Question no. 1

→ **#include <algorithm>**

**#include <fstream>**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**#include <vector>**

**// a. Class to handle date**

**class dateType {**

**int day, month, year;**

**public:**

**dateType(int d = 1, int m = 1, int y = 1900) : day(d), month(m), year(y) {**

**if (!isValidDate()) {**

**day = 1;**

**month = 1;**

**year = 1900;**

**}**

**}**

**void setDate(int d, int m, int y) {**

**day = d;**

**month = m;**

**year = y;**

**if (!isValidDate()) {**

**day = 1;**

**month = 1;**

**year = 1900;**

**}**

**}**

**bool isLeapYear() {**

**return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);**

**}**

**bool isValidDate() {**

**if (month < 1 || month > 12 || day < 1 || day > 31 || year < 1900)**

**return false;**

**if (month == 2)**

**return day <= (isLeapYear() ? 29 : 28);**

**if (month == 4 || month == 6 || month == 9 || month == 11)**

**return day <= 30;**

**return true;**

**}**

**int getMonth() const { return month; }**

**};**

**// b. Class to handle address**

**class addressType {**

**std::string streetAddress, city, state, zipCode;**

**public:**

**addressType(std::string sa = "", std::string c = "", std::string s = "",**

**std::string zc = "")**

**: streetAddress(sa), city(c), state(s), zipCode(zc) {}**

**void printAddress() {**

**std::cout << streetAddress << ", " << city << ", " << state << " "**

**<< zipCode << std::endl;**

**}**

**};**

**// c. Class to handle extended person type**

**class extPersonType {**

**std::string firstName, lastName, phoneNumber, personType;**

**dateType dob;**

**addressType address;**

**public:**

**extPersonType(std::string fn = "", std::string ln = "", std::string pn = "",**

**std::string pt = "", dateType dob = dateType(),**

**addressType address = addressType())**

**: firstName(fn), lastName(ln), phoneNumber(pn), personType(pt), dob(dob),**

**address(address) {}**

**void printPerson() {**

**std::cout << firstName << " " << lastName << ", " << phoneNumber << ", "**

**<< personType << std::endl;**

**address.printAddress();**

**}**

**std::string getLastName() const { return lastName; }**

**int getBirthMonth() const { return dob.getMonth(); }**

**std::string getFirstName() const { return firstName; }**

**std::string getPersonType() const { return personType; }**

**};**

**// d. Class to handle address book**

**class addressBookType {**

**std::vector<extPersonType> addressBook;**

**public:**

**void addPerson(const extPersonType &person) {**

**if (addressBook.size() < 500)**

**addressBook.push\_back(person);**

**}**

**void sortByName() {**

**std::sort(addressBook.begin(), addressBook.end(),**

**[](const extPersonType &a, const extPersonType &b) {**

**return a.getLastName() < b.getLastName();**

**});**

**}**

**extPersonType \*searchByLastName(const std::string &lastName) {**

**for (auto &person : addressBook) {**

**if (person.getLastName() == lastName)**

**return &person;**

**}**

**return nullptr;**

**}**

**void printPersonDetails(const std::string &lastName) {**

**extPersonType \*person = searchByLastName(lastName);**

**if (person != nullptr)**

**person->printPerson();**

**}**

**void printBirthdaysInMonth(int month) {**

**for (auto &person : addressBook) {**

**if (person.getBirthMonth() == month)**

**std::cout << person.getFirstName() << " " << person.getLastName()**

**<< std::endl;**

**}**

**}**

**void printNamesBetween(const std::string &lastName1,**

**const std::string &lastName2) {**

**for (auto &person : addressBook) {**

**if (person.getLastName() >= lastName1 &&**

**person.getLastName() <= lastName2)**

**std::cout << person.getFirstName() << " " << person.getLastName()**

**<< std::endl;**

**}**

**}**

**void printByPersonType(const std::string &type) {**

**for (auto &person : addressBook) {**

**if (person.getPersonType() == type)**

**person.printPerson();**

**}**

**}**

**void printAll() {**

**for (auto &person : addressBook) {**

**person.printPerson();**

**}**

**}**

**};**

**int main() {**

**// Create some dateType objects**

**dateType dob1(1, 1, 2000);**

**dateType dob2(2, 2, 2002);**

**dateType dob3(3, 3, 2003);**

**// Create some addressType objects**

**addressType address1("123 Main St", "Anytown", "Anystate", "12345");**

**addressType address2("456 Maple Ave", "Sometown", "Somestate", "67890");**

**addressType address3("789 Oak Dr", "Othertown", "Otherstate", "11223");**

**// Create some extPersonType objects**

**extPersonType person1("John", "Doe", "111-111-1111", "Family", dob1,**

**address1);**

**extPersonType person2("Jane", "Doe", "222-222-2222", "Friend", dob2,**

**address2);**

**extPersonType person3("Jim", "Smith", "333-333-3333", "Business", dob3,**

**address3);**

**// Create an addressBookType object**

**addressBookType addressBook;**

**// Add the extPersonType objects to the addressBookType object**

**addressBook.addPerson(person1);**

**addressBook.addPerson(person2);**

**addressBook.addPerson(person3);**

**// Print the details of the people in the address book before sorting**

**std::cout << "Address Book Entries Before Sorting:\n";**

**addressBook.printAll();**

**std::cout << std::endl;**

**// Sort the address book by last name**

**addressBook.sortByName();**

**// Print the details of the people in the address book after sorting**

**std::cout << "Address Book Entries After Sorting:\n";**

**addressBook.printAll();**

**std::cout << std::endl;**

**// Search for a person by last name "Doe"**

**std::cout << "Searching for last name 'Doe':\n";**

**extPersonType \*person = addressBook.searchByLastName("Doe");**

**if (person != nullptr)**

**person->printPerson();**

**else**

**std::cout << "Person not found.\n";**

**std::cout << std::endl;**

**// Print the address and phone number of a person**

**std::cout << "Details for 'Doe':\n";**

**addressBook.printPersonDetails("Doe");**

**std::cout << std::endl;**

**// Print the names of people whose birthdays are in January**

**std::cout << "Birthdays in January:\n";**

**addressBook.printBirthdaysInMonth(1);**

**std::cout << std::endl;**

**// Print the names of all people between "Doe" and "Smith"**

**std::cout << "Names between 'Doe' and 'Smith':\n";**

**addressBook.printNamesBetween("Doe", "Smith");**

**std::cout << std::endl;**

**// Print the names of all family members**

**std::cout << "All Family Members:\n";**

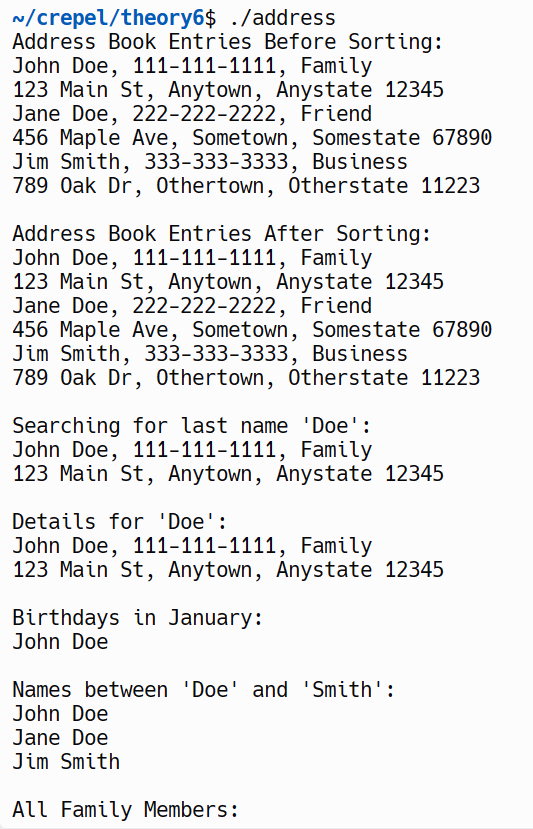
**addressBook.printByPersonType("Family");**

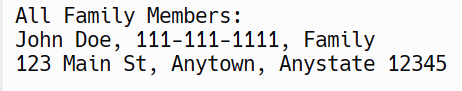
**std::cout << std::endl;**

**return 0;**

**}**

Output:





**Question no.2**

→ **#include <iostream>**

**#include <vector>**

**// Abstract class**

**class CarbonFootprint {**

**public:**

**virtual double getCarbonFootprint() const = 0; // Pure virtual function**

**};**

**class Building : public CarbonFootprint {**

**private:**

**double electricityUsed; // in kWh**

**public:**

**Building(double elec) : electricityUsed(elec) {}**

**double getCarbonFootprint() const override {**

**return electricityUsed \* 0.92; // Assuming 0.92kg CO2 emitted per kWh**

**}**

**};**

**class Car : public CarbonFootprint {**

**private:**

**double fuelBurned; // in liters**

**public:**

**Car(double fuel) : fuelBurned(fuel) {}**

**double getCarbonFootprint() const override {**

**return fuelBurned \* 2.3; // Assuming 2.3kg CO2 emitted per liter of petrol**

**}**

**};**

**class Bicycle : public CarbonFootprint {**

**public:**

**double getCarbonFootprint() const override {**

**return 0; // Bicycles do not emit CO2**

**}**

**};**

**int main() {**

**std::vector<CarbonFootprint\*> items;**

**items.push\_back(new Building(1200));**

**items.push\_back(new Car(40));**

**items.push\_back(new Bicycle());**

**for(const auto& item : items) {**

**std::cout << "Carbon footprint: " << item->getCarbonFootprint() << " kg CO2" << std::endl;**

**}**

**// Clean up**

**for(const auto& item : items) {**

**delete item;**

**}**

**return 0;**

**}**

Output:

