CONSTRUCTORS

Workshop 4 (10 marks – 3.75% 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 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 `Traveler` module for an airline application.

In the `Traveler.h` header file, predefine the following constants as integers:

- `max_destination_size` with the value 32. This represents the maximum number of characters for the destination of the traveler
- `max_name_size` with the value 16. This represents the maximum number of characters for the name of the traveler

Define a class named `Traveler` in the `sict` namespace. The class defines the structure of a traveler's information for an airline company. The class holds the following private information:

- `The traveler's first name`: an array of characters of size `max_name_size` (including `'\0'`);
- `The traveler's last name`: an array of characters of size `max_name_size` (including `'\0'`);
- `The destination`: an array of characters of size `max_destination_size` (including `'\0'`);

Your `Traveler` type 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 `Traveler` object to a safe empty state;
- **constructor with 3 parameters**: This constructors set the attributes to the data from the parameters, if the parameters are valid.
  - The first parameter receives the address of a null-terminated C-style string containing the first name of the traveler.
  - The second parameter receives the address of a null-terminated C-style string containing the last name of the traveler.
  - The third parameter receives the address of a null-terminated C-style string containing the name of their destination.
  - This constructor copies the data at the received addresses to the object instance variables, **only if that data is valid**. Data is valid if the address refers to a non-empty string; that is, data is not valid if its address is the null address or the string at that address 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 `Traveler` object is in a safe empty state.

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

```
FIRST-NAME LAST-NAME goes to DESTINATION<ENDL>
```

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

```
--> Not a valid traveler! <--<ENDL>
```

Using the `w4_in_lab.cpp` implementation file of the main module shown below, test your code and make sure that it works. 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 "Traveler.h"
#include "Traveler.h" // this is intentional

using namespace std;
using namespace sict;

int main() {
  Traveler travelers[] = {
    Traveler(nullptr, nullptr, "Toronto"),
    Traveler(nullptr, "",       "Toronto"),
    Traveler(nullptr, "Smith", "Toronto"),

    Traveler("",      nullptr, "Toronto"),
    Traveler("",      "",       "Toronto"),
    Traveler("",      "Smith", "Toronto"),

    Traveler("John",   nullptr, "Toronto"),
    Traveler("John",   "",       "Toronto"),
    Traveler("John",   "Smith", "Toronto"), // valid

    Traveler("John",   "Smith", nullptr),
    Traveler("John",   "Smith", ""),
    Traveler(nullptr, nullptr, nullptr),
    Traveler("",      "",       ""),
    Traveler()
  };
  cout << "----------------------------------------" << endl;
  cout << "Testing the validation logic" << endl;
  cout << "(only traveler 9 should be valid)" << endl;
  cout << "----------------------------------------" << endl;
  for (int i = 0; i < 14; ++i)
  {
    cout << "Traveler " << i + 1 << ": "
         << (travelers[i].isEmpty() ? "not valid" : "valid") << endl;
  }
  cout << "----------------------------------------" << endl << endl;

  sict::Traveler vanessa("Vanessa", "Williams", "Paris"),
                 mike("Mike",       "Jones",    "Tokyo"),
                 alice("Alice",     "Miller",   "Rome");

  cout << "----------------------------------------" << endl;
  cout << "Testing the display function" << endl;
  cout << "----------------------------------------" << endl;
  vanessa.display();
  mike.display();
  alice.display();
  travelers[0].display(); // not valid
  travelers[8].display();
  travelers[13].display(); // not valid
  cout << "----------------------------------------" << endl << endl;

  return 0;
}
```

In-Lab Expected Output

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

----------------------------------------
Testing the display function
----------------------------------------
```

```
Vanessa Williams goes to Paris
Mike Jones goes to Tokyo
Alice Miller goes to Rome
--> Not a valid traveler! <--
John Smith goes to Toronto
--> Not a valid traveler! <--
----------------------------------------
```

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 `Traveler.h`, `Traveler.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 244XXX_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%)

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

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

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

To manage this data, declare in your `Traveler` class definition, the following new member functions and implement them in the `.cpp` file of your `Traveler` module:

constructor with 6 parameters: this constructor receives the addresses of the null-terminated C-style strings containing the traveler's first name, last name and destination along with the year, month and day of departure.

Like the other constructors, 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 2019, 2020, 2021, 2022 (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 first name of the traveler; OR the address of an empty string if the `Traveler` object is in a safe empty state.

`bool canTravelWith(const Traveler&) const`: a query that receives an unmodifiable reference to a `Traveler` object and checks if the traveler referenced can travel with the current `Traveler` (two traveler 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 3 parameters: Same parameters as for in-lab portion. This constructor copies this data from parameters into the instance variables *and sets the departure date to July 1st, 2019*, only if the data is valid. Data is valid if the address refers to a non-empty string; that is, data is not valid if its address is the null address or the string at that address 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 `Traveler` 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

```
 LAST-NAME, FIRST-NAME goes to DESTINATION on YEAR/MM/DD<ENDL>
```

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 "Traveler.h"

using namespace std;
using namespace sict;

int main()
{
  Traveler travelers[] = {
    Traveler(nullptr, "Smith", "Toronto", 2020,  4, 20),
    Traveler("",      "Smith", "Toronto", 2020,  4, 20),
    Traveler("John",  "",      "Toronto", 2020,  4, 20),
    Traveler("John",  "Smith", nullptr,   2020,  4, 20),
    Traveler("John",  "Smith", "",        2020,  4, 20),
    Traveler("John",  "Smith", "Toronto", 2020,  4, 20), // valid
    Traveler("John",  "Smith", "Toronto", 2028,  4, 20),
    Traveler("John",  "Smith", "Toronto", 2014,  4, 20),
    Traveler("John",  "Smith", "Toronto", 2022, 12, 31), // valid
    Traveler("John",  "Smith", "Toronto", 2020, 40, 20),
    Traveler("John",  "Smith", "Toronto", 2020,  0, 20),
    Traveler("John",  "Smith", "Toronto", 2019,  1,  1), // valid
    Traveler("John",  "Smith", "Toronto", 2020,  4,  0),
    Traveler("John",  "Smith", "Toronto", 2020,  4, 32),
    Traveler(nullptr, nullptr, nullptr,      0,  0,  0),
    Traveler("John",  "Smith", "Toronto"),               // valid
    Traveler()
  };
  cout << "-----------------------------------------" << endl;
  cout << "Testing the validation logic" << endl;
  cout << "(only travelers 6, 9, 12 and 16 should be valid)" << endl;
  cout << "-----------------------------------------" << endl;
  for (int i = 0; i < 17; ++i)
  {
    cout << "Traveler " << i + 1 << ": "
         << (travelers[i].isEmpty() ? "not valid" : "valid") << endl;
  }
  cout << "-----------------------------------------" << endl << endl;

  Traveler david("David", "Davis", "Toronto", 2019, 6, 20);
  Traveler friends[] = {
    Traveler("Vanessa",  "Miller",  "Toronto", 2019,  6, 20),
    Traveler("John",     "Miller",  "Toronto", 2019,  6,  6),
    Traveler("Alice",    "Turner",  "Toronto", 2019, 10, 20),
    Traveler("Bob",      "Moore",   "Paris",   2019,  6, 20),
    Traveler("Jennifer", "Hill",    "Toronto", 2020,  6, 20),
    Traveler("Mike",     "Flores",  "Toronto", 2019,  6, 20),
    Traveler("Sarah",    "Stewart", "Toronto", 2019,  6, 20),
    Traveler("Mark",     "Simmons", "Toronto")
  };

  cout << "-----------------------------------------" << endl;
  cout << "Testing Traveler::display(...)" << endl;
  cout << "-----------------------------------------" << endl;
  for (int i = 0; i < 8; ++i)
    friends[i].display();
  cout << "-----------------------------------------" << endl << endl;

  cout << "-----------------------------------------" << endl;
  cout << "Testing Traveler::canTravelWith(...)" << endl;
  cout << "-----------------------------------------" << endl;
  cout << david.name() << " can travel with: " << endl;
  for (int i = 0; i < 8; ++i)
  {
    if (david.canTravelWith(friends[i]))
      cout << "  - " << friends[i].name() << endl;
  }
  cout << "-----------------------------------------" << endl << endl;

    return 0;
}
```

At-Home Expected Output

```
----------------------------------------
Testing the validation logic
(only travelers 6, 9, 12 and 16 should be valid)
----------------------------------------
Traveler 1: not valid
Traveler 2: not valid
Traveler 3: not valid
Traveler 4: not valid
Traveler 5: not valid
Traveler 6: valid
Traveler 7: not valid
Traveler 8: not valid
Traveler 9: valid
Traveler 10: not valid
Traveler 11: not valid
Traveler 12: valid
Traveler 13: not valid
Traveler 14: not valid
Traveler 15: not valid
Traveler 16: valid
Traveler 17: not valid
----------------------------------------

----------------------------------------
Testing Traveler::display(...)
----------------------------------------
Miller, Vanessa goes to Toronto on 2019/06/20
Miller, John goes to Toronto on 2019/06/06
Turner, Alice goes to Toronto on 2019/10/20
Moore, Bob goes to Paris on 2019/06/20
Hill, Jennifer goes to Toronto on 2020/06/20
Flores, Mike goes to Toronto on 2019/06/20
Stewart, Sarah goes to Toronto on 2019/06/20
Simmons, Mark goes to Toronto on 2019/07/01
----------------------------------------

----------------------------------------
Testing Traveler::canTravelWith(...)
----------------------------------------
David can travel with:
  - Vanessa
  - Mike
  - 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 another state as the safe empty state?
2) Describe how you have minimized code duplication.
3) Explain why the `canTravelWith(...)` member function can access the private data of the object referenced in its parameter.

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`, `Traveler.h`, `Traveler.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 244XXX_w4_home<ENTER>
```

and follow the instructions.

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.