Assignment 4: Heaps and priority queues

Objectives

You must implement a heap and priority queue API and use them to write a waiting queue manager.

Requirements

You will construct a program `queue` that demonstrates your heap and priority queue API. We describe the required behavior of the program in the following section.

You must implement your heap structure with the array functions defined in `array.h`. These functions provide a dynamic array implementation.

You must submit your work as a tarball. `make tarball` will create the tarball for you.

Waiting queue

Your program will manage a queue at the doctor's office, which does not take appointments. The situation at this office is as follows:

- patients arrive at the front door in random order and enter the waiting area;
- every session, the doctor picks the first patient in alphabetic order;
- at the end of every day, all remaining patients return home.

To translate this into a program, you will work with the following:

- the program receives the incoming patients on its standard input. Every line gives information on a single patient. That information is the first name followed by their age, separated by a space character.
- the program will work by running a loop, performing the following steps:
    1. accept all the patients "waiting at the door" at that time: read all the input lines until you reach a "`.`" on a separate line.
    2. place the patients in a priority queue ordered by name;
    3. pick the first patient in alphabetic order using the priority queue;
    4. treat the patient;
    5. make the patient leave: print the patient's name on the standard output and remove the patient from the queue.
    6. print a "`.`" on a single line to signal the end of the hour-long session.
- if there are no patients to pick, the doctor will do nothing during the session, but you should still print a "`.`" to show that time has passed.
- "at the end of the day" (after ten iterations of the loop, so ten hours later), "all patients leave". Print the names of all patients still in the queue in alphabetical order and exit.

Below we show a typical day of 10-hour sessions in the doctor's office. Carl, Bob, Albert, and Barbara show up early and get treated by the doctor in alphabetical order. Jim, Tom, and Zoe arrive late in the day, so Tom and Zoe leave without being treated:

Input	Output

Carl 22 Bob 68 Albert 35 .	Albert .
Barbara 40 .	Barbara .
.	Bob .
.	Carl .
.	.
.	.
.	.
Alice 28 .	Alice .
.	.
Jim 30 Tom 31 Zoe 29 .	Jim .
	Tom Zoe

Getting started

1. Unpack the provided source code archive; then run `make`.
2. Try out the generated `queue` program and familiarize yourself with its interface.
3. Read the file `prioq.h` and understand the interface.
4. Implement the data structure in `heap.c`.
5. Implement the missing code in `main.c`.

Grading

Your grade starts from 0, and the following tests determine your grade:

- +0,5pt if you have submitted an archive in the right format.
- +0,5pt if your source code builds without errors and you have modified `heap.c` in any way.
- +2pt if your priority queue API processes insertions and removal properly.
- +3pt if your `queue` program orders the patients properly.
- +1pt if your `queue` program properly forces every patient to leave at the end of the day in the correct order.
- -0,5pt if your `queue` program misbehaves if no patient has arrived since the last patient left.
- -1pt if `valgrind` reports errors while running your program.
- -1pt if the provided `Makefile` reports warnings when compiling your code.
- +1pt if your implementation has the correct style and the correct complexity.

- +1pt if your `queue` program accepts a single command-line argument `-y`, which causes, when specified, to change all the patient processing based on age. It will pick the youngest patient instead of selecting based on alphabetical order. If two or more patients have the same age, use the patient's name as a secondary sorting key with alphabetical ordering. Patients leaving at the end of the day are also sorted by increasing age in the same way.
- +1pt if your `queue` program also accepts a duration of the patient's appointment on the input (after the age), and keeps the doctor occupied for that duration when that patient is treated. If a patient is still being treated at the end of the day, their treatment is not completed, and the patient is forced to leave first. We show an example with the duration extension below:

Input	Output
Zoe 22 4 .	.
Martha 50 1 .	.
.	.
Albert 11 1 .	Zoe .
.	Albert .
.	Martha .
etc..	