**COSC 120 Practice Final Exam Answers**

\* Yes, this is a repeat question of the last practice exam. It’s a good question to make sure you understand the concept of structures!

1. Initialize a new structure called Hospital. This structure will hold a 2D array of booleans - 5 rows and 30 columns (for the 5 floors and 30 rooms per floor of the hospital). The array will be used to signify if a room is currently occupied. It will also contain an integer for the number of patients currently at the hospital and another integer for the number of doctors at the hospital.

**struct Hospital {**

**bool occupied[5][30];**

**int numOfPatients;**

**};**

2. Using the structure that you created in 1, make an array of those structures. The array will be of size 10 and represent all of the hospitals in a state.

**Hospital hospitals[10];**

Write the code to change the number of patients at the fourth hospital to 67.

**hospitals[3].numOfPatients = 67;**

Write the code to change the 13th room on the 4th floor of the 7th hospital to occupied (set that position to true).

**hospitals[6].occupied[3][12] = true;**

Write a for loop that will change every room of the 5th hospital to be vacant.

**for (int r = 0; r < 5; r++)**

**for (int c = 0; c < 30; c++)**

**hospitals[4].occupied[r][c] = false;**

3. Reinitialize your Hospital structure. This new structure will no longer contain an array of booleans; it will instead be a 2D array of Patient structure. The array will be the same size (5 rows, 30 columns). The Patient structure contains firstname (string), lastname (string), and total bill (double) for the patient’s stay at the hospital. Assume that ALL rooms of the hospital are occupied so each element in the Patient array has all of the attributes listed above. You can also remove the total number of patients at the library from the previous questions because every room is filled. Define both the Patient structure and the Hospital structure.

**struct Patient {**

**string firstName;**

**string lastName;**

**double totalBill;**

**};**

**struct Hospital {**

**Patient patients[5][30];**

**};**

4. Initialize your Hospital structure and write the code to change the name of the patient in the fourth room on the fifth floor to “David Jackson”. Update his total bill to $1500.00.

**Hospital hospital;**

**hospital.patients[4][3].firstName = “David”;**

**hospital.patients[4][3].lastName = “Jackson”;**

**hospital.patients[4][3].totalBill = 1500.00;**

5. Write the header of a function that will accept only a Hospital structure and reinitialize it to empty. Call the function **emptyHospital**. It does not need to return anything.

**void emptyHospital (Hospital h)**

Call the function **emptyHospital** passing the Hospital structure defined in 4 into the function.

**emptyHospital(hospital);**

6. Define a new class called Car. Your Car class will have 5 private members - Make, Model, Year, Nickname, Mileage. Make public getters and setters for each of the private members (you will have 10 in total). You will also need a default constructor and a constructor that will take in all 5 private members to set them when the object is created in the order listed above. The default constructor will set Make to “Default”, Model to “Default”, Year to “0”, Nickname to “Default”, and Mileage to 0. Define a destructor as well. The destructor will just need to print out “The car has been destroyed”. Write two files. The first will be the header file with just the declarations of the methods and members. The second will be the cpp implementation of each of the methods.

**Car.h**

**class Car**

**{**

**private:**

**string make;**

**string model;**

**int year;**

**string nickName;**

**int mileage;**

**public:**

**Car();**

**Car(string, string, int, string, int);**

**~Car();**

**void setMake(string nMake);**

**string getMake();**

**void setModel(string nModel);**

**string getModel();**

**void setYear(int nYear);**

**int getYear();**

**void setNickName(string nNickName);**

**string getNickName();**

**void setMileage(int nMileage);**

**int getMileage();**

**};**

**Car.cpp**

**Car::Car()**

**{**

**make = “Default”;**

**model = “Default”;**

**year = 0;**

**nickName = “Default”;**

**mileage = 0;**

**}**

**Car::Car(string nMake, string nModel, int nYear, string nNickName, int nMileage)**

**{**

**make = nMake;**

**model = nModel;**

**year = nYear;**

**nickName = nNickName;**

**mileage = nMileage;**

**}**

**Car::~Car()**

**{**

**cout << “The car has been destroyed\n”;**

**}**

**void Car::setMake(string nMake)**

**{**

**make = nMake;**

**}**

**string Car::getMake()**

**{**

**return make;**

**}**

**void Car::setModel(string nModel)**

**{**

**model = nModel;**

**}**

**string Car::getModel()**

**{**

**return model;**

**}**

**void Car::setYear(int nYear)**

**{**

**year = nYear;**

**}**

**int Car::getYear()**

**{**

**return year;**

**}**

**void Car::setNickName(string nNickName)**

**{**

**nickName = nNickName;**

**}**

**string Car::getNickName()**

**{**

**return nickName;**

**}**

**void Car::setMileage(int nMileage)**

**{**

**mileage = nMileage;**

**}**

**int Car::getMileage()**

**{**

**return mileage;**

**}**

7. What will the following program print out?

int main()

{

Car a = new Car();

Car b = new Car(“Honda”, “Civic”, 2005, “Cujo”, 125000);

cout << a.getYear() << “ “ << a.getMake() << “ “ << a.getModel() << “\n”;

cout << b.getYear() << “ “ << b.getMake() << “ “ << b.getModel() << “ “ << b.getNickName() << “\n”;

b.setNickName(“Apple Pie”);

cout << b.getNickName() << “\n”;

return 0;

}

**0 Default Default**

**2005 Honda Civic Cujo**

**Apple Pie**

**The car has been destroyed**

**The car has been destroyed**

8. Using the header and cpp you wrote above, overload the output operator (“<<”) to output the following: Nickname, the Year Make Model has Mileage miles. ie. “Cujo, the 2005 Honda Civic has 125000 miles.”.

**ostream &operator<<(ostream &strm, const Car &obj)**

**{**

**strm << obj.nickName << ", the " << obj.year << " " << obj.make << " " << obj.model << " has " << obj.mileage << " miles.";**

**return strm;**

**}**

9. Define a static variable for called numCars to hold the total number of Cars instantiated. Each time you define a new car, this should be incremented in the constructor. When a car is destroyed, decrement this value in the destructor. Also define a static member function that will return the total number of Cars instantiated.

**class Car**

**{**

**private:**

**string make;**

**string model;**

**int year;**

**string nickName;**

**int mileage;**

**static int numCars;**

**public:**

**Car();**

**Car(string, string, int, string, int);**

**~Car();**

**void setMake(string nMake);**

**string getMake();**

**void setModel(string nModel);**

**string getModel();**

**void setYear(int nYear);**

**int getYear();**

**void setNickName(string nNickName);**

**string getNickName();**

**void setMileage(int nMileage);**

**int getMileage();**

**int getNumCars();**

**};**

**int Car::numCars = 0;**

**Car::Car()**

**{**

**make = “Default”;**

**model = “Default”;**

**year = 0;**

**nickName = “Default”;**

**mileage = 0;**

**numCars++;**

**}**

**Car::Car(string nMake, string nModel, int nYear, string nNickName, int nMileage)**

**{**

**make = nMake;**

**model = nModel;**

**year = nYear;**

**nickName = nNickName;**

**mileage = nMileage;**

**numCars++;**

**}**

**Car::~Car()**

**{**

**cout << “The car has been destroyed\n”;**

**numCars--;**

**}**

**static int Car::getNumCars()**

**{**

**return numCars;**

**}**

10. Given the following class declaration Distance and main, define a function called **func**. This function will be a friend to the Distance class. When called, **func** will set the private member meter to 5 and then return meter.

class Distance  
{  
private:  
 int meter;  
public:  
 Distance(): { meter = 0; }

**friend int func(Distance);**  
};

**int func(Distance d)  
{  
 d.meter=5;**

**return d.meter;**

**}**

int main()  
{  
 Distance D;

cout << "Distace: " << **func(D);**

return 0;  
}