/\*

Course\_Name: OOP 345

Section: A

Title: Workshop 5 Part 2

File\_Name: reflect.txt

Student\_Name: Zhaokai\_Guan

Student\_ID: 130988215

\*/

This workshop is less intense compared to the previous one, but still I was able to learn numerous important skills and knowledges about functions and error handling.

1. the difference between the implementations/utilizations of a functor and a lambda expression. When is appropriate to use each one?

First of all, functors is an object with function call operator overloaded. So we could implement the function logic inside the overloaded function call operator. It is pretty sick because we can use an function like an object. To be more specific, please take a look inside my code.

const unsigned ARRAY\_SIZE = 6;

class SpellChecker {

std::string m\_badWords[ARRAY\_SIZE]{};

std::string m\_goodWords[ARRAY\_SIZE]{};

size\_t m\_counts[ARRAY\_SIZE]{{}};

public:

SpellChecker(const char\* filename);

void operator ()(std::string& text);

void showStatistics(std::ostream& out) const;

};

We actually put the function logic into the overloaded operator(), and the member attributes are the data required from the function logic.

However, we could have some status attributes like function call counter. And create a getCallCounter() as a public query method.

In this way, we could know how many times this functor object has been called.

Anyways, this is more powerful and flexible compared to the functions we wrote in oop244.

As for the appropriate usage, I would say when you have function logic depends on the external data (like this spellchecker module of this ws). To enhance the flexibility, it is better to have a functor like this. Also, it is important to notice that functor’s implementation is more time consuming than implementing a conventional function logic.

As for the lambda expression, it is very good for writing a function with the relatively simple and concise function logic. And also with the knowledge of function pointer, we could actually store the function into a variable and use it latter just like what I did in this workshop.

auto fixPrice = [=](Book& src) {

if (src.country() == "US") {

src.price() \*= usdToCadRate;

}

else if (src.country() == "UK" && (src.year() >= 1990 && src.year() <= 1999)) {

src.price() \*= gbpToCadRate;

}

};

And latter on, I used a for loop to loop through all the elements from library in order to have fix the prices by calling this lambda.

Also, it should not be neglected that lambda can capture values from outside the function scope by using the [=] or [&] in order to get the read access or read and write access respectively. It makes this even more powerful when we are implementing a light-weighted function inside a module. ( possibly that function won’t be used outside of that module)

1. the constructor for SpellChecker generates an exception in certain conditions. How would you change your code to achieve the same result, but without exceptions (both on the constructor and on the client side)? Compare the two solutions.

Without creating an exception, we will probably use our old friend dummy object. And in the main code, we will be using if clause to check if we create the dummy object. If a dummy object was created, we will probably warn the user about bad file name.

Here is the code from my workshop.

SpellChecker::SpellChecker(const char\* filename){

std::ifstream ifs(filename);

if (ifs){

unsigned index{};

while (index<ARRAY\_SIZE){

std::string temp{};

std::getline(ifs, temp);

if (ifs){

m\_badWords[index] = temp.substr(0, temp.find\_first\_of(' '));

m\_goodWords[index] = temp.substr(temp.find\_last\_of(' ')+1);

index++;

}

}

}

else{

throw ("Bad file name!");

}

}

But I would say the logic based on the exception is more legitimate. There is no actual object created when there is a bad file name. Think about it, we should probably not to waste any memory on an object created from a wrong file name. Even it is a dummy object, it occupies memory, right? Also, I would say throwing exception’s logic is more concise, and it won’t need any comment to explain the how the programmer defines a dummy object.

1. the classes Movie and Book contain almost identical logic in loading data from the file. How would you redesign the classes in order **not** to duplicate that logic?

I would actually create an interface first and define all the required methods according to this workshop.

Then, Movie and Book can inherit that interface the publicly. If there is more member attributes or methods required while implementing the Movie module and Book module, we could just simply add more specially into that child class. By having an interface, we could already set the rules for the concrete classes before we even implement it. Also, it saves a lot of unnecessary typing for the duplicated logics.

1. the classes Movie and Book are instantiated from the main() function using a custom constructor, but they also contain a default constructor. Is the default constructor necessary? Could you remove it? Justify your answer.

At first, I did some researches about this, and some resources were stating that the compiler will give insert an empty default constructor for the class if there is none. However, I tested in my code. Let’s take the example from the Book module.

class Book {

std::string m\_Author{};

std::string m\_Title{};

std::string m\_PublicationCountry{};

size\_t m\_PublicationYear{};

double m\_Price{};

std::string m\_bookSummary{};

public:

/\*Book() {};\*/

Book(const std::string& strBook);

const std::string& title() const;

const std::string& country() const;

const size\_t& year() const;

double& price();

template<typename T>

void fixSpelling(T& spellChecker);

friend std::ostream& operator << (std::ostream& ostr, const Book& src);

};

I commented out the default constructor and tried to build, and of course it fails. The compiler is warning me that the in the collection module. When I was trying to resize the array of collector ( typename is Book), there is no appropriate default constructor to use, so the compilation failed.

So, you could actually not take the default constructor off when there is a template has the dependency to it.

Further question:

Prof, would you agree with the opinion =>compiler will insert a default constructor for you if there is none declared. <= If that is true, how come on my visual studio it complains when I take off the default constructor? Where did that compiler inserted default constructor go?