3460:209 Project Part 3

# **Project Assignment Part 3: My Airline**

**Overview**

The purpose of this part of the project is to make sure that you know how to write a program that utilizes arrays and array processing, and that you know how to write a program that utilizes pointers and pointer processing.

**PROGRAM SPECIFICATION**

Your airline is now publicly traded! More growth requires more improvements, and that means more software modifications. We will again pick up from the last part, and add more functionality.

For this part of the development of this project, we are going to add array processing. You are going to have a separate array for each one of the flights. These arrays will represent the physical capacity that the airline has for each of the types of flight. For example, the airline now owns 19 helicopters, hence the array for the helicopters will be allocated at a size of 19, and so on. Our arrays will allow us to control and limit the capacity for the number of users correspondingly with the amount of equipment. Don’t go creating these arrays just yet. Below, you’ll find more details, including the fact that these arrays will be dynamically allocated parallel arrays. The table below illustrates what our arrays supporting our current airline look like:

|  |  |
| --- | --- |
| Flight types | Active aircraft numbers |
| 1 | 1-19 |
| 2 | 1-15 |
| 3 | 1-24 |
| 4 | 1-33 |

SETUP NOTES:

The following global constants must be used in your program:

const int HELICOPTER\_SIZE = 19;

const int TURBO\_PROP\_SIZE = 15;

const int MIDSIZE\_JET\_SIZE = 24;

const int HEAVY\_JET\_SIZE = 33;

To make our programming more efficient and thrifty, you are also given the following code that will create a constant fixed array for the activities. This CONSTANT array called AIRCRAFT\_SIZES holds the number of possible aircrafts for each flight request. This is a simple array of integers. NUM\_AIRCRAFTS is also created to enable us to add or subtract a flight using the particular aircraft. Using these structures allows our program to be much more flexible in terms of modification efforts, limiting coding, and giving us more control.

// Global Constants

// Number of aircraft types available

const int NUM\_AIRCRAFTS = 4;

// Number of slots or vehicles of each of the aircraft types

const int AIRCRAFT\_SIZES[NUM\_AIRCRAFTS] = { HELICOPTER\_SIZE,

TURBO\_PROP\_SIZE, MIDSIZE\_JET\_SIZE, HEAVY\_JET\_SIZE };

Note: you will need to research (or maybe not) on passing the activity sizes array to your functions as a parameter. We will refer to these constants hereon.

THE ARRAYS:

Back to the array holding activities: the elements (values) within the array are the flight numbers that we generated. So this array is the type integer. As each new booking for one of the flights is added through our current system, we will now add that new flight number into the appropriate (next available) slot in the array. Your program will have to check the arrays for the next available place, and insert the value of the flight number as an element of the array (at the index). So we are simply adding sequentially to each array as new users sign up for the use of flights. Initially, let’s fill all unused slots with a value of 0. This will be the value of the empty slots and serve as a placeholder for empty and available items. This array will continue to add items until full. When the limit has been reached, *issue a message* and begin inserting again from the beginning, i.e. from index 0. The message is whatever you feel appropriate, and should be brief.

So, to illustrate, whenever a user apportions a flight, that flight number is input into the appropriate array. For example, if flight number 49193 uses a helicopter (choice 1), they are input into the next available slot in the helicopter array. Let’s say that flight number 49193 is input into number 3, the index value of 2, in the helicopter array. Then the next request for a helicopter should place the next, new flight number, into the 4th slot (index 3), and so on. This is the expected behavior for all of the arrays. Your program would fill up the arrays until full, and then start overlaying the flights starting at the front again (index 0) until reaching the back again (index 18).

Another illustration is given here that offers a sample view of the flight number arrays, where flight number 49193 is found in slot 2 in the 3rd array (midsize jet) and flight number 99577 is found in slot 1 of activity 4 (heavy jet):

|  |  |
| --- | --- |
| Flight type | Active aircraft numbers |
| 1 (helicopter) | 1: empty 2: empty 3: empty 4: empty …… 19: empty |
| 2 (turbo prop) | 1: empty 2: empty 3: empty 4: empty … 15: empty |
| 3 (midsize jet) | 1: empty 2: 49193 3: empty 4: empty ………… 24: empty |
| 4 (heavy jet) | 1: 99577 2: empty 3: empty 4: empty ……………… 33: empty |

You are going to additionally going to create parallel arrays to hold the other values in our airline system. As we collect the flight ID, we also collect the wind speeds and the destinations. We will design relational arrays or parallel arrays for those values. Thus, two additional arrays will need to be created to hold the wind speed and route. These parallel arrays are tied together by the index.

The parallel arrays for the above flight numbers might look like this:

|  |  |
| --- | --- |
| Flight type | Wind speeds |
| 1 (helicopter) | 1: empty 2: empty 3: empty 4: empty …… 19: empty |
| 2 (turbo prop) | 1: empty 2: empty 3: empty 4: empty … 15: empty |
| 3 (midsize jet) | 1: empty 2: 22 3: empty 4: empty ………… 24: empty |
| 4 (heavy jet) | 1: -9 2: empty 3: empty 4: empty ……………… 33: empty |

|  |  |
| --- | --- |
| Flight type | Destinations |
| 1 (helicopter) | 1: empty 2: empty 3: empty 4: empty …… 19: empty |
| 2 (turbo prop) | 1: empty 2: empty 3: empty 4: empty … 15: empty |
| 3 (midsize jet) | 1: empty 2: CLE-ORD 3: empty 4: empty …… 24: empty |
| 4 (heavy jet) | 1: JFK-MIA 2: empty 3: empty 4: empty ………… 33: empty |

We are going to allocate all 3 of these arrays dynamically (on the heap). To do this we will have to use pointers, each “pointing to” a corresponding location where the parallel array is located (on the heap). Your program will need to create the first array, the array of flight numbers or IDs, and then create the two parallel arrays to hold the wind speed and the routes. These arrays all use pointers to allocate them. The arrays are all tied together through the index, and the size of them is determined by the AIRCRAFT\_SIZES array. For creating these arrays you will allocate each of them by (using **new**).

How it works? So, for example, if we were allocating 33 heavy jets, let’s examine how that works for the 3 arrays. The 3 arrays will have 33 slots, and so the AIRCRAFT\_SIZES[3]should be the value of 33. And it is! That is the upper bound of the array. We can also allocate each array (flight ids, wind speeds, destinations) dynamically using this value AIRCRAFT\_SIZES[3] for jets. We would not want to hard code the subscripts, of course, but want to use an iterative coding construct to loop through and create them. This is where the NUM\_AIRCRAFTS will come in handy!

Make sure to allocate them using **new**. Make sure to not hard-code anything and use the structure like AIRCRAFT\_SIZES and NUM\_AIRCRAFTS to help you code dynamically, that is using counts with upper bounds.

PRINTING:

After getting the arrays in order, the next part adds to your menu logic the ability for the arrays to be printed. Thus, you will add an option to the menu as “Print Aircraft Usage”. This will assume the option 6. You should create a function to handle the printing of the array information. Follow the same format as (below). This function will have the task of reading through each of the flight number arrays separately and sequentially perform counts of users (see illustration below). You also will not need to access or read those parallel arrays, just the flight number array.

We now know that an array is an indexed data structure that is given to iterative processing, thus we will likely use iteration to do this task. We really do not want to process the ‘dead’ part of the array, that is, the unused part of this array. Counting those empty items would be inefficient. Yes, counting 20 or so seems innocuous, however, what if the airline grows to where we have to keep track of 5,000,000 helicopters... When designing processes such as this we should always consider the most effective algorithm for the worst case.

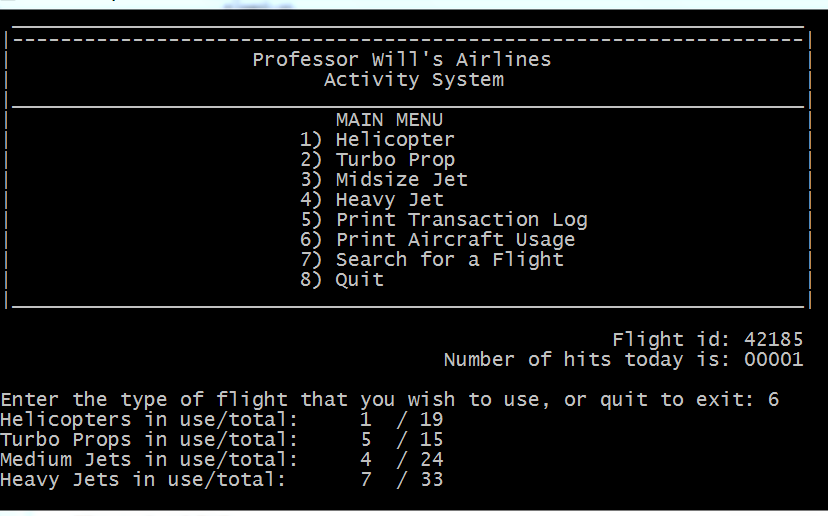
SEARCHING:

The next part again adds to your menu logic the ability for the arrays to be printed. Add an option to the menu as “Search for a Flight”. This will assume the option 7, and option 8 will be added to support “Quit”. Don’t forget to change all the parts of your program, such as validation, to incorporate these menu logic changes.

You will develop a search option and write a supporting search function so that the user can type in a flight number, and the system will output what aircraft it is that they are using and route (destination) that flight is taking, such as “Flight 45332 is taking a heavy jet for the route of JFK-MIA. Use message “that flight is not active in the system” if the flight number is not using (found) in any aircraft slot. Specific to coding, you are asked to perform a binary search for this option. Obviously, you only need to read through the array that holds flight numbers for this search function and you can then locate the destination information using the index where the flight number was found to be located (if it was found).

That’s it! That’s a lot to change, and at a high level here’s what we are doing: 1) create parallel arrays (use pointers to allocate them on the heap); 2) when the limit has been reached in any array, issue a message and begin inserting over again from the beginning, 3) display the output of the arrays with new menu option #6 as illustrated; 4) add the search functionality via menu option #7, and; 5) update the menu accordingly.

The new display after adding option 6 should exactly reflect this example:



Retain all of your prior logic for input validations, outputting, formatting, displaying the s, functions, file processing, etc…

Make sure that your programs follow good documentation standards and follow the requirements for assignments. Reference the rubric standards on Brightspace. Do not use namespace std, do validate data. DO NOT USE VECTORS, use arrays. Projects using vectors will not get credit. ☹ Design your application that includes breaking out the code into functions where possible. This code could be reused as a whole or each routine could be reused.

Submission Instructions – for programming solutions

On Brightspace, go to the matching Assignments for the **PROJECT PART-#**, where # is the number or character of the project part assigned (eg., 3 for project part 3), and submit your cpp, and any hpp files (if they exist). Unless otherwise specified, they may be submitted under any name that you prefer (such as main.cpp).

*Last updated 5.22.2016 by Will Crissey.*

*Be aware that programming falls under all of the rules of plagiarism. Be careful when using any coding found in the outside world that is not your own. Any evidence of plagiarism is subject to sanctions like forfeits, suspension, and even ejection, as determined by the Department of Student Conduct and Community Standards.*

**COPYRIGHT © 2016 Will Crissey, Jr., Assistant Professor, All rights reserved.**

This document is intended exclusively for use by The University of Akron's Department of Computer Science. Do not distribute document copies to the public in any way.

***THIS PROGRAM'S CONTENT IS PROTECTED***

The document and associated content are either the property of, or used with permission by, The University of Akron, Department of Computer Science and may be protected by copyright and other restrictions. Copyrights in the site content may also be owned by individuals and entities other than, and in addition to, The University of Akron, Department of Computer Science. The use of this document by you, or anyone else authorized by you, is prohibited, except for purposes of completing the homework/project. Any unauthorized use of the document may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.