# San Francisco Bay University

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

**Due day: 4/14/2024**

**Kritika Regmi**
**ID: 19702**

1. Using classes, design an online address book to keep track of the names, addresses, phone numbers, and dates of birth of family members, close friends, and certain business associates. Your program should be able to handle a maximum of 500 entries.

a. Design the class *dateType* was designed to implement the date in a program, but the member function *setDate* and the constructor do not check whether the date is valid before storing the date in the member variables. Rewrite the definitions of the function *setDate* and the constructor so that the values for the month, day, and year are checked before storing the date into the member variables. Add a member function, *isLeapYear*, to check whether a year is a leap year. Moreover, write a test program to test your class.

ANS:

```
5  //number a
6  class dateType {
7  public:
8      void setDate(int month, int day, int year);
9      bool isLeapYear() const;
10     void printDate() const;
11     dateType(int month = 1, int day = 1, int year = 1900);
12 private:
13     int dMonth;
14     int dDay;
15     int dYear;
16     bool isValidDate(int month, int day, int year) const;
17 };
18 void dateType::setDate(int month, int day, int year) {
19     if (isValidDate(month, day, year)) {
20         dMonth = month;
21         dDay = day;
22         dYear = year;
23     } else {
24         cout << "Invalid date. No changes made." << endl;
25     }
26 }
27 bool dateType::isLeapYear() const {
28     return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
29 }
```

```cpp
void dateType::printDate() const {
    cout << dMonth << "-" << dDay << "-" << dYear << endl;
}

dateType::dateType(int month, int day, int year) {
    setDate(month, day, year); // Utilize setDate to validate upon object creation
}

bool dateType::isValidDate(int month, int day, int year) const {
    if (year < 1 || month < 1 || month > 12) return false;
    int daysInMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    daysInMonth[1] = isLeapYear(year) ? 29 : 28; // February leap year check
    return day >= 1 && day <= daysInMonth[month - 1];
}
```

b. Define a class, *addressType*, that can store a street address, city, state, and ZIP code. Use the appropriate functions to print and store the address. Also, use constructors to automatically initialize the member variables.

```cpp
// number b
#include <iostream>
class addressType {
public:
    addressType(string street = "", string city = "", string state = "", string zip = "")
    : streetAddress(street), city(city), state(state), zipCode(zip) {}

    void printAddress() const {
        cout << streetAddress << ", " << city << ", " << state << " " << zipCode << endl;
    }

private:
    string streetAddress;
    string city;
    string state;
    string zipCode;
};
```

c. Define a class *extPersonType* using the class *personType* as follows, the class *dateType*, and the class *addressType*. Add a member variable to this class to classify the person as a family member, friend, or business associate. Also, add a member variable to store the phone number. Add (or override) the functions to print and store the appropriate information. Use constructors to automatically initialize the member variables.

```
// number c
class extPersonType : public personType {
public:
    extPersonType(string first, string last, string phone, string relation, const addressType& address, const dateType& dob)
    : personType(first, last), phoneNumber(phone), relationship(relation), address(address), dob(dob) {}

    void printPerson() const {
        print(); // Print name
        cout << " (" << relationship << ")" << endl;
        address.printAddress();
        cout << "Phone: " << phoneNumber << endl;
        cout << "Date of Birth: ";
        dob.printDate();
    }
private:
    string phoneNumber;
    string relationship; // family, friend, or business
    addressType address;
    dateType dob;
};
```

d. Define the class *addressBookType* using the previously defined classes. An object of the type *addressBookType* should be able to process a maximum of 500 entries.

```
//number d
class addressBookType {
public:
    addressBookType() : numEntries(0) {}

    void loadFromFile(const string& filename); // Implement file reading
    void sortByName();
    extPersonType* searchByLastName(const string& lastName);
    void printPersonDetails(const string& lastName);
    void printBirthdaysByMonth(int month);
    void printNamesInRange(const string& startLastName, const string& endLastName);
    void printByRelationship(const string& relationship);

private:
    extPersonType entries[500];
    int numEntries;
};
```

Testing the code:

```
98  int main() {
99      // Test dateType
100     dateType myDate(2, 29, 2020);
101     myDate.printDate();  //print a valid leap year date
102
103     // Test addressType
104     addressType myAddress("1234 Elm St", "Smalltown", "TX", "79999");
105       myAddress.printAddress();  // print the address
106
107       // Test extended person type
108       extPersonType myPerson("John", "Doe", "123-456-7890", "friend", myAddress,
    myDate);
109       myPerson.printPerson();  // print full person details
110
111       return 0;
112   }
```

Output:

```
Console  Shell
~/CS360HW6/NUM1$ g++ num1.cpp -o result1
~/CS360HW6/NUM1$ ./result1
2-29-2020
1234 Elm St, Smalltown, TX 79999
John Doe (friend)
1234 Elm St, Smalltown, TX 79999
Phone: 123-456-7890
Date of Birth: 2-29-2020
~/CS360HW6/NUM1$
```

2. Using an abstract class with only pure virtual functions, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance -- classes *Building, Car* and *Bicycle*. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an abstract class *CarbonFootprint* with only a pure virtual *getCarbonFootprint* method. Have each of your classes inherit from that abstract class and implement the *getCarbonFootprint* method to calculate an appropriate carbon footprint for that class *(check out a few websites that explain how to calculate carbon footprints, such as https://www.youtube.com/watch?v=wWeI0c1m14Y)*. Write an application that creates objects of each of the three classes, places pointers to those objects in a vector of *CarbonFootprint* pointers, then iterates through the vector, polymorphically invoking

each object's *getCarbonFootprint* method. For each object, print some identifying information and the object's carbon footprint.

**ANS:**

OUTPUT