# CSC 225

**Program 2**

Due: Tuesday, December 13, 2022 at 3:00pm – No late days may be used.

# Instructions

|  |
| --- |
| 1. **You must work on this assignment individually.** You may ask another person to look over your project for errors, but they cannot instruct you the correct way to write. (For example: the person helping can say “The problem is in the ‘switch’ statement” or “Your logic is incorrect in the second ‘if’ statement.”) In addition, you may not use code from other sources, such as the Web or other students. 2. You will need to upload the .cpp and .h files from your solution to eLearn. |

Your task in this program will be to develop a program that simulates the First Come First Served and Round Robin methods of CPU scheduling. CPU scheduling is the procedure of assigning processes waiting to use the CPU an order in which to use it. FCFS is self-explanatory. Round Robin makes use of a *time quantum*, which is the maximum amount of time a process is run before it is sent to the end of the list waiting to use the CPU. One metric used to measure the effectiveness of a CPU scheduling algorithm is the amount of time each process spends waiting in the list to use the CPU before it finishes. The time a process spends running is not included in waiting time.

Consider the processes below, which are waiting in the order given in the list to use the CPU:

|  |  |
| --- | --- |
| **Process** | **Duration** |
| A | 3 |
| B | 2 |
| C | 5 |
| D | 4 |

Using FCFS scheduling, process A will run to completion, then B will run to completion, and so on. Process A has no waiting time since it is first. Process B has a waiting time of 3 since it had to wait for Process A to finish. Process C has a waiting time of 5, since it had to wait for Processes A and B. Process D has a waiting time of 10.

Using Round Robin scheduling with a time quantum of 2 seconds, Process A will run for 2 seconds and then the CPU will move on to the next process. Process B will be able to complete within its time quantum, having waited for 2 seconds, so it will be removed from the list. Then, Processes C and D will each run for 2 seconds. Next, Process A will finish up by running 1 more second, having waited 6 total seconds. Then, Process C will run for 2 seconds. Process D will then finish, having waited a total of 9 seconds. Finally, Process C will finish with a total wait time of 9. When it is removed, the list of processes waiting for the CPU will be empty.

Using Round Robin scheduling with a time quantum of 3, Process A will be able to complete within its time quantum, not having waited, and will be removed from the list. Process B will be able to complete within its time quantum and be removed, having waited for 3 seconds. Then, Process C and D will each run for 3. Next, Process C will run for 2 and finish, having waited for a total of 8 seconds. Finally, Process D will finish, having waited a total of 10 seconds.

You will need to define a Process class for this program, based on the slides from November 28.

When the program begins, it should ask the user for the name of a file containing information about the processes to be used in the program. The first value in the file should be the number of processes to be added to the system. Then, the file should contain the name and duration of each process. These processes should be added to the waiting list (represented as a circular linked list of Process objects) as they are read in. After reading in the file, the user should be able to select whether to use FCFS or Round Robin scheduling, and the program should ask for the time quantum if Round Robin is chosen. Then, it should run the specified algorithm, and add the Process objects to a different circular linked list *in the order in which they finish*. Finally, the program should print a well-formatted table summarizing the results (see below). The processes should appear in the table in the order in which they finished.

You must also create the following functions as specified and use them in your main program. You may create other functions as you see fit.

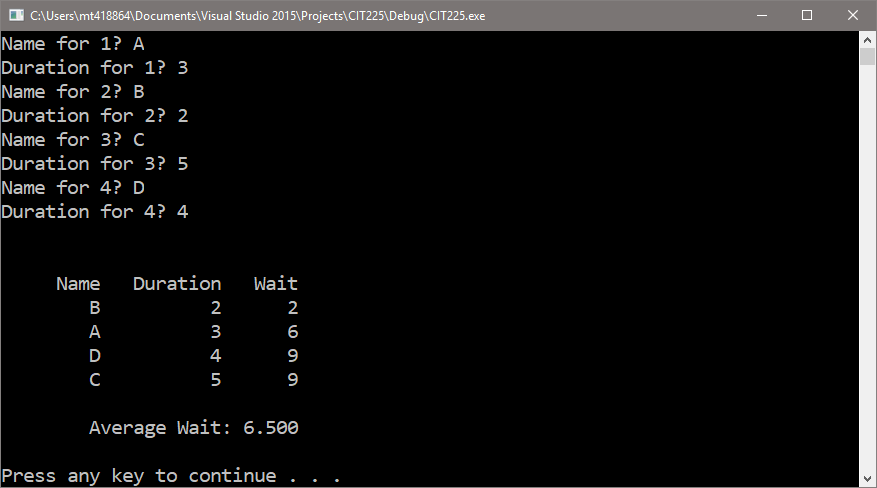
| **Function Prototype** | **Description** |
| --- | --- |
| void appendProcess(ProcessNode\*&, ProcessNode\*&, Process) | Appends a process to the circular list |
| void removeProcess(ProcessNode\*&, ProcessNode\*&, Process) | Removes (but does not delete) the Process from the circular list having the same name as the Process passed to the function |
| void printProcesses(ProcessNode\*) | Prints the output of the program, given the head of the list of finished processes |

In addition to the program functioning correctly as described above, you will also be graded on:

* Using appropriate variable names, data types, and constants
  + Limit globals
  + Declare variables at the beginning of your functions
  + Name constants using all capital letters
* Using comments appropriately, including header comments
* Formatting your code appropriately using tabs and blank lines
* Having user-friendly and well-formatted input and output
* Appropriate data validation

Additional information and requirements:

* Make sure you understand the above requirements before you begin. Dr. Thompson can provide any clarifications you need
* The results should look like the screenshot below:



* To clear the screen, use the call system(“cls”)
* Since you will be using an object as part of a linked list, you will be using notation similar to  
  head->data.getName()since head->data is a Process object
* Start with your Process class and then getting FCFS to work completely, then move on to Round Robin
* Destroy your second list when the program is finished
* Both linked lists must be circular and have tails

In order to receive *any* credit for this assignment, **all** of the following must be completed:

* Your *Process* class should compile and pass all the tests in *TestProcess.cpp* (on eLearn)
* Your main program should compile with no errors and:
  + The *appendProcess*, *removeProcess*, and *printProcesses* functions should be implemented correctly
  + It should read the processes from the file and print them in some order