CSCI 2226 Spring 2017

Program 4: Airport Check-in

1 Goals

1. To implement and use a linked-list queue and a linked-list priority queue.
2. To use a controller class, a container class with two derived classes, and a polymorphic data class in one application.
3. To program a simple simulation.

2 The Idea

Jetstream Airlines wants to run their airport check-in counter as efficiently as possible without causing passengers to miss flights. They have collected data, which they claim are typical, about their groups of passengers and their arrival habits. Your job is to simulate the check-in process and find out how many agents must be on duty to handle a flight. Your program will implement a variable number check-in agents, and you will find out experimentally how many are needed for the given input file.

2.1 The Person Class

This is a polymorphic base class from which both check-in agents and groups of customers are derived. It should have one data member, a name (a string).

In this class, put all the functions that are the same for the Group class and the Agent class. This includes:

1. A constructor with one string parameter that will initialize the name.
2. A default destructor that must be virtual.
3. `Person pop()`: Return the Person (Agent or Group) at the head of the list, then update the head.
4. `bool isempty()` Return true if the list is empty.
5. `virtual void print()` (This is to help you debug.) Walk through the people in the queue, starting with the head. Use each object in the queue to call the appropriate Group::print() or Agent::print() function. Note: printing a data structure should never change it. So use a local temporary scanner.

2.2 The Group Class

This data class is derived from Person. It should have two data members: the number of people in the group and the number of bags they have. Define these functions:

1. A constructor with three parameters, a name and the two numbers. The constructor must pass the group name in a ctor to the constructor of the base class (Person).
2. A virtual default destructor.
3. Two get-functions for the two numbers.
4. `virtual void print()`: print the Group's name, number of people, and number of bags.

2.3 The Agent Class

This data class is derived from Person. It needs one additional data member, a float = the time at which the agent will finish with the current group of customers. Define the functions below and other functions, if needed.

1. Define a constructor with one parameter, a name. Pass that parameter in a ctor to the constructor of the base class, Person.
2. A virtual default destructor.
3. `virtual void print()`: print the Agent's name and the time at which he will be free.
4. `void handle(Group gp).` Calculate the number of minutes it should take to check in `gp`, as follows:
   (a) 4 minutes to process the first person in a Group.
   (b) 1 minute to handle each additional group member.
   (c) .5 minute per bag that must be checked.

   Add this number of minutes to the Agent's current time to get the next time at which the agent will be free. Store this number in the Agent object.

2.4 The PQueue Class

Derive this container class should implement a linked-list priority queue, with a head pointer, as discussed in class. Initialize to nullptr in the class definition.

Implement these functions:

1. A default constructor is enough.
2. `push(Agent ag)`: Add a Cell containing ag at the proper insertion slot in the queue. Update the head of the queue if that insertion was before the first Cell.
3. `Group pop()`: Return the Agent at the head of the queue, then update the head.
4. `bool isempty()` Return true if the list is empty.
5. `print()` Print the groups in the queue, starting with the head. (This is to help you debug.) Note: printing a data structure should never change it. So use a local temporary scanner.

This class and the Queue class could both be derived from a linked list class because several of he functions are identical.. I am not recommending that because I want you to get the job done and not focus on deriving classes.

2.5 The Queue Class

This container class should implement a linked-list queue, as shown in the lecture notes, with head and tail pointers. Initialize to nullptr in the class definition.

Implement these functions:

1. A default constructor is enough.
2. `push(Group g)`: Add a Cell containing g to the end of the queue and update the tail.
3. `Group pop()`: Return the Group at the head of the list, then update the head.
4. `bool isempty()` Return true if the list is empyt.
5. `print()` Print the groups in the queue, starting with the head. (This is to help you debug.) Note: printing a data structure should never change it. So use a local temporary scanner.

2.6 The Simulation Class

This class is the application controller. It will have five data members, an open input stream, a Queue for the Customers, a PQueue for the Agents, the number of Agents, and a clock (initially 0).

Implement these functions:

1. A constructor with one parameter, a string, which is the name of the input file. Open the file and initialize the the stream variable in the class. Accept input from the keyboard for n, the number of counter agents to use. Instantiate the empty Queue and PQueue.

   Create n Agents who will all be ready at time 0.0 ; insert them onto the Pueue. The order does not matter at this time, since all these agents will start their jobs 120 minutes (2 hours) before departure time. Finally, call the getGroups() function.
2. `getGroups()`. The input file will have one family per line, and the lines will be in order of arrival time. Each line gives the family name, the number of people traveling together, and the number of bags to check:

Harrison	3	5
Kaufman	1	0
Zhang	4	9
Amin	1	2

   Read each line of the file, instantiate a Group, and push it into the Queue. When this is done, you are ready to start the simulation.
3. `CheckIn()`. The check-in counter opens two hours (120 minutes) before the scheduled flight time. The goal is to get the crowd through the checkin and out to the security checkpoint by 30 minutes before departure time. Set the current time to 0. Execute the following loop until the Queue is empty:

   (a) Remove the first Group from the Queue.
   (b) Remove the first agent from the PQueue.
   (c) Set the simulation clock to the time stored in the Agent's object.
   (d) Call that agent's handle function to handle the group. The function will return a time at which the Agent will be free.
   (e) Insert the agent back into the PQueue using the free-time.
   (f) Print a line of output with the current time, the agent's name, the passenger's name, and the minutes he will require to check in.

   When everyone in the file has been processed, print the current time and say whether the check-in was on time or not. (All passengers are supposed to reach security at least half an hour before flight time.)

   Experiment with different numbers of agents and find the smallest number of agents that get all the people through the process on time (in 90 minutes or less).

2.7 The main function

Write a `main()` function that will instantiate a Simulation using the name of a data file. Then call its CheckIn() function, which will carry out the simulation.

3 Submitting Your Work

1. I have made a short input file for use during debugging. Run your program on it until it works.
2. Then run it on the long file, which lists well more than 200 passengers.
3. Experiment with different numbers of agents and find the smallest number of agents that get all the people through the process on time (in 90 minutes or less).
4. Hand in your code and the output from the final run that shows how many agents you need.
5. Your name and "P4" should be part of the name of the zip file you submit.