CMSC 202 Spring 2024

Project 3 – Airline Manager

Assignment: Project 3 – Airline Manager

Due Date: Tuesday, April 2nd @ 8:59pm on GL

Value: 80 points

1. Overview

In this project you will:

- Implement a linked-list data structure,
- Use dynamic memory allocation to create new objects,
- Practice using C++ class syntax,
- Practice object-oriented thinking.

2. Background

An airline route encapsulates the rights granted to the airline operator to operate scheduled air services between specified airports. International routes are governed by multilateral agreements between countries. The airline route grants exclusive rights to the airline operator to generate economic benefits which may not be available to other airline operators. For example, an airline could pay an airport for the rights to travel between JFK in New York to MCO in Orlando.

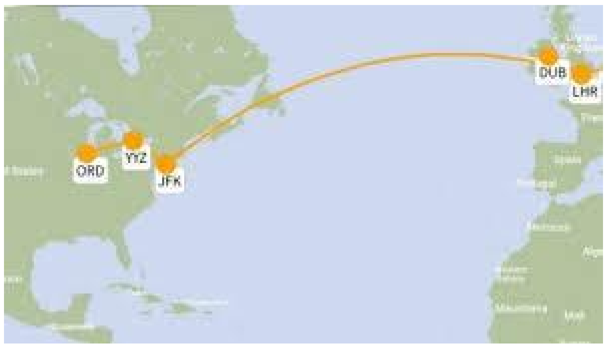


Figure 1. Example Airline Route

For this project, we are going to be designing a tool that allows us to create routes for our fledgling airline. Additionally, as fuel is so expensive, we are going to be able to calculate the distance for each designed route. There is a provided file of airport information, and we can create routes for each of the container airplanes to travel from city to city.

3. Assignment Description

Your assignment is to build an application that will allow the user to create multiple routes. Each route would be made up of at least two airports.

4. Requirements:

Initially, you will have to use the following files **`Airport.h, Route.h, Navigator.h, makefile, proj3.cpp`**, and one input file (**`proj3_data.txt`**). You can copy the files from Prof. Dixon's folder at:

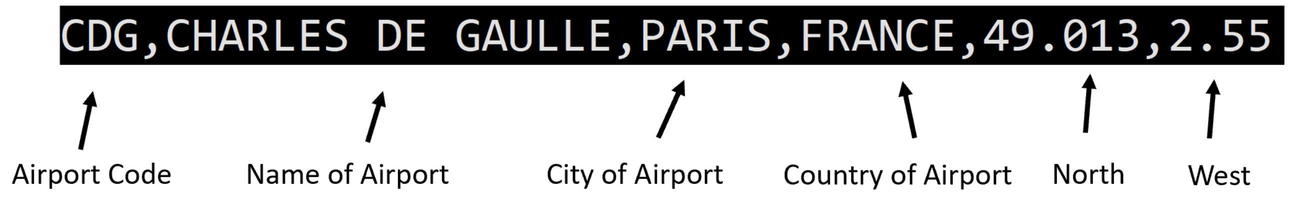
```
/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3
```

To copy it into your project folder, just navigate to your project 3 folder in your home folder and use the command:

```
cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3/* .
```

Notice the trailing period is required (it says copy it into this folder).

- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. No **`breaks`** (except in switch statements), **`continues`**, or **`exit()`**. Libraries we have learned include **`<iostream>, <fstream>, <iomanip>, <vector>, <cstdlib>, <time.h>, <cmath>,`** ~~**`<list>`**~~,and **`<string>`**. You should only use **`namespace std`**.
- You must use the function prototypes as outlined in the **`Airport.h, Route.h and Navigator.h`** header file. Do not edit the header files.
- There is one input file of ports to use including **`proj3_data.txt.`** You can see how the input file is organized in figure 2 below.



```
CDG,CHARLES DE GAULLE,PARIS,FRANCE,49.013,2.55
```

Figure 2. Sample Input File

- The node class is called **`Airport`** and contains six pieces of data: The airport code (**`m_code`**), name of the airport (**`m_name`**), the city of the airport (**`m_city`**), the country of the airport (**`m_country`**), the degrees north of the airport (**`m_north`**), the degrees west of the airport (**`m_west`**), and a pointer to the next airport (**`m_next`**). All variables in the **`Airport`** class are private and must be accessed using getters.

  - **`Airport`** (default constructor) – creates a "**`New Airport`**" from a "**`New Location`**" with north and west of 0 and **`m_next`** = **`nullptr`**;
  - **`Airport`** (Overloaded constructor) – uses data passed to populate member variables
  - Setters and Getters (may not be used explicitly but need to be implemented)
  - Overloaded **`<< operator`** – Allows user to print a specific node. Provided in .h file.

- The linked list class is called **`Route`** and contains four pieces of data: A name for the route (**`m_name`**), an **`Airport`** pointer tracking the front of the **`Route`** (**`m_head)`**, an **`Airport`** pointer tracking the end of the **`Route`** (**`m_tail)`**, and an integer tracking the size of the **`Route`** (**`m_size)`**.
  - **`Route`** (Default) – creates a new route with name of "**`Test`**" and pointers = **`nullptr`** and **`m_size`** = 0;
  - **`~Route`** – Destructs the whole route.
  - **`InsertEnd`** – Inserts a new **`Airport`** into the route at the end of the route.
  - **`SetName()`** – May not be used explicitly but you must implement. Updates the name of the route based on the string passed.
  - **`UpdateName()`** – Used to update the name of the route based on the name of the first airport and the last airport of the route. So, if the first airport was Baltimore and the last airport was Boston, the name would be Baltimore to Boston.
  - Getters (size and name) – May not be used explicitly in project but you must implement them.

- **ReverseRoute** – Reverses the route so that the airports are in reverse order. Can be called multiple times. Must rotate airports in the route (cannot update just data).
- **GetData** – Returns an airport pointer at a specific index. Passed the number of the node you would like to return from 0 to size-1. May not be used explicitly in project but you must implement it.
- **RemovePort** – Removes an airport from an existing route if there are more than two airports in the route. Passed the index of the airport to remove.
  Hint: Don't forget about the special cases (first node, last node, or middle node). Don't forget to update **m_tail**.

- The class managing the loading of files, the user input, and the routes is called **Navigator**. It has three pieces of data: a vector to hold the **Routes** (**m_routes**), a vector to hold the **Airports** (**m_airports**), and a string holding the name of the input file (**m_fileName**).
  - **Navigator**(Overloaded) – Creates a new Navigator object that is passed the file name to load.
  - **~Navigator** – Destructor for the manager
  - **Start** – Reads in the file and calls the main menu.
  - **ReadFile** – Reads in the file and dynamically allocates ports and inserts them into **m_airports**.
  - **MainMenu** – Lists each of the options: 1. Create New Route, 2. Display Route, 3. Remove Airport from Route, 4. ReverseRoute and 5. Exit. Calls the corresponding functions as needed.
  - **DisplayAirports** – Displays numbered list of each airport in **m_airports**. Uses provided **<< operator**.
  - **DisplayRoute** – Checks to see if there are any routes to display. If there are, asks user which route to display using **ChooseRoute**. Calls **DisplayRoute** in Route. Shows total miles in route.
  - **ChooseRoute** – Asks the user which route they would like to work with. Returns index of route chosen.

- **ReverseRoute** – Allows user to choose one route to reverse using **ChooseRoute**.
- **InsertNewRoute** – Allows user to insert dynamically allocated new route into **m_routes**. User continues to add airports (minimum two airports with no maximum number) until they enter a -1. Sets the name of the new route to the name of the first airport and the name of the last airport as in "Boston to Camden".
- **RemoveAirportFromRoute** – Allows user to remove an airport from an already existing route in **m_routes**. User selects an airport to remove from numbered list. Cannot remove an airport if there are two or fewer airports in the route. Will update the name of the route to be the name of the first airport and the name of the last airport as in "Boston to Camden".
- **RouteDistance** – Iterates over each airport in a specific route in **m_routes** and uses the provided **CalcDistance** to calculate the distance between two airports. Calls **CalcDistance** for airport 1 to airport 2, then airport 2 to airport 3, and so on until it reaches the end of the route.
- **CalcDistance** – Provided (do not edit). Used to calculate the distance between two points when provided with the north and west for each point.

- All user inputs will be assumed to be the correct data type. For example, if you ask the user for an integer, they will provide an integer.
- Regardless of the sample output below, all user input must be validated. If you ask for a number between 1 and 5 with the user entering an 8, the user should be re-prompted.

5. Sample Input and Output

5.1. Sample Run

One additional files named **proj3_sample.txt** is available in

/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3

A normal run of the compiled code would look like this with user input highlighted in blue:

1

1

2

3

4

```
-1










2



1



















5
```

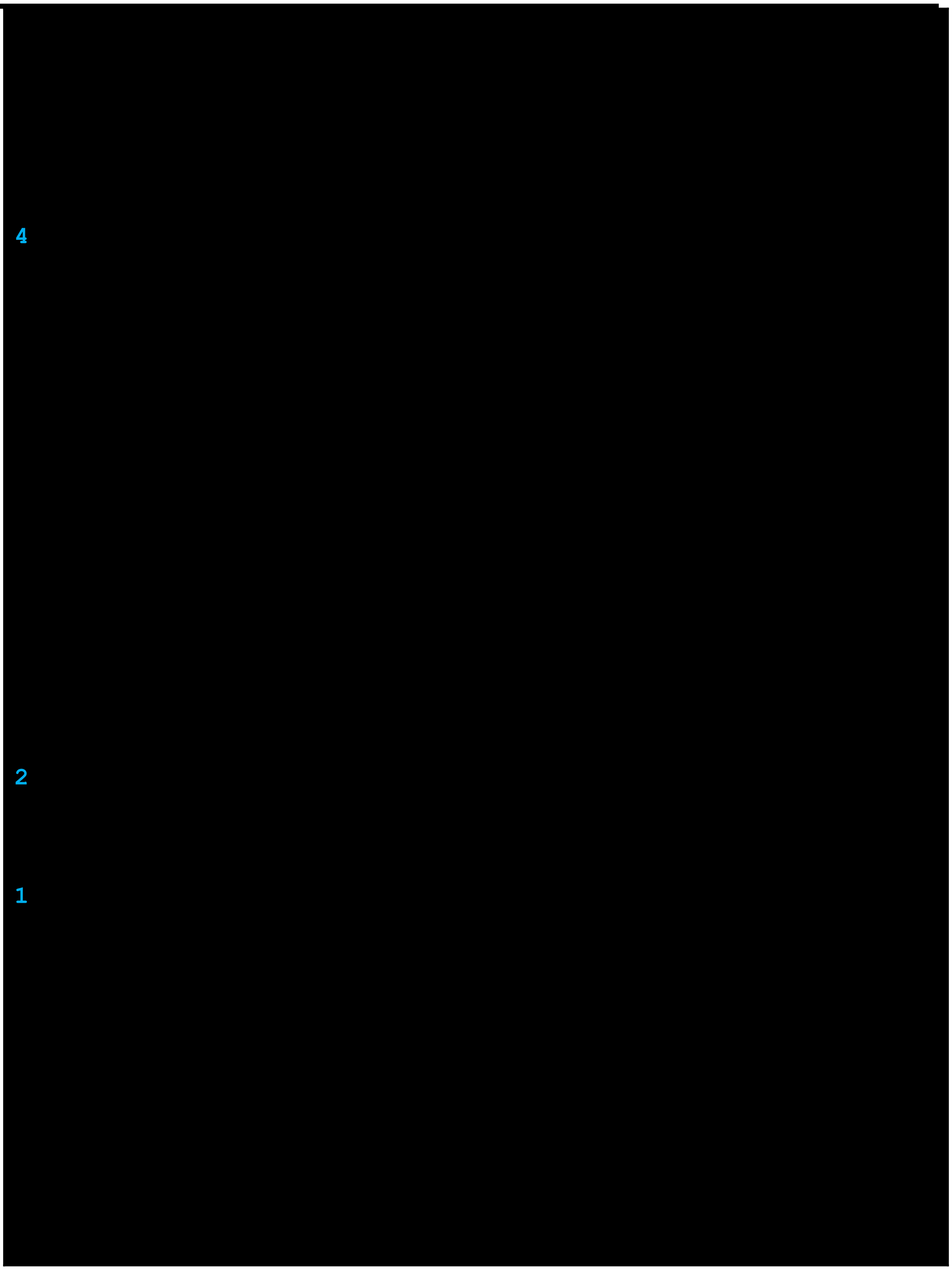
Here is the rest of that run where we remove an airport from a route and then we reverse the route.

```
2

1

3

1
```



```
4

2

1
```

```
4

1

2

1
```

A much longer run with additional input validation is included in **`proj3_sample.txt`** in the starting directory.

6. Compiling and Running

Because we are using a significant amount of dynamic memory for this project, you are required to manage any memory leaks that might be created. For a linked list, this is most commonly related to the dynamically allocated nodes. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

```
valgrind ./proj3 proj3_data.txt
```

Note: If you accidently use **`valgrind make run`**, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file. The **`makefile`** should include **`make val`** (which is ok).

If you have no memory leaks, you should see output like the following:

The important part is "in use at exit: 0 bytes 0 blocks," which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **`valgrind`** can be found here: http://valgrind.org/docs/manual/quick-start.html

Once you have compiled using your **`makefile`**, enter the command **`./proj3`** to run your program. You can use **`make val`** to test each of the input files using **`valgrind`** (do NOT use **`valgrind make run`**!). They have differing sizes. If your executable is not **`proj3`**, you will lose points. It should look like the sample output provided above.

7. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit these files to the **`proj3`** subdirectory:

`proj3.cpp` — should be unchanged.

`Airport.h` — should be unchanged.

`Airport.cpp` – should include your implementations of the class functions.

`Route.h` – should be unchanged

`Route.cpp` – should include your implementations of the class functions.

`Navigator.h` — should be unchanged.

`Navigator.cpp` – should include your implementations of the class functions.

As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder. You can try **`make submit`** too. You should turn in all **`.h`** and **`.cpp`** files for this project.

a. cd to your project 3 folder. An example might be cd **`~/202/projects/proj3`**

b. **`cp proj3.cpp Airport.h Airport.cpp Route.cpp Route.h Navigator.h Navigator.cpp ~/cs202proj/proj3`**

You can check to make sure that your files were successfully copied over to the submission directory by entering the command:

```
ls ~/cs202proj/proj3
```

You can check that your program compiles and runs in the `proj3` directory, but please clean up any `.o` and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading or generation of any `.gch or vgcore*` files in your submit directory will result in a severe penalty.

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj3 folder, it is because the due date has passed. You should be able to see your proj3 files, but you can no longer edit or copy the files in to your proj3 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj3-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj3-late2`**
- If it is after 48 hours late, it is too late to be submitted.