### **Pseudocode for Round Robin Scheduling**

1. **Input**:
   * n: Number of processes.
   * timeQuantum: Time quantum.
   * For each process P[i], take input:
     + arrivalTime[i]: Arrival time of the process.
     + burstTime[i]: Burst time of the process.
2. **Initialize**:
   * Set remainingTime[i] = burstTime[i] for all processes.
   * Set waitingTime[i] = 0, turnAroundTime[i] = 0.
   * Create a queue to manage processes based on arrival time.
   * Set currentTime = 0, completed = 0.
3. **Start Execution**:
   * Add processes to the queue in the order of their arrival time.
4. **While there are incomplete processes**:
   * **Step 4.1**: Dequeue the process P[i] from the front of the queue.
   * **Step 4.2**: Execute the process for a minimum of:
     + timeQuantum, or
     + remainingTime[i].
   * Update:
     + remainingTime[i] -= timeExecuted.
     + currentTime += timeExecuted.
   * **Step 4.3**: If the process completes (remainingTime[i] == 0):
     + Compute:
       - completionTime[i] = currentTime.
       - turnAroundTime[i] = completionTime[i] - arrivalTime[i].
       - waitingTime[i] = turnAroundTime[i] - burstTime[i].
       - Increment completed.
   * **Step 4.4**: If the process is not completed:
     + Re-enqueue the process to the end of the queue.
   * Add any newly arrived processes to the queue.
5. **Output Results**:
   * Print:
     + Process ID, Arrival Time, Burst Time, Waiting Time, and Turnaround Time for each process.
   * Print:
     + Average Waiting Time.
     + Average Turnaround Time.

#include <stdio.h>

#include <stdbool.h>

struct Process {

int id;

int arrivalTime;

int burstTime;

int remainingTime;

int waitingTime;

int turnAroundTime;

int completionTime;

};

int main() {

int n, timeQuantum;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

int queue[100], front = 0, rear = 0;

printf("Enter Time Quantum: ");

scanf("%d", &timeQuantum);

// Input process details

for (int i = 0; i < n; i++) {

processes[i].id = i;

printf("Enter Arrival Time and Burst Time for Process P%d: ", i);

scanf("%d%d", &processes[i].arrivalTime, &processes[i].burstTime);

processes[i].remainingTime = processes[i].burstTime;

processes[i].waitingTime = 0;

processes[i].turnAroundTime = 0;

}

int currentTime = 0, completed = 0;

bool isInQueue[n];

for (int i = 0; i < n; i++) {

isInQueue[i] = false;

}

// Enqueue the first process

queue[rear++] = 0;

isInQueue[0] = true;

while (completed < n) {

// Dequeue the process

int currentProcess = queue[front++];

front %= 100;

// Execute the process

int timeExecuted = (processes[currentProcess].remainingTime > timeQuantum) ? timeQuantum : processes[currentProcess].remainingTime;

processes[currentProcess].remainingTime -= timeExecuted;

currentTime += timeExecuted;

// Check for new arrivals during the execution

for (int i = 0; i < n; i++) {

if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0 && !isInQueue[i]) {

queue[rear++] = i;

rear %= 100;

isInQueue[i] = true;

}

}

// If the process is completed

if (processes[currentProcess].remainingTime == 0) {

processes[currentProcess].completionTime = currentTime;

processes[currentProcess].turnAroundTime = processes[currentProcess].completionTime - processes[currentProcess].arrivalTime;

processes[currentProcess].waitingTime = processes[currentProcess].turnAroundTime - processes[currentProcess].burstTime;

completed++;

isInQueue[currentProcess] = false;

} else {

// Re-enqueue the process if not completed

queue[rear++] = currentProcess;

rear %= 100;

}

}

// Output Results

printf("\nPROCESS\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

float totalWaitingTime = 0, totalTurnAroundTime = 0;

for (int i = 0; i < n; i++) {

totalWaitingTime += processes[i].waitingTime;

totalTurnAroundTime += processes[i].turnAroundTime;

printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id,

processes[i].arrivalTime,

processes[i].burstTime,

processes[i].waitingTime,

processes[i].turnAroundTime);

}

printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);

printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);

return 0;

}