

1. Define and explain the following issues related to C++ game engine development. Illustrate them with appropriate examples.

(a) Find out what is the traditional assert macro of ANSI C. What is the purpose of it and how is it used? How can asserts be turned off? Why would we want to turn them off? Compare to the assert facility in Java.

**Answer:** Assert is used like a function to check the value of an expression that we expect to be true under normal circumstances, if expression is a nonzero value, the assert macro does nothing. If expression is zero, the assert macro writes a message to stderr and terminates the program by calling abort. Assertion checks can be turned off at compile time by defining the macro NDEBUG either in the source code or in the command-line interface, (using /DNDEBUG and #define NDEBUG before <assert.h> is included). We would want to turn them off as too much checking would make the program significantly slower and it might greatly reduce the performance of the program.

Compared to Java, you need to explicitly enable assertion validation by using -enableassertions or -ea command line argument, Java assertion is different in the sense that the assert statement can be followed by a string to act as the additional comment on the error that the assertion throw. Such as

**assert I != null : "Connection is null"** in which the error message would be **Exception in thread "main" java.lang.AssertionError: Connection is null**

(b) What is the relation of the assert facility to using pre- and post-conditions? When should we use (1) assert, and when should we (2) throw exceptions from a function? Give some examples of both. What is meant by a class invariant and what is it used for? Should we use assert or exceptions with class invariants? Explain.

**Answer:** We can use assert pre condition to ensure that what we have assumed to be true is true before the function is executed, and we can use assert post condition to specify what will happen if the preconditions are met- and what the function would return to the caller. Assertion should only be used to verify conditions that should be logically impossible to be false, in which our inputs are generated by our own code, to guarantee the preconditions are true. Exception should be thrown when the checks are based on external inputs, and we would try to handle to exception gracefully by using try-catch block or just 'throw' it (delegate it to another class/method).

```
int average (arr){  
    assert arr.length == 0 ;  
    int avg = IntStream.of(arr).sum();  
    return avg;  
}
```

The above computes the average of the integer array and it checks if the arr length is 0 in which we would know there is no element in the array and would result in an assertion error.

#### Using assert as ctrl flw

```
private void search() {  
    for (...) {  
        ... if (found) // will always happen return;
```

```
    }  
  
    assert false; // should never get here  
}
```

This ensures that the code would never reach outside of the loop

### Throwing exception example

```
static void testMethod() throws Exception {  
    String test = null;  
    test.toString();  
}  
  
public class Example {  
    public static void main(String[] arg) {  
        try {  
            testMethod();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

A class invariant is a condition that all objects of that class must satisfy while it can be observed by clients, say we have a balanced tree data structure and we have a private method (private Boolean `balanced()`) that checks if the tree was balanced. We need to invoke the method before and after any methods are completed by using `assert balanced()`. Hence, it is not sensible to catch the exception since we still need to check the condition to be true even after the execution of each method.

2. (a) Write a C++ Complex class, with two floating-point attributes x and y, of your own. Overload typical binary operators of your choice for complex numbers. Consider which operations are best declared as members, and which ones are better declared as helper functions outside the class. Can we depend on automatically generated default copy operations? Consider utilizing inline operations or default parameters. Motivate and explain your implementation decisions.

(b) Write a driver program that reads in floating-point numbers, makes complex numbers out of pairs of numbers read, calculates new complex values from existing complex numbers, and writes out the complex numbers. If possible, compile and try out your program.

**Answer:**

```
#include <iostream>
```

```
using namespace std;
```

```
class Complex
```

```
{
```

```
private:
```

```
    float x;
```

```
    float y;
```

```
public:
```

```
    Complex() : x(0), y(0) { }
```

```
    void input()
```

```
{
```

```
    cout << "Enter real and imaginary number one at a time: ";
```

```
    cin >> x;
```

```
    cin >> y;
```

```
}
```

```
Complex operator - (Complex c2)
```

```
{
```

```
    Complex temp;
```

```
    temp.x = x - c2.x;
```

```
    temp.y = y - c2.y;
```

```
    return temp;
```

```
}
```

Complex operator + (Complex c2)

```
{  
    Complex temp;  
    temp.x = x + c2.x;  
    temp.y = y + c2.y;  
  
    return temp;  
}
```

Complex operator \* (Complex c2)

```
{  
    Complex temp;  
    temp.x = x * c2.x - y * c2.y;  
    temp.y = x * c2.y + y * c2.x;  
  
    return temp;  
}
```

void output()

```
{  
    if (y < 0)  
        cout << "Output: " << x << y << "i" << endl;  
    else  
        cout << "Output: " << x << "+" << y << "i" << endl;  
}  
};
```

int main()

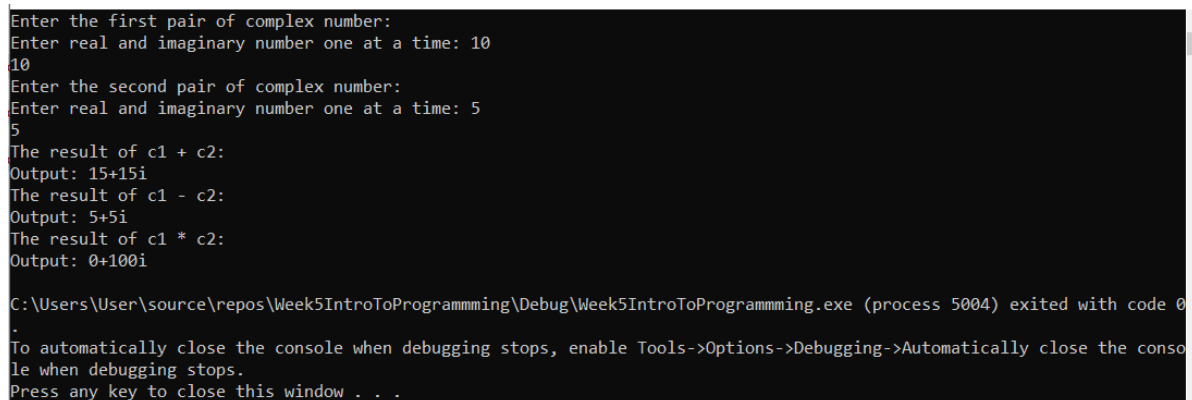
```
{  
    Complex c1, c2, result1, result2, result3;  
  
    cout << "Enter the first pair of complex number:\n";
```

```
c1.input();

cout << "Enter the second pair of complex number:\n";
c2.input();

result1 = c1 + c2;
result2 = c1 - c2;
result3 = c1 * c2;
cout << "The result of c1 + c2:\n";
result1.output();
cout << "The result of c1 - c2:\n";
result2.output();
cout << "The result of c1 * c2:\n";
result3.output();

return 0;
}
```



The screenshot shows a Windows command prompt window with a black background and white text. It displays the execution of a program that takes two complex numbers as input and outputs their sum, difference, and product. The user enters '10' for the first number and '5' for the second. The program outputs the results: '15+15i' for the sum, '5+5i' for the difference, and '0+100i' for the product. At the bottom, a message indicates the program exited successfully with code 0, followed by instructions on how to close the console window.

```
Enter the first pair of complex number:
Enter real and imaginary number one at a time: 10
10
Enter the second pair of complex number:
Enter real and imaginary number one at a time: 5
5
The result of c1 + c2:
Output: 15+15i
The result of c1 - c2:
Output: 5+5i
The result of c1 * c2:
Output: 0+100i

C:\Users\User\source\repos\Week5IntroToProgramming\Debug\Week5IntroToProgramming.exe (process 5004) exited with code 0
.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

I would use inline member functions for all of them since I would need access to the object data and it would be more convenient for me to

3. Describe the two-phase processing (pipeline) of game assets from content-creation tools to a running game. Explain all the flows and transformations of data in this pipeline (usually) needed for (rigid) 3D models.

**Answer:**

The rendering pipeline to get game assets from content-creation tools to a running game consists of tools and asset conditioning stages which would deal with the geometry data of the meshes and materials of the 3D models. In the tools stage, meshes are built by 3d modelers in a digital content creation tool like Maya, 3ds, Max, Lightwave, Softimage/XSI etc. The models are defined by using either NURBS, quads, triangles etc, which are tessellated into triangles prior to rendering. The materials of the 3d models are also authored in this stage like the shader, textures, configuration parameters, vertex attributes are defined. In the asset conditioning stage, the model is exported, processed and linked together multiple types of assets into a cohesive whole. For 3d model, its geometry (vertex and index buffers), materials, textures and optional skeletons are loaded by the engine. The geometric and material data are stored in a platform-independent intermediate format and is further processed into the format compatible with the game engine.

4. Explain the motivation of using versions of the "Model-View-Controller" architecture for a game program. Explain the potential benefits. How can we apply this architecture for implementing video games? What is model, controller, and what are views in this case? When implementing such a game architecture what additional design patterns are required? Describe them briefly.

**Answer:**

Using versions of the MVC is simple and helps us to decouple the classes with each other, by having one class that maintains a list of pointers to instances of some interface (the observers in this case), which would help in observing the list of observers and sending the notifications to them once some triggering conditions have been met.

We can apply this architecture to the game achievements system in the game, in this case, the model would be the classes that represent the game data, the views would be the current interface/game state rendered, and controllers would be the user's input or triggering conditions. The game would wait for the user's input or other triggering condition, in which that will invoke the notification of those events to be sent and updated the model.

Additional design patterns should be considered such as using an asynchronous communication method using Event Queue, since synchronous communication might lock up the games for too long and could potentially deadlock the game, especially in a highly threaded game.

Another design pattern can be having an object pool that we reallocate to avoid dynamic allocation of the observers, so that we can have a fixed-size pile of list nodes of observers to use and reuse as we need without having to deal with manually allocating/deallocating memory.

5. Explore the Service locator game programming pattern. Explain its general purpose, structure, and consequences. For what situations or purposes would Service locator be used in game programs? Sketch a sample design for Service locator. Implement it in a suitable pseudolanguage and explain verbally what is needed to make it work. Discuss the other (game) programming patterns related to this one. How do they resemble or differ from Service locator?

**Answer:**

Service locator provides a global point of access to a service without coupling users to the concrete class that implements it. There is a service class defines an interface to a set of operations, a concrete service provider that implements this interface, and a separate service locator that provides access to the service by finding an appropriate provider while hiding both the provider's concrete type and the process used to locate it. This is best suited for services that need to be available to the entire game such as an audio system.

For example to play an audio with the interface to be as follow:

```
class ConsoleAudio : public Audio  
  
{  
  
public:  
  
virtual void playSound(int soundID)  
  
{  
  
// Play sound using console audio api...  
  
}  
  
};
```

We can have the service locator to be defined this way.

```
class Locator  
  
{  
  
public:  
  
static Audio* getAudio() { return service_; }  
  
static void provide(Audio* service)  
  
{  
  
service_ = service;  
  
}  
  
private:  
  
static Audio* service_;  
  
};
```



We can then call the service locator anywhere in the code and it will give us an instance of our audio service to use.

```
Audio *audio = Locator::getAudio();
```

```
audio->playSound(soundEffect);
```

Another similar one would be singleton design pattern in which exactly one object is needed to coordinate actions across the system, which can help to hide the constructor of the class and have a public static operation getInstance() that returns the sole instance of the class, so that the class itself is responsible of controlling its instantiation and it would only be instantiated only once. Continued to the example above, if we have a singleton pattern instead, we might have the following:

```
class AudioSystem
```

```
{
```

```
public:
```

```
static AudioSystem & instance()
```

```
{
```

```
static AudioSystem *instance = new AudioSystem ();
```

```
return *instance; Yong Jin Cheng 15/12/2020 Intro to game programming exam
```

```
}
```

```
private:
```

```
AudioSystem () {}
```

```
};
```

In which we would call the playSound with this:

```
AudioSystem::instance()->playSound(soundEffect);
```

As we can see from the codes above, singleton and service locator both provide us with global access of the object with service locator being more flexible.