# In class Practice 10/18/2019

Group three or four people team.

Part I

1. Class members are accessed via the __dot(.)____ operator in conjunction with the name of an object (or reference to an object) of the class or via the ___arrow(->)operator in conjunction with a pointer to an object of the class.
2. Class members specified as _private___ are accessible only to member functions of the class and friends of the class.
3. A nonmember function must be declared by the class as a(n) ___friend_____ of a class to have access to that class's private data members.
4. A constant object must be __initiated_; it cannot be modified after it's created.
5. A(n) _____static______ data member represents classwide information.
6. An object's non-static member functions have access to a "self pointer" to the object called the ___this______ pointer.
7. If a member initializer is not provided for a member object of a class, the object's ___default constructor is called.
8. Member objects are constructed ___before_____ their enclosing class object.
9. When a member function is defined outside the class definition, the function header must include the class name and the _______, followed by the function name to "tie" the member function to the class definition. scope resolution operator

## Part II

Find the error in each of the following program segments.

a) Define two constructors as the following for class Time.

```
Time(int h = 0, int m = 0, int s = 0);
Time();
```

Cannot define like this. Cause it's ambiguity when the user used like Time(), it will be confused for the compiler.

b) Assume the following prototype is declared in class Car:

```
void ~Car(int);
```

destructor has no return type and no parameter.

c) Find error(s) from the following class definition

```
class C1  {
public:
        C1 (int y = 10) : data(y) { }
        int getIncrementedData() const {
                return ++data;
        }
        static int getCount() {
                cout << "Data is " << data << endl;
                return count;
        }
private:
        int data;
        static int count;
};
```

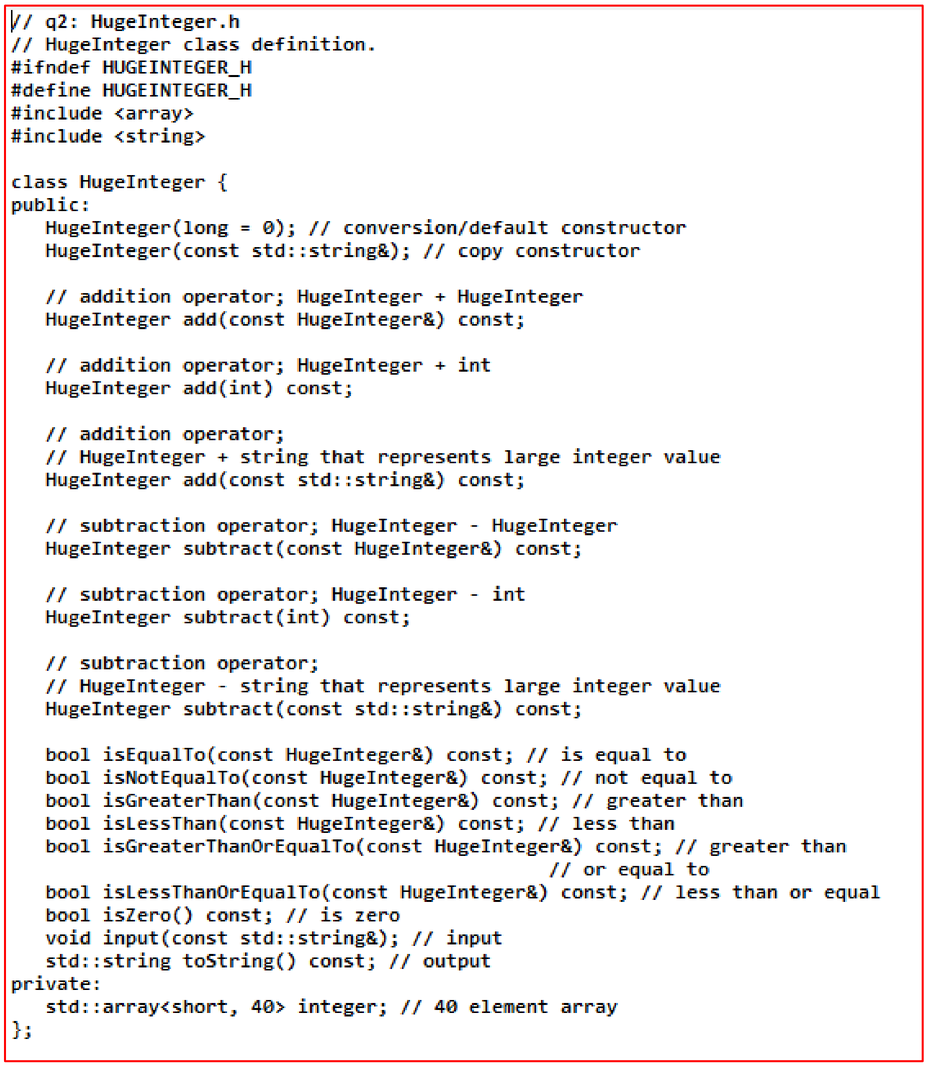
delete the const here.

the static function cannot use non-static attribute

## Part III

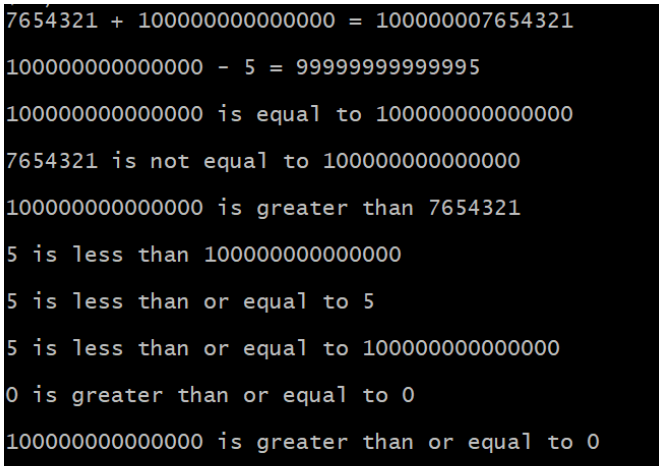
Q1 Modify class definition of Time of Fig9.5 and 9.6. Instead of using three integers to represent the current time of hour, minute and second. We would like to use just ONE integer to represent the time. This change of implementation shouldn’t affect the original public interface. I.e., the clients who use this class can still use the class without modify their application code.

Q2 Implement a class call it *HugeInteger*. The following is the class definition which contains its member function prototypes.



```cpp
// q2: HugeInteger.h
// HugeInteger class definition.
#ifndef HUGEINTEGER_H
#define HUGEINTEGER_H
#include <array>
#include <string>

class HugeInteger {
public:
   HugeInteger(long = 0); // conversion/default constructor
   HugeInteger(const std::string&); // copy constructor

   // addition operator; HugeInteger + HugeInteger
   HugeInteger add(const HugeInteger&) const;

   // addition operator; HugeInteger + int
   HugeInteger add(int) const;

   // addition operator;
   // HugeInteger + string that represents large integer value
   HugeInteger add(const std::string&) const;

   // subtraction operator; HugeInteger - HugeInteger
   HugeInteger subtract(const HugeInteger&) const;

   // subtraction operator; HugeInteger - int
   HugeInteger subtract(int) const;

   // subtraction operator;
   // HugeInteger - string that represents large integer value
   HugeInteger subtract(const std::string&) const;

   bool isEqualTo(const HugeInteger&) const; // is equal to
   bool isNotEqualTo(const HugeInteger&) const; // not equal to
   bool isGreaterThan(const HugeInteger&) const; // greater than
   bool isLessThan(const HugeInteger&) const; // less than
   bool isGreaterThanOrEqualTo(const HugeInteger&) const; // greater than
                                                 // or equal to
   bool isLessThanOrEqualTo(const HugeInteger&) const; // less than or equal
   bool isZero() const; // is zero
   void input(const std::string&); // input
   std::string toString() const; // output
private:
   std::array<short, 40> integer; // 40 element array
};
```

So when you test this class and might generate output such as the following. Note two huge integers.



```
7654321 + 100000000000000 = 100000007654321

100000000000000 - 5 = 99999999999995

100000000000000 is equal to 100000000000000

7654321 is not equal to 100000000000000

100000000000000 is greater than 7654321

5 is less than 100000000000000

5 is less than or equal to 5

5 is less than or equal to 100000000000000

0 is greater than or equal to 0

100000000000000 is greater than or equal to 0
```