THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming

(Computer Science 2211a)

ASSIGNMENT 5

Due date: Wednesday, December 01, 2021
(changed from Course Outline Date: date has been extended 2 days)
11:55 pm Eastern Standard Time – 4:55 am Greenwich Mean Time)
allow up to one day late ONLY – assignment closed Dec 02, 2021 11:55pm est : 4:55am GMT

Assignment Overview:

The purpose of this assignment is to provide experience in managing linked lists using pointers and structures.

When embarking on a detailed assignment like assignment five (5) there are two ways to approach the task.

The first is to just start programming and write code that accomplishes many if not all the tasks all at once. This is the way we all started to code in first year. The assignments were fairly uncomplicated. This meant the code could be approached as a single task and worked on as a unit.

But this does not lend itself to larger and more complex projects.

The second method is to understand the task and break it down into smaller, more manageable sections. Then code one section to completion at a time.

It is highly recommended that the second approach be used for this assignment. It will save much time and also very much frustration.

The hints provided at the end of this document will detail one possible approach to this second method.

PREPERATION:
For this assignment, create a new directory under the assignments directory created in the first assignment. Label this new directory: asn5

All work should be performed in this directory. Use a UNIX editor like vi to create and compile the C code.

The code MUST compile in the UNIX environment to be considered correct.

This time the assignment will be contained within multiple files.

Assignment Five (5)

This assignment will simulate the use of a printer queue to manage the processing of printer jobs.

Every time a document is requested to be printed, a printer job is created and placed in the printer queue. The printer will process one job at a time by printing the page(s) of a document. Once all the pages of the current printer job have been processed, the request is released, and the printer will inspect the printer queue to retrieve and remove the next job in the queue.

The simulation will operate on a per cycle basis. Whereas a real printer queue is continuous, this simulation will execute for a given finite number of cycles.

The main program will loop through this process for some finite set of cycles denoted by the defined preprocessor definition of ITERATIONS representing number of cycles.

Each print job will have four factors:

a) – document number (next number in sequence)

b) – request priority:
1 – high priority
2 – average priority
3 – low priority

c) – Number of pages in the requested document

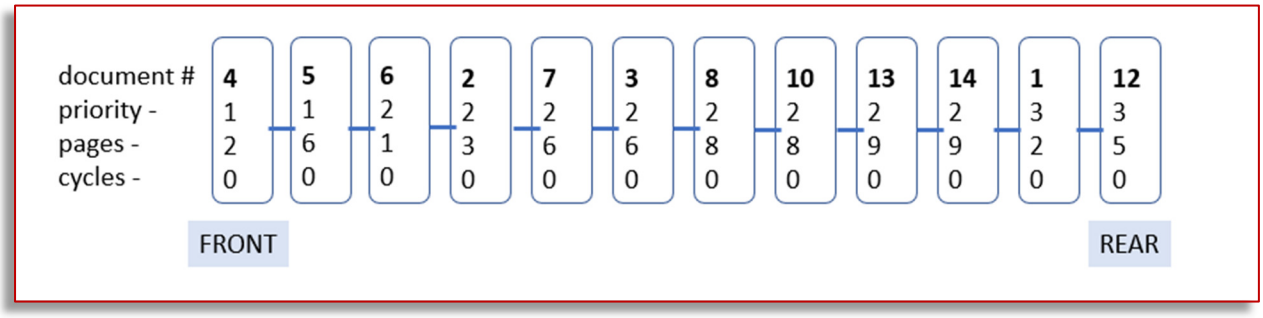
d) – Number of cycles the print job has been in the queue.

Each cycle will consist of:

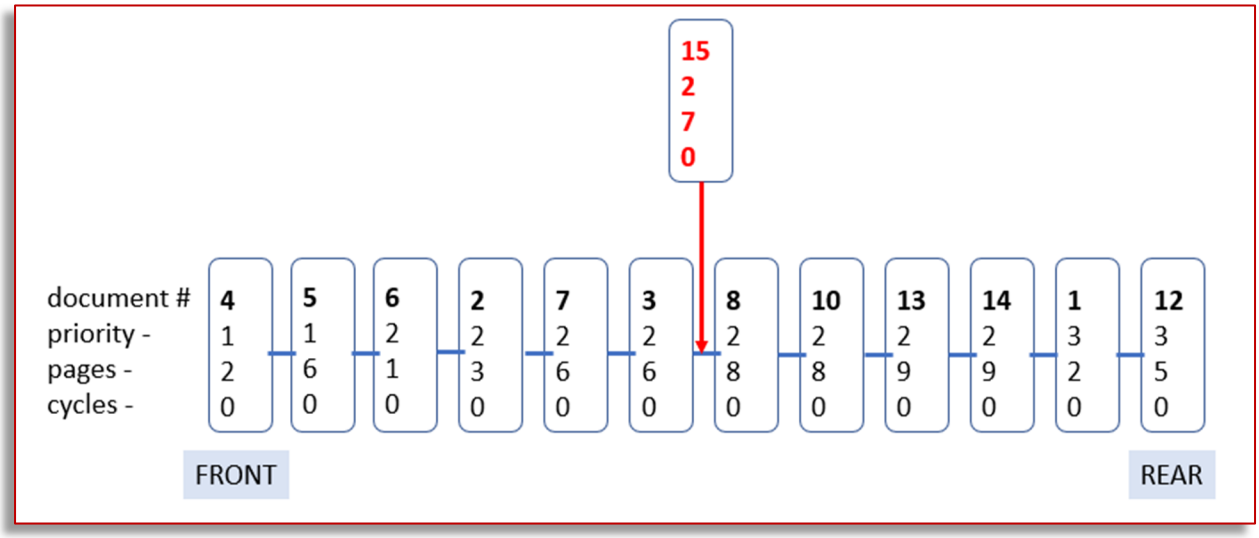
a) - creating a new print job request:
This request will be based on a 10% chance in each cycle that a new request will be generated.
The request priority will be distributed as:
1 – high priority (10% probability)
2 – average priority (70% probability)
3 – low priority (20% probability)
The number of pages will be a random number between 0 and MAXPAGES
The document number will be the next number in the sequence, starting with the number one (1).

b) - the new print job request will be placed in the queue:
The position in the queue will be based on:
- first - priority (lower number (representing higher priority) before a larger (lower priority) number.
- second - within the priority, the number of pages, with the documents with a lower number of pages being positioned before documents in the queue with larger number of pages.

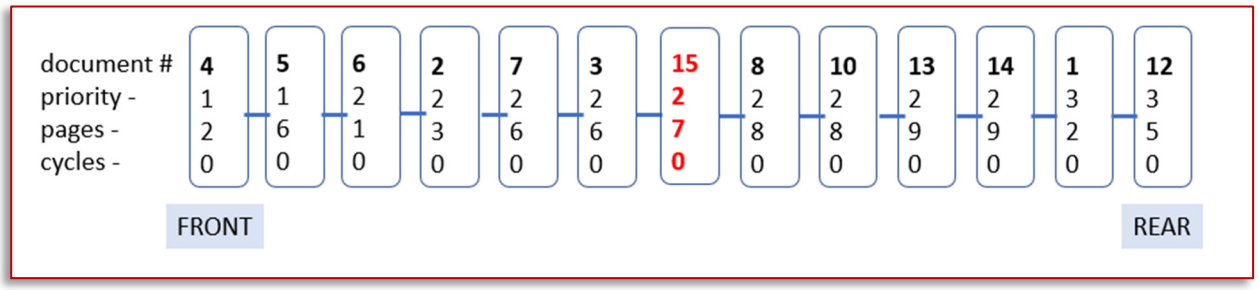
(assume the existing printer queue is represented as the image below)



(if a new printer job request (document 15) is created with an average priority (2) and seven (7) pages, then the new printer job request will be placed immediately before the first existing job of priority 2, page count of 8. (before request for document 8) see image below.)
note: if the new printer job priority was 2 and the number of pages was 6 then the new printer job would be placed in the exact same position – immediately before the higher page count.

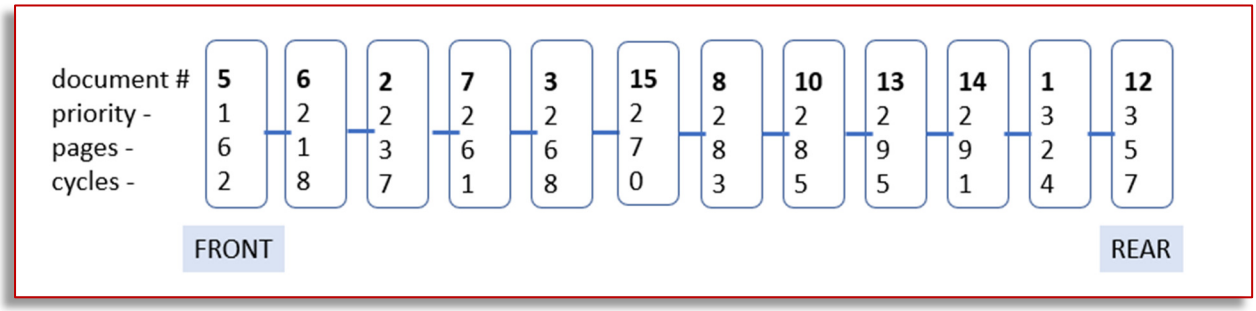


(below is an image of the existing printer queue once the new printer job has been added to the list.)



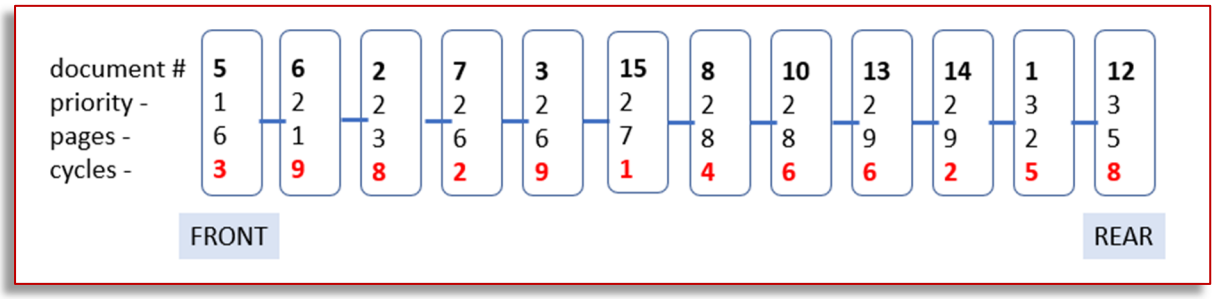
c) – print page(s) of the current active print job:
Check that there is a print job being processed by the printer.
-If there is not a current active printer job, then **remove** the next printer job request from the front of the printer queue. Document 4 has been removed and sent to the printer.

(below is an image of the existing printer queue once the front printer job
(document 4 - priority 1 – pages 2) has been popped off the list and sent to the printer.)



- Decrement the number of pages of the current active print job by the PAGEPERMINUTE value.
- if all the pages are printed after decrementing, then remove the print job from the printer.

d) – increment the cycle count for each print job request in the queue:
Transverse the entire printer queue and increment the cycle count by 1.



After incrementing the cycle count for a print job, check if the cycle count for a print job request has exceeded MAXCYCLES. If so, print out a message if DEBUG_SHOW_EXCEEDED is set to 1.

OUTPUT:

The program will display a message every time a print process is completed.
The message will be:

```
Print Job Completed – Document Number:  xx – Cycle Count:  yy
```

where xx is the document number just completed and yy is the number of cycles the print job required to be completed.

Print out a message indicating the program has completed execution
(note: change to your name).

```
End of Program – * MAX MAGGUILLI *
Number of printer jobs left in queue: xx
```

The program will also have a series of debug statements controlled by preprocessor definitions.

DEBUG_LIST
- print out a listing of printer jobs currently in the queue:

```
Current Printer Queue Size: xx
Current Printer Queue : DocNum: xx  NumofPages xx  PriorityLevel xx  NumOfCycles xx
Current Printer Queue : DocNum: xx  NumofPages xx  PriorityLevel xx  NumOfCycles xx
Current Printer Queue : DocNum: xx  NumofPages xx  PriorityLevel xx  NumOfCycles xx
...
END OF LIST
```

- or if queue is empty:

```
    EMPTY QUEUE – no print jobs currently in queue
```

DEBUG_ADDING
- printout every time a new printer job is added to the queue:

```
    Adding to Queue – Doc: xx  NoPages: xx Priority: xx
```

DEBUG_PRINT_PROCESS
- print out current printing process of active printing document:

```
    PRINTING – DOCUMENT: xx   PAGE: xx   priority: xx
```

DEBUG_SHOW_CYCLES
- print out a list of every printer job after the cycle count has been increased by 1.

```
Increment Cycle – Document: xx Pages: xx Priority: xx Cycle Count: xx
```

DEBUG_SHOW_EXCEEDED
- print out the **first time** a print job cycle count exceeds MAXCYCLES
- prints only **once** when that specific print job cycle count exceeds MAXCYCLES

```
EXCEEDED CYCLE COUNT – Document: xx    Pages: xx    Priority: xx    Cycle Count: xx
```

BONUS – (optional) worth 20% extra added to your assignment five (5) grade.

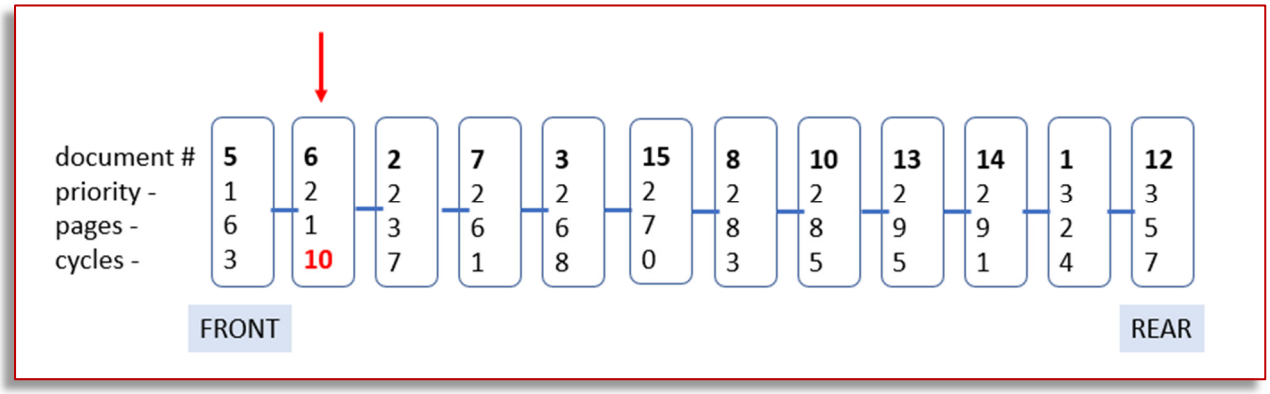
e) – reposition a printer job if the cycle count of an individual printer job has exceeded the specified number of maximum cycles:

After incrementing the cycle count for a print job, check if the cycle count for a print job request has exceeded MAXCYCLES.

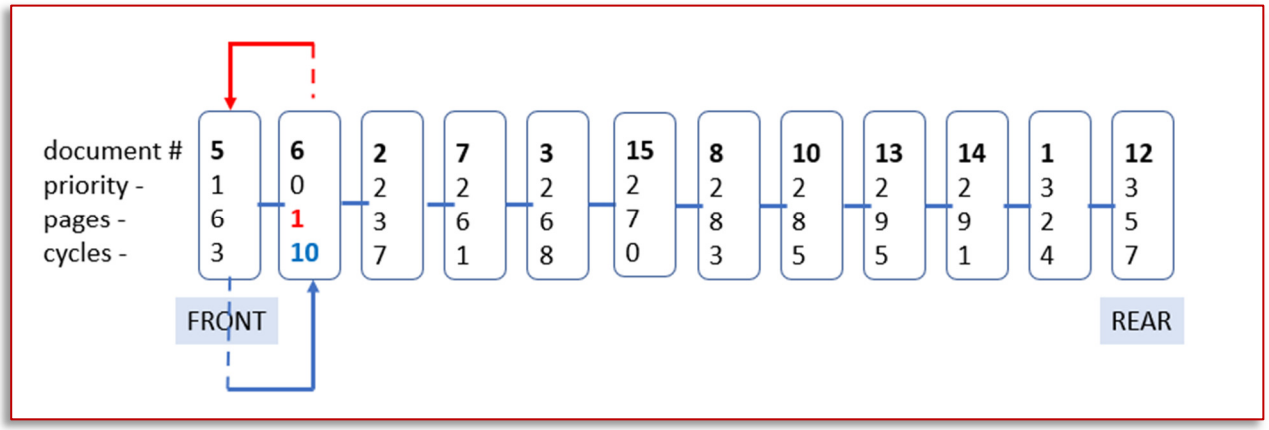
If it has, then change the priority to zero (0) and move the print job request to the front of the queue. BUT! This request MUST be placed at the end of any existing print jobs that also have a priority of zero. If it was placed at the very front as the first item, then some print jobs would never be sent to the printer.

(assume that MAXCYCLES is set to 10. Then once the second print job above (document 6) is processed, its priority will be changed to zero (0) and it will be repositioned (moved) towards the beginning of the printer queue.)

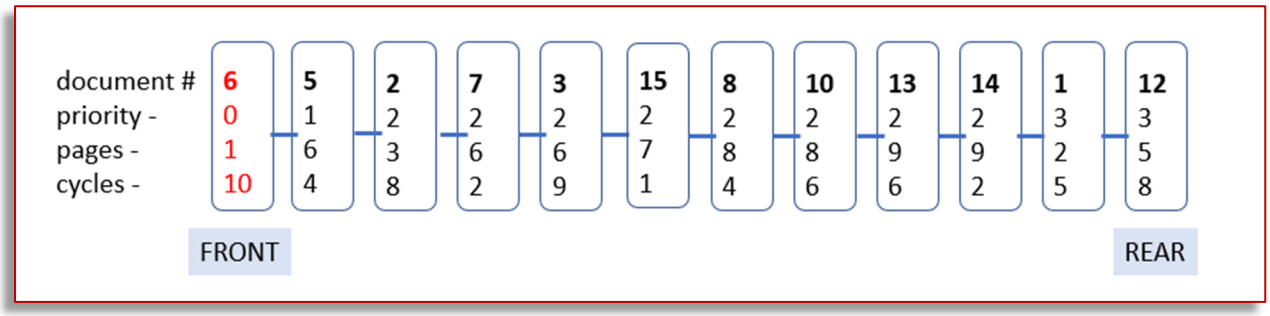
[first, document number 5's cycle count was increased from 2 to 3, then document number 6's cycle count is increased from 9 to 10. The count of 10 exceeds MAXCYCLES …)



so document number 6 has its priority level changed to zero (0) ...

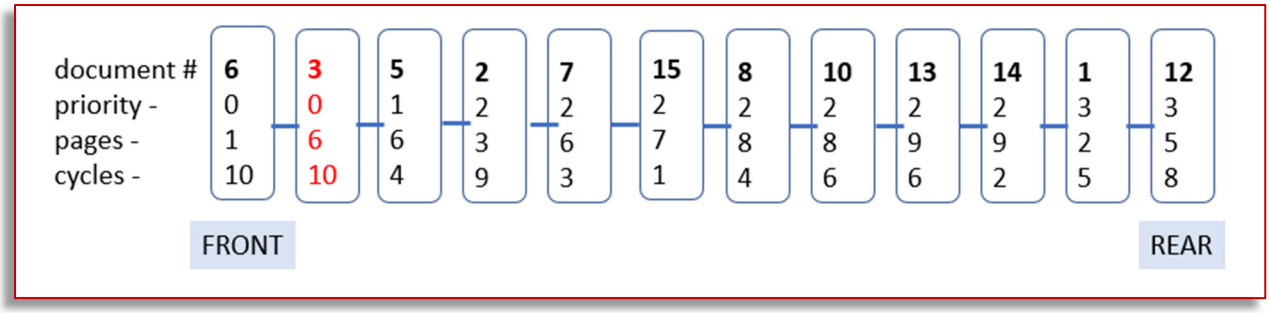


then printer job for document number 6 is moved to the end of all the existing printer jobs with a priority level of zero (0).]



Continue down the print queue and increment the cycle count for the remaining print jobs.
If another print job cycle exceeds MAXCYCLES, repeat the process, but move the print job to the end of the print jobs with priority zero (0).

In the image above, when the cycle count for document 3 (the fifth print job in the queue) is incremented, it will exceed MAXCYCLES and will be moved to the end of any of the print jobs with a priority of zero (0). NOT to the front of the printer queue.



Notice that the print job for document number 3 is now the **second member of the list**. This is because it was placed at the end of all the existing priority zero (0) members. This is regardless of the number of pages in document number 3. The number of pages is not taken into consideration when number of cycles exceeds MAXCYCLES. The print job must be placed at the end of all the existing priority zero (0) members even if the other members with zero (0) priority have a higher page count.

Coding Practices:

Regarding modularity, each of these operations will be contained in their own function and each function will reside in their own **.c file**. The main function will be the program control section.
Regarding incremental programming, each operation will be a separate project, where, after the first project, each one will build off the last.

Correct use of controls will be introduced by forbidding the use of infinite loops with conditional break statements imbedded in the control.

Example:

```
while (1) {
      ... c statements
      ... c statements
      ... c statements
       if (something)
            break;
      ... c statements
      ... c statements
}
```

This is tantamount to using a hammer to drive in a screw. Yes, it works, but it is NOT pretty, and it does NOT do the job correctly. This causes the programmer to look and attempt to validate the logic of the loop termination. This practice will result in major deductions on your assignments.

The break and continue statements are powerful and useful parts of the C language when used correctly. Using it in the above example is counter to the use of the while loop. This assignment may contain a possible location creating an example of the correct usage of the break statement in C code (this is optional).

Your main.c must only contain minimal computation (as demonstrated in class).

Code Standards:

1.) create two (2) header files (similar as was required for assignment four).
one file labeled headers.h
the other file is labeled definitions.h
2.) Global variables are not allowed.
3.) Linked list must be singly linked list (double linked list not allowed).
4.) All structures (and/or unions) are dynamically allocated.
5.) Members of the printer queue are nodes that only contain pointers to other nodes and a pointer to the data item representing a document. (Review the notes for details)

Each print job node in the printer queue will be based on the following definition:

```
// ADT Type Definitions
    typedef struct node
    {
       void*          dataPtr;  // pointer to document structure
       struct node*   next;     // must be a singly linked list
    } LIST_NODE;
```

5.) The preprocessor definitions are to be set as follows:

```
    #define PAGESPERMINUTE 1
    #define MAXCYCLES 200
    #define MAXPAGES 30
    #define ITERATIONS 1000
```

Code Standards:
1.) create two (2) header files.
one file labeled headers.h with the following contents:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "definitions.h"
```

2.) the other file is labeled definitions.h with the following contents:

```
- any preprocessor definitions (e.g. DEBUG)
- all function prototypes.
```

3.) all arrays are to be dynamically allocated based on user input.

Required Coding Standards
All code is to be indented correctly.
Comments at the very beginning (top – first lines) of each of the C code files must be:

```
/* CS2211a 2021 */
/* Assignment 05 */
/* your name */
/* your student number */
/* your UWO User Name */
/* Date Completed */
```

Your program is to be submitted as C code file.
Your script will be a script file created in UNIX.

All variables MUST have a comment describing their intended use(s) demonstrating an understanding of what each variable is used for.

A comment describing the code for each major part of the code is expected. Comment(s) can be added to describe any complete statements you create. They can be brief but must convey what that section of code performs.

Any questions or ambiguities are to be asked through the Forums only. Any individual emails containing questions on this assignment will be replied to with a request to restate the question in the Forums in Owl where they will be addressed.

Working in UNIX.

Save the files in your asn5 directory.

NEXT: Follow the steps below to complete Part 2.

1. Type the following to begin recording your session in a file called yourUserName_asn5.output

   script YourUserName_asn5.output

   note - (using your actual user name).

2. Display the current date and time using the appropriate command

3. Display your username using the appropriate command

4. Display the contents of the current working directory using the 'l' switch (lower case L).

5. Display the contents of the file main.c (i.e. show the main() function in your C program)

6. Compile the program again ensuring the executable is labeled: **`asn5`**
   (yes, even though we have not covered makefiles, it is recommended you use one if you wish)

7. Run the program.

8. Type **`exit`** to stop your screen capture session.

9. Compress the code files and the listed required submission files into a single **.tar.gz** file.
   (i.e. Ensure you have a complete copy of the Asn5_SubmissionForm in your asn5 directory.)
   main.c
   headers.h
   definitions.h
   <all other .c and .h files required to compile the program>
   *YourUserName*_asn5.output
   Asn5_SubmissionForm.txt (or .pdf)

10. Copy (i.e. using an FTP or any method of your choice) the .tar.gz file to your computer so you can upload it through OWL for submission.

Submission Instructions:

Complete the *CS2211 Assignment Submission Form* Name (or rename) that form to **Asn5_SubmissionForm.txt (or Asn5_SubmissionForm.pdf)**

Save this form in your asn5 directory as a text file or as **PDF** (most word processors have this option).

Submit via the CS2211 OWL Web Site the files in your asn5 directory using the instructions on how to compress and submit the single compressed .tar.gz file.

Submit via the CS2211 OWL Web Site the following files inside your compressed submission file:

(all your program files (.c and .h)
*YourUserName*_asn5.output
Asn5_SubmissionForm.txt (or .pdf)

note: do CAN (optional) include your executable file: asn5 (it is easier than trying to remove it ...)
note: you are submitting one tar file (and only one file).
this is a compressed file that contains the files listed above.
do NOT send any of the three above files separately.
note: If you have elected to use an SRA – it must be denoted in the Asn5_SubmissionForm or the SRA will not be applied.
note: marks will deducted if the Asn5_SubmissionForm is omitted.

It is the student's responsibility to ensure the work was submitted and posted in OWL.
OWL replies with a summation verification email (every time).

Submission date and time is based on the last file submitted. So if you re-submit after the due date, the entire assignment will be graded as late based on that timestamp.

The teaching assistant grading your assignment will compile and run your program.
If the program does not compile under UNIX, the TA will NOT attempt to correct or fix your program so it will run.

(*yourUserName* - example: assume my UWO email is kdoit373@uwo.ca
i.e. if my email is – **kdoit373@uwo.ca** then my user name will be – **kdoit373**
So, my UWO User Name is: **kdoit373** and this assignment is **asn5**

It is the student's responsibility to ensure the work was submitted and posted in OWL.
OWL replies with a summation verification email (every time).

Any assignment **not** submitted correctly will **not** be graded.

PS: remember: do your own work – you will need to know all this for the exam to pass !!!!

Please check CS2211 Assignment Submission Guidelines.
note: Guidelines mention the example for assignment 1 – please substitute a 5 for the 1.

Hints on assignment five (5).

How to proceed?

The program tasks have already been broken down in the assignment.

1) Write the code that will dynamically create a print job based on the specifications given.
Test this in a finite loop and ensure
- the frequency (10% chance (i.e. on average, one print job per 10 iterations)).
- the priority distribution is 10% high, 70% average and 20% low priority.
- the random number of pages is in the range of 1 to MAXPAGES

This will be the structure that the *dataPtr in your node will reference (point to)

2) Develop the code that will create and populate the linked list with members consisting of a print job. Ignore insertion order. Simply add each new print job to the front of the list.
Review the notes and use as much of the code as you find useful. This will help keep the code as generic as possible. Each printer job will be a node in the linked list. Each printer job will reference (point to) a document. Creating a dynamically allocated linked list is one of the main tasks of the assignment. Ensure this section of code is completed and tested before moving to the next steps.

algorithm (suggestion only):

```
dynamically allocate a document (using the code from step one)
create a print job node
assign the address of the document to the print job node
add the print job node to the front (top) of the printer queue.
```

3) Write the code that will print out the existing list in order from the front of the printer queue until the end.

algorithm (suggestion only):

```
make the first printer node the current node
loop until done
   print out the details of the data referenced by the current node
   make the next node the current node
```

4) Once all the steps above are done and tested (and work), then write the code to pop the front print job off the printer queue and also to process the current printing document.
hint: use a static declaration for the print document. This way, every time the program loops to the function that processes the current printing document, it retains all the values. This will remove the necessity of passing the current document back to the function.

algorithm (suggestion only):

```
test if there is a document being printed
  if no document currently being be printed
  take the print job node document off the linked list
  make the document referenced by the removed node as current
    print job in printer

  decrement the number of pages by the value in PAGESPERMINUTE
  if number of pages is 0 (or less)
       remove document from printer
```

5) Once all the steps above are done and tested (and work), then write the code to add a new print job to the queue in the correct order based **ONLY** on the priority.

algorithm (suggestion only):

```
make the front print job the current node
loop until new node is added to the printer queue
  if the new node priority level is less than the
      current node priority level
           add the new node to the left of the current node
  else
  make the next node the current node
```

note: what happens if the new node is placed in the front or at the end of the printer queue?

6) Once all the steps above are done and tested (and work), then write the code to add a new print job to the queue in the correct order based on the number of pages.

algorithm (suggestion only): amend the code from the previous step

```
make the front print job the current node
loop until new node is added to the printer queue
  if the new node priority level is equal to the
      current node priority level
          if the new node page number is less than the current page
            number
              add the new node to the left of the current node
          else
             make the next node the current node
  else
  make the next node the current node
```

note: what happens if the new node is placed in the front or at the end of the printer queue?

7) Once all the steps above are done and tested (and work), then write the code to increment the number of cycles for each print job in the queue and test if the cycle count exceeds the MAXCYCLES. If the debug switch is set for this output, print the message to the screen.

algorithm (suggestion only):

```
make the first printer node the current node
loop until done
   increment the cycle count by 1
   if cycle count = MAXCYCLES
     if DEBUG_SHOW_EXCEEDED == 1
          print out message
```

BONUS) Once all the steps above are done and tested (and work), then write the code to increment the number of cycles for each print job in the queue and move the print job if the cycle count exceeds the MAXCYCLES.

algorithm (suggestion only):

```
make the first printer node the current node
loop until done
   increment the cycle count by 1
   if cycle count = MAXCYCLES
      change current print job priority to zero (0)
      find new position in queue
```

```
      move the current print job to the new position
   make the next node the current node
```

This method of breaking down the assignment into manageable steps changes the process from a large task down to a series of smaller, more simple programs. Each of the pieces are created and tested before proceeding to the next.

This will isolate your errors and make them quicker to resolve. One of the more likely errors will be segmentation faults probably due to referencing print job nodes that do not exist. Sometimes this is from going too far past the existing nodes, other times it is from confusing which each node points to during a change. By doing one section at a time, it is easier to trace just that part of the code.

Programs that are attempted in large chunks usually create cascading errors. A segmentation fault might be caused by earlier code and only manifest in a different section.

The key to this approach is that each and every step is completely finished and tested with every possible case before moving on to the next step.

It is very tempting to combine two or more steps in order to ‘save time’. Trust me, this will only lead to taking more time to complete the entire project.

Good luck and remember to have fun with this.