Scenario

You have been hired by a top-tier company as an OS specialist. The company is in the process of developing its own operating system, and your responsibility is to create an efficient scheduler for this operating system currently in development. The decision has been made to categorize processes into two groups: background processes and foreground processes. Processes are designated as "Background" or "Foreground" based on their priority. If a process's priority is less than 5, it falls into the background category, while those with a priority of 5 or higher are considered foreground processes.

The number of processes can vary, and their burst times are unknown. However, it is known that there are fewer background processes and a larger number of foreground processes.

Selecting the right scheduling algorithm for this task has been a challenging decision for you. The primary goal is well-defined: to maximize CPU utilization while minimizing both process starvation and the average waiting time for processes.

To fulfill the above requirement, you have finally selected two algorithms: Shortest Remaining Time First and Priority based scheduling.

You are required to write a program which asks the user about the number of processes, their respective priority, their burst, and arrival times. Calculate the average waiting time using the following algorithms and compare the results:

- Priority based scheduling. [35 minutes]
- Shortest Remaining Time First. [35 minutes]
- By using a multilevel feedback queue. The foreground processes are placed in a queue named foreground queue while the background processes are placed in a queue named background queue. The background queue uses Shortest Remaining Time First algorithm to schedule the processes while the foreground queue uses Priority based algorithm to schedule the processes present in the background queue. At the start of the program, the CPU is always given to the foreground queue, after 10 seconds it is given to the background queue, after 3 seconds it is again given to the foreground queue and so on. [60 minutes]

Hints:

- Use built-in STL queue library for maintaining background and foreground queues. See the sample queue code on next page.
- You can use sleep function to simulate that the CPU is given to a specific process.
- If process’s burst time is not zero, and CPU is taken away from that process, add it to the queue with the updated burst time.

```cpp
#include <iostream>
#include <queue>

using namespace std;

// Print the queue
void showq(queue<int> gq)
{
    queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}

// Driver Code
int main()
{
    queue<int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);

    return 0;
}
```