Assignment 4: Heaps and priority queues

3-03-2017

Deadline: 11-03-2017 23:59

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. The required behavior of the program is given in the following section.

It is strongly recommended to implement your heap structure as an array. We provide a dynamic array implementation in `array.h` and `array.c`. We advise you to use these array functions to implement your heap.

You must submit your work as a tarball[1]. Next to the source code, your archive must contain a text file file named "`AUTHORS`" containing your name and Student ID(s).

Waiting queue

Your program will manage queuing at a 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 go back home.

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

- the program receives the incoming patients on its standard input. Each patient is given as a line of input text containing the patient's first name followed by his/her age, separated by a space character.

[1] http://lmgtfy.com/?q=how+to+make+a+tarball

- 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 complete session, but you should still print a "." to show that time has passed.
- "at the end of the day" (after 10 iterations of the loop, so 10 hours later), "all patients leave": print the names of all patients already in the queue in alphabetical order and exit.

A typical day of 10 hour long sessions in the doctors office is shown below. 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 files `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 with an `AUTHORS` file.
- +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 arrived since the last patient left.
- -1pt if `valgrind` reports errors while running your program.
- -1pt if `clang -W -Wall` reports warnings when compiling your code.

The following extra features will be tested to obtain higher grades, but only if you have obtained a minimum of 5 points on the list above already:

- +1.5pt if your `queue` program accepts a single command-line argument `-y` which causes, when specified, to change all the patient processing based on age to pick the youngest patient instead of alphabetical order. Patients leaving at the end of the day are also sorted by increasing age.
- +1.5pt if your `queue` program also accepts a duration of the patient's appointment on the input (after the age), and keeps the doctor busy for that duration when that patient is picked. 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. An example with the duration extension is shown below:

Input	**Output**
Zoe 22 4 .	.

... continued on next page

Input	**Output**
Martha 50 1 .	.
.	.
Albert 11 1 .	Zoe .
.	Albert .
.	Martha .
etc..	