CONSTRUCTORS

Workshop 4 (10 marks – 3.75% of your final grade)

In this workshop, you initialize the data within an object at its creation.

LEARNING OUTCOMES

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

- define a constructor that initializes an object's data at creation time
- define a default constructor that sets an object to a safe empty state
- describe 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. You must be present at the lab in order to get credit for the *in-lab* portion.

If you do not attend the lab, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is four days after your scheduled *in-lab* workshop (@23:59) (even if that day is a holiday).

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 `Passenger` module for an airline application. Your module contains a class named `Passenger` in the `sict` namespace. This class defines the structure of a passenger's information for an airline company. The class holds the following private information:

`The passenger's name`: an array of characters of size 19 (including `'\0'`);
`The destination`: an array of characters of size 19 (including `'\0'`);

The class also includes the following member functions (which you need to implement — make sure to reuse existing code wherever possible instead of duplicating existing code):

default constructor (a no-argument constructor): this constructor sets the `Passenger` object to a safe empty state;

constructor with 2 parameters: The first parameter receives the address of an unmodifiable null-terminated C-style string containing the name of the passenger and the second parameter receives the address of an unmodifiable null-terminated C-style string containing the name of their destination. This constructor copies the data at the received addresses to the object's instance variables, only if all data is valid. An address is valid if it can be dereferenced and points to a non-empty string; conversely, an address is invalid if it is the null address or points to an empty string. If any data is invalid, 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>
```

If the object is in a safe empty state, this function outputs the following message

```
No passenger!<ENDL>
```

Test your code and make sure that it works using the `w4_in_lab.cpp` implementation file of the main module shown below. The expected output from your program

is listed below this source code. The output of your program should match **exactly** this expected output.

IN-LAB MAIN MODULE

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

using namespace std;
using namespace sict;

int main() {
	const int no_passengers = 8;
	Passenger travellers[] = {
		Passenger(nullptr, "Toronto"),
		Passenger("", "Toronto"),
		Passenger("John Smith", nullptr),
		Passenger("John Smith", ""),
		Passenger("John Smith", "Toronto"), // valid
		Passenger("Christopher Szwartenegger", "Toronto"), // valid
		Passenger(nullptr, nullptr),
		Passenger()
	};
	cout << "----------------------------------------" << endl;
	cout << "Testing the validation logic" << endl;
	cout << "(only passengers 5 and 6 should be valid)" << endl;
	cout << "----------------------------------------" << endl;
	for (int i = 0; i < no_passengers; ++i)
	{
		cout << "Passenger " << i + 1 << ": "
		     << (travellers[i].isEmpty() ? "not valid" : "valid") << endl;
	}
	cout << "----------------------------------------" << endl << endl;

	Passenger vanessa("Vanessa", "Paris"),
		  mike("Mike", "Tokyo"),
		  chris("Christopher Szwartzenneger", "Toronto"),
		  alice("Alice", "Paris");

	cout << "----------------------------------------" << endl;
	cout << "Testing the display function" << endl;
	cout << "----------------------------------------" << endl;
	vanessa.display();
	mike.display();
	alice.display();
	chris.display();
	travellers[0].display();
	cout << "----------------------------------------" << endl << endl;

	return 0;
}
```

IN-LAB OUTPUT

```
---------------------------------------
Testing the validation logic
(only passengers 5 and 6 should be valid)
---------------------------------------
Passenger 1: not valid
Passenger 2: not valid
Passenger 3: not valid
Passenger 4: not valid
Passenger 5: valid
Passenger 6: valid
Passenger 7: not valid
Passenger 8: not valid
---------------------------------------

---------------------------------------
Testing the display function
---------------------------------------
Vanessa - Paris
Mike - Tokyo
Alice - Paris
Christopher Szwart - Toronto
No passenger!
---------------------------------------
```

IN-LAB SUBMISSION

To test and demonstrate execution of your module use the main function 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.

Then, run the following command from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace `XXX`, i.e., SAA, SBB, etc.):

~profname.proflastname/submit 200XXX_w4_lab<ENTER>

and follow the instructions generated by the command and your program.

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

AT-HOME (30%)

In the "at home" part of this workshop, you enhance your `Passenger` class by adding date information.

Copy your `Passenger` module from your in-lab solution. Add data members that store the following additional information to your `Passenger` class:

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

To manage this data, declare 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 unmodifiable null-terminated C-style strings containing the passenger's name and destination and values containing the year, month and day of departure. Like the other constructors, this constructor validates the parameters before accepting any data. This constructor stores the data in the object's instance variables only if all the data received is valid. If any data is invalid, this constructor sets the object to a safe empty state.

- Each address is valid if it can be dereferenced and does not point to an empty string;
- The valid **years** are 2019, 2020, 2021 (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 the passenger referenced can travel with the current `Passenger` (two passengers can travel together if they are flying to the same destination on the same date).

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

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

constructor with 2 parameters: This constructor copies the strings pointed to by the parameters into the instance variables *and sets the departure date to October 1st, 2019*, only if all the data is valid. Date data is valid if it meets the criteria defined above.

`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 `w4_at_home.cpp` implementation file of the 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() {
	const int no_travellers = 16;
	Passenger travellers[] = {
		Passenger(nullptr, "Toronto", 2019, 10, 20),
		Passenger("", "Toronto", 2019, 10, 20),
		Passenger("John Smith", nullptr, 2019, 10, 20),
		Passenger("John Smith", "", 2019, 10, 20),
		Passenger("John Smith", "Toronto", 2019, 10, 20), // valid
		Passenger("John Smith", "Toronto", 2029, 10, 20),
```

```
		Passenger("John Smith", "Toronto", 2014, 10, 20),
		Passenger("John Smith", "Toronto", 2020, 12, 31), // valid
		Passenger("John Smith", "Toronto", 2019, 40, 20),
		Passenger("John Smith", "Toronto", 2019, 0, 20),
		Passenger("John Smith", "Toronto", 2019, 10, 1), // valid
		Passenger("John Smith", "Toronto", 2019, 10, 0),
		Passenger("John Smith", "Toronto", 2019, 10, 32),
		Passenger("Christopher Szwartenegger", "Toronto"), // valid
		Passenger(nullptr, nullptr, 0, 0, 0),
		Passenger()
	};
	cout << "----------------------------------------" << endl;
	cout << "Testing the validation logic" << endl;
	cout << "(only passengers 5, 8, 11 and 14 should be valid)" << endl;
	cout << "----------------------------------------" << endl;
	for (unsigned int i = 0; i < no_travellers; ++i)
	{
		cout << "Passenger " << i + 1 << ": ";
		travellers[i].display();
	}
	cout << "----------------------------------------" << endl << endl;

	Passenger david("David", "Toronto", 2019, 10, 20);
	const int no_friends = 8;
	Passenger friends[] = {
		Passenger("Vanessa", "Toronto", 2019, 10, 20),
		Passenger("John", "Toronto", 2019, 10, 20),
		Passenger("Alice", "Toronto", 2019, 10, 20),
		Passenger("Bob", "Paris", 2019, 11, 20),
		Passenger("Jennifer", "Toronto", 2019, 10, 20),
		Passenger("Mike", "Toronto", 2019, 10, 20),
		Passenger("Christopher Szwartenegger", "Toronto", 2019, 10, 20), // valid
		Passenger("Sarah", "Toronto", 2019, 10, 20)
	};

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

	cout << "----------------------------------------" << endl;
	cout << "Testing Passenger::canTravelWith(...)" << endl;
	cout << "----------------------------------------" << endl;
    for (int i = 0; i < no_friends; ++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, 11 and 14 should be valid)
----------------------------------------
Passenger 1: No passenger!
Passenger 2: No passenger!
Passenger 3: No passenger!
Passenger 4: No passenger!
Passenger 5: John Smith - Toronto on 2019/10/20
Passenger 6: No passenger!
Passenger 7: No passenger!
Passenger 8: John Smith - Toronto on 2020/12/31
Passenger 9: No passenger!
Passenger 10: No passenger!
Passenger 11: John Smith - Toronto on 2019/10/01
Passenger 12: No passenger!
Passenger 13: No passenger!
Passenger 14: Christopher Szwart - Toronto on 2019/10/01
Passenger 15: No passenger!
Passenger 16: No passenger!
----------------------------------------

----------------------------------------
Testing Passenger::display(...)
----------------------------------------
Vanessa - Toronto on 2019/10/20
John - Toronto on 2019/10/20
Alice - Toronto on 2019/10/20
Bob - Paris on 2019/11/20
Jennifer - Toronto on 2019/10/20
Mike - Toronto on 2019/10/20
Christopher Szwart - Toronto on 2019/10/20
Sarah - Toronto on 2019/10/20
----------------------------------------

----------------------------------------
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 Christopher Szwart
David can travel with Sarah
----------------------------------------
```

REFLECTION (40%)

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time.**

Create a file named `reflect.txt` that contains your **detailed description of the topics that you have learned** in completing this workshop and mention any issues that caused you difficulty. Include in your explanation—**but do not limit it to**—the following points:

1) What is a safe empty state? Could you define a safe empty state that differs from the empty state that you defined?
2) Identify the parts of your source code where to minimized code duplication.
3) The `canTravelWith(...)` member function accesses the private data of the object referenced in its parameter. Explain why C++ allows this access.
4) What statement did you add to ensure that the `strncpy(...)` function executes correctly?
5) 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 command from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace XXX, i.e., SAA, SBB, etc.):

`~profname.proflastname/submit 200XXX_w4_home`<ENTER>

and follow the instructions generated by the command and your program.

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