CS 221 C++ Advanced Evaluation Assignment

Lily Larsen

December 12, 2022

1.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-8/1.cpp

```cpp
#include <algorithm>
#include <cmath>
#include <memory>
#include <string>
#include <tuple>
#include <vector>
using namespace std;

// Previous homework
class Car {
private:
  double x;
  double y;
  double milesPerGallon;
  double fuelTankCapacityGallons;
  double fuelGallons;

protected:
  void setX(double toX) { x = toX; }
  void setY(double toY) { y = toY; }
  void setFuelGallons(double gallons) { fuelGallons = gallons; }
  void burnGallons(double gallons) { fuelGallons -= gallons; }

public:
  Car(double x, double y, float milesPerGallon, double fuelTankCapacityGallons,
      double fuelGallons)
      : x(x), y(y), milesPerGallon(milesPerGallon),
        fuelTankCapacityGallons(fuelTankCapacityGallons) {}
  double getMilesPerGallon() { return milesPerGallon; }
  double getFuelTankCapacityGallons() { return fuelTankCapacityGallons; }
  double getFuelGallons() { return fuelGallons; }
  double getX() { return x; }
  double getY() { return y; }
  bool moveTo(double destinationX, double destinationY) {
    double gallonCost = hypot(getX() - destinationX, getY() - destinationY) *
                        getMilesPerGallon();
    if (gallonCost > getFuelGallons()) {
      return false;
    }

    burnGallons(gallonCost);
    setX(destinationX);
    setY(destinationY);
    return true;
  }
  double refillTank() {
    double refilledGallons = fuelTankCapacityGallons - fuelGallons;
    return refilledGallons;
  }
};

vector<Car> moveToPoint(vector<Car> cars, double destinationX,
                        double destinationY) {
```

```
  vector<Car> result;
  for (Car car : cars) {
    if (car.moveTo(destinationX, destinationY)) {
      result.push_back(car);
    }
  }

  return result; // Return array with size and a lot of other helpful things
                 // like god (the C++ standard library) intended us to do. This
                 // helpful construct is called a vector. I know its not an
                 // array. I know its not an array. I know its not an array. I
}

class GasStation {
private:
  double x;
  double y;
  double pricePerGallon;

public:
  GasStation(double x, double y, double pricePerGallon)
      : x(x), y(y), pricePerGallon(pricePerGallon) {}
  double getX() { return x; }
  double getY() { return y; }
  double getPricePerGallon() { return pricePerGallon; }
};

tuple<vector<Car>, vector<double>>
moveToPointGasStations(vector<Car> cars, vector<GasStation> gasStations,
                       double destinationX, double destinationY) {
  vector<Car> resultCar;
  vector<double> resultCost;

  for (Car car : cars) {
    double cost = 0;

    for (int i = gasStations.size(); i > 0; i--) {
      if (car.moveTo(destinationX, destinationY)) {
        resultCar.push_back(car);
        resultCost.push_back(cost);
        continue;
      }

      GasStation gasStation = gasStations.at(i);
      if (car.moveTo(gasStation.getX(), gasStation.getY())) {
        cost += car.refillTank() * gasStation.getPricePerGallon();
      } else {
        continue;
      }
    }
  }

  return {resultCar, resultCost};
}

// Homework 8
class Bus : public Car {
```

```
111 private:
112   int maxPassengers;
113   vector<int> passengers;
114   double milesPerGallonPenaltyPerPassenger;
115
116 public:
117   double getMilesPerGallon() {
118     return Car::getMilesPerGallon() -
119            passengers.size() * milesPerGallonPenaltyPerPassenger;
120   }
121
122   bool moveTo(double destinationX, double destinationY) {
123     double gallonCost = hypot(getX() - destinationX, getY() - destinationY) *
124                         getMilesPerGallon();
125     if (gallonCost > getFuelGallons()) {
126       return false;
127     }
128
129     burnGallons(gallonCost);
130     setX(destinationX);
131     setY(destinationY);
132     return true;
133   }
134
135   vector<int> generateRoute() {
136     vector<int>::iterator it = unique(passengers.begin(), passengers.end());
137     vector<int> results = passengers;
138     results.erase(it);
139
140     return results;
141   }
142 };
143
144 class Provider {
145 private:
146   string name;
147   string role;
148 };
149
150 class Patient {
151 private:
152   string name;
153   string condition;
154 };
155
156 class MedicalCenter {
157 private:
158   vector<Provider> providers;
159   vector<Patient> patients;
160
161 public:
162   vector<Patient> getPatients() { return patients; }
163
164   vector<Provider> getProviders() { return providers; }
165 };
166
167 class Ambulance : Car, MedicalCenter {
```

```
168 private:
169   int patientCapacity;
170   int providerCapacity;
171   double gasMileagePenalty;
172
173 public:
174   double getMilesPerGallon() {
175     return Car::getMilesPerGallon() -
176            getPatients().size() * getProviders().size() * gasMileagePenalty;
177   }
178
179   bool moveTo(double destinationX, double destinationY) {
180     double gallonCost = hypot(getX() - destinationX, getY() - destinationY) *
181                         getMilesPerGallon();
182     if (gallonCost > getFuelGallons()) {
183       return false;
184     }
185
186     burnGallons(gallonCost);
187     setX(destinationX);
188     setY(destinationY);
189     return true;
190   }
191 };
192
193 vector<Car> moveToPoint(vector<unique_ptr<Car>> cars, double destinationX,
194                         double destinationY) {
195   vector<Car> result;
196   for (auto &car : cars) {
197     if ((*car).moveTo(destinationX, destinationY)) {
198       result.push_back(*car);
199     }
200   }
201
202   return result; // Return array with size and a lot of other helpful things
203                  // like god (the C++ standard library) intended us to do. This
204                  // helpful construct is called a vector. I know its not an
205                  // array. I know its not an array. I know its not an array. I
206 }
207
208 int main() { return 0; }
```

2.

https://github.com/Id405/cs-221-eval-assignments/blob/main/eval-8/2.cpp

```cpp
#include <iostream>
#include <memory>
#include <vector>
using namespace std;

template <typename T> class BSTNode {
private:
  unique_ptr<BSTNode<T>> left;
  unique_ptr<BSTNode<T>> right;
  unique_ptr<T> data;

public:
  BSTNode(T value) {
    data = make_unique<T>(value);
    left = nullptr;
    right = nullptr;
  }

  ~BSTNode() {
    left.reset();
    right.reset();
    data.reset();
  }

  T getData() { return data; }

  BSTNode<T> getLeft() { return *left; }

  BSTNode<T> getRight() { return *right; }

  bool leftIsNull() { return *left == nullptr; }
  bool rightIsNull() { return right == nullptr; }

  void resetLeft() { left.reset(); }
  void resetRight() { right.reset(); }

  void releaseLeft() { left.release(); }
  void relesaeRight() { right.release(); }

  BSTNode<T> setLeft(BSTNode<T> node) {
    left.reset();
    left = make_unique<T>(node);
  }

  BSTNode<T> setRight(BSTNode<T> node) {
    right.reset();
    right = make_unique<T>(node);
  }
};

template <typename T> class BST {
private:
  unique_ptr<BSTNode<T>> trunk;
```

```
54    int size;
55
56  public:
57    BST(vector<T> values) {
58      for (T value : values) {
59        insert(value);
60      }
61    }
62
63    ~BST() { trunk.reset(); }
64
65    void insert(T value) {
66      unique_ptr<BSTNode<T>> node = make_unique<T>(BSTNode<T>(value));
67
68      if (trunk == nullptr) {
69        trunk = node;
70        size++;
71        return;
72      }
73
74      BSTNode<T> currentNode = *trunk;
75
76      bool side = value < currentNode.getData();
77
78      while (side ? !currentNode.leftIsNull() : !currentNode.rightIsNull()) {
79        currentNode = side ? currentNode.left() : currentNode.right();
80        side = value < currentNode.getData();
81      }
82
83      if (side) {
84        currentNode.setLeft(node);
85      } else {
86        currentNode.setRight(node);
87      }
88
89      size++;
90      return;
91    }
92
93    void insertNode(T node) {
94      T value = node.getData();
95
96      if (trunk == nullptr) {
97        trunk = node;
98        size++;
99        return;
100     }
101
102     BSTNode<T> currentNode = *trunk;
103
104     bool side = value < currentNode.getData();
105
106     while (side ? !currentNode.leftIsNull() : !currentNode.rightIsNull()) {
107       currentNode = side ? currentNode.left() : currentNode.right();
108       side = value < currentNode.getData();
109     }
110
```

```
111       if (side) {
112         currentNode.setLeft(node);
113       } else {
114         currentNode.setRight(node);
115       }
116
117       size++;
118       return;
119     }
120
121     void remove(T value) {
122       if (trunk == nullptr) {
123         return;
124       }
125
126       bool currentNodeIsRoot = true;
127       BSTNode<T> currentNodeBranch = *trunk;
128       BSTNode<T> currentNode = *trunk;
129
130       bool side = value < currentNode.getData();
131
132       while (currentNode.getData() != value) {
133         currentNodeBranch = currentNode;
134         currentNodeIsRoot = false;
135
136         side = value < currentNode.getData();
137         if (side ? currentNode.leftIsNull() : currentNode.rightIsNull()) {
138           return;
139         }
140
141         currentNode = side ? currentNode.left() : currentNode.right();
142       }
143
144       if (currentNode.leftIsNull()) {
145         if (currentNode.rightIsNull()) {
146           if (currentNodeIsRoot) {
147             trunk.reset();
148           } else {
149             if (side) {
150               currentNodeBranch.resetLeft();
151             } else {
152               currentNodeBranch.resetRight();
153             }
154           }
155         } else {
156           if (currentNodeIsRoot) {
157             BSTNode<T> *newTrunk = trunk.releaseRight();
158             delete trunk;
159             trunk = make_unique<T>(newTrunk);
160           } else {
161             BSTNode<T> *node;
162             if (side) {
163               node = currentNodeBranch.releaseLeft();
164             } else {
165               node = currentNodeBranch.releaseLeft();
166             }
167
```

```
168             currentNodeBranch.setRight((*node).getRight());
169             delete node;
170           }
171         }
172       } else {
173         if (currentNodeIsRoot) {
174           BSTNode<T> *newTrunk = trunk.releaseLeft();
175           BSTNode<T> right = *trunk.releaseRight();
176           delete trunk;
177           trunk = make_unique<T>(newTrunk);
178           insert(right);
179         } else {
180           BSTNode<T> *node;
181           if (side) {
182             node = currentNodeBranch.releaseLeft();
183           } else {
184             node = currentNodeBranch.releaseLeft();
185           }
186
187           currentNodeBranch.setRight((*node).getRight());
188           insertNode((*node).releaseRight());
189           delete node;
190         }
191       }
192
193       return;
194     }
195   };
196
197   int main() {
198     return 0;
199   }
```