CS 3113 Fall 2025 – Project 1
Due Wednesday, September 24th at 11:59 PM

In this project, you will simulate a simple kernel keeping track of processes context in their Process Control Blocks (PCBs) while managing processes using round-robin scheduling.

For this simple kernel simulator, let's assume that each process only run CPU bound instructions. Hence its state will only be Ready', 'Running', or 'Terminated'. Each PCB will contain the following information:

- process id (pid)
- program counter (pc)
- state
- total work units needed.

Round-robin scheduling is a simple method that can be used by operating systems to manage multiple processes sharing the CPU. It works by giving each process a small, fixed amount of CPU time called a time quantum (or time slice). Processes take turns running in a circular order, like taking turns in a game. Here's how it works:

- Processes are placed in a ready queue.
- The first process in the queue gets to run for the time quantum (e.g., 2 units).
- If the process finishes within the time quantum, it's done and removed from the queue.
- If it's not finished, it's moved to the back of the queue, and the next process runs.
- This repeats until all processes are complete.

Think of it like a relay race where each runner (process) runs for a short time before passing the baton to the next runner, ensuring everyone gets a fair chance without any single process hogging the CPU.

Your task:

You are going to complete **pcb_simulation.cpp**. A print and main functions are provided for you. Do not modify these functions except the TODO part in the main function.
The program reads process details from standard input via redirection (e.g.,./pcb_simulation < input.txt), creates PCBs dynamically, and simulates execution with a fixed time quantum of 2 units. After each time slice, it outputs the state of each process (Ready, Running, Terminated) in a standardized format (check the output samples). Your primary task is to implement the kernelSimulator function, which manages the round robin scheduling including the context switching. You can create additional helper functions to support your implementation.

Input and Output Explanations

Input Format

The input is read from standard input (std::cin) via redirection, typically from a text file (e.g., input.txt). The format consists of:

- First Line: A single positive integer N, representing the number of processes ($1 \leq N \leq 10$, to keep the simulation manageable).
- Subsequent N Lines: Each line contains two positive integers separated by a space:
    - PID: A unique process ID (positive integer, e.g., 1, 2, 3).
    - Work Units: The total number of work units the process requires to complete (positive integer, e.g., 1–20 for simplicity).

Constraints:

- N must be positive (N > 0).
- PIDs must be unique within the input.
- Work units must be positive (work > 0).
- Input must contain valid integers (no non-numeric values).
- No extra spaces, blank lines, or invalid characters should be present, as std::cin expects clean input.

Sample Input File (input.txt)

```
3
1 6
2 3
3 4
```

Line 1: 3 indicates there are 3 processes.
Line 2: 1 6 defines a process with PID 1 requiring 6 work units.
Line 3: 2 3 defines a process with PID 2 requiring 3 work units.
Line 4: 3 4 defines a process with PID 3 requiring 4 work units.
Total Work Units: 6 + 3 + 4 = 13.

Expected output for the given input

```
Interrupt 1:
PID 1: Running, at pc 2
PID 2: Ready, at pc 0
PID 3: Ready, at pc 0
Interrupt 2:
PID 1: Ready, at pc 2
PID 2: Running, at pc 2
PID 3: Ready, at pc 0
Interrupt 3:
PID 1: Ready, at pc 2
PID 2: Ready, at pc 2
PID 3: Running, at pc 2
Interrupt 4:
PID 1: Running, at pc 4
PID 2: Ready, at pc 2
PID 3: Ready, at pc 2
Interrupt 5:
PID 1: Ready, at pc 4
PID 2: Terminated, at pc 3
PID 3: Ready, at pc 2
Interrupt 6:
PID 1: Ready, at pc 4
PID 2: Terminated, at pc 3
PID 3: Terminated, at pc 4
Interrupt 7:
PID 1: Terminated, at pc 6
PID 2: Terminated, at pc 3
PID 3: Terminated, at pc 4
All processes completed.
```

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 pcb_simulator.cpp -o pcb_simulator

./pcb_simulator < input.txt
```

You can also replace gcc with g++

MacOS User
Open terminal and execute the following commands:

```
clang++ -std=c++11 pcb_simulator.cpp -o pcb_simulator
./pcb_simulator < 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://www.freecodecamp.org/news/how-to-install-c-and-cpp-compiler-on-windows/
- https://medium.com/@_arupbasak_/setup-c-c-compiler-easiest-way-470db3f1000c

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.