# ***EXPERIMENT-1***

## ***Q Write a program to implement Round Robin Scheduling Algorithm***

```
#include <iostream>
#include <queue>
using namespace std;
struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int remainingTime;
};
void roundRobinScheduling(Process processes[], int n, int quantum) {
    queue<Process*> readyQueue;
  int currentTime = 0;
    int totalWaitingTime = 0;
    int totalTurnaroundTime = 0;
  int completedProcesses = 0;
  // Initialize remaining time for each process
    for (int i = 0; i < n; ++i) {
        processes[i].remainingTime = processes[i].burstTime;
    }

    // Run until all processes are completed
    while (completedProcesses < n) {
        // Add processes to the ready queue that have arrived by the current time
        for (int i = 0; i < n; ++i) {
            if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0) {
                readyQueue.push(&processes[i]);
            }
        }
```

```
 // If ready queue is empty, move time forward to the arrival time of next process
        if (readyQueue.empty()) {
            currentTime++;
            continue;
        }
// Process current process in the ready queue
        Process* currentProcess = readyQueue.front();
        readyQueue.pop();
          // Execute process for the quantum time or for its remaining time, whichever is smaller
        int executionTime = min(quantum, currentProcess->remainingTime);
        currentProcess->remainingTime -= executionTime;
        currentTime += executionTime;
          // If the process is completed, calculate waiting and turnaround times
        if (currentProcess->remainingTime == 0) {
            completedProcesses++;
            int waitingTime = currentTime - currentProcess->arrivalTime - currentProcess->burstTime;
            int turnaroundTime = currentTime - currentProcess->arrivalTime;
            totalWaitingTime += waitingTime;
            totalTurnaroundTime += turnaroundTime;
        cout << "Process " << currentProcess->id << ": Waiting Time = " << waitingTime << ", Turnaround Time = "
<< turnaroundTime << endl;
        } else {
            // If process is not completed, put it back in the ready queue
            readyQueue.push(currentProcess);
        }
    }
      // Calculate average waiting and turnaround times
    double avgWaitingTime = static_cast<double>(totalWaitingTime) / n;
    double avgTurnaroundTime = static_cast<double>(totalTurnaroundTime) / n;
   cout << "Average Waiting Time = " << avgWaitingTime << endl;
    cout << "Average Turnaround Time = " << avgTurnaroundTime << endl;
```

```
}
int main() {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
   Process processes[n];
 for (int i = 0; i < n; ++i) {
        processes[i].id = i + 1;
        cout << "Enter arrival time and burst time for process " << i + 1 << ": ";
        cin >> processes[i].arrivalTime >>
```

## *Q2 Write a program to implement priority scheduling algorithm.*

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Process {
    int id;
    int arrivalTime;
    int burstTime;
    int priority;
    int waitingTime;
    int turnaroundTime;
};
bool compareArrivalTime(const Process& p1, const Process& p2) {
    return p1.arrivalTime < p2.arrivalTime;
}
bool comparePriority(const Process& p1, const Process& p2) {
    return p1.priority < p2.priority;
}
void priorityScheduling(Process processes[], int n) {
    // Sort processes based on arrival time
    sort(processes, processes + n, compareArrivalTime);
```

```
    int currentTime = 0;
    int totalWaitingTime = 0;
    int totalTurnaroundTime = 0;
    for (int i = 0; i < n; ++i) {
        // Process arrives after the current time, wait for it
        if (currentTime < processes[i].arrivalTime) {
            currentTime = processes[i].arrivalTime;
        }
    processes[i].waitingTime = currentTime - processes[i].arrivalTime;
        totalWaitingTime += processes[i].waitingTime;

        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
        totalTurnaroundTime += processes[i].turnaroundTime;

        currentTime += processes[i].burstTime;
    }

    // Calculate average waiting and turnaround times
    double avgWaitingTime = static_cast<double>(totalWaitingTime) / n;
    double avgTurnaroundTime = static_cast<double>(totalTurnaroundTime) / n;

    cout << "Process\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < n; ++i) {
        cout << processes[i].id << "\t" << processes[i].waitingTime << "\t\t" << processes[i].turnaroundTime << endl;
    }

    cout << "Average Waiting Time = " << avgWaitingTime << endl;
    cout << "Average Turnaround Time = " << avgTurnaroundTime << endl;
}
int main() {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;
```

```
    Process processes[n];

    for (int i = 0; i < n; ++i) {
        processes[i].id = i + 1;
        cout << "Enter arrival time, burst time, and priority for process " << i + 1 << ": ";
        cin >> processes[i].arrivalTime >> processes[i].burstTime >> processes[i].priority;
    }

    priorityScheduling(processes, n);

    return 0;
}
```