ECS36B Winter 2020 2020-02-06

Homework 2 due Thu Feb 13 at 23:59

Use the handin directory hw2 to submit your work

Part 1: Aircraft Maintenance System

The regular maintenance of aircraft engines is critical to ensure the high level of safety of air travel. Maintenance activities are scheduled for each engine of an aircraft after specific periods of use, which depend on the aircraft and engine model. In this assignment, you are asked to write a program that determines the maintenance schedule of aircraft engines. The program must be able to process different types of aircrafts (having different numbers of engines), requiring maintenance at different intervals. In this assignment, we will only consider two types of aircraft: Airbus A380 (four engines, requiring maintenance every 750 hours) and Boeing 737 (two engines, requiring maintenance every 500 hours).

Description

The input of the program is read from standard input. Each input line consists of a description of an aircraft, including the model (encoded as one character), the aircraft name (or "tail code"), and the number of hours recorded since the last maintenance check for each of its engines. The number of hours of use may differ among the engines of a given aircraft due to the fact that not all engines are subjected to maintenance at the same time.

Assignment

Aircrafts are represented as instances of the classes **`A380`** and **`B737`** that are derived from a base class **`Aircraft`**. The main program **`maintenance.cpp`** is provided, together with the header file **`Aircraft.h`** . Another program **`testAircraft.cpp`** is provided that tests the functionality of the **`Aircraft`** derived classes, and should compile without warnings. You should not modify these files. You will implement the base class **`Aircraft`**, and the derived classes **`A380`** and **`B737`** in the file **`Aircraft.cpp`** . The **`Makefile`** is provided and should not be modified. The **`maintenance`** program reads a list of aircraft descriptions from standard input and prints a description of the aircraft and its maintenance schedule. See the example input files and corresponding output files for details. Note that if an engine's maintenance is past due, a special output message is printed instead of a time. If an unknown aircraft code is used, the program prints an error message and exits (see example files).
Example: the input line:

```
A VH-OQF 630 550 470 690
```

describes an Airbus A380 named VH-OQF whose four engines have 630, 550, 470 and 690 hours of use since their last maintenance. Given this input, the maintenance program will print the following output:

```
Aircraft: VH-OQF type: Airbus A380 has 4 engines
engine 1: 630 hours
engine 2: 550 hours
engine 3: 470 hours
engine 4: 690 hours
```

```
Maintenance schedule for VH-OQF
engine 1: maintenance due in 120 hours
engine 2: maintenance due in 200 hours
engine 3: maintenance due in 280 hours
engine 4: maintenance due in 60 hours
```

If the input contains multiple lines, the program should process them sequentially until the end of file is reached in input.

Specification of the Aircraft classes

The **`Aircraft`** base class is an abstract class from which the classes **`A380`** and **`B737`** are derived. The base constructor takes two arguments: the number of engines and the aircraft name. In addition, the following member functions must be defined:

`virtual const std::string type(void) const`
A pure virtual function returning a string (**`"Airbus A380"`** or **`"Boeing B737"`**).

`virtual const int maxHours(void) const`
A pure virtual function returning the maximum duration of use of an engine before maintenance is due.

`const std::string name(void) const`
A member function returning the name of the aircraft.

`int numEngines(void) const`
A member function returning the number of engines of the aircraft.

`void setHours(int i, int h)`
A member function used to set the current number of hours of use for engine **`i`** .

`void print(void) const`
A member function that prints the description of the aircraft on standard output (see first part of the above example).

`void printSchedule(void) const`
A member function that prints the maintenance schedule of the aircraft on standard output (see above example).

`static Aircraft* makeAircraft(char ch, std::string name_str)`
A static factory function that creates an **`Aircraft`** object of the appropriate type (determined by the character **`ch`**) and returns a pointer to it. If **`ch`** is '**`A`**', an **`A380`** must be created. If **`ch`** is '**`B`**', a **`B737`** must be created.

All compilations should complete without warnings (use the **`–Wall`** option). Submit your project using:

```
$ handin cs36b hw2 hw2p1.tar
```

Part 2: Air Cargo Logistics

An airline is using airplanes of various sizes to ship containers between two airports. Each shipment consists of a number of standardized containers that must be loaded on the airplanes so as to minimize the amount of unused space. As several shipments of different sizes arrive at the airport, the airline must decide which shipment is loaded on which plane. The following rules are applied:

- All airplanes are flying to the same destination, i.e. a shipment can be added to any airplane.
- A shipment cannot be split among different airplanes.
- Once a shipment is loaded on a plane, it cannot be unloaded before departure.

You are tasked with writing a program that will assign shipments to airplanes in the order of the shipments' arrival at the airport. The airline you work for operates four Airbus A321 with a capacity of 10 containers each, two Boeing B777 with a capacity of 32 containers each, and one Boeing B787 with a capacity of 40 containers.

Description

The input of the program consists of the size of the shipments (i.e. number of containers in the shipments), in the order in which they appear in the queue. The assignment program must first print a list of available slots in all empty airplanes. It should then read the size of each shipment to be loaded and report on attempts to assign the shipment to an airplane. Finally, it should print a summary of all assignments. See the example input and output files for details. If a shipment size is not valid (i.e. not a positive number of containers) an exception should be thrown and an error message printed before the program proceeds with the next shipment.

Assignment

The program **`assignshipments.cpp`** is provided, together with the header files **`Airline.h`** and **`Airplane.h`** . You should not modify these files. You will implement the classes **`Airline`** and **`Airplane`** in the files **`Airline.cpp`** and **`Airplane.cpp`** respectively. A **`Makefile`** is provided and must not be modified. Use the test input files to verify that your program reproduces *exactly* the example output files. Use the Unix **`diff`** command to compare your output to the example output.

Specification of the Airline class

The **`Airline`** class constructor creates the list of airplanes using dynamic allocation of the (private) pointer array **`airplaneList`**. The airplanes should appear in the list in order of increasing size. The **`Airline`** constructor takes three arguments (the number of A321, B777 and B787 airplanes). The **`nAirplanes`** data member is the total number of airplanes used by the airline. The **`Airline`** class has a member function **`addShipment`** that tries to assign a shipment to an airplane by inspecting the list of airplanes and by assigning the shipment to the first airplane that can accommodate the shipment, starting at the beginning of the **`airplaneList`** array. The **`Airline`** class also has a member function **`printSummary`** that prints a list of airplanes with their current and maximum load. Empty airplanes should not be printed in the summary. See the examples of output files for details on the output format.

Specification of the Airplane class

The **`Airplane`** class constructor takes one argument (the maximum capacity of the airplane). It has accessor member functions **`maxLoad`** and **`currentLoad`** returning the maximum and

current number of containers respectively. The **addContainers** member function attempts to add containers to the airplane and modifies the load accordingly. It returns **true** if the operation is successful and **false** otherwise.

All compilations should complete without warnings (use the **–Wall** option). Submit your project using:

```
$ handin cs36b hw2 hw2p2.tar
```