AERSP 424: Advanced Computer Programming

Submission Instructions: Homework 5
Due: 2/26/19

Submission Instructions:

- Submit the file with the h/.cpp extension containing your C++ source code.
- Notes on submissions:
  - Submit single h/cpp file for questions 1-3
  - You can submit an cpp file or a pdf for question 4. NO HANDWRITTEN SUBMISSIONS.
  - Complete submissions should include an h/cpp file and a pdf.
- Teams of up to 3 are allowed. Put the names of all team members on a single submission

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 a data member of type **`static int`** called **`iNumberOfRobots.`** Create setters and getters for this variable.
    b. Initialize this data member to 0. Increment the variable inside the constructor.
    c. Add a destructor which decrements the variable.
    d. Create a robot in main. Create a pointer to a new dynamic robot in main. Show that **`iNumberOfRobots`** accurately keeps track of the number robots. Destroy the dynamic robot. Again show that **`iNumberOfRobots`** accurately keeps track of the number robots.

In Robot.h:

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

   private:      //data members
      static int iNumberOfRobots;
};
#endif
```

In Robot.cpp:

```
#include "Robot.h"
Robot::Robot() {
      iNumberOfRobots++;
}
Robot::~Robot() {
      iNumberOfRobots--;
}
int Robot::iNumberOfRobots =0;
int Robot::getNumberOfRobots() const  {
      return iNumberOfRobots;
}
```

In main

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

int main() {
   Robot aRobot;
   Robot* pRobot = new Robot();
   cout << aRobot.getNumberOfRobots() << endl;
   delete pRobot;
   cout << aRobot.getNumberOfRobots() << endl;

   return 0;
}
```

2. Implement the following statements (you should begin from the robot you create in Q1):
    a. Create a data member of type **`const float`** called **`fLength`**. 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. Add a constructor which sets the **`fLength`** variable and an error bound indicating the amount of difference in length for two lengths to be considered different.
    b. Overload the **`==`** operator returning true if the robot lengths.
    c. Overload the **`>`** operator returning true if the robot's length is greater than the right hand length.
    d. Overload the **`-`** operator returning a robot with length that is the difference of two robots.
    e. Overload the **`!=`** operator returning a Boolean indicating whether or not the lengths are the same.

In Robot.h:

```
#ifndef ROBOT_H
#define ROBOT_H
class Robot {
   public:       //member functions
      Robot();
      Robot(float,float);
      ~Robot();
      int getNumberOfRobots() const;
      int getLength() const;
      bool operator==(const Robot &);
      bool operator>(const Robot &);
      Robot operator-(const Robot &);
      bool operator!=(const Robot &);

   private:      //data members
      static int iNumberOfRobots;
```

```
        const float fLength;
        float fError;
};
#endif
```

In Robot.cpp:

```
#include "Robot.h"
Robot::Robot() {
        iNumberOfRobots++;
}
Robot::Robot(float f, float e):fLength(f)   {
        iNumberOfRobots++;
        fError=e;
}
Robot::~Robot() {
        iNumberOfRobots--;
}
int Robot::iNumberOfRobots =0;
int Robot::getNumberOfRobots() const  {
        return iNumberOfRobots;
}
float Robot::getLength() const {
        return fLength;
}
bool Robot::operator==(const Robot& bot) {
        if(fabs(this->fLength - bot.getLength())< fError)
                return true;
        else
                return false;
}
bool Robot::operator>(const Robot & bot) {
        if(((this->fLength - bot.getLength()) > fError)
                return true;
        else
                return false;
}
Robot Robot::operator-(const Robot & bot) {

        return Robot(this->fLength - bot.fLength, fError);
}
bool Robot::operator!=( const Robot& bot) const {
        return fabs((this->fLngth - bot.fLength)>= fError);
}
```

In main

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

```
int main() {
   Robot aRobot;
   Robot* pRobot = new Robot();
   cout << aRobot.getNumberOfRobots() << endl;
   return 0;
}
```

3. Implement the following statements (you should begin from the robot you create in Q2):
    a. Overload the **=** operator returning a robot type object.
    b. Add a data member of type **`float*`** called **`ptrHistory`**. Add setter and getter functions for **`ptrHistory.`** In main use the new key word to create an array of 5 floats. Set the value of the floats to 0 in the constructor.
    c. Create a deep copy constructor for the robot class. Create a robot and set its history to 4.0. Constructing another robot using the copy constructor. For each robot print BOTH the value and address for each value of the **`ptrHistory`** variable. Are the addresses and/or values different? Why or why not? Did a memory leak result?
    d. Create a shallow copy constructor for the robot. Create a robot and set its history name to 2.0. Constructing another robot using the copy constructor. For each robot print BOTH the value and address for each value of the **`ptrHistory`** variable. Are the addresses and/or values different? Why or why not? Did a memory leak result? (You will need to comment out the deep constructor to compile the shallow constructor).
    e. Add a destructor which prevents memory leaks.

In Robot.h:

```
#ifndef ROBOT_H
#define ROBOT_H

#include <string>
using namespace std;

class Robot {
   public:   //member functions
      Robot();
      Robot(float,float);
      ~Robot();
      int getNumberOfRobots() const;
      float* getHistory() const;
      void setHistory( float* );
      int getLength() const;
      bool operator==(const Robot &);
      bool operator>(const Robot &);
      Robot operator-(const Robot &);
```

```
        bool operator!=(const Robot &) const;

    private:  //data members
        static int iNumberOfRobots;
        const float fLength;
        float fError;
};
#endif
```

In Robot.cpp:

```
#include "Robot.h"
Robot::Robot() {
   iNumberOfRobots++;
   ptrHistory=new float[5]();
}
//SHALLOW
Robot::Robot(const Robot& bot) {
   this->iNumberOfRobots = bot.getNumberOfRobots();
   ptrHistory = bot.getHistory();
}

/*
//DEEP
Robot::Robot(const Robot& bot) {
   this->iNumberOfRobots = bot.getNumberOfRobots();
   ptrHistory = new float[ 5 ](); //allocate memory
   float* fPtr = bot.getHistory();
   for(int i=0;i<5;i++)
      *(ptrHistory+i)=*(fPtr+i);
}
Robot::Robot(float f, float e):fLength(f)    {
   iNumberOfRobots++;
   fError=e;
   ptrHistory=new float[5]();
}
Robot::~Robot() {
   iNumberOfRobots--;
   delete[] ptrHistory;
}
int Robot::iNumberOfRobots =0;
int Robot::getNumberOfRobots() const    {
   return iNumberOfRobots;
}
float Robot::getLength() const {
   return fLength;
}
```

```
float* Robot::getHistory() const {
   return ptrHistory;
}
void Robot::setHistory(float* ptr) {
   if(ptrHistory) {
      delete[] ptrHistory;
   }
   ptrHistory=new float[5]();
   for(int i=0; i<5;i++) {
      *(prtHistory+i)=*(ptr+i);
   }
}
bool Robot::operator==(const Robot& bot) {
   if(fabs(this->fLength - bot.getLength())< fError)
      return true;
   else
      return false;
}
bool Robot::operator>(const Robot & bot) {
   if(((this->fLength - bot.getLength()) > fError)
      return true;
   else
      return false;
}
Robot Robot::operator-(const Robot & bot) {

   return Robot(this->fLength - bot.fLength, fError);
}
bool Robot::operator!=( const Robot& bot) const {
   return fabs(this->fLngth - bot.fLength)>= fError;
}
```

In main

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


int main()
{
      float* deepRobotHistory = new float[5]();
      for(int i=0;i<5;i++)
            *(deepRobotHistory+i)=4;

      Robot aRobot(10,0.1);
      aRobot.setHistory(deepRobotHistory);
```

```
    Robot newRobot(aRobot);

    float* fPtr = aRobot.getHistory();

    cout << "Value: "<<*fPtr <<" Address: "<<fPtr << endl;
    cout << "Value: "<<*(fPtr+1) <<" Address: "<<(fPtr+1) << endl;
    cout << "Value: "<<*(fPtr+2) <<" Address: "<<(fPtr+2) << endl;
    cout << "Value: "<<*(fPtr+3) <<" Address: "<<(fPtr+3) << endl;
    cout << "Value: "<<*(fPtr+4) <<" Address: "<<(fPtr+4) << endl;

    float* fPtr2 = newRobot.getHistory();
    cout << "New Robot's copy" <<endl;
    cout << "Value: "<<*fPtr2 <<" Address: "<<fPtr2 << endl;
    cout << "Value: "<<*(fPtr2+1) <<" Address: "<<(fPtr2+1) << endl;
    cout << "Value: "<<*(fPtr2+2) <<" Address: "<<(fPtr2+2) << endl;
    cout << "Value: "<<*(fPtr2+3) <<" Address: "<<(fPtr2+3) << endl;
    cout << "Value: "<<*(fPtr2+4) <<" Address: "<<(fPtr2+4) << endl;
    return 0;
}
```

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

   a.

```
Cat operator==(const Cat&);

bool operator==(const Cat&);
```

   b.

```
const Dog & Dog::Dog(const Dog& rht) {

    this->iPaws = rht.getPaws();
    this->iTeeth = rht.getTeeth();

    return *this;
}
Dog::Dog(const Dog& rht) {

    this->iPaws = rht.getPaws();
    this->iTeeth = rht.getTeeth();

}

No return on a constructor.
```

c.

```
bool Dog::operator?(const Dog& rht) {

   if(this->iPaws == rht.getPaws())
     return true;
   else
     return false;
}
```

The symbol ? cannot be overloaded.