National Yang Ming Chiao Tung University
Department of Electrical Engineering
Data Structures, Fall 2021

October 12, 2021
Instructor: Mark P.-H. Lin
TA: C.-L. Ko

Programming Assignment #2

Stacks & Queues

1 Problem Description

In parallel computing, ***work stealing*** is a scheduling strategy for multithreaded computer programs on multiple processors, or multiple processor cores. Each processor has a queue of pending tasks to be executed. When a processor runs out of tasks in its queue, it looks at other processors' queues and "steals" a task from them to avoid idling. Consequently, the task load is dynamically balanced over the whole processors.

In this programming assignment, you are asked to write a program to simulate the behavior of "work stealing" in a computer system, which contains a set of N processors, where $0 < N < 100$. The size of each processor's queue could be as large as 10^6, which means that a processor can store at most 10^6 pending tasks in the queue.

Initially, each processor's queue is empty, which means that there is no pending task for each processor. Your program, or the task scheduling simulator, will receive a sequence of the following commands.

- **ASSIGN** [**processor ID**] [**new task name**]: When the simulator receives this command, it will assign the new task to the corresponding processor by storing the task in the processor's queue. The "processor ID" is an integer ranging from 1 to N, while the "task name" is represented by a character of any capital letter ranging from "A" to "Z". Different new tasks may have the same task name.
- **EXEC** [**processor ID**]: When the simulator receives this command, it will execute the first task in the processor's queue by removing that task from the queue. However, if that processor's queue is empty, it will steal one task from another processor according to the following rules: 1) Select the processor having the maximum number of queuing tasks; 2) If two or more processors have the same maximum number of queuing tasks at that moment, select the one with the smallest processor ID among them; 3) Steal the last pending task from the selected processor by removing the last task from that processor's queue.
- **STOP**: When the simulator receives this command, it will stop simulation and output both queuing and executed tasks of all processors at that moment.

2 Input Format

The first line of the input file (e.g. "sample.in") to your program gives the number of processors in the computer system, and then followed by a sequence of **ASSIGN** and **EXEC** commands. The **STOP** command will appear in the last line of the input file.

Sample Input

```
4
ASSIGN 1 F
EXEC 4
ASSIGN 4 J
ASSIGN 1 G
EXEC 3
ASSIGN 1 C
ASSIGN 3 R
ASSIGN 4 G
ASSIGN 3 J
ASSIGN 4 A
EXEC 1
EXEC 2
EXEC 4
STOP
```

Based on the processor number and command sequence given in the sample input, the queuing and executed tasks of each processor after receiving each command are demonstrated below.

Command	Processor 1	Processor 2	Processor 3	Processor 4
ASSIGN 1 F	$\boxed{F}$			
EXEC 4				F
ASSIGN 4 J				$\boxed{J}$ F
ASSIGN 1 G	$\boxed{G}$			$\boxed{J}$ F
EXEC 3			G	$\boxed{J}$ F
ASSIGN 1 C	$\boxed{C}$		G	$\boxed{J}$ F
ASSIGN 3 R	$\boxed{C}$		$\boxed{R}$ G	$\boxed{J}$ F
ASSIGN 4 G	$\boxed{C}$		$\boxed{R}$ G	$\boxed{J}\boxed{G}$ F
ASSIGN 3 J	$\boxed{C}$		$\boxed{R}\boxed{J}$ G	$\boxed{J}\boxed{G}$ F
ASSIGN 4 A	$\boxed{C}$		$\boxed{R}\boxed{J}$ G	$\boxed{J}\boxed{G}\boxed{A}$ F
EXEC 1	C		$\boxed{R}\boxed{J}$ G	$\boxed{J}\boxed{G}\boxed{A}$ F
EXEC 2	C	A	$\boxed{R}\boxed{J}$ G	$\boxed{J}\boxed{G}$ F
EXEC 4	C	A	$\boxed{R}\boxed{J}$ G	$\boxed{G}$ F J
STOP	C	A	$\boxed{R}\boxed{J}$ G	$\boxed{G}$ F J

3 Output Format

Your program will generate the output file (e.g. "sample.out"), which contains both queuing and executed tasks of each processor in the following format, "[**processor ID**] [**task names**]". The list of processors must be in the ascending order of their IDs, and the list of each processor's tasks must follow the executing and queuing sequence. There is no spacing between task names.

Sample Output
Queuing
1
2
3 RJ
4 G
Executed
1 C
2 A
3 G
4 FJ

4 Command-line Parameter

In order to correctly test your program, you are asked to add the following command-line parameters to your program.

[executable file name] [input file name] [output file name]

(e.g., StudentID.exe sample.in sample.out)

5 Submission Information

Your program must be written in the C/C++ language, and can be compiled on the Linux platform. The source files of your program must be named with "[your student ID].h" and "[your student ID].cpp". The executable file name of your program must be "[your student ID].exe". To submit your program, please archive both executable and source files of your program into a single zip file, named "[your student ID].zip", and upload to E3.

For this programming assignment, you are NOT allowed to use the following containers in the standard template library (STL), including stack, queue, priority_queue, deque, list, forward_list, and string. Therefore, the following headers CANNOT be included in your source code: <stack>, <queue>, <deque>, <list>, <forward_list>, <string>.

6 Due Date

The zip file must be submitted through E3 before 23:59, October 23, 2021.

7 Grading Policy

The programming assignment will be graded based on the following rules:

- Pass sample input with compilable source code (50%)
- Pass five hidden test cases (50%)

The submitted source codes, which are copied from or copied by others, will NOT be graded. There will be 25% penalty per day for late submission.