CIS 22B Lab 6 Itty Bitty Airfreight (IBA)

#### 200 Points

Topics:

Linked Lists

Files

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

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

**Lab 6.1**

Utilizing the code from Lab 5.2 rewrite your program to read the data from **lab6data.txt** and create two

linked lists of objects, one for each aircraft type.

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

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. This will facilitate correct inheritance.

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.

The unit and type data are different for each class as is the aircraft type. Maxload is **46000** pounds for our 737 aircraft,

and **116000** pounds for our 767.

The main function calls the input function and does nothing else.

The data file **lab6data.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 total.

In the input function, the 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 add that unit to the appropriate linked list. 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 linked lists, ensure that the units are compatible with the aircraft. For example, the 737 cannot hold an AQF type container. Reject units that are not compatible with the aircraft type. You should send an error message which reads something like this “The AQF container is not compatible with the 737 aircraft”. 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, showing an error message saying “Unit XXX not added due to weight restrictions”, where XXX is the unit type.

Print out the contents of both linked lists with appropriate header information, similar to this:

**Load out of 737 aircraft IBA0123**

**Pallet/Container Type Identifier Weight Destination**

**Load out of 767 aircraft IBA0456**

**Pallet/Container Type Identifier Weight Destination**

\* Use the **lab6data.txt** file which will contain data similar to

the following three lines of data; **lab6data.txt** will have no

more than 20 lines of data total.

**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 data 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)

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

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

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); // #include <stdlib>, or use return

}

while (inputFile.peek()!= EOF)

{…}