Part C – Encapsulation

Class with a Resource

Workshop 6 (1.0 at_home)

In this workshop, you will work with a class that holds dynamic resources as well as static ones. In addition to having regular member functions, this class will also be copied. Copying their data members will involve the use of copy constructors and assignment operators. The classes used in this workshop will be the Figurine class for which we will mass produce figures based on a master mold.

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities

- To implement a copy constructor and copy assignment operator
- To manage a class with dynamic resources
- To use existing objects to create new ones or to assign values to existing objects based on other existing ones.
- To describe to your instructor what you have learned in completing this workshop

Submission Policy

The workshop is divided into 2 sections;
in-lab - 50% of the total mark
To be completed before the end of the lab period and submitted from the lab.
at-home - 50% of the total mark
To be completed within 2 days after the day of your lab.

The *in-lab* section is to be completed after the workshop is published, and before the end of the lab session. The *in-lab* is to be submitted during the workshop period form the lab.

If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.
If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is 2 days after your scheduled *in-lab* workshop (23:59) (even if that day is a holiday).
All your work (all the files you create or modify) must contain your name, Seneca email and student number.
You are responsible to back up your work regularly.
Ask your professor if there are any additional requirements for your specific section.

CITATION AND SOURCES

When submitting the DIY part of the workshop, Project and assignment deliverables, a file called sources.txt must be present. This file will be submitted with your work automatically.
You are to write either of the following statements in the file "sources.txt":
I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.
Then add your name and your student number as signature
OR:
Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.
You need to mention the workshop name or assignment name and also the file name and the parts in which you received the code for help.
Finally add your name and student number as signature.
By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

LATE SUBMISSION PENALTIES:

-*In-lab* portion submitted late, with *at-home* portion:
    0 for *in-lab*. Maximum of **at-home**/10 for the workshop.
-at-home submitted late:
    1 to 2 days, -20%, 3 to 7 days -50% after that submission rejected.

-If any of *the at-home* or in-lab portions is missing, the mark for the whole workshop will be **0/10**

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:
Run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

```
~profname.proflastname/submit 244/NXX/WS06/in_lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS06/at_home -due<ENTER>
```

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:

```
g++ -Wall -std=c++11 -o ws (followed by your .cpp files)
```

After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws")

```
valgrind ws <ENTER>
```

IN-LAB (50%)

FIGURINE MODULE

The **Figurine** module is a class that represent a collectible figurine.

To accomplish the above follow these guidelines:

Design and code a class named `Figurine`.

Follow the usual rules for creating a module: (i.e. Compilation safeguards for the header file, namespaces and coding styles your professor asked you to follow).

Create the following constants in the **Figurine** header:

`MAX_NAME_LEN` with a value of 20.
This constant number is used to set the maximum length of the name of the Figurine.

`DEFAULT_PRICE` with a value of 25.25
This constant number is used to set the default price of a Figurine if an invalid value is given

PRIVATE MEMBERS:

A Figurine will have the following data members:

- `name`
  This is a **character array** representing the nickname of the Figurine. Utilize the MAX_NAME_LEN to set the size of this array. Remember to consider the **nullbyte**.
- `pose`
  This is a **character pointer** representing the pose of the Figurine.
- `price`
  This is a **double value** that represents the price of the Figurine.

PUBLIC MEMBERS:

Constructors/Destructors

Figurine objects will be making use of **constructors** to handle their creation. There are three **constructors** that are required:

1. Default constructor – This constructor should set a Figurine object to **a safe empty state**. The notion of what the safe empty state will be left up to your design but choose reasonable values.
2. 3 Argument Constructor – This constructor will take in 3 parameters:
   a. A **constant character pointer** that represents the name of the Figurine
   b. A **constant character pointer** representing the pose of the Figurine
   c. A **double value** that represents the price of the Figurine

   A little bit of validation needs to occur in the 3 arg constructor. **If the provided constant character pointer** for the name or the pose is either **nullptr** or an **empty string** then set the Figurine to **a safe empty state**.

   Otherwise set the data members to be the values of the parameters as they

are. If the provided name from the parameter is greater than **MAX_NAME_LEN** then simply copy over **MAX_NAME_LEN** characters.

Recall that the pose data member is a character pointer. This then means that **dynamic memory** will be required. When initializing the pose use the length of the respective parameter to determine how much memory is required. Additionally, remember that this same memory **will need to be cleaned up at some point** when a Figurine object goes out of scope.

Remember to close off the string with a **nullbyte** at the last index for all the character array data members.

If the price value provided from the parameter is less than 1 then set the price to the DEFAULT_PRICE constant.

3. Copy Constructor – This constructor will take in 1 parameter:
    a. A **constant Figurine reference** that will be treated as a source object from which we will create a new Figurine.

In essence we are copying all the parameters of the parameter Figurine and using their values to create a new Figurine with the same values. As a Figurine is composed of resources where some of them are static and others are dynamic, attention will need to be paid when copying values.

For the name and the price data members the copying can be done via **shallow copying.** Whereas the pose needs to apply **deep copying** (recall the need for independence of resources).

The steps to follow for this constructor should be the following:

- **Shallow** copy all the static resources first
- Check if the parameter Figurine's **pose** is in an **empty** state or not
- If it is in an **empty** state then simply set the **pose** of the current object being created to an **empty** state as well.
- If it isn't empty then prepare to **deep** copy by allocating memory for the pose based on the source Figurine's pose
- After allocation **copy** the value from the source's pose into the current object's pose
- Remember to append a null byte to the end of the pose.

Other Members

```
void setName(const char*);
```
This member function is used to set the name of the Figurine based on the parameter. Recall the need for a **nullbyte** at the end of the name character array.

```
void setPose(const char*);
```
This member function is used to set the pose of the Figurine based on the parameter. Recall that the pose is a character pointer thus pay attention to the need for **allocation** and **deallocation** of **memory** when setting the pose. Recall the need for a **nullbyte** at the end of the pose **dynamic** character array.

```
void setPrice(double);
```
This member function is used to set the price of the Figurine based on the parameter. If the **price is less than 1** then set the price to the DEFAULT_PRICE.

```
ostream& display() const;
```
This member function will display the current details of the Figurine. If the Figurine is in an **empty** state the following will be printed:

```
"This Figurine has not yet been sculpted" <newline>
```

If the Figurine isn't empty then it will print:

```
"Figurine Details" <newline>
"Name: [name]" <newline>
"Pose: [pose]" <newline>
"Price: [price]" <newline>
```

The price should have two decimal point precision.

Refer to the sample output for details.

Lastly at the end of the function **return the cout object**.

Operator Overload Members

```
operator bool() const;
```
This **boolean conversion** operator will define the behavior of converting a

Figurine into a bool value. For our context, we will use the notion of if the name is in an **empty** state then we will return **true**. Otherwise we return **false**.

IN-LAB MAIN MODULE

```
/*****************************************************************************
// OOP244 Workshop 6: Class with Resources
// File FigurineTester.cpp
// Version 1.0
// Date       2019/10/22
// Author     Hong Zhan (Michael) Huang
// Description
// Tests the Figurine class and the ability to create new objects by
// copying
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*****************************************************************************/

#include <iostream>
#include "Figurine.h"
#include "Figurine.h"

using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
  for (int i = 0; i < len; i++, cout << ch);
  return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}

int main() {

  cout << "Figurine default constr" << endl;
  line(64, '-') << endl;
  number(1) << endl;
  Figurine f1;
  f1.display() << endl;

  cout << "Figurine 3 arg constr (invalid)" << endl;
  line(64, '-') << endl;
  number(2) << endl;
  Figurine f2("nullptr", nullptr, 12);
  Figurine f3(nullptr, "nullptr", 12);
  f2.display();
  f3.display() << endl;
```

```
  cout << "Figurine 3 arg constr (valid)" << endl;
  line(64, '-') << endl;
  number(3) << endl;
  Figurine f4("Shiva", "Lotus", 50);
  Figurine f5("Grimnir", "Child", -99);
  f4.display() << endl;
  f5.display() << endl;

  cout << "Figurine copy constructor (empty)" << endl;
  line(64, '-') << endl;
  number(4) << endl;
  Figurine f6 = f2;
  f6.display() << endl;

  cout << "Figurine copy constructor (non-empty)" << endl;
  line(64, '-') << endl;
  number(5) << endl;
  Figurine f7("Europa", "Crane", 90);
  Figurine f8 = f7;
  f7.display() << endl;
  f8.display() << endl;

  cout << "Figurine change values of copy" << endl;
  line(64, '-') << endl;
  number(5) << endl;
  f8.setName("Alexiel");
  f8.setPose("Triangle");
  f8.setPrice(200);
  f7.display() << endl;
  f8.display() << endl;
}
```

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Figurine default constr
----------------------------------------------------------------
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
This Figurine has not yet been sculpted

Figurine 3 arg constr (invalid)
----------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
This Figurine has not yet been sculpted
This Figurine has not yet been sculpted

Figurine 3 arg constr (valid)
----------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
Figurine Details
Name: Shiva
Pose: Lotus
Price: 50.50

Figurine Details
Name: Grimnir
Pose: Child
Price: 25.25
```

```
Figurine copy constructor (empty)
-----------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
This Figurine has not yet been sculpted

Figurine copy constructor (non-empty)
-----------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Figurine Details
Name: Europa
Pose: Crane
Price: 90.90

Figurine Details
Name: Europa
Pose: Crane
Price: 90.90

Figurine change values of copy
-----------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Figurine Details
Name: Europa
Pose: Crane
Price: 90.90

Figurine Details
Name: Alexiel
Pose: Triangle
Price: 200.20
```

IN-LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload Figurine module and the FigurineTester.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS06/in_lab<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your

implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT_HOME (50%)

FIGURINE MODULE (UPDATED)

The **Figurine** module will be updated to include some new data members and functions.

PRIVATE MEMBERS:

Add the following new data member to the Figurine class:

- `copy`
  This is a **Boolean value** used to determine whether the current object is a copy or not.

PUBLIC MEMBERS:

Constructors

Update the existing constructors to accommodate for the new data member in the following ways:

1. Default Constructor – The default constructor should set the value of the **copy** data member to **false** as we are not creating a copy.
2. 3 Argument Constructor – Similarly the 3-argument constructor should set the value of **copy** to **false** as we are not creating a copy.
3. Copy Constructor – The copy constructor will set the current object's value of **copy** to **true** after it completes the deep copying.

Copy Assignment Operator

Implement a copy assignment operator for the Figurine class that has the following signature:

```
Figurine& operator=(const Figurine&)
```

Recall that the actions performed here will be very similar to that of the copy constructor. There are however a few additions:

1. Prior to doing any copying, we should check for **self-assignment**. If the parameter is the same object as the current object, there is no need to do anything but return the current object.
2. Passing the **self-assignment** check then proceed to **deallocate** any old memory that was allocated to our dynamic resources.
3. Proceed to **shallow** copying of static resources.
4. Proceed to **deep** copying of dynamic resources.
5. Set the **copy** data member of the current object to **true** as we have made a copy.
6. Return the current object.

Consider how you may best add this overload but keep redundancy to a minimum.

Other Members

```
bool isCopy() const;
```
Create a new query function that returns the value of the **copy** data member.

```
ostream& display() const;
```
Update the display function to incorporate the **copy** data member.

If the object is **empty** and a **copy** print the following:

```
"This Figurine has not yet been sculpted" <newline>
“This is a replica of emptiness” <newline>
```

If the Figurine isn’t **empty** but is a **copy** then it will print:

```
"Figurine Details" <newline>
"Name: [name]" <newline>
"Pose: [pose]" <newline>
"Price: [price]" <newline>
“This is a replica” <newline>
```

Helper Functions

`bool operator==(const Figurine&, const Figurine&)`
Create a helper function that compares two Figurines against one another. It returns a **true** value if and only if:

1. The name, pose and price are the same between the two
2. Neither Figurine is a copy of the other.

Any other case will return **false**.

Create any public member functions needed to allow this helper to work. Do not incorporate friendship or otherwise allow direct access to private data

AT-HOME MAIN MODULE

```
/*************************************************************************
// OOP244 Workshop 6: Class with Resources - At Home
// File FigurineTester2.cpp
// Version 1.0
// Date       2019/10/22
// Author     Hong Zhan (Michael) Huang
// Description
// Tests the Figurine class and the ability to create new objects by
// copying
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*************************************************************************/

#include <iostream>
#include "Figurine.h"
#include "Figurine.h"

using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
  for (int i = 0; i < len; i++, cout << ch);
  return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}

int main() {

  cout << "Figurine default constr" << endl;
```

```
line(64, '-') << endl;
number(1) << endl;
Figurine f1;
f1.display() << endl;

cout << "Figurine copy empty object via CC" << endl;
line(64, '-') << endl;
number(2) << endl;
Figurine f2 = f1;
f2.display() << endl;

cout << "Figurine copy empty object via CA" << endl;
line(64, '-') << endl;
number(3) << endl;
f1 = f2;
f1.display() << endl;

cout << "Figurine 3-arg constr and copy via CC" << endl;
line(64, '-') << endl;
number(4) << endl;
Figurine f3("Yuel", "Fox", 88.88);
Figurine f4 = f3;
f3.display() << endl;
f4.display() << endl;

cout << "Figurine copy non empty object via CA" << endl;
line(64, '-') << endl;
number(5) << endl;
Figurine f5("Societte", "Ninetails", 99.99);
f4 = f5;
f4.display() << endl;
f5.display() << endl;

cout << "Figurine compare original and original" << endl;
line(64, '-') << endl;
number(6) << endl;
if (f5 == f5)
  cout << "This is the real deal" << endl << endl;

cout << "Figurine compare original and copy" << endl;
line(64, '-') << endl;
number(7) << endl;
if (!(f4 == f5))
  cout << "This is very well made" << endl << endl;

cout << "Figurine compare copy and copy" << endl;
line(64, '-') << endl;
number(8) << endl;
Figurine f6 = f5;
if (!(f5 == f6))
  cout << "It's like looking into a mirror" << endl << endl;
```

}

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Figurine default constr
----------------------------------------------------------------
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
This Figurine has not yet been sculpted

Figurine copy empty object via CC
----------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
This Figurine has not yet been sculpted
This is a replica of emptiness

Figurine copy empty object via CA
----------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
This Figurine has not yet been sculpted
This is a replica of emptiness

Figurine 3-arg constr and copy via CC
----------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
Figurine Details
Name: Yuel
Pose: Fox
Price: 88.88

Figurine Details
Name: Yuel
Pose: Fox
Price: 88.88
This is a replica

Figurine copy non empty object via CA
----------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Figurine Details
Name: Societte
Pose: Ninetails
Price: 99.99
This is a replica

Figurine Details
Name: Societte
Pose: Ninetails
Price: 99.99

Figurine compare original and original
----------------------------------------------------------------
6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6 - 6
This is the real deal

Figurine compare original and copy
----------------------------------------------------------------
```

```
7 - 7 - 7 - 7 - 7 - 7 - 7 - 7 - 7 - 7
This is very well made

Figurine compare copy and copy
-----------------------------------------------------------------
8 - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8 - 8
It's like looking into a mirror
```

AT-HOME SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload Figurine module and the FigurineTester2.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS06/at_home<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.