ECS36B Fall 2018 2018-10-29

Homework 2 due Tue Nov 6 at 23:59

Use the handin directory hw2 to submit your work

Part 1: Aircrew scheduling system

The scheduling of aircrew members must follow strict regulations on how long each person may work in a given week. Each aircrew member can work on varying number of flights and hours depending on their job responsibilities. In this assignment, you are asked to write a program that determines whether a crew member may be scheduled for a specific flight based on their current workload. The program must be able to process different types of crew members (having different limitation on number of flights and hours). In this assignment, we will only consider three types of crew members: Pilot (5 flights, 60 hours/week), Attendant (8 flights, 60 hours/week), and a "Tagged" Attendant (8 flights, 60 hours/week). A tagged attendant is working as a backup for other fulltime attendants on the fight and registers only half the hours, meaning on a flight that physically takes 7 hours, a tagged attendant will only register 3.5 hours of work, whereas an Attendant or a Pilot will register the full 7 hours of work. A flight having stops counts as multiple flights. For example, a flight having 2 stops counts as 3 flights.

Description

The input of the program is read from standard input. Each input line consists of a description of an aircrew member, including the job title (encoded as one character), the name, number of flights recorded so far this week, and the number of hours worked during this week. Followed by the flight information, namely number of stops, and then total length of flight in hours.

Assignment

Aircrew members are represented as instances of the classes **`Pilot`**, **`Attendant`** and **`TaggedAttendant`** derived from a base class **`Aircrew`**. The main program **`scheduling.cpp`** is provided, together with the header file **`Aircrew.h`** . Another program **`testAircrew.cpp`** is provided that tests the functionality of the **`Aircrew`** derived classes. You should not modify these files. You will implement the base class **`Aircrew`**, and the derived classes **`Pilot`**, **`Attendant`** and **`TaggedAttendant`** in the file **`Aircrew.cpp`** . You will create a **`Makefile`** that includes a target **`all`**, which builds the executables **`scheduling`** and **`testAircrew`**. The **`scheduling`** program reads a list of aircrew descriptions from standard input and prints a description of the aircrew and his/her availability. See the example input files and corresponding output files for details. Note that the last line of output should reflect whether the crew member can be assigned to the specific flight or not. If an unknown crew code is used, the program prints an error message and exits (see example files).
Example: the input line:

```
P Alex 3 20 1 20
```

describes a Pilot named Alex who has worked on 3 flights this week for a total of 20 hours. And the system is attempting to schedule Alex to a 1-stop flight (therefore, counted as 2 flights) lasting a total of 20 hours. Given this input, the scheduling program will print the following output:

```
Pilot: Alex has operated 3 flights for a total of 20 hours this
week
Available flights: 2
Available hours: 40
Attempting to schedule for 1 stop(s) 20 hours flight...
This crew member can be scheduled
```

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

Specification of the Aircrew classes

The **`Aircrew`** base class is an abstract class from which the classes **`Pilot`**, **`Attendant`** and **`TaggedAttendant`** are derived. The base constructor takes one argument: the name of the crew member.
In addition, the following member functions must be defined:

`virtual const std::string type(void) const`
A pure virtual function returning a string (**`"Pilot"`**, **`"Attendant"`** or **`"TaggedAttendant"`**).

`virtual const int maxFlights(void) const`
A pure virtual function returning the maximum number of flights a crew member can work on per week.

`const double maxHours(void) const`
A member function returning the maximum number of hours a crew member can work on per week.

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

`void setFlights(int i)`
A member function used to set the number of flights worked.

`void setHours(double h)`
A member function used to set the current number of hours worked.

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

`virtual void scheduleFlight(int i, double h) const`
A member function that prints whether the aircrew can be scheduled for the flight specified or not on standard output (see above example, starting from the line **`"Attempting to schedule..."`**).

`static Aircrew* makeAircrew(char ch, std::string name_str)`

A static factory function that creates an **Aircrew** object of the appropriate type (determined by the character **ch**) and returns a pointer to it. If **ch** is '**P**', a **Pilot** must be created. If **ch** is '**A**', an **Attendant** must be created, and if **ch** is '**T**', a **TaggedAttendant** must be created.

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

```
$ handin cs36b hw2 aircrew.tar
```

Part 2: The All or Nothing Restaurant

Welcome to the one-of-a-kind "All-or-Nothing-Restaurant", where if guests are lucky enough to get seated, then they can eat as much as they want. But, there's a catch. The arriving group can only be seated all together at one table. If the current seating arrangement cannot accommodate them, they need to try coming back another night. Each table in the restaurant can only be used once in the entire evening.

Given the restaurant's popularity, there is large queue of groups waiting to be seated everyday. The restaurant however only has 3 types of tables - Large (which can accommodate 10 guests, Medium which can accommodate 7 guests and Small which can accommodate 5 guests). Currently, the restaurant has only 1 Large sized table, 2 Medium sized tables and 3 small sized tables. The owner of the restaurant has hired you to write a program to assign guests to tables.

Description

The input of the program consists of the size of the groups, in the order in which they appear in the queue. The size of the first group to be seated appears first. The table assignment program must first print a list of available tables in the restaurant. It should then read the sizes of each group to be seated and report on attempts to assign the group to a table. Finally, it should print a summary of seat assignments. See the example input and output files for details. If a group size is not valid (i.e. not a positive number) an exception should be thrown and an error message printed before the program proceeds with the next group.

Assignment

The program **assigntables.cpp** is provided, together with the header files **Restaurant.h** and **Table.h** . You should not modify these files. You will implement the classes **Restaurant** and **Table** in the files **Restaurant.cpp** and **Table.cpp** respectively. You will create a **Makefile** that includes a target **assigntables** (that builds the executable) and a target **clean** (that removes the executable and all object files). 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 Restaurant class

The **Restaurant** class constructor creates the list of tables using dynamic allocation of the (private) pointer array **tableList**. The tables should appear in the list in order of decreasing size. The **Restaurant** constructor takes three arguments (the number of large, medium and small tables in the restaurant). The **size** data member is the total number of tables in the restaurant. The **Restaurant** class has a member function **addGuests** that tries to assign a group of guests to a table by inspecting the list of tables and by assigning the group to the first table that can accommodate the group, starting at the beginning of the **tableList** array. The

`Restaurant` class also has a member function **`printSummary`** that prints a list of tables with their current and maximum occupancy. Empty tables should not be printed in the summary. See the examples of output files for details on the output format.

Specification of the Table class

The **`Table`** class constructor takes one argument (the size of the table). It has accessor member functions **`maxOccupancy`** and **`currentOccupancy`** returning the maximum and current number of guests respectively. The **`addGuests`** member function attempts to add a group of guests to the table and modifies the occupancy 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 restaurant.tar
```