Part C – Encapsulation

Constructors

Workshop 4

In this workshop, you are to initialize the data within instances of class type upon their creation.

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- define the special member function that initializes the data in simple objects;
- define a default constructor that sets an object to a safe empty state;
- overload a constructor to receive information from a client;
- describe to your instructor what you have learned in completing this workshop.

Submission Policy

The *"in-lab"* section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the *"in-lab"* portion of the workshop during that period, ask your instructor for permission to complete the *"in-lab"* portion after the period. If you do not attend the workshop, you can submit the *"in-lab"* section along with your *"at-home"* section (with a penalty; see below). The *"at-home"* portion of the lab is **due on the day of your next scheduled workshop** (at 23:59).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly back up your work.

Late submission penalties:

- *"in-lab"* submitted late, with *"at-home"* submitted on time: maximum of 20/50 for *"in-lab"* and maximum of 50/50 for "*at-home*"*;*
- *"in-lab"* on time, with *"at-home"* and *"reflection"* submitted at most one week late: maximum of 50/50 for *"in-lab"* and maximum of 20/50 for *"at-home"*;
- workshop late for at most one week: *"in-lab"*, *"at-home"* **and** *"reflection"* must all be submitted for maximum of 50/100;
- workshop late for more than one week: *"in-lab"*, *"at-home"* **and** *"reflection"* must all be submitted for maximum of 30/100;
- if any of *"in-lab"*, *"at-home"* or *"reflection"* is missing, the mark will be **zero**.

IN-LAB (50%):

Design and code a class named `Passenger` in the namespace `holiday`. The class represents passengers for an airline company that go on vacation. Add to the class the following **private** attributes:

`m_name`: an array of characters of size 32 (including `'\0'`) that holds the name of the passenger;

`m_destination`: an array of characters of size 32 (including `'\0'`) that holds the destination where the passenger is travelling;

`m_departureYear`, `m_departureMonth`, and `m_departureDay`: three integers that together represent the date when the passenger goes on vacation;

The class `Passenger` has the following member function **already implemented** in `passenger.cpp` (look at this function and make sure you understand its implementation):

`void display(bool onlyNames = false) const`: a query that displays the contents of a `Passenger` instance.

Add to the type `Passenger` the following member functions (and implement them—make sure to reuse existing code instead of duplicating it):

default constructor (a constructor with no parameters): this constructor should set the instance to the safe empty state (the safe empty state is when all numeric attributes are set to zero and the strings are set to the empty string);

constructor with 2 parameters: The first parameter contains the name of the passenger and the second parameter contains the destination of the journey. This constructor should set the `m_name` attribute to the first parameter, the `m_destination` attribute to the second parameter, and the departure date to July 1st, 2017. This constructor validates the parameters before accepting them; if the parameters are not valid (any of them), then this function should set the instance to the safe empty state.

- The **strings** are valid if they are not null and not empty;

`bool isEmpty() const`: a query that checks if the `Passenger` instance is in the safe empty state;

`bool canTravelWith(const Passenger&) const`: a query that checks if two passengers can travel together (two passengers can travel together if they go to the same destination on the same date).

Using the sample implementation of the `w4_in_lab.cpp` main module shown below, test your code and make sure that it works. Below the source code is the expected output from your program. The output of your program should match **exactly** the expected one.

```
#include <iostream>
#include "passenger.h"
#include "passenger.h" // this is intentional

using namespace std;
using namespace holiday;

int main()
{
    Passenger travellers[] = {
        Passenger(nullptr, "Toronto"),
        Passenger("", "Toronto"),
        Passenger("John Smith", nullptr),
        Passenger("John Smith", ""),
        Passenger("John Smith", "Toronto"), // valid
        Passenger(nullptr, nullptr),
        Passenger()
    };
    cout << "----------------------------------------" << endl;
    cout << "Testing the validation logic" << endl;
    cout << "(only passenger 5 should be valid)" << endl;
    cout << "----------------------------------------" << endl;
    for (unsigned int i = 0; i < 7; ++i)
    {
        cout << "Passenger " << i + 1 << ": "
             << (travellers[i].isEmpty() ? "not valid" : "valid") << endl;
    }
```

```
    cout << "----------------------------------------" << endl << endl;

    Passenger vanessa("Vanessa", "Paris"),
              mike("Mike", "Tokyo"),
              alice("Alice", "Paris");

    cout << "----------------------------------------" << endl;
    cout << "Testing the display function" << endl;
    cout << "----------------------------------------" << endl;
    vanessa.display();
    mike.display();
    alice.display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing the travelling together logic" << endl;
    cout << "----------------------------------------" << endl;
    cout << "Can Vanessa and Mike travel together (should be NO)? "
         << (vanessa.canTravelWith(mike) ? "YES" : "NO") << endl;
    cout << "Can Vanessa and Alice travel together (should be YES)? "
         << (vanessa.canTravelWith(alice) ? "YES" : "NO") << endl;
    cout << "Can Alice and Vanessa travel together (should be YES)? "
         << (alice.canTravelWith(vanessa) ? "YES" : "YES") << endl;
    cout << "Can Mike and Alice travel together (should be NO)? "
         << (mike.canTravelWith(alice) ? "YES" : "NO") << endl;
    cout << "----------------------------------------" << endl << endl;

    return 0;
}
```

IN-LAB OUTPUT:

```
---------------------------------------
Testing the validation logic
(only passenger 5 should be valid)
---------------------------------------
Passenger 1: not valid
Passenger 2: not valid
Passenger 3: not valid
Passenger 4: not valid
Passenger 5: valid
Passenger 6: not valid
Passenger 7: not valid
---------------------------------------

---------------------------------------
Testing the display function
---------------------------------------
Vanessa will travel to Paris. The journey will start on 2017-7-1.
Mike will travel to Tokyo. The journey will start on 2017-7-1.
Alice will travel to Paris. The journey will start on 2017-7-1.
---------------------------------------

---------------------------------------
Testing the travelling together logic
---------------------------------------
Can Vanessa and Mike travel together (should be NO)? NO
Can Vanessa and Alice travel together (should be YES)? YES
Can Alice and Vanessa travel together (should be YES)? YES
Can Mike and Alice travel together (should be NO)? NO
---------------------------------------
```

IN-LAB SUBMISSION:

To submit the *in-lab* section, demonstrate execution of your program with the exact output as in the example above. Upload `passenger.h`, `passenger.cpp` and `w4_in_lab.cpp` to your matrix account. Compile and run your code and make sure everything works properly. To submit, run the following script from your account (and follow the instructions):

~profname.proflastname/submit 200_w4_lab<ENTER>

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT-HOME (40%):

To the `Passenger` class you have created for the *"in-lab"* portion, add the following member functions (and implement them):

constructor with 5 parameters: this constructor sets the instance attributes to the values of the parameters. The first parameter contains the name passenger, the second parameter contains the destination of the journey and the last three parameters contains the year, month and day of the departure (in that order). This constructor validates the parameters before accepting them; if the parameters are not valid (any of them), then this function should set the instance to the safe empty state.

- The **strings** are valid if they are not null and not empty;
- The only valid **years** are 2017, 2018, 2019, 2020;
- The valid **months** are between 1 and 12 (inclusive);
- The valid **days** are between 1 and 31 (inclusive);

`void travelWith(const Passenger* arrPassengers, int size)`: a query that receives as its first parameter an array of passengers representing the friends of the current passenger. The size of the array is specified in the second parameter. This function checks if the current passenger can go on vacation with any of the friends specified in the array.

- If no friend can join the current passenger on vacation, this function should display the following message:

  ```
  Nobody can join PASSENGER_NAME on vacation!<ENDL>
  ```

- If all friends can join the current passenger on vacation, this function should display the name of the friends in the following format:

  ```
  Everybody can join PASSENGER_NAME on vacation!<ENDL>
  PASSENGER_NAME will be accompanied by FRIEND_1,
  FRIEND_2, •••, FRIEND_N.<ENDL>
  ```

- If only some friends can join the current passenger on vacation, this function should display only the name of the friends that can join in the following format:

```
PASSENGER_NAME will be accompanied by FRIEND_1,
FRIEND_2, •••, FRIEND_N.<ENDL>
```

NOTE: Use the `Passenger::display(...)` function to print the name of a passenger in the examples above.

NOTE: Use the `Passenger::canTravelWith(...)` function to check if two passengers can go together on vacation.

NOTE: Redesign your constructors to reuse the code instead of duplicating it.

Using the sample implementation of the `w4_at_home.cpp` main module shown below, test your code and make sure that it works. Below the source code is the expected output from your program. The output of your program should match **exactly** the expected one.

```
#include <iostream>
#include "passenger.h"

using namespace std;
using namespace holiday;

int main()
{
    Passenger travellers[] = {
        Passenger(nullptr, "Toronto", 2018, 4, 20),
        Passenger("", "Toronto", 2018, 4, 20),
        Passenger("John Smith", nullptr, 2018, 4, 20),
        Passenger("John Smith", "", 2018, 4, 20),
        Passenger("John Smith", "Toronto", 2018, 4, 20), // valid
        Passenger("John Smith", "Toronto", 2028, 4, 20),
        Passenger("John Smith", "Toronto", 2014, 4, 20),
        Passenger("John Smith", "Toronto", 2020, 12, 31), // valid
        Passenger("John Smith", "Toronto", 2018, 40, 20),
        Passenger("John Smith", "Toronto", 2018, 0, 20),
        Passenger("John Smith", "Toronto", 2017, 1, 1), // valid
        Passenger("John Smith", "Toronto", 2018, 4, 0),
        Passenger("John Smith", "Toronto", 2018, 4, 32),
        Passenger(nullptr, nullptr, 0, 0, 0),
        Passenger()
    };
    cout << "----------------------------------------" << endl;
    cout << "Testing the validation logic" << endl;
    cout << "(only passengers 5, 8 and 11 should be valid)" << endl;
    cout << "----------------------------------------" << endl;
    for (unsigned int i = 0; i < 15; ++i)
    {
        cout << "Passenger " << i + 1 << ": "
             << (travellers[i].isEmpty() ? "not valid" : "valid") << endl;
    }
    cout << "----------------------------------------" << endl << endl;
```

```
    Passenger david("David", "Toronto", 2018, 4, 20);
    Passenger friends[] = {
        Passenger("Vanessa", "Toronto", 2018, 4, 20),
        Passenger("John", "Toronto", 2018, 4, 20),
        Passenger("Alice", "Toronto", 2018, 4, 20),
        Passenger("Bob", "Paris", 2018, 1, 20),
        Passenger("Jennifer", "Toronto", 2018, 4, 20),
        Passenger("Mike", "Toronto", 2018, 4, 20),
        Passenger("Sarah", "Toronto", 2018, 4, 20)
    };

    cout << "----------------------------------------" << endl;
    cout << "Testing the display function" << endl;
    cout << "----------------------------------------" << endl;
    for (int i = 0; i < 4; ++i)
        friends[i].display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing the travelWith(...) function." << endl;
    cout << "----------------------------------------" << endl;
    david.travelWith(nullptr, 0);
    david = Passenger("Ross", "Toronto", 2018, 4, 20);
    david.travelWith(friends, 3);
    david = Passenger("Isabela", "Toronto", 2018, 4, 20);
    david.travelWith(friends, 7);
    cout << "----------------------------------------" << endl << endl;

    return 0;
}
```

AT-HOME OUTPUT:

```
---------------------------------------
Testing the validation logic
(only passengers 5, 8 and 11 should be valid)
---------------------------------------
Passenger 1: not valid
Passenger 2: not valid
Passenger 3: not valid
Passenger 4: not valid
Passenger 5: valid
Passenger 6: not valid
Passenger 7: not valid
Passenger 8: valid
Passenger 9: not valid
Passenger 10: not valid
Passenger 11: valid
Passenger 12: not valid
Passenger 13: not valid
Passenger 14: not valid
Passenger 15: not valid
---------------------------------------

---------------------------------------
Testing the display function
---------------------------------------
Vanessa will travel to Toronto. The journey will start on 2018-4-20.
John will travel to Toronto. The journey will start on 2018-4-20.
Alice will travel to Toronto. The journey will start on 2018-4-20.
Bob will travel to Paris. The journey will start on 2018-1-20.
---------------------------------------

---------------------------------------
Testing the travelWith(...) function.
---------------------------------------
Nobody can join David on vacation!
Everybody can join Ross on vacantion!
Ross will be accompanied by Vanessa, John, Alice.
Isabela will be accompanied by Vanessa, John, Alice, Jennifer, Mike, Sarah.
---------------------------------------
```

AT-HOME REFLECTION (10%):

Create a file `reflect.txt` that contains the answers to the following questions:

1) What is meant by a safe empty state? What other safe empty state you would choose for the type `Passenger`? Explain why.
2) Describe a case where having multiple constructors for the type `Passenger` would simplify the code of clients that use it.

3) Explain what have you learned in this workshop.

AT-HOME SUBMISSION:

To submit the *at-home* section, demonstrate execution of your program with the exact output as in the example above. Upload `reflect.txt`, `passenger.h`, `passenger.cpp` and `w4_at_home.cpp` to your matrix account. Compile and run your code and make sure everything works properly. To submit, run the following script from your account (and follow the instructions):

~profname.proflastname/submit 200_w4_home<ENTER>

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.