15-122: Principles of Imperative Computation, Fall 2014
Lab 12: Priority Queues in C

Tom Cortina (tcortina@cs) and Rob Simmons (rjsimmon@cs)

Monday, November 17, 2014

For this lab, you will show your TA your answers as you complete each activity. Autolab is not used for this lab. You should be able to complete the first task and at least part of the second task for full lab credit.

Make a directory `lab12` in your private `15122` directory and copy the required `pq*.c` files to your `lab12` directory:

```
cd $HOME/private/15122
mkdir lab12
cp /afs/andrew.cmu.edu/usr9/tcortina/public/15122-f14/pq*.c lab12
```

Then, while you are in your `lab12` directory, make a `lib` directory and copy the required support files to your `lib` directory:

```
mkdir lib
cp /afs/andrew.cmu.edu/usr9/tcortina/public/15122-f14/lib/* lib
```

This lab involves using (unbounded, resizing) priority queues, specified by the following C interface:

```
typedef void *elem;
typedef bool higher_priority_fn(elem e1, elem e2);
typedef struct pq_header* pq;

pq pq_new(size_t capacity,            /* > 0 */
          higher_priority_fn *prior); /* != NULL, (*prior)(e1, e2)
                                         returns true if e1 is
                                         STRICTLY higher priority
                                         than e2 */
bool pq_empty(pq P);
void pq_add(pq P, elem e);
elem pq_rem(pq P);         /* P must not be empty */
void pq_free(pq P);        /* P must be empty */
```

1 Sorting using priority queues in C

Look through the files to see what code you are given. You are given a file `heaps.c` that implements the priority queue interface in `pq.h`. Note that the header for a priority queue now requires a function pointer to a function that determines if one element has a higher priority than another. The type definition for this function is given in `pq.h`.

Exercise 1. Complete the `pqsort.c` file so that it sorts an array of strings using a priority queue. You will need to define a separate function that determines if one string has a higher priority than another string, and pass a pointer to this function when you create your new priority queue. We provide a simple `main` function for you to test your implementation.

Compile your function as follows:

```
gcc -Wall -Wextra -Werror -std=c99 -pedantic -DDEBUG lib/*.c pqsort.c
```

In addition, use `valgrind` to make sure you have no memory leaks.

2 Stacks using priority queues in C

(HARDER) Use priority queues to implement stacks as defined in `lib/stack.h`.

Exercise 2. Complete the file `pqstack.c` with your implementation of the stack functions. Stacks are to be implemented using a priority queue to hold the data. Each element of the priority queue will be a stack element along with its "priority". We give you a main function you can use to test your implementation.

Note that your stack header consists of a priority queue (to hold all of the elements) and a `pushcount` field. The `pushcount` field keeps track of how many elements have been pushed on to the stack. In order for your priority queue to act like a stack, you will need to use the `pushcount` field in some way. You still need a function to test for higher priority as you did in the previous exercise.

For your stack operations that you implement, the only thing you need to check with your data structure invariant is that the given stack pointer isn't NULL, so you can just write preconditions like `REQUIRES(S != NULL);` instead of writing an `is_stack` function.

Compile your function as follows:

```
gcc -Wall -Wextra -Werror -std=c99 -pedantic -DDEBUG lib/*.c pqstack.c
```

In addition, use `valgrind` to make sure you have no memory leaks.

Something to ponder: Does your solution still have a subtle bug that the main function does not trigger?