we see a third constructor for the Volunteer class, called the copy constructor. What kind of data do you think we use as a parameter to Volunteer's copy constructor? It may also help to review the previous modules on std::vectors and std::maps when we first saw copy constructors.

A Volunteer object.

8. What does a copy constructor do?

It copies the member variables of an existing object to the object being initialized.

9. Write the code for the Food class' copy constructor.

Food(const Food &other): name_(other.Name()), quantity_(other.Quantity()) { }

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 4. Overloading (9 min) Start time: _______

Object Instantiation

Instantiation	**Volunteer constructor**

`Volunteer tuffy;`	`Volunteer(): name_(""),`
	`    branch_("CSU Fullerton") { }`

`Volunteer titan("Titan",`
`                 "CSUF");`
	`Volunteer(const std::string &name,`
	`          const std::string branch)`
	`    : name_(name), branch_(branch) {`
	`}`

`Volunteer`
`tuffy_clone(tuffy);`
	`Volunteer(const Volunteer &other)`
	`    : name_(other.Name()),`
	`      branch_(other.Branch()) { }`

Member function call

Member function call	**Member function**

`tuffy.RequestFood("Milk"`
`,`
`                  2);`
	`void RequestFood(const std::string`
	`&name,`
	`                 int quantity);`

`Food f("Milk", 2);`
`tuffy.RequestFood(f);`
	`void RequestFood(const Food &f);`

10. Developers can instantiate a Volunteer object using any of its three constructors. Which constructor was called on the first row of the Object instantiation table? The concept of providing multiple constructors and the ability to choose which one to use is called *constructor overloading*.

Default constructor

11. Which instantiation uses the copy constructor? Provide the row number from the Object instantiation table.

3

12. How do you think the compiler knows which constructor it should call? Place a check (✓) beside your answer.

    a. It checks the name of the member function that was called.
    b. It compares the number of arguments and their data types with the constructor's parameters. ✓
    c. It always calls the overload with the least number of parameters.

13. Do you think it is possible to create another constructor with the same set of parameters? For example, adding the following constructor to Volunteer. Explain your reasoning.

```
Volunteer(): name_("Default"), branch_("CSU Long Beach") { }
```

No, because the compiler will not know which of the two default constructor it should call.

14. Much like constructors, member functions can also be overloaded. Write the name of Volunteer's overloaded member function. Take note that member function overloads need to have the same return type.

RequestFood

15. How do you think the compiler knows which member function it should call? Place a check (✓) beside your answer.

    a. It checks the name of the member function that was called.
    b. It compares the number of arguments and their data types with the member function's parameters. check
    c. It always calls the overload with the least number of parameters.

16. Create a member function overload for Food's DisplayName member function that takes in a bool parameter called show_quantity. The member function should work exactly like the first DisplayName member function if show_quantity is true, but it should only return the name of the food if show_quantity is false. Write only the member function overload below.

```
std::string DisplayName(bool show_quantity) const {
  if (show_quantity) {
    return std::to_string(quantity_) + " x " + name_;
  } else {
    return name_;
  }
}
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Extra challenge (6 min) Start time: ______

```
class Volunteer {
 public:
  Volunteer(): Volunteer("", "CSU Fullerton") { }
  Volunteer(const std::string &name, const std::string branch)
      : name_(name), branch_(branch) { }
  Volunteer(const Volunteer &other)
      : name_(other.Name()), branch_(other.Branch()) { }

  std::string Name() const { return name_; }
  void SetName(const std::string &name) { name_ = name; }
  std::string Branch() const { return branch_; }
  void SetBranch(const std::string &branch) { branch_ = branch; }

  void RequestFood(const std::string  &name, int quantity) const {
    std::cout << std::to_string(quantity) << " x "
              << name << "requested.\n";
  }
  void RequestFood(const Food &f) const {
    RequestFood(f.Name(), f.Quantity());
  }

 private:
  std::string name_;
  std::string branch_;
};
```

17. Analyze the default constructor. What do you think happens when we instantiate an object using the default constructor?

It calls the second constructor passing in "" and "CSU Fullerton" as arguments.

18. Analyze the RequestFood member function with a Food parameter. What do you think happens when we call it?

It calls the other RequestFood member function passing in the name and quantity of the parameter f as arguments.

Reflector questions

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

2. What did the group think about the member initializer list syntax?

3. Consult your group to rate the difficulty of understanding overloading, where 1 is easy, and 5 is difficult. Explain your rating.