CSCE 221 Cover Page

Programming Assignment #5

Due November 20th at midnight to eCampus

First Name Last Name UIN

User Name E-mail address

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Date

Programming Assignment 5 (100 points)

Objectives:

- Implementation of the Minimum Priority Queue (MPQ) ADT
- Testing MPQ operations using the binary heap data structure
- Application of MPQ for scheduling computer processes (jobs)

This programming assignment consists of three phases.

1. **(20 points) Phase 1**: Implement the binary heap operations: `remove_min()`, `is_empty()`, and `insert()`. Test the binary heap ADT using input objects described in the phase 3. In your report provide implementation details, the running time in terms of big-O asymptotic notation of each operation, and an evidence of testing the operations. You can use the STL vector to implement binary heap data structure.

2. **(20 points) Phase 2**: Implement the minimum priority queue ADT with the operations: `remove_min()`, `is_empty()`, and `insert()`. Test the priority queue data structure based on the binary heap from the phase 1. In your report provide implementation details of MPQ, the running time in terms of big-O asymptotic notation of each operation, and an evidence of testing the operations.

3. **(20 points) Phase 3**: Write an application based on the MPQ to simulate the scheduling of computer processes (jobs) to run on a single CPU (see also Problem P-8.7, page 366 in the textbook).

   (a) The simulation of CPU running is done by using a loop where one iteration corresponds to the unit of the CPU time called a "*time slice*." The value of the loop index is the current CPU time, and it goes from 1 to a certain integer limit (say, the number of jobs to run). The CPU selects a job with the highest priority (see the definition of the highest priority below) and run it.

   (b) The jobs to run the CPU are read from a file called `cpu-jobs.txt` where each line represents a job. The format of each line consists of 3 integers: *job ID, length n, priority p.*

      i. Job IDs are unique positive integers (they do not need to be consecutive numbers).

      ii. Assume that the job length n is measured in time slices and is an integer between 1 and 10, inclusive.

      iii. A job priority p is represented also by an integer between -20 (highest priority is negative 20) and 19 (lowest priority is 19), inclusive as well. Jobs are scheduled based on their priorities: highest priority jobs will be scheduled first. Pay attention to the fact that the *higher* priority means *lower* numerical value (therefore we can use Minimum PQ). If two or more jobs have the same priority value, break ties by job IDs: a job with a lower ID is scheduled first.

   File input format:

job ID	length	priority
281	5	12
825	2	2
111	10	19
382	3	-7

   (c) For simplicity, we assume that jobs cannot be interrupted – once a job is scheduled on the CPU, it runs for a number of time slices equal to its length.

   (d) Your program should create an output file `cpu-jobs-output.txt`. For each time slice, print one line to this file containing job ID of the job currently running the CPU, its priority, and the remaining time to run (in time slices). With every loop iteration the length of the job running the CPU is decreased by 1. The format of the output line is as follows: "*Job [job ID here] with length n and priority p*". For example:

   ```
   Job 382 with length 3 and priority -7
   Job 382 with length 2 and priority -7
   Job 382 with length 1 and priority -7
   Job 825 with length 2 and priority 2
   Job 825 with length 1 and priority 2
   ...
   ```

   (e) If there are no jobs to run, the program issues the output: "*no more jobs waiting*", and stops.

4. (10 pts) Reading data from the input file and writing data into the output file with the required format. Your program should run correctly on TA's input.

5. Instructions

   (a) Your files should be named as follows: `main.cpp`, `cpu-sim.cpp`, `cpu-sim.h`

   (b) Compile your program by

   ```
   g++ -std=c++11 *.cpp
   g++ -std=c++11 -o cpu-sim *.o
   ```

   or

   ```
   make all
   ```

   (c) Run your program by executing `./cpu-sim`

   (d) Test your code and collect output data for your report.

6. Submission

   (a) Tar the files above and any additional files. Please create your tar file following the instructions.

   (b) "turnin" your tar file to eCampus.

   (c) (30 points) Submit an electronic copy of cover page, report (see below) in lab to your lab TA. Find a cover page. Typed report made preferably using "LyX/LaTeX"

      i. (2 pts) Program description; purpose of the assignment.

      ii. (2 pts) Instructions to compile and run your program; input and output specifications.

      iii. (5 pts) Programming style, and program organization and design: naming, indentation, whitespace, comments, declaration, variables and constants, expressions and operators, error handling and reporting, files organization, operators overloading. Please refer to the PPP-style document.

      iv. (15 pts) Discussion of the implementation and running time in the terms of big-O of each phase of the CPU simulator.

      v. (6 pts) Presenting the testing results, the screenshots of the input and output files.