Module 8: Objects and References

Learning Objectives

1. Distinguish when to use pass-by-value vs. pass-by-reference.
2. Write code that passes object references to functions.

Process Skills

1. Critical thinking. Use prior knowledge to interpret and explain related concepts.
2. Oral and written communication. Explain the reasoning for making a design decision.

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.	

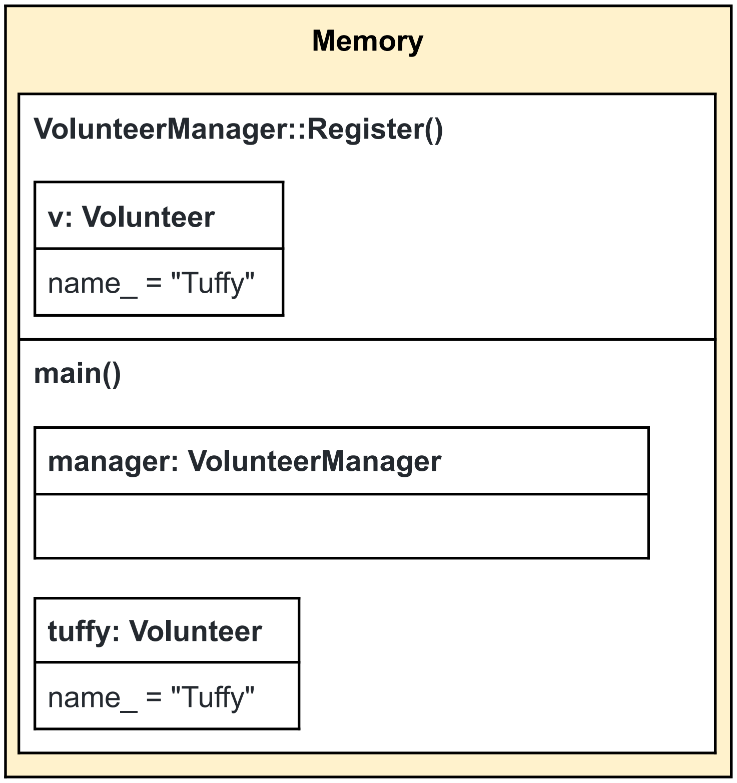
For virtual activities: Once you select your roles, change your Zoom name using the format and example below.
Format: Group X: First name, Last name initial / Role
Example: Group 1: Paul I / Presenter

Model 1. Pass-by-value (6 min)

Start time: ______

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

class Volunteer {
public:
 Volunteer() { }
 std::string Name() { return name_; }
 void SetName(std::string name) { name_ = name; }

private:
 std::string name_;
};

class VolunteerManager {
 public:
   VolunteerManager() { }
   void Register(Volunteer v) {
     std::cout << "Welcome " << v.Name() << "!\n";
   }
};
```

```
// main.cc
#include <iostream>
#include "manager.h"

int main() {
 VolunteerManager
manager;
 Volunteer tuffy;
 tuffy.SetName("Tuffy");

 manager.Register(tuffy);
 return 0;
}
```



1. The model shows a snapshot of the computer's memory when manager's Register member function was called. How many Volunteer objects are inside the main() section of the memory?

1

2. When the tuffy object was passed as the argument to manager's Register member function, it creates a copy of the tuffy object inside the VolunteerManager::Register() section of the memory. This method of passing arguments is called *pass-by-value*. How many Volunteer objects are in memory?

2

3. Passing tuffy to the sizeof (https://en.cppreference.com/w/cpp/language/sizeof) function tells us that a Volunteer object takes up 32 bytes of memory on an Ubuntu (Linux) operating system. How many bytes of memory in total do the Volunteer objects in this program occupy?

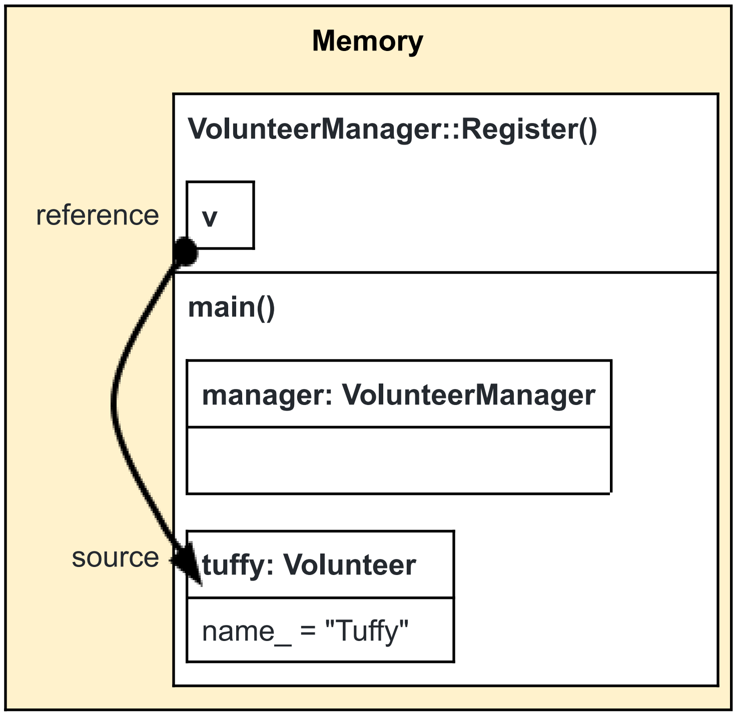
64 bytes

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

Model 2. Object references (12 min) Start time: _______

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

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

private:
 std::string name_;
};

class VolunteerManager {
 public:
   VolunteerManager() { }
   void Register(const Volunteer &v) {
     std::cout << "Welcome " << v.Name() << "!\n";
   }
};
```

```
// main.cc
#include <iostream>
#include "manager.h"

int main() {
 VolunteerManager
manager;
 Volunteer tuffy;
 tuffy.SetName("Tuffy");

 manager.Register(tuffy);
 return 0;
}
```



4. Compare VolunteerManager's Register member function in Model 2 and Model 1. What symbol did we add before the parameter's name? This is called the *reference declarator*.

&

5. When we use the *reference declarator* on a parameter, the compiler uses a different method of passing arguments called *pass-by-reference*. Based on the illustration, how many Volunteer objects do you think are in memory when calling the manager's Register member function?

1

6. Reference variables do not take up space in memory because the compiler only references the source variable. How many bytes of memory in total do the Volunteer objects in this version of the program occupy?

32

7. Recall the const keyword that makes constants. What do you think is the effect of making the Volunteer parameter v constant? Place a check (✓) beside your answer.

    a. There is no difference if we place a const keyword or not.
    b. The passed object cannot be modified inside the function. ✓
    c. The passed object will be passed-by-value instead of passed-by-reference.

8. If we only plan to retrieve information from a passed object, we prefer to make it const. Why do you think Register's parameter v was declared constant? Place a check (✓) beside your answer.

    a. The function only retrieved the name of v. ✓
    b. The function modified v's name.
    c. We should always make parameters constant.

9. Member functions that do not modify a class' member variable should be marked as a const function (the const keyword is written after the parameter list and before the curly brace). The compiler may produce an error if a const object parameter's member function is called, but the function was not marked const. In our example what will happen if we remove the const keyword from the Name member function? Place a check (✓) beside your answer.

    a. The code will compile successfully.
    b. The compiler produces an error because the Register function declares v as a const and calls its Name function, but Name is not declared const. ✓
    c. The compiler produces an error because the Register function declares v as a const, but SetName is not declared const.

Object passing checklist

1. Go over each member function and add the const keyword to functions that do not modify any member variables.
2. When accepting objects as parameters, prefer to pass them by reference.
3. Add a reference declarator (&) before the identifier name of an object parameter.
4. Use the dot notation to call a const object's member function.

10. Work as a team to apply the Object Passing Checklist. Modify food.h and volunteer.h so it properly uses object references.

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

class Food {
 public:
  Food() { }
  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;
  }

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

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

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

  void RequestFood(const Food &f) const {
     std::cout << f.Quantity() << " x "
               << f.Name() << " requested.\n";
  }

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

Extra challenge (5 min)

Start time: _______

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

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

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

11. You may have noticed that the name parameter in SetName was also changed to a const reference. Why do you think it was changed? *Hint: Is std::string a primitive data type or a class? See cppreference.com.*

std::string is a class so it should also be passed-by-reference.

Reflector questions

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

2. What previous topic(s) helped you understand today's discussion? List them below.

3. Describe how your group made decisions while applying the Object Passing Checklist (e.g., how did you find a common answer? how did you resolve different opinions).