CS 3113 Fall 2025 – Project 2
Due Monday, October 27th at 11:59 PM

In this project you need to implement a CPU scheduler simulator that use the preemptive priority with round robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. Each process may arrive at different time. Processes with higher priority will always be scheduled ahead of the lower priority processes. If there are multiple processes have the same priority level, then they will be scheduled using round robin. If a process is currently using the CPU yet another process with higher priority arrives, the running process will be preempted by the new arrival process. In addition to the processes, the system also has an idle task (which consumes no CPU resources. This task has priority 0 and is scheduled whenever the system has no other available processes to run.

Below is the sample input.txt

```
q = 10
P1 40 20 0
P2 30 25 25
P3 30 25 30
P4 35 15 60
P5 5 10 100
P6 10 10 105
```

Input explanation:
q 10 means time quantum for round robin = 10
P1 40 20 0 means Process P1 with Priority 40, CPU Burst Time 20, and Arrival Time 0.
The same meaning applies to P2, P3, P4, P5, and P6.
The output must look as follows **(do not add extra space at the end of the line)**:

```
Time 0-20: P1 (Priority 40)
Time 20-25: Idle
Time 25-35: P2 (Priority 30)
Time 35-45: P3 (Priority 30)
Time 45-55: P2 (Priority 30)
Time 55-60: P3 (Priority 30)
Time 60-75: P4 (Priority 35)
Time 75-80: P2 (Priority 30)
Time 80-90: P3 (Priority 30)
Time 90-100: Idle
Time 100-105: P5 (Priority 5)
Time 105-115: P6 (Priority 10)
Time 115-120: P5 (Priority 5)

Turnaround Time
P1 = 20
P2 = 55
P3 = 60
P4 = 15
P5 = 20
P6 = 10

Waiting Time
P1 = 0
P2 = 30
P3 = 35
P4 = 0
P5 = 10
P6 = 0

CPU Utilization Time
105/120
```

Write the CPU Scheduler simulator in C++. You can modify the PCB structure used in Project 1 as follows:

```
// Structure to represent a process control block
struct PCB {
    string id;
    int priority;
    int burst_time;
    int arrival_time;
    int remaining_time;
    // feel free to add more variables here
};

int main() {
    string line;
    char dummy;
    int time_quantum;
    vector<PCB> processes;

    // Read time quantum from standard input
    cin >> dummy >> time_quantum;

    // Read processes from standard input
    while (cin >> line) {
        PCB p;
        p.id = line;
        cin >> p.priority >> p.burst_time >> p.arrival_time;
        p.remaining_time = p.burst_time;
        p.last_run_time = -1;
        p.remaining_quantum = time_quantum;
        processes.push_back(p);
    }

    // TODO: Create CPU scheduler simulator
    // You can create any data structures, classes, functions helpers as you wish
    // Do not forget to include comments describing how your simulator works.
    return 0;
}
```

Test your program

Once you modify your program, you should compile and run it. Make sure you keep the input file (e.g input.txt) in the same directory where your program is located.

Linux User
Open terminal and execute the following commands:

g++ -std=c++11 project2.cpp -o project2
./project2 < input.txt

MacOS User
Open terminal and execute the following commands:

clang++ -std=c++11 project2.cpp -o project2
./project2 < input.txt

If you do not have clang++ in your MacOS, it means you first need to install XCode by running the following command on the terminal:

xcode-select --install

Windows User
If you have installed VSCode for the Data Structure class, you can use its terminal to run the same command as in Linux.

If you do not have C, C++ compiler installed in your Windows machine, you can try installing it using one of the following options:

- https://code.visualstudio.com/docs/cpp/config-mingw

Rules and Submission Policy

All projects in this course are **individual assignments** and are not to be completed as group work. Collaboration with others or the use of outside third parties to complete this project is strictly prohibited. Submissions must be made through **GradeScope**, where automated grading will be conducted. Additionally, manual grading may be performed to ensure correctness and adherence to requirements.

Several input files will be used to evaluate your program. While a subset of these input files will be provided to you for testing, additional files not shared beforehand will also be used during grading. Your score on this project will depend on producing correct results for all input files, including the undisclosed ones used in GradeScope evaluation.

All programs must be written in **C or C++** and must compile successfully using the **GCC** or **GNU C++ compiler**. It is your responsibility to ensure that your program adheres to these requirements.
The course syllabus provides more details on the late and submission policy. You should also go through that.