CIS 22B Lab 5 Itty Bitty Airfreight (IBA)

#### 200 Points

Topics:

Inheritance

Virtualization

Files

-----------------------------------------------------------------------------------------------------------------------------------------

Refer to the file airfreight.doc or airfreight.docx for sample class information before you rewrite your class.

**Lab 5.1**

Utilizing the code from Lab 4.2, replace your Cargo class with a new base class. This will be the base for two

classes created through inheritance. The Cargo class will need to have virtual functions in order to have them

redefined in the child classes. You will be able to use many parts of the new Cargo class in the child classes

since they will have the same arguments/parameters and the same functionality.

Child class one will have the information for our 737 aircraft, with the addition of maxload, which is 46000 pounds.

Child class two will have the information for our **new 767 aircraft**, which has the following information:

Unit and type: Container or Pallet: AKE, APE, AKC, AQP, AQF, AAP Containers and P1P, P6P Pallets

Unit ID: Container or Pallet type: five digits + airline code; our ID code is IB, e.g. AYF12345IB

Aircraft type: Ours is a 767

Weight: The weight, in pounds, of the loaded container or pallet

Destination: A three alpha character IATA string, e.g. MFR (Medford, OR), or BUR (Burbank, CA)

Maxload: 767-300 max load is 116000 pounds

The 737-200 is a single hold, narrow body aircraft, designed to hold up to approximately 23 tons of cargo.

The 767-300 is a dual hold (upper and lower) designed to handle approximately 58 tons of cargo. IBA does not

carry refrigerator (reefer) containers or animal containers due to handling limitations.

Change private to **protected** in the Cargo class only.

Make two classes which inherit from the Cargo class:   **Boeing737** and **Boeing767**.   
Each class will need to inherit a default constructor, a copy constructor, and a constructor that has six parameters. Only one more function will be built in each class; all the rest will be inherited.   
The unit and type data are different for each class as is the aircraft type. Maxload is provided for both aircraft. Our

737 can hold up to 46000 pounds, and our 767 can hold up to 116000 pounds.

Create two new global functions named load737 and load767. These are used to build a unit for Boeing737 or a Boeing767, respectively.

Revise the main function to call the input function and do nothing else.

The data file **lab5data.txt** is provided for your use. Your program will be tested with different data, but the format

Is guaranteed to be correctly formatted, and not to exceed 20 lines.

In the input function loop reads one line from the file each time through the loop, look at the aircraft field in the record and call the corresponding build function to build that type of unit. Before calling the appropriate build function, print a header giving the sequence number of the unit read, with the number 1 for the first unit and incremented for each successive unit. For both child classes, ensure that the units are compatible with the aircraft. For example, the 737 cannot hold an AQF type container. Ensure that the weight totals for both types of aircraft are not exceeded by the units assigned to them. If they are at or near capacity, check to ensure that another unit would not put them over the weight limit, if it would, then reject the unit and move on to the next unit in the file.

Store each unit object in an **array of objects** of the appropriate type, either for the 737 or the 767.

Once input has completed, you will have a number of objects of two different types stored in two different arrays.

Use output to send the contents of each array’s objects to the screen, taking care to output all of the 737 units

together and all of the 767 units together as well as the weight totals for both aircraft.

\* Use the **lab5data.txt** file which will contain data similar to

the following three lines of data; **lab5data.txt** will have no

more than 20 lines of data

**Pallet PAG PAG45982IB 737 4978 OAK**

**Container AYF AYF23409AA 767 2209 LAS**

**Container AAA AAA89023DL 767 5932 DFW**

\* All weights are in pounds, don’t worry about kilograms.

\* In the input function, declare an object of type ifstream named

inputFile, which we will use to read from the file.

\* At the beginning of the code for the input function, open the

file. If the open fails, send a message to stderr and exit the

program.

\* In all the reads within the input function, remove the user

prompt and read from the inputFile object, rather than reading

from the stdin object.

\* **Hint: We need to use getline when reading the strings.**

using **>>** skips leading white space before reading the data.

getline does not skip this leading whitespace. So, **before** using

getline use the following code:

while(inputFile.peek() == ' ')

inputFile.get();

**peek** looks at the next character you are about to read. If it is

a space, **get** is used to read the space character, to get it out

of the way. Your output will then be much neater.

\* Use a loop to read each line from the file. To do this use a

while loop including all the reading in the input function, as

well building and output of the Cargo.

**Hint: you can do this with the following while statement:**

while(inputFile.peek() != EOF) or

while(!inputFile.eof) or

while (inputFile >> type)

The peek function will return EOF is there is no next character.

The .eof will turn true when eof is reached

The read of type with >> if true when there is data present, else it is false

\* At the bottom of the input function, close the file.

This is good code for opening a file and testing to see if it opened successfully.

ifstream inputFile;

inputFile.open("lab5data.txt");

// If the open fails, send a message and exit

if(!inputFile)

{

std::cerr << "Error opening the file" << endl;

exit(0);

}

while (inputFile.peek()!= EOF)

{…}