MSIM 406/506: Introduction to Distributed Simulation
Homework 3: Communicating Objects with MPI

Purpose:

To develop an ability to communicate objects between processes.

Problem Statement:

For this assignment, you are to communicate airplanes between processes. Each process in the simulation creates 3 airplanes. Each airplane will then be communicated to another process, selected randomly. On arrival, a new process will be selected to send the aircraft. This basic functionality is provided to you by

```
int GetNextDest()
{
   int rank = CommunicationRank();
   int size = CommunicationSize();
   int offset = rand() % (size - 1);
   return( (rank + 1 + offset) % size);
}
```

The structure of the airplane class is defined below.

Your task is to implement the communication of the airplane. Since at any point in time, we do not know where a communication is coming from, the system must detect where a message is coming from. You also must use a non-blocking send, else you could hang the system as everyone waits for acknowledgement on the send.

Your communication must be functional with all others in the course, as a version of everyone's program will be run to act as an airport capable of receiving and sending airplanes.

Airplane:

An airplane is defined by the following attributes (note: your implementation of the class MUST change the order of the attributes):

- ID – the id is defined by the following 2 values, which when treated as a tuple, should be unique:
  - Rank of creating process
  - Unique integer for that process
- Number of flights – number of flights the aircraft has completed
- Origin of last flight – indicates where the last flight originated from (rank of process). Should be set just prior to sending the airplane to a new airport.
- Cargo – cargo is defined by the following 3 values:
  - Quantity – integer value indicating the number of pieces of cargo on the airplane.
  - Capacity – double value indicating quantity of cargo the airplane can handle, such as square feet or cubic feet, not to change for the lifetime of the airplane.
  - Size – double value indicating the size of the cargo currently being carried by the aircraft, in the same units as the capacity and should be less than or equal to the capacity.

The airplane should have the following public methods defined in a virtual class which must be implemented:

- Constructor(double capacity)
  - Sets the ID (rank and unique ID)
  - Sets the number of flights to 0.
  - Sets cargo quantity and size values to 0.
  - Sets cargo capacity to capacity.

- Constructor(int source) – this constructor populates the new airplane by receiving its contents through an MPI communication. It should receive the message buffer from MPI and populate the object using the CommunicationPattern Recv function.
- SendFlight(int rank) – This should send the airplane to the process indicated by rank using MPI through the CommunicationPattern.
- AddFlight() – increments the number of flights. This should be called just before the airplane is sent to another airport.
- AddFlightOrigin() – sets the origin of the last flight. Should be called just before the airplane is sent to the next airport.
- AddCargo(double size) – increments the quantity of cargo and adds size to utilized.
- RemoveCargo(double size) – decrements the quantity of cargo and subtracts size from utilized.
- int GetCargoQuantity() – returns the cargo quantity.
- double GetCargoSize() – returns the cargo size.
- bool Fits(double size) – returns true if cargo of dimension size will fit in the available space on the airplane.
- PrintAirplane() – print the contents of the airplane, using the following format on a single line:
    - "rank_of_current_process, rank_of_flight_origin, id_rank, id_unique_int, number_of_flights, cargo_quantity, cargo_capacity, cargo_utilized"
    - This will allow us to sort the output messages, isolating a single process, or a single airplane. But EVERYONE must use the same format for this to work.

  This should be called every time an airport arrives at an airport.

You are required to implement the above functions. The airplane should also inherit the CommunicationPattern and implement the necessary functionality to support communication through MPI. There should be no reference to MPI outside the CommunicationPattern.

Note: Where possible, perform memory cleanup on your internal buffers once a communication is complete, but do not fret on this. While memory management is important, it is not a focus of this effort. However, without proper memory cleanup, it is suggested that you periodically shut down Visual Studio if running from the IDE to clean things up, else you may crash Visual Studio or hang your program.

Simulation Termination:

Each aircraft should complete 10 flights. Therefore, if you receive an aircraft that has completed 10 flights, you should not send the aircraft on. But you need to let all other processes know that the aircraft has completed. Therefore, you should send a message to all other processes with a tag of 0 indicating a completed aircraft (normal messages carrying an aircraft will have a tag of 1). By summing up how many aircraft have terminated on your process and how many termination messages you receive, you can determine when to terminate your program.

Message Format:

The class is required to define and document a common message format. That is to include the tag definition and the structure of information in the data buffer. The data buffer should include all info from the aircraft's attributes.

- ID – the id is defined by the following 2 values, which when treated as a tuple, should be unique:
    - Rank of creating process
    - Unique integer for that process

- Number of flights – number of flights the aircraft has completed
- Origin of last flight – indicates where the last flight originated from (rank of process). Should be set just prior to sending the airplane to a new airport.
- Cargo – cargo is defined by the following 3 values:
    - Quantity – integer value indicating the number of pieces of cargo on the airplane.
    - Capacity – double value indicating quantity of cargo the airplane can handle, such as square feet or cubic feet, not to change for the lifetime of the airplane.
    - Size – double value indicating the size of the cargo currently being carried by the aircraft, in the same units as the capacity and should be less than or equal to the capacity.

Running the program:
Each student should name his program Airport"LAST NAME" replacing "LAST NAME" with your last name. You can then test your own program with itself by running

```
mpiexec -n 5 AirportLastName
```

and then run all student airports using

```
mpiexec -n 1 AirportDilinila : -n 1 AirportPoteat : -n 1 AirportRichardson : -n 1 AirportTracey : -n 1
        AirportWebster
```

Any subset can be used by modifying the above. It is suggested to test in pairs and then slowly add more students. There is a file exchange on Blackboard to support sharing executables.

Deliverables:

1.	Design.	(1/4 of grade)
2.	Individual code (zipped project directory) and ability to execute with itself.	(1/2 of grade)
3.	Ability to execute with the rest of the class.	(1/4 of grade)