CS 305: HW 3 (Mother of All PrintShops: MOAP)
Fall 2019

During the semester free tutoring is available to you for CS 305. Email StepUP@up.edu to make an appointment. You should also seek help of the tutors working on the weekday nights in Shiley 208. Of course, you can always come to office hours.

Due Date: Sunday, October 27th, NLT midnight. Your code files (.h and .c files), sample output file, and makefile should be submitted as a single username_hw3.zip file and short answer questions in a pdf or docx file should be submitted as a separate file to Moodle.

This assignment is meant to give you practice with linked lists and to compare/contrast a linked list implementation with an array implementation.

This assignment should be completed **individually**. Advice: start as soon as the homework assignment is assigned, so you have plenty of time to get the code to compile, have time to test, have time to debug, have time to re-test, and have time to complete the summary report.

Specification

A local Mother of All PrintShops: MOAP print shop asked you to build an application to manage their print shop jobs. Your task is to implement print queue logic and the job management. In particular your application should support reading and adding jobs to the printer queues, deleting a particular job from a queue, merging queue when a printer goes offline, and distributing the load when a printer comes back online.

The final code should include the following files:

- `printer.h` – defines the printer struct and the function prototypes that operate on the printer.
- `printJob.h` – defines the print job struct and the function prototypes that operates on print jobs.
- `main.c` – test program runs the simulation
- `printer.c` – implements the functions on the printer (you can add additional helper function on as needed basis)
    - add_job
    - update_printer
    - offline
    - online
    - print
- `printJob.c` – implements the functions on the print job
    - create_printJob
    - print

A `makefile` should include a directive to build executable called 'run':

```
make run
```

To run:

```
./run <initial setup file required>
```

To run the code and direct output to a file:

```
./run <initial setup file required> > sample_output.txt
```

To clean:

```
make clean
```

Requirements:

Your program should read an input file with initial configuration and starting print jobs. The file name is passed into the program from the command line and read into the file. The file structure is a decimal value that is the maximum number (N) of printers that the MOAP has. Followed by the list of N printers and their print capacity (in pages per minute). Assume that at the start of the program all printers are online. The rest of the lines in the file are print jobs with the name and a number of pages the print job has. The jobs are distributed into the printer queues in a round robin fashion. After the jobs are read in, the program will hang on a command line to accept and process one of the user defined commands.

`setup.txt`

```
3
P1 1
P2 2
P3 1
J1 3
J2 5
J3 2
J4 1
J5 2
J6 3
J7 1
```

1. `printer.h` defines the following struct with the printer name (maximum of 10 characters, no spaces allowed), the print queue that is the head of a linked list that points to the printer's linked list of print jobs (pointer is null if the printer does not have any jobs), the printer's capacity in the pages per minute and the variable online to store the printer's state if it is online or not.

```
printer
------------------------------
name: char*
printQueue: printJob*
capacity: int
online: true/false
```

2. `printJob.h` implements the linked list data structure with the payload. Each printJob is one node of a linked list. Each node contains the print job name (a maximum of 10 characters, no spaces allowed), the size of the job (in pages) and a pointer to the next print job object.

```
printJob
---------------------------
name : char*
size: int
next: printJob*
```

3. `main.c` will define an array of printer structs, then read in the printers and distribute the print jobs into the printer queues in a round robin assignment. The main will then hang to process the next command line argument. The above setup.txt file will result in the following queue assignment (the following also shows the functionality of the print function):

```
P1:1-> J1:3, J4:1, J7:1
P2:2-> J2:5, J5:2
P3:1-> J3:2, J6:3
```

4. `printer.c` will implement the following functionality on the printer's queue:

- add_job: If the printer is online, the command line 'a <printer name> <printJob name> <print job size>' directive will add a printJob object to the end of the printer's queue. If the queue is empty (Null) it adds the printJob at the head. Illegal or incorrect printer name will result in no-op. PrintJob name duplicates are allowed and the printJob size has to be >0.

```
P1:1-> J1:3, J4:1, J7:1
P2:2-> J2:5, J5:2
P3:1-> J3:2, J6:3
a P3 J9 3
P1:1-> J1:3, J4:1, J7:1
P2:2-> J2:5, J5:2
P3:1-> J3:2, J6:3, J9:3
```

- update_printer: is executed on the tick (command t = one minute measure entered as command line directive) and it processes the next print job from the printer's queue by the number of pages equal to the printer's capacity. If the printer's capacity is larger than the next job in the queue, the subsequent job(s) are processed until the printer's capacity for the current minute tick is exhausted. The print jobs that are fully processed must be deallocated from the memory and a print message issued "Done: <printer name> <printJob name>". If the whole printJob cannot be completed in the current minute, the printJob size is updated accordingly in the printJob's size. If the printer queue is empty execute no-op. Example:

```
P1:1-> J1:3, J4:1, J7:1
P2:2-> J2:5, J5:2
P3:1-> J3:2, J6:3
t
P1:1-> J1:2, J4:1, J7:1
P2:2-> J2:3, J5:2
P3:1-> J3:1, J6:3
t
P1:1-> J1:1, J4:1, J7:1
P2:2-> J2:1, J5:2
P3:1-> J6:3
Done: P3 J3
t
P1:1-> J4:1, J7:1
P2:2-> J5:1
P3:1-> J6:2
Done: P1 J1
Done: P2 J2
```

- offline: executed on the command line directive ‘f <printer name> which will set the printer’s variable online to false and no jobs can be added to the printer’s queue. If the printer queue is not empty, the print jobs are weaved into the rest of the printers’ queue in a round robin with skipping one when possible. The tick is not advanced. See example below:

  ```
  P1:1-> J1:3, J4:1, J7:1
  P2:2-> J2:5, J5:2
  P3:1-> J3:2, J6:3, J8:4, J9:1, J10:1, J11:1
  f P3
  P1:1-> J1:3, J3:2, J4:1, J8:4, J7:1, J10:1
  P2:2-> J2:5, J6:3, J5:2, J9:1, J11:1
  P3:1->
  ```

- online: executed on the command ‘o <printer name>’ from the command line. The command will set the set particular printer back online if the printer is offline and no-op otherwise. The tick is not advanced.
- print: will print the printer(s) in the following format:
  <printer name>:<printer capacity>-><printJob name>:<printJob remaining size>, …

HINT: when working on the offline functionality, I suggest passing in the array of printer structs, making a temporary array of pointers to the head of each print queue list and advancing the temporary pointers as you are weaving in the jobs from the printer that just went offline.

HINT2: make sure you are passing enough information to the methods described above to be able to implement the required functionality. For example, the update function will need all printers which means pass in the array of printer structs. The add function on the other hand only needs one printer struct from the array of printers and one allocated+instantiated printJob struct.

5. `printJob.c` will allocate, instantiate or deallocate the printJob objects.

At the top of all source files, please include useful comment that at least includes your name and information about this file and implementation specifics. Then, include <stdio.h>, <stdlib.h>, and all necessary headers.

Copy and paste the function prototypes from the header files `*.h` to the respective source files `*.c`. Remove the semi-colon at the end of the lines and replace them with braces { }.

Implement the above described functionality. I’m giving you a specification level description which is pretty close to the design, otherwise you are free to design and implement the solution on your own.

<u>It is recommended that you draw pictures as you implement code to ensure the pointers in the linked list are updated correctly.</u>

Command line commands:
t -- tick increments the global counter by one minute and processes non-empty printer queues
o <printer name> -- set <printer name > online
f <printer name> -- set <printer name > offline and distributes jobs
a <printer name> <job name> <job duration> -- add a print job with its size to the printer’s queue
p – explicit print call to print all printers and their queues. After every printer or queue update, the print function of all printers+queues is executed by default. ‘p’ is an explicit print command for debugging.
q – quit the application and deallocate all allocated memory

Testing: Although the command line arguments require the setup file for the initial input, I suggest generating no fewer than 5 test case files for the command line functionality test. Recall the redirection command that will allow you to pipe in the content of a file to appear as if it was typed in on a command line. For example:

```
./run setup.txt < commandTests.txt
```

Documentation
The code should be well-commented, well indented, and include enough whitespace to be readable. The .c file should have a block comment at the top that includes information about the author and the code.

Example Output
generate sample output for your tested command files:

```
./run setup.txt < commandTests.txt > commandTests_output.txt.
```

Additional Enrichment
If you have time, feel free to add more components to your program. But, keep the original code saved!!! You can submit multiple folders of code. If you do any enrichment, please put the original code files in a folder called `hw3` and put the enriched program in a folder called `hw3optional`. Then zip both folders into a single file.

Please document the additional features in your code and your written summary. This is not extra credit, but a chance for you to do some exploration and learning.

- Add more attributes to MOAP, include deleting items from the queue, cancelling all jobs assigned to a printer, keeping statistics on the total number of pages printed by a printer, giving print jobs different priorities (careful to avoid starvation).

Logistics

1. You may use your personal computer to write, edit, and compile code. However, your code *must compile* with gcc on Mobaxterm in Shiley 206 or the Linux VDI kiosk. It is to your advantage to make sure your code compiles on this platform.
2. When you are finished and ready to submit:
    1. Create a new folder called username_HW3.
        1. (For example, Martin's would be cenek_HW3.)
    2. Copy the header files and code files into the folder.
    3. Copy your sample commandTests.txt and commandTests_output.txt file into this folder.
    4. Zip the folder by right-clicking and Send To->Compressed Folder (on Windows). There should also be a zip utility on Mac.
3. **What to turn in:** Submit your **username_HW3.zip** file and **username_HW3.docx or .pdf** to Moodle before the due date and time. After logging in to learning.up.edu, navigate to CS 305. You will find a link to submit this file. You do not need to print anything for this homework submission.

Grading Guidelines (total of 20 points)
Your files will be graded in three separate categories:

- (5 points) Code Quality: Design, commenting, whitespace, readability, proper indentation, consistent naming conventions
- (10 points) Code Operation: Does code do what is listed in the specification? Note: code that does NOT compile will receive 0 points for code operation.
    - online (0.5 pts)
    - offline (4 pts)

    - update (3 pts)
    - add (2 pts)
    - free (0.5 pts)
- (5 points) Summary report

HW 3 Report Guidelines and Format – use the template provided below. Include the questions in your write-up.

Name:

CS 305 HW 3 Report

1. Questions (include these in your write-up):

1a. (.5 point) If your program does not meet the specifications, please note the differences and anything that is not working correctly. If it works, write "Meets all specifications."

1b. (.5 point) Include the input and output file called `commandTests.txt and commandTests_output.txt` in your zip file from executing your program.

> You can create an output file of what is printed to the screen with the command line command (assuming your executable is called run):
> ```
> ./run setup.txt < commandTests.txt > commandTests_output.txt
> ```

2. (.5 point) Show test cases (such as `commandTests2, commandTests3 etc.`) that demonstrate how you tested that all cases defined in the above specifications for update, offline, and add functionality are working properly.

3. (.5 point) Why is datastructure to keep track of the printers the same/different from the print job queue implementation?

4a. (.5 point) Print a simple setup.txt file and draw a picture of the MOAP application. The picture should include all components of the printers, jobs, and how are these objects linked in the memory **.**

4b. (.5 point) Draw the picture of the above MOAP application in 4a followed by:
't, t, a < >< >< >, t, f<>,t,t'

5a. (.5 point) What is the big-O running time for the best case and the worst case for the `add` MOAP function (CLI command 'f <>')? Assume arbitrary number of items in the queue (N = print jobs).

5b. (.5 point) What is the big-O running time for the best case and the worst case for the `offline` MOAP function (CLI command 'f <>')? Assume arbitrary number of items in each of the queues (N = all print jobs).

5c. (.25 point) Suppose a print queue is created as an array of 5000 printJob structs and the printer has only 10 print jobs. Suppose a print queue is created as a linked list and stores 10 items. How much wasted storage (in bytes) does the array vs. linked list implementation use in this example?

5d. (.25 point) What is one advantage (pro) of using linked lists to store the print jobs?

5e. (.25 point) What is one advantage (pro) of using arrays to store the print jobs?

6. (.25 point) How much time did you spend in total on this homework assignment in hours (including the report)?

Appendix A: (copy this statement if it applies to you) I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files.
Appendix B: Copy and paste your `source and header files` here (use Courier New 8pt font so the characters line up correctly)