CONSTRUCTORS

Workshop 4 (10 marks – 3% of your final grade)

In this workshop, you are to initialize the data within an object of class type upon its 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 at creation time
- to define a default constructor that sets an object to a safe empty state
- to overload a constructor to receive information from a client
- to 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 that is two days before your next scheduled workshop (23:59:59).

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

You are responsible to back up your work regularly.

LATE SUBMISSION PENALTIES:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of 7/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

IN-LAB (30%)

Design and code a class named `Passenger` in the `sict` namespace. The class represents passengers for an airline company. The class holds the following information privately:

`passenger's name`: an array of characters of size 32 (including `'\0'`);
`destination`: an array of characters of size 32 (including `'\0'`);

The `Passenger` type includes the following member functions (which you need to implement — make sure to reuse existing code instead of duplicating it):

default constructor (a constructor with no parameters): this constructor sets the object to a safe empty state;

constructor with 2 parameters: The first parameter receives the address of a C-style string containing the name of the passenger and the second parameter receives the address of a C-style string containing the name of the destination. This constructor copies the parameter data to the object instance variables, only if the data is valid. Data is not valid if its address is the null address or its string is empty. If the data is not valid, this constructor sets the object to a safe empty state.

`bool isEmpty() const`: a query that reports if the `Passenger` object is in a safe empty state.

`void display() const`: a query that displays the contents of the `Passenger` object in the following format (see also the output listing below).

```
PASSENGER-NAME - DESTINATION<ENDL>
```

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.

In-Lab Main Module

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

using namespace std;
using namespace sict;

int main()
{
    sict::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 (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;

    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 - Paris
Mike - Tokyo
Alice - Paris
----------------------------------------
```

IN-LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, 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.

Then, run the following script from your account: (use your professor's Seneca `userid` to replace `profname.proflastname`)

~profname.proflastname/submit 244_w4_lab<ENTER>

and follow the instructions.

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

AT-HOME (30%)

To your `Passenger` class of your in-lab solution add data members that store the following additional information:

`year of departure`: an integer

`month of departure`: an integer
`day of departure`: an integer

Include in your `Passenger` class definition, the following new member functions and implement them in the `.cpp` file of your `Passenger` module:

constructor with 5 parameters: this constructor receives the addresses of the C-style null-terminated strings containing the passenger's name and destination along with the year, month and day of departure. This constructor validates the parameters before accepting them. This constructor stores the data in the object's instance variables only if all of the data received is valid. If any data is invalid, this constructor sets the object to a safe empty state.

- Each **string** is valid if its address is not null and it is not empty;
- The valid **years** are 2017, 2018, 2019, 2020 (inclusive);
- The valid **months** are between 1 and 12 (inclusive);
- The valid **days** are between 1 and 31 (inclusive);

`const char* name() const`: a query that returns the address of the name of the passenger; the address of an empty string if the `Passenger` object is in a safe empty state.

`bool canTravelWith(const Passenger&) const`: a query that receives an unmodifiable reference to a `Passenger` object and checks if that passenger referenced can travel with the current `Passenger` (two passengers can travel together if they go to the same destination on the same date).

Modify your implementations of the constructors and `display()` member function to include the date of departure in the format shown below (see also the output listing below):

default constructor (a constructor with no parameters): this constructor sets the object to a safe empty state*, including the date variables*;

constructor with 2 parameters: The first parameter receives the address of a C-style string containing the name of the passenger and the second parameter receives the address of a C-style string containing the name of the destination. This constructor copies this data from into the instance variables *and sets the departure date to July 1st, 2017*, only if

the data is valid. Data is not valid if its address is a null address or its string is empty. If the data is not valid, this constructor sets the object to a safe empty state.

`void display() const`: a query that displays the contents of the `Passenger` object in the following format (see also the output listing below). Note that the month and day values are in two-digit format zero-filled if necessary

```
PASSENGER-NAME – DESTINATION on YEAR/MM/DD<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.

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

AT-HOME MAIN MODULE

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

using namespace std;
using namespace sict;

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 Passenger::display(...)" << endl;
    cout << "----------------------------------------" << endl;
    for (int i = 0; i < 7; ++i)
        friends[i].display();
    cout << "----------------------------------------" << endl << endl;

    cout << "----------------------------------------" << endl;
    cout << "Testing Passenger::canTravelWith(...)" << endl;
    cout << "----------------------------------------" << endl;
    for (int i = 0; i < 7; ++i) {
        if (david.canTravelWith(friends[i]))
            cout << david.name() << " can travel with " << friends[i].name() << endl;
    }
    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 Passenger::display(...)
---------------------------------------
Vanessa - Toronto on 2018/04/20
John - Toronto on 2018/04/20
Alice - Toronto on 2018/04/20
Bob - Paris on 2018/01/20
Jennifer - Toronto on 2018/04/20
Mike - Toronto on 2018/04/20
Sarah - Toronto on 2018/04/20
---------------------------------------

---------------------------------------
Testing Passenger::canTravelWith(...)
---------------------------------------
David can travel with Vanessa
David can travel with John
David can travel with Alice
David can travel with Jennifer
David can travel with Mike
David can travel with Sarah
---------------------------------------
```

REFLECTION (40%)

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 states could you choose for the `Passenger` type? Explain why.
2) Describe how you would modify the code to minimize duplication using the syntax that you have covered in the course to date.
3) Explain the principal benefit of minimizing code duplication.
4) Explain why the `canTravelWith(...)` member function can access the private data variables of the referenced object.
5) Describe how you would improve your code to allow for changes in the sizes of the arrays holding the name and destination data.
6) What do you need to correct when you use the `strncpy(...)` function?
7) Explain what you have learned in this workshop.

QUIZ REFLECTION:

Add a section to `reflect.txt` called Quiz X Reflection. Replace the X with the number of the last quiz that you received and list the numbers of all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

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 244_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.