CSC 250
Program 4: Idle Time

Assigned: 4/15/15 **Due: 5/1/15 at Midnight**

Problem Statement:

A local law office as asked you to write a simulation for a printer. The employees have been complaining that they are spending a lot of time waiting for their documents to be printed. The law firm as had the printer checked out and there is nothing wrong with the printer. The current printer server has no way to gather statistics on how long the printer was active or inactive. The office manager would like to know how much the printer is idle. If it is idle a substantial amount of time during the day, it is just unfortunate coincidence that the employees are printing their documents at the same time. Otherwise, the law firm may wish to invest in another printer to accommodate the office staff.

You will work in teams of 4. The teams will be assigned by the instructor.

Program Execution:

Your program will be executed from the command line with 3 command line arguments. You must validate the arguments and if the program was started up incorrectly output a usage statement and exit the program.

```
C:\> prog4.exe   40    3   -r
```

The first argument is the number of seconds on average that documents arrive at the printer. This number will not be less than 31 (guaranteed). In the above command, documents arrive within 40 seconds of each other on average. The manager also states that this will vary with +- 30 seconds. So the range for document arrival is between 10 and 70 seconds inclusive.

The second argument is how many seconds it takes the printer to print one page of a document.

The third argument is where you will retrieve your values from. A –r switch states that any random number will be generated using the random function within C++. A –f states that all random numbers will be retrieved from files. If files are used, you will use two files, "arrival.rand" and "pages.rand". Just hard code in the names and check that they were open successfully.

The Simulation:

Your program will need to store the pages being printed along with some statistics inside of a queue. Your queue will be a template class and manage the arrival and departure of the documents. As the day progresses, pages will arrive at the print queue. You will need to insert this document into the queue. Through out the day, if the queue is empty, the printer is idle.

Otherwise, the printer is printing the document at the front of the queue. Other documents may arrive while the printer is busy printing the current document.

The Queue:

You will write a **template queue** class that can hold any data type. This queue class is to know nothing about arrival times, departure times, or any thing else. It will just hold the information about a document. Be sure to test your queue class with floats and integers before using it within the simulation. Once you have tested your queue, set up the printing queue to hold the following structure.

```
struct document
{
   int pages;                 // number of pages in the document
   int time_arrived;          // number of seconds from the start of day
   int time_dequeued;         // unknown
   int time_started_print;    // unknown
   int time_end_printing;     // unknown
};
```

NOTICE: The last three fields will never be known within the queue. Those fields will be used when a document is removed from the queue. ***DO NOT ATTEMPT*** TO MAKE YOUR QUEUE CLASS FILL IN THESE VALUES. No credit will be given if you do.

A Sample Run:

`Prog4.exe  40 2 -r`

I am using different numbers for easier math.

0 seconds is the start of the day, and a work day contains 8 hours (28800 seconds).
The first document arrives at 15 seconds into the day and contains 10 pages. It is placed into the queue. Since the queue was empty, 15 seconds is added to an idle counter. With each page taking 2 seconds to print, the printer will not become idle for 20 seconds.

The next document arrives 10 seconds after the last document and contains 1 page. It is placed into the queue.
Queue:

	1	2	3	4	5	6	7	8	9
Pages	10	1							
Arrival	15	25							

The next document arrives 60 seconds after the last document and contains 10 pages.
Since its arrival time is 85, all documents in the queue will need to be processed before this document is inserted.

The first document will finish printing at 35 (arrived at 15 seconds and took 20 seconds to print) seconds into the day.

Queue:

	2	3	4	5	6	7	8	9	10
Pages	1								
Arrival	25								

The time is now 35 seconds into the day. The queue is not empty, when you fill in the other values for the document that just got removed, you would notice that the document at the front of the queue would start printing immediately and would take 2 seconds to print this document. The time is now 37 seconds into the day.

There are no more documents to be printed and the next document arrives at 85 seconds into the day. This document is then placed in the queue. Since the queue was empty, 48 seconds (85 – 37) will be added to the idle counter, (now at 63 seconds (15 + 48)).

Queue:

	3	4	5	6	7	8	9	10	11
Pages	10								
Arrival	85								

The next document is determined to arrive at 10 seconds after the last document and has 3 pages. This document is placed into the queue with and arrival time of 95 seconds.

Queue:

	3	4	5	6	7	8	9	10	11
Pages	10	3							
Arrival	85	95							

The next document is determined to arrive at 5 seconds the last document and has 7 pages to print. (Note, the document at the front is not done printing so this gets enqued.)

Queue:

	3	4	5	6	7	8	9	10	11
Pages	10	3	7						
Arrival	85	95	100						

The next document is determined to arrive at 40 seconds after the last document. All documents that would finish printing should be removed, before inserting this document.

You continue to enque and deque documents until the end of day is reached (28800 seconds). Once your timer reaches this point, output the following.

```
Number of seconds between printing documents (arrival): 40
Number of seconds to print a page: 2
Documents printed: 586
Number of seconds printer idle: 5421
```

```
Number of documents still in queue: 1
```

Note: the first two are from the command line.

Determining the arrival time:

Write a function that returns the number of seconds until the next arrival time of a document. If the command line switch was set to be a –r, you will use the random number generator. If it was a –f you will read in an integer from a file. The file will be full of random numbers for you to use as if you were calling a random number generator. This will enable you to repeatedly run your code with the same values for testing as well as enable me to grade your program. You will use the mod operator to get the value down into the range specified +/- 30 seconds.

Example: number generated or read from a file is 3278

If the arrival time is 50 then this number must be mapped to a number between 20 and 80.

Determining the Page count:

Write a function that returns the number of pages in a document. The number of pages will be between 10 and 20 inclusively, ALWAYS – NO EXCEPTIONS. Again, this number can come from the random number generator or a file. You will need to use the mod operator to get the value down into the range 10 to 20 inclusive.

Example: number generated or read from a file is 78

This number then needs to be mapped to a value between 10 and 20.

Random number Generator:

You will need to include to system header files <ctime> and <cstdlib>.
To retrieve a random number you will use the rand() function.
This function returns a value between 0 and 32765.
The rand function will return the same number series each time the program is run. After you have your program working change this behavior by adding the line

`srand( (int) time(NULL));` near the top of main.

This will initialize the random number generator using the current time as a seed.

Use:
Suppose you were to roll a dice for a board game. Values should be between 1 and 6 inclusive. The rand function does not give us these values, but you can use the mod operator to achieve this.

```
rand() % 6; //  gives values between 0 and 5 inclusive
```

So

```
rand() % 6 + 1; // gives a value between 1 and 6 inclusive
```

Suppose you need a value between 50 and 100 inclusive

```
Rand() % (100 – 50 ); // gives a values 0 and 49 inclusive
```

```
Rand() % (100 - 50) + 50; // gives a values 50 and 99
```
Almost
```
rand() % (100 - 50 + 1 ); // gives values 0 -> 50
rand() % (100 - 50 + 1) + 50; // gives values 50 to 100
```

Algorithm:

1. process command line
2. generate document and insert into queue if necessary
   If queue is empty, increment counter
3. remove all printed documents necessary.
4. move clock counter to time along as necessary.
5. repeat from step 2 until clock counter becomes 28800.
6. Print stats

Hints and Other Notes:

1. Do not do long session of programming, this leads to silly errors and mistakes.
2. Save and compile often.
3. Work with your partners and listen to them when they have an idea.
4. Do not wait until the last minute.
5. Be sure to follow the coding style guidelines that can be found on the web server.
6. I do not fix programs on the day they are due.
7. Your source file must be called **prog4.cpp** and **queue.h**
8. Do not wait until the last minute
9. Seek help when stuck on a problem.
10. Your program sources will be named queue.h and prog4.cpp. Place these files in a zip compressed file and submit via the website.
11. Each Partner is to evaluate their partner(s) performance and their own performance. This will be submitted via the website by each partner. Be honest. Your partner will never know what is said.

Timeline:

4 / 15:	Get team members contact information:
4 / 17:	Process command line arguments
4 / 24:	Have queue written and tested
4 / 26:	Have get_arrival and get_pages written and tested
4 / 28:	Generate documents and fill to end of day. (do not worry about idle time)
4 / 30:	Have simulation complete (idle time)
5 / 1:	Submit cpp , h file , and teammate evaluations via the mcs web site.

Each Partner is to evaluate their partner(s) performance and their own performance. This will be submitted via the website by each partner. Be honest. Your partner will never know what is said.