Samir Khadka
CS360 - Programming in C and C++
Homework Assignment #6

Question 1:

Question1.cpp

Question1.cpp > ...

Format

```cpp
#include <iostream>
#include <string>
using namespace std;

// Class representing a date
class dateType {
public:
    dateType(int m = 1, int d = 1, int y = 2000); // Constructor with default values

    // Getters for month, day, and year
    int getMonth() const;
    int getDay() const;
    int getYear() const;

    // Function to check if a year is a leap year
    bool isLeapYear(int year);

    // Function to set the date with validation
    void setDate(int m, int d, int y);

private:
    int month;
    int day;
    int year;
};

// Constructor definition
dateType::dateType(int m, int d, int y) {
    setDate(m, d, y);
}

// Getter definitions
int dateType::getMonth() const {
    return month;
}

int dateType::getDay() const {
    return day;
}
```

Question1.cpp

Question1.cpp > ...

Format

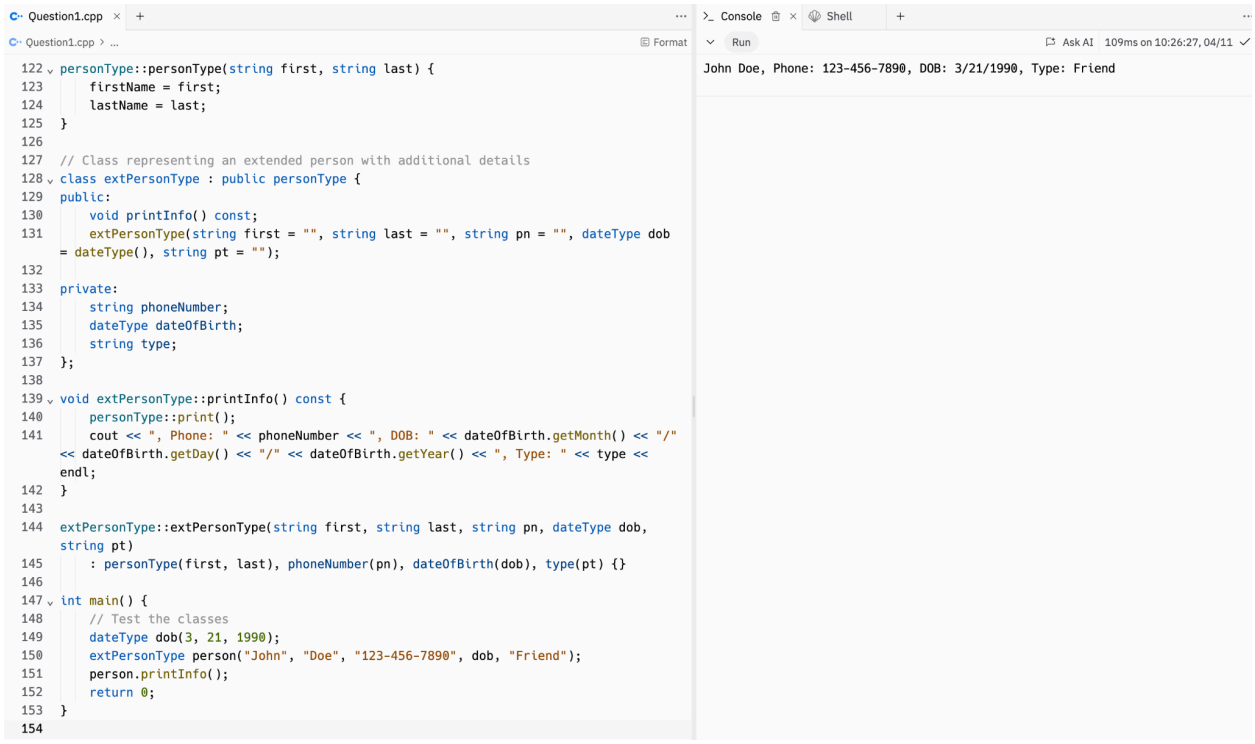
```cpp
// Function to check if a year is a leap year
bool dateType::isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

// Function to set the date with validation
void dateType::setDate(int m, int d, int y) {
    if (m < 1 || m > 12 || d < 1 || d > 31 || y < 0) {
        cerr << "Invalid date. Setting to default date (1/1/2000)." << endl;
        month = 1;
        day = 1;
        year = 2000;
        return;
    }

    // Validate day based on month
    if ((m == 4 || m == 6 || m == 9 || m == 11) && (d > 30)) {
        cerr << "Invalid date for the given month. Setting to default date (1/1/2000)." << endl;
        month = 1;
        day = 1;
        year = 2000;
        return;
    }

    if (m == 2) {
        if (isLeapYear(y) && d > 29) {
            cerr << "Invalid date for February in a leap year. Setting to default date (1/1/2000)." << endl;
            month = 1;
            day = 1;
            year = 2000;
            return;
        } else if (!isLeapYear(y) && d > 28) {
            cerr << "Invalid date for February in a non-leap year. Setting to default date (1/1/2000)." << endl;
            month = 1;
            day = 1;
            year = 2000;
            return;
```

Question1.cpp

Question1.cpp > ...

Format

```
82          }
83      }
84
85      // Date is valid, set the date
86      month = m;
87      day = d;
88      year = y;
89  }
90
91  // Class representing a person
92  class personType {
93  public:
94      void print() const;
95      void setName(string first, string last);
96      string getFirstName() const;
97      string getLastName() const;
98      personType(string first = "", string last = "");
99
100 private:
101     string firstName;
102     string lastName;
103 };
104
105 void personType::print() const {
106     cout << firstName << " " << lastName;
107 }
108
109 void personType::setName(string first, string last) {
110     firstName = first;
111     lastName = last;
112 }
113
114 string personType::getFirstName() const {
115     return firstName;
116 }
117
118 string personType::getLastName() const {
119     return lastName;
120 }
121
```



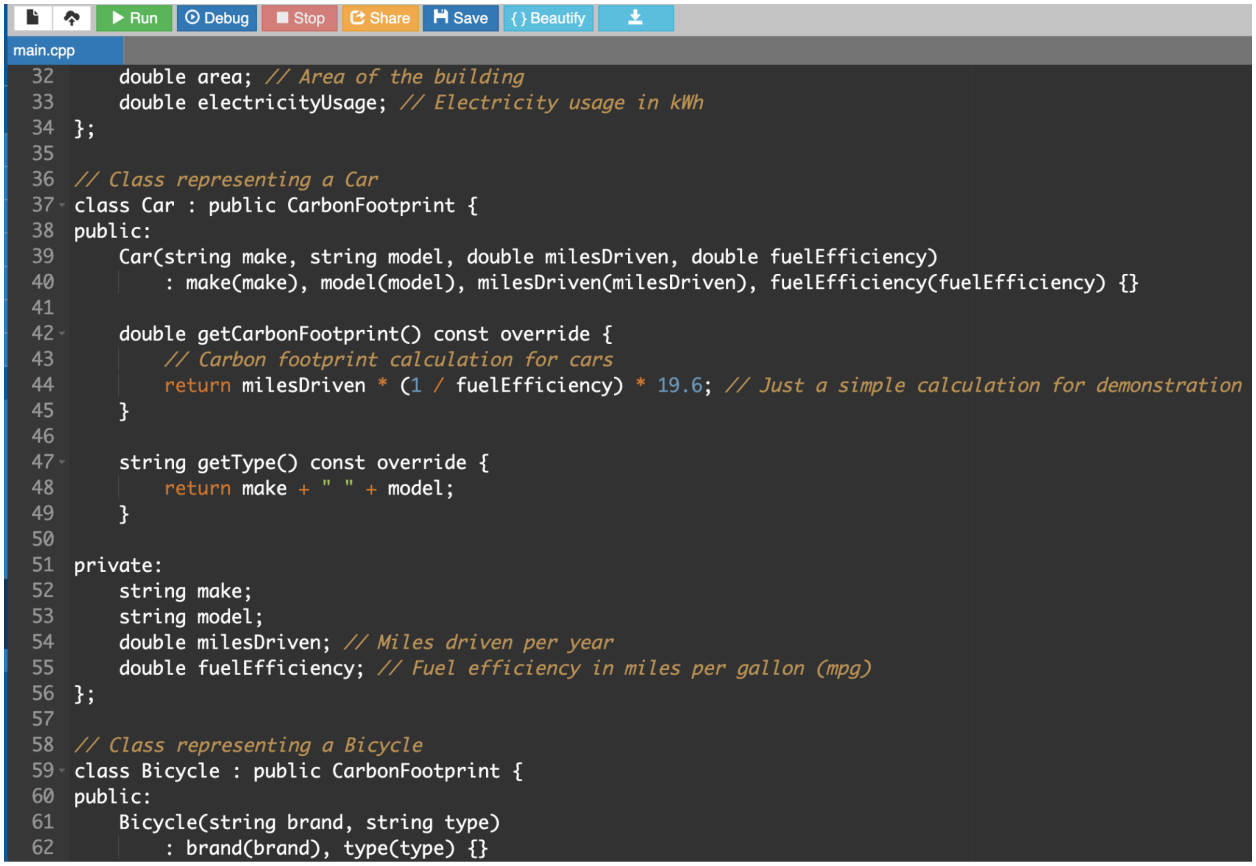

```cpp
personType::personType(string first, string last) {
    firstName = first;
    lastName = last;
}

// Class representing an extended person with additional details
class extPersonType : public personType {
public:
    void printInfo() const;
    extPersonType(string first = "", string last = "", string pn = "", dateType dob = dateType(), string pt = "");

private:
    string phoneNumber;
    dateType dateOfBirth;
    string type;
};

void extPersonType::printInfo() const {
    personType::print();
    cout << ", Phone: " << phoneNumber << ", DOB: " << dateOfBirth.getMonth() << "/" << dateOfBirth.getDay() << "/" << dateOfBirth.getYear() << ", Type: " << type << endl;
}

extPersonType::extPersonType(string first, string last, string pn, dateType dob, string pt)
    : personType(first, last), phoneNumber(pn), dateOfBirth(dob), type(pt) {}

int main() {
    // Test the classes
    dateType dob(3, 21, 1990);
    extPersonType person("John", "Doe", "123-456-7890", dob, "Friend");
    person.printInfo();
    return 0;
}
```

Question 2:

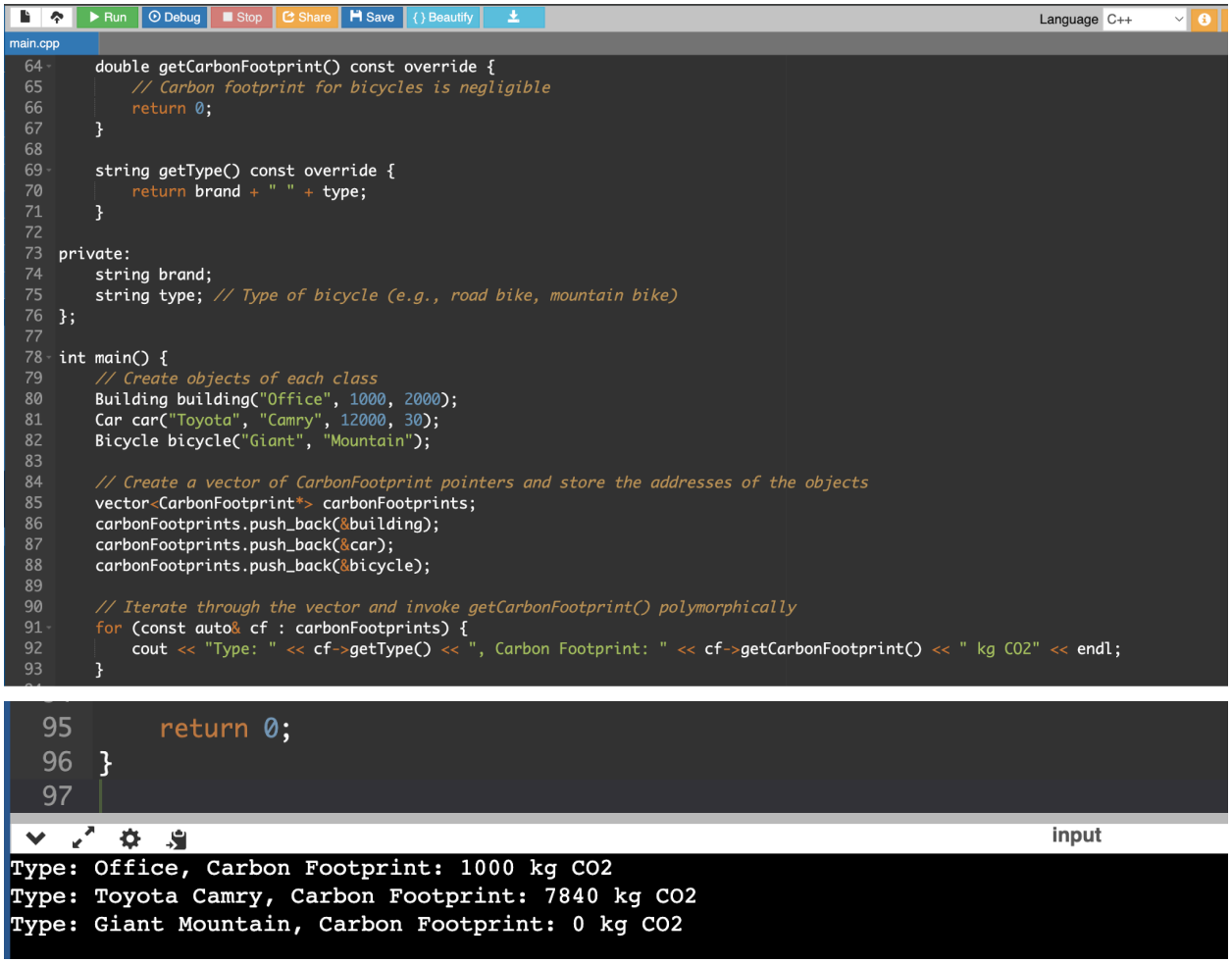
Run | Debug | Stop | Share | Save | { } Beautify

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <cmath>

using namespace std;

// Abstract class representing a CarbonFootprint
class CarbonFootprint {
public:
    virtual double getCarbonFootprint() const = 0; // Pure virtual function
    virtual string getType() const = 0; // Pure virtual function to get type
};

// Class representing a Building
class Building : public CarbonFootprint {
public:
    Building(string type, double area, double electricityUsage)
        : type(type), area(area), electricityUsage(electricityUsage) {}

    double getCarbonFootprint() const override {
        // Carbon footprint calculation for buildings
        return electricityUsage * 0.5; // Just a simple calculation for demonstration
    }

    string getType() const override {
        return type;
    }

private:
    string type;
```




```cpp
    double area; // Area of the building
    double electricityUsage; // Electricity usage in kWh
};

// Class representing a Car
class Car : public CarbonFootprint {
public:
    Car(string make, string model, double milesDriven, double fuelEfficiency)
        : make(make), model(model), milesDriven(milesDriven), fuelEfficiency(fuelEfficiency) {}

    double getCarbonFootprint() const override {
        // Carbon footprint calculation for cars
        return milesDriven * (1 / fuelEfficiency) * 19.6; // Just a simple calculation for demonstration
    }

    string getType() const override {
        return make + " " + model;
    }

private:
    string make;
    string model;
    double milesDriven; // Miles driven per year
    double fuelEfficiency; // Fuel efficiency in miles per gallon (mpg)
};

// Class representing a Bicycle
class Bicycle : public CarbonFootprint {
public:
    Bicycle(string brand, string type)
        : brand(brand), type(type) {}
```


Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
double getCarbonFootprint() const override {
// Carbon footprint for bicycles is negligible
return 0;
}
string getType() const override {
return brand + " " + type;
}
private:
string brand;
string type; // Type of bicycle (e.g., road bike, mountain bike)
};
int main() {
// Create objects of each class
Building building("Office", 1000, 2000);
Car car("Toyota", "Camry", 12000, 30);
Bicycle bicycle("Giant", "Mountain");
// Create a vector of CarbonFootprint pointers and store the addresses of the objects
vector<CarbonFootprint*> carbonFootprints;
carbonFootprints.push_back(&building);
carbonFootprints.push_back(&car);
carbonFootprints.push_back(&bicycle);
// Iterate through the vector and invoke getCarbonFootprint() polymorphically
for (const auto& cf : carbonFootprints) {
cout << "Type: " << cf->getType() << ", Carbon Footprint: " << cf->getCarbonFootprint() << " kg CO2" << endl;
}
return 0;
}
input
Type: Office, Carbon Footprint: 1000 kg CO2
Type: Toyota Camry, Carbon Footprint: 7840 kg CO2
Type: Giant Mountain, Carbon Footprint: 0 kg CO2