AERSP 424: Advanced Computer Programming - Solution

Submission Instructions: Homework 4
Due: 2/12/19

Use the Robot.h and Robot.cpp files we have developing in class for this home. Specifically, add the following.

In Robot.h:

```
#ifndef ROBOT_H
#define ROBOT_H
class Robot {
   public:       //member functions
      Robot();

   private:      //data members

};
#endif
```

In Robot.cpp:

```
#include "Robot.h"

Robot::Robot() {

}
```

In main

```
#include <iostream>
#include "Robot.h"
using namespace std;

int main() {

   return 0;
}
```

1. Implement the following statements:
    a. Create two data members of type **`float`** called **`fJointAngleOne`** and **`fJointAngleTwo`** initialize these data members to 0.0 in the constructor.

in Robot.h

```
private:   //data members
      float fJointAngleOne;
      float fJointAngleTwo;
```

in Robot.cpp

```
      Robot::Robot() {
            fJointAngleOne =0.0;
            fJointAngleTwo =0.0;
      }
```

b. Create setters and getters for these variables.

in Robot.h

```
public:    //member functions
      Robot();
      float getJointAngleOne() const;
      float getJointAngleTwo() const;
      void setJointAngleOne( float );
      void setJointAngleTwo( float );
```

in Robot.cpp

```
float Robot::getJointAngleOne() const { return fJointAngleOne;    }
float Robot::getJointAngleTwo() const { return fJointAngleTwo;    }
void Robot::setJointAngleOne( float x){ fJointAngleOne =x;  }
void Robot::setJointAngleTwo( float y){ fJointAngleTwo=y;   }
```

c. Add member functions called **`IncrementJointAngles`** and **`DecrementJointAngles`** which increments or decrements both the robot's joint angles respectively.

in Robot.h

```
public:    //member functions
      Robot();
      void IncrementJointAngles();
      void DecrementJointAngles();
```

in Robot.cpp

```
      void Robot::IncrementJointAngles() {
            fJointAngleOne = fJointAngleOne +1;
            fJointAngleTwo = fJointAngleTwo +1;
      }
      void Robot::DecrementJointAngles () {
            fJointAngleOne = fJointAngleOne -1;
            fJointAngleTwo = fJointAngleTwo -1;
      }
```

d. Add a member function called **PolyInter** which sets the joint angles to the power of the parameter k, an **int** parameter for the member.

in Robot.h

```
public:    //member functions
      Robot();
      void PolyInter(int k);
```

in Robot.cpp

```
      void Robot::PolyInter(int k) {
            fJointAngleOne = pow(fJointAngleOne,k);
            fJointAngleTwo = pow(fJointAngleTwo,k);
      }
```

2. Implement the following statements (you can either begin from the robot you create in Q1 or start fresh with the robot described above Q1):
   a. Create a data member of type **const int** called **iIDNumber**.

in Robot.h

```
private:   //data members
      const int iIDNumber;
```

   b. Create a getter for this data member. The getter function for this member should be of type **const** ensuring that the getter cannot manipulate the object's data.

in Robot.h

```
public:    //member functions
      Robot();
      int getIDNumber() const;
```

in Robot.cpp

```
      int Robot::getIDNumber () const {
            return iIDNumber;     }
```

   c. Add a constructor which sets the **iIDNumber** variable to whatever number is passed to the constructor. In the constructor print the following message to the console: "Creating Robot: " and then the ID number.

in Robot.h

```
public:    //member functions
      Robot(int);
```

in Robot.cpp

```
      Robot::Robot(int s):iIDNumber(s) {
      cout<< "Creating robot "<< iIDNumber <<endl;
      };
```

3. Implement the following statements (begin from the robot you create in Q1):
   a. Add a data member to the **Robot** class of type **Robot*** named **ptrToTeamLeader.** Create a setter for the data member.

in Robot.h

```
private:   //data members
      Robot* ptrToTeamLeader;
```

in Robot.cpp

```
      void Robot::setTeamLeader(Robot* ldr) {
            ptrToTeamLeader=ldr;
      }
      Robot* Robot::getTeamLeader() {
            return ptrToTeamLeader; }
```

   b. Add a member function called **DisplayLeaderIDNumber** which prints the team leader's ID number to the console.

in Robot.cpp

```
      void Robot::DisplayLeaderIDNumber() {
            cout << ptrToTeamLeader->getIDNumber() << endl;
      }
```

   c. Create a destructor for the robot object. Inside the destructor print the message "Destroying robot " and then the robot's ID number.

in Robot.h

```
public:    //member function
~Robot() { cout<< "Destroying robot "<< iIDNumber <<endl; }
```

4. Implement the following statements (begin from the robot you create in Q1):
   a. In main create two robot objects with random ID numbers. Name these robots **FollowerOne** and **FollowerTwo.**

```
int main() {

   default_random_engine engine{
   static_cast<unsigned int> (time(0))};
     uniform_int_distribution<unsigned int> randomInt(1,1000);

   Robot FollowerOne(randomInt(engine));
   Robot FollowerTwo(randomInt(engine));

   Robot* ptrLeaderRobot = new Robot(randomInt(engine));

   return 0;
}
```

b. Create a third new robot with random ID using the **new** keyword. The pointer to this new robot should be named **ptrLeaderRobot.**

```
int main() {

      default_random_engine engine{
            static_cast<unsigned int> (time(0))};
              uniform_int_distribution<unsigned          int>
      randomInt(1,1000);

      Robot FollowerOne(randomInt(engine));
      Robot FollowerTwo(randomInt(engine));

      Robot* ptrLeaderRobot = Robot(randomInt(engine));


      return 0;
}
```

c. Set the follower robot's **ptrToTeamLeader** member to the dynamically created robot object named **ptrLeaderRobot**.

```
int main() {

      default_random_engine engine{
            static_cast<unsigned int> (time(0))};
              uniform_int_distribution<unsigned          int>
      randomInt(1,1000);

      Robot FollowerOne(randomInt(engine));
```

```
        Robot FollowerTwo(randomInt(engine));

        Robot* ptrLeaderRobot = Robot(randomInt(engine));

        FollowerOne->setTeamLeader(ptrLeaderRobot);
        FollowerTwo->setTeamLeader(ptrLeaderRobot);

        return 0;
}
```

d. Use the **FollowerOne** and **FollowerTwo** objects to get and print the follower's ID numbers to the console. Use the **FollowerOne** and **FollowerTwo** objects to get and print each object's leader ID number to the console.

```
int main() {

        default_random_engine engine{
              static_cast<unsigned int> (time(0))};
                uniform_int_distribution<unsigned                int>
        randomInt(1,1000);

        Robot robotOne(randomInt(engine));
        Robot robotTwo(randomInt(engine));

        Robot* ptrLeaderRobot = Robot(randomInt(engine));

        FollowerOne->setTeamLeader(ptrLeaderRobot);
        FollowerTwo->setTeamLeader(ptrLeaderRobot);

        cout << FollowerOne->getIDNumber() << endl;
        cout << FollowerTwo->getIDNumber() << endl;

cout << FollowerOne->getTeamLeader()->getIDNumber() << endl;
cout << FollowerTwo->getTeamLeader()->getIDNumber() << endl;

        return 0;
}
```

5. Identify and correct all errors in each of the following statements. Potential errors include compile errors, link errors, and logical errors.

a.

```
double ~Fawn();
```

```
~Fawn();
```

b.

```
class Bridge {
public:
     Bridge(int);
     Bridge(double) const;

private:
     int iAge;
     int iHeight;
     int iWeight;
};
```

```
class Bridge {
public:
     Bridge(int);
     Bridge(double);

private:
     int iAge;
     int iHeight;
     int iWeight;
};
```

c.

```
class Horse {
public:
     Horse(float);
     ~Horse(float);
     float getWeight() const;
     void setWeight(float t);

private:
     Saddle* ptrSaddle;
     float fWeight;
};
```

```
class Horse {
public:
     Horse(float);
     ~Horse();
     float getWeight() const;
     void setWeight(float t);

private:
```

```
        Saddle* ptrSaddle;
        float fWeight;
};
```

d.

```
class Tunnel {
        Tunnel()  { iLength=200; bClosed=false;}

private:
        int iLength;
        const bool bClosed;
};
```

```
class Tunnel {
        Tunnel(); //bClosed needs to be initialized in an
initialization list

private:
        int iLength;
        const bool bClosed;
};
```

e.

```
class BathTub {
public:
        BathTub();
        double getDepth() const { return dDepth; }
        void setDepth(double t) const { dDepth=t; }

private:
        double dDepth;
};
```

```
class BathTub {
public:
        BathTub();
        double getDepth() const { return dDepth; }
        void setDepth(double t) { dDepth=t; }

private:
        double dDepth;
};
```